Visual Studio 2005 Visual C#

Visual C#

Microsoft Visual C# 2005 (C シャープと読みます) は、.NET Framework で動作するような幅広いアプリケーションを構築するようにデザインされたプログラミング言語です。C# は、シンプルかつ強力で、タイプ セーフのオブジェクト指向言語です。革新的な機能を多数備えた C# を使用すると、C 形式の言語が持つ表現力と簡潔さを維持したままで、アプリケーションの開発速度を向上できます。

Visual Studio は、完全な機能を備えたコード エディタ、プロジェクト テンプレート、デザイナ、コード ウィザード、強力で使いやすいデバッガなどのツールを装備した Visual C# をサポートしています。.NET Framework クラス ライブラリを使用すると、幅広いオペレーティング システム サービスをはじめ、他の有効な設計クラスにアクセスでき、開発サイクルが大幅に短縮されます。

このセクションの内容

Visual C# について

C# 言語または Visual Studio で初めて開発するプログラマ向けに、C# 2.0 の機能を紹介します。また、Visual Studio のヘルプを検索するためのロードマップも説明します。ここには、"操作方法" のページもあります。

Visual C# IDE の使用

Visual C# 開発環境について紹介します。

Visual C# によるアプリケーションの作成

C# と .NET Framework を使用した一般的なプログラミング タスクのわかりやすい概要を紹介し、詳細なドキュメントへのリンクを提供します。

Visual C# への移行

C# 言語を Java や C++ と比較し、Java Language Conversion Assistant を使用して Java アプリケーションおよび Visual J++ アプリケーションを Visual C# に変換する方法について説明します。

C# プログラミング ガイド

C# 言語構成要素の使用方法に関する情報および実際の例を提供します。

C# リファレンス

C# プログラミング概念、キーワード、型、演算子、属性、プリプロセッサ ディレクティブ、コンパイラ スイッチ、およびコンパイラのエラーと警告に関する詳細な参考情報を紹介します。

C# 言語仕様

Microsoft Word 形式で作成された最新バージョンの C# 仕様へのリンクを紹介します。

Visual C# のサンプル

Visual C# を使用したプログラミング方法を示すサンプル ソース コードを示します。

関連するセクション

C# 2.0 言語およびコンパイラの新機能

C# 言語の新機能を説明します。

Visual C# 2005 の新機能

新しいコード エディタ、開発環境、コード ウィザード、およびデバッグ機能について説明します。

Visual C# アプリケーションから Visual Studio 2005 へのアップグレード

既存プロジェクトの Microsoft Visual Studio 2005 への更新について説明します。

参照

その他の技術情報

Visual Studio

Visual C# について

以下のトピックでは、Microsoft Visual C# 2005 を使用してアプリケーションの開発を開始するときに役立つ情報を紹介します。また、C# 言語の Version 2 と共に Microsoft Visual Studio 2005 に組み込まれているさまざまな新機能についても説明します。

このセクションの内容

Visual C# ドキュメントのロードマップ

Visual C# ドキュメントの内容について概説します。

C# 言語と .NET Framework の概要

C# 言語および .NET プラットフォームの概要を示します。

Visual C# 2005 の新機能

Microsoft Visual C# 2005 の新機能について説明します。

C# 2.0 言語およびコンパイラの新機能

C# Version 2.0 の新機能について説明します。

Visual C# アプリケーションから Visual Studio 2005 へのアップグレード

既存プロジェクトの Microsoft Visual Studio 2005 への更新について説明します。

初めて C# アプリケーションを作成する場合

簡単な C# アプリケーションの記述、コンパイルおよび実行について説明します。

C# スタート キットの使用

C# スタート キットの使い方について説明します。

その他の関連資料 (Visual C#)

他のヘルプ リソースへのリンクを示します。

C# での操作方法

さまざまな特定のタスクを実行する方法を説明したトピックへのリンクを示します。

関連するセクション

Visual C# IDE の使用

Visual C# 開発環境の使用ガイドです。

Visual C# への移行

Java アプリケーションと Visual J++ アプリケーションの Visual C# への変換について説明します。

Visual C# によるアプリケーションの作成

C# と .NET Framework を使用した一般的なプログラミング タスクの概要を紹介し、詳細なドキュメントへのリンクを提供します。

C# プログラミング ガイド

C# プログラミングの概念に関する情報を提供し、C# でさまざまなタスクを実行する方法について説明します。

C# リファレンス

C# のキーワード、演算子、プリプロセッサ ディレクティブ、コンパイラ スイッチ、およびコンパイラのエラーと警告に関する詳細なリファレンス情報を紹介します。

Visual C# のサンプル

Visual C# を使用したプログラミング方法を示すサンプル ソース コードを示します。

C# 用語集

C# 用語集です。

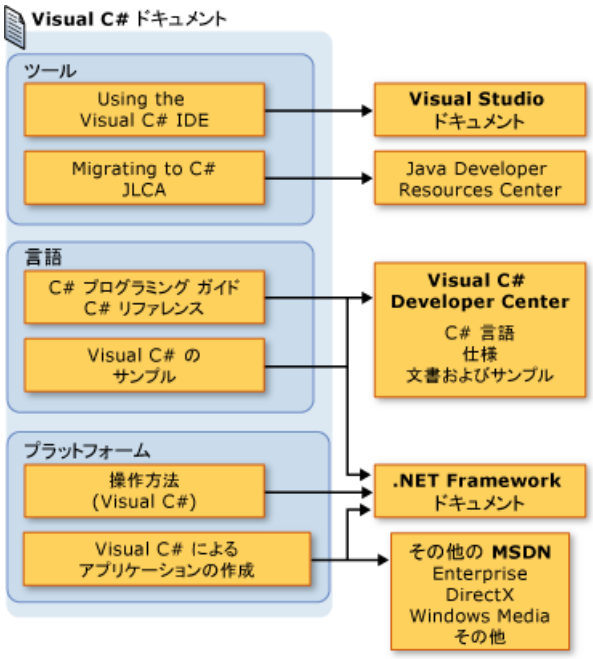
参照

その他の技術情報
Visual C#
Visual Studio

Visual C# ドキュメントのロードマップ

Microsoft Visual C# 2005 ドキュメントには、キーワード、コンパイラ オプション、エラー メッセージ、プログラミングの概念など、C# 言語に固有の情報が含まれます。また、統合開発環境 (IDE) の使用方法の概要も説明されています。さらに、.NET Framework クラス、ASP.NET Web 開発、デバッグ、SQL データベース プログラミングなど、より詳細なヘルプへのリンクも数多く紹介されています。

次の図は、Visual C# ドキュメントの内容を概念的に表し、Visual Studio ドキュメントと MSDN オンラインの関連セクションとの関係を表しています。



C# 固有のドキュメントへのクイック リンク

Visual C# について

C# での操作方法

Visual C# への移行

Visual C# IDE の使用

Visual C# によるアプリケーションの作成

C# リファレンス

Visual C# のサンプル

C# コンパイラ オプション

関連ドキュメントへのクイック リンク

Windows アプリケーションの作成の詳細については、「Windows ベースのアプリケーション、コンポーネント、サービス」を参照してください。

Web アプリケーションの作成の詳細については、「Visual Web Developer」を参照してください。

.NET Framework クラス ライブラリの詳細については、「.NET Framework クラス ライブラリの概要」を参照してください。

.NET Framework の共通言語ランタイム、共通型システム、および他の関連する概念の詳細については、「.NET Framework の概要」を参照してください。

参照

その他の技術情報

Visual Studio

C# 言語と .NET Framework の概要

C# は、タイプ セーフで洗練されたオブジェクト指向言語です。C# を使用すると、.NET Framework で稼動する、安全で信頼性の高いさまざまなアプリケーションを構築できます。C# を使用すると、従来の Windows クライアント アプリケーション、XML Web サービス、分散コンポーネント、クライアント/サーバー アプリケーション、データベース アプリケーションなど、さまざまなアプリケーションを作成できます。Microsoft Visual C# 2005 には、高度なコード エディタ、便利なユーザー インターフェイスのデザイナ、統合デバッガなど多数のツールが用意されています。Version 2.0 の C# 言語と .NET Framework を基にしてアプリケーションを迅速に開発できます。

メモ：
Visual C# のドキュメントは、基本的なプログラミング概念を理解している方を対象に書かれています。初心者である場合、Web で入手できる Visual C# Express Edition が参考になります。C# には参考資料として優れた書籍や Web リソースがいくつかあるため、実践的なプログラミング スキルを身に付けるときに利用することもできます。

C# 言語

C# の構文は表現力が豊かですが、キーワード数は 90 未満です。単純ですぐに覚えることができます。C、C++、または Java に慣れていれば、C# の中かっこ ({}) 構文をすぐに理解できます。これらの言語のいずれかを理解していると、一般に、C# での開発を短期間で始めることができます。C# 構文では、C++ の複雑な部分が数多く簡略化されているだけでなく、null 許容値型、列挙体、デリゲート、匿名メソッド、ダイレクト メモリ アクセスなど、Java にはない強力な機能が実装されました。また C# は、ジェネリック メソッドおよびジェネリック型をサポートしているため、タイプ セーフおよびパフォーマンスが向上し、反復子もサポートしているため、クライアント コードで簡単に使用できるカスタムの反復動作をコレクション クラスの実装側が定義できます。

C# は、オブジェクト指向言語として、カプセル化、継承、およびポリモーフィズムの概念をサポートしています。アプリケーションのエントリ ポイントである `Main` メソッドなど、変数とメソッドのすべてがクラス定義内でカプセル化されています。親クラスから直接継承できるクラスは 1 つのみですが、クラスで実装できるインターフェイスの数は任意です。親クラスの仮想メソッドをオーバーライドする場合、誤って再定義しないように、**override** キーワードを指定する必要があります。C# の構造体はコンパクトなクラスのようなものです。インターフェイスを実装できるスタック割り当て型ですが、継承はサポートされていません。

C# には、基本的なオブジェクト指向の原則とは別に、次のように革新的な言語構成要素が用意されているため、ソフトウェア コンポーネントの開発が容易になります。

- メソッドのシグネチャをカプセル化するデリゲート。タイプ セーフなイベント通知を実行できます。
- プロパティ。プライベート メンバ変数へのアクセサとして機能します。
- 属性。実行時に型に関する宣言のメタデータを提供します。
- インライン XML ドキュメントのコメント。

COM オブジェクトやネイティブの Win32 DLL など、他の Windows ソフトウェアと対話するには、C# で "相互運用" というプロセスを使用します。C# プログラムで相互運用機能を使用すると、ネイティブの C++ アプリケーションで実行できる機能をすべて実行できます。さらに、ダイレクト メモリ アクセスが必須の場合でも、ポインタと "unsafe" コードの概念がサポートされます。

C# のビルド処理は、C や C++ よりも単純で Java よりも柔軟です。ヘッダー ファイルは分かれていません。また、メソッドや型を特定の順序で宣言する必要はありません。C# のソース ファイルでは、クラス、構造体、インターフェイス、およびイベントをいくつでも定義できます。

その他に、C# の参照ドキュメントを紹介します。

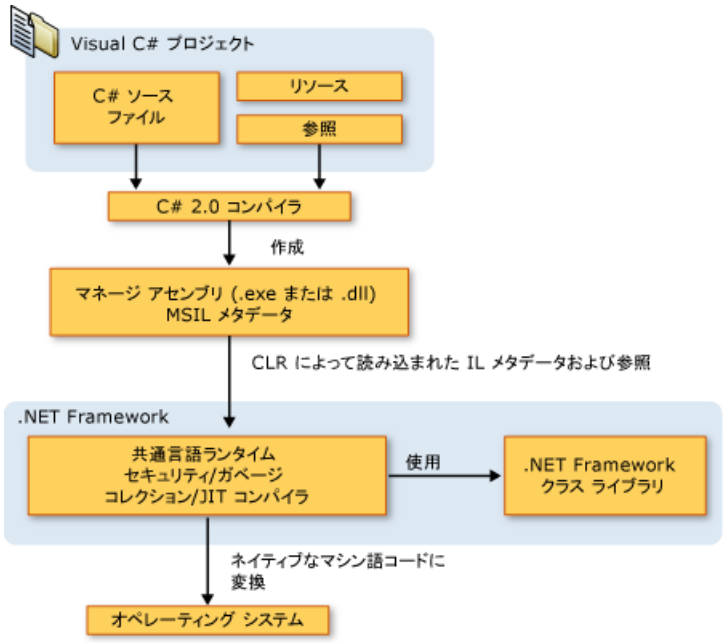
- C# 言語の概略については、「C# 言語仕様」の第 1 章を参照してください。
- C# 言語の具体的な側面の詳細については、「C# リファレンス」を参照してください。
- C# の構文と Java や C++ の構文との比較については、「Java 開発者のための C# プログラミング言語」および「C++ 経験者が C# で開発する場合」を参照してください。

.NET Framework のプラットフォーム アーキテクチャ

C# プログラムは、.NET Framework で実行されます。これは、共通言語ランタイム (CLR) と呼ばれる仮想実行システムや統合的なクラス ライブラリを備えた Windows の統合コンポーネントです。CLR は、共通言語基盤 (CLI) をサポートする Microsoft のシステムです。CLI は、複数の言語やライブラリをシームレスに連携する実行環境と開発環境を構築するための、基本的な国際規格です。

C# で記述したソース コードは、CLI 仕様に準拠する中間言語 (IL) にコンパイルされます。IL コードは、ビットマップや文字列などのリソースと共に、アセンブリと呼ばれる実行可能ファイルのあるディスクに保存されます。アセンブリの拡張子は、一般的に .exe か .dll です。アセンブリに含まれるマニフェストには、アセンブリの種類、バージョン、カルチャ、およびセキュリティ要件に関する情報が規定されています。

C# プログラムを実行すると、アセンブリが CLR に読み込まれ、マニフェストの情報に基づいてさまざまな処理が実行されます。このとき、セキュリティ要件に一致すると、CLR で Just-In-Time (JIT) のコンパイルが実行され、IL コードはネイティブのマシン語命令に変換されます。CLR には、自動的なガベージ コレクション、例外処理、およびリソース管理に関するサービスも用意されています。CLR で実行されるコードは、"マネージ コード" と呼ばれることがあります。反対に、特定のシステムを対象にしたネイティブのマシン語にコンパイルされたコードは、"アンマネージ コード" と呼ばれることがあります。C# ソース コード ファイル、基本クラス ライブラリ、アセンブリ、および CLR について、コンパイル時間と実行時間の関係を次の図に示します。



言語の相互運用性は、.NET Framework の主要機能です。C# コンパイラで生成された IL コードは共通型の仕様 (CTS: Common Type Specification) に準拠しています。そのため、C# の IL コードは、Visual Basic、Visual C++、Visual J# を始めとした 20 を超える CTS 準拠言語で生成されたコードと相互運用性があります。1 つのアセンブリには、異なる .NET 言語で記述されたモジュールを複数含めることができます。また、同じ言語で記述されている場合と同様に、型を参照することもできます。

.NET Framework には、実行時のサービス以外にも、4,000 クラスを超える多数のライブラリが用意されています。このライブラリは、ファイルの入出力、XML 解析のための文字列操作、Windows フォーム コントロールなど、役に立つさまざまな機能を備えた名前空間に構成されています。C# アプリケーションでは、一般に、.NET Framework クラス ライブラリを広範囲に使用して、一般的な "配管工事" のような作業を処理しています。

.NET Framework プラットフォームの詳細については、「.NET Framework の概要」を参照してください。

参照

その他の技術情報

Visual C#

Visual C# によるアプリケーションの作成

Visual C# 2005 の新機能

Microsoft Visual C# 2005 では、次の領域で新機能が追加されました。

- 言語およびコンパイラ
- コード エディタ
- 開発環境
- ドキュメントおよび言語仕様
- デバッグ

言語およびコンパイラ

C# 言語で、ジェネリック型、反復子、および部分型をサポートするようになりました。最新バージョンの C# コンパイラにも、新しい機能とオプションがあります。詳細については、「C# 2.0 言語およびコンパイラの新機能」を参照してください。

コード エディタ

コード エディタでは、Visual C# 2005 向けに次の新機能が追加されました。

コード スニペット

コード スニペットで入力用のテンプレートを用意すると、共通して使用するコード コンストラクタを迅速に追加できます。スニペットは、XML 形式のファイルで格納されているため、編集とカスタマイズが簡単です。

- コード スニペット (C#)
- 方法 : コード スニペットを使用する (C#)
- 方法 : ブロックの挿入コード スニペットを使用する

リファクタリング

リファクタリング ツールを使用すると、ソース コードが自動的に再構築されます。たとえば、ローカル変数をパラメータに昇格した場合や、コードのブロックをメソッドに変換した場合に再構築されます。

- 方法 : ローカル変数をパラメータへ上位変換する
- メソッドの展開
- フィールドのカプセル化
- インターフェイスの展開
- 名前の変更
- パラメータの削除
- パラメータ順序の再変更

開発環境

開発環境では、Visual C# 2005 向けに次の機能が強化されました。

IntelliSense

IntelliSense では、次の新機能が追加されました。

- オブジェクトの前にあるスコープ演算子までカーソルを戻したときや、完了したアクションを元に戻したときに、[メンバの一覧]のリスト全体が自動的に表示されます。
- エラー処理コードを記述するときに [メンバの一覧]を使用すると、try-catch (C# リファレンス) 句のリスト全体から関係のないメンバをフィルタできるため、キャッチする例外を検出するときに役立ちます。
- 標準的なコードを挿入する場合、自動コード生成を使用すると、IntelliSense からコードを選択して挿入できます。
- IntelliSense は、Web アプリケーションを編集するときに使用できます。

クラス デザイナ

クラス デザイナは、クラスや型を図で表示する新しいエディタで、メソッドを追加または修正できます。また、[クラス デザイナ] ウィンドウからリファクタリング ツールを呼び出すこともできます。

- クラスと型のデザインおよび表示 を参照してください。

オブジェクト テスト ベンチ

オブジェクト テスト ベンチは、オブジェクト レベルの簡単なテストを行うために設計されています。オブジェクトのインスタンスを作成し、そのメソッドを呼び出すことができます。

- オブジェクト テスト ベンチ を参照してください。

ClickOnce の配置

ClickOnce の配置を使用すると、Windows アプリケーションを Web サーバーまたはネットワーク ファイル共有に発行して、インストールを簡略化できます。

- ClickOnce の配置 を参照してください。

厳密な名前のアセンブリをサポートするツール

[プロジェクトのプロパティ] ダイアログ ボックスの内容が変更され、アセンブリへの署名がサポートされました。

- プロジェクトのプロパティ を参照してください。

コード ウィザード

次のコード ウィザードは、使用されなくなりました。

- C# メソッド ウィザード
- C# プロパティ ウィザード
- C# フィールド ウィザード
- C# インデクサ ウィザード

ドキュメントおよび言語仕様

C# の関連ドキュメントは大幅に書き換えられ、一般的な情報が詳しくなっただけでなく、開発者が C# でアプリケーションを作成するときに必要となる高度な用法に関する質問も追加されました。

C# 言語仕様は、ヘルプ環境に統合されなくなりましたが、2 つの .doc ファイルで提供されます。これらのファイルは、既定で **\\Microsoft Visual Studio 8\vcsharp\specifications\1033** にインストールされます。最新バージョンは、MSDN の C# Developer Center でダウンロードできます。詳細については、「C# 言語仕様」を参照してください。

C# 固有のデバッグ機能強化

エディット コンティニュなど、C# 開発者に役立つ新機能が追加されました。

- Visual Studio 2005 デバッガの新機能 を参照してください。

参照

概念

Visual Studio 2005 の新機能

その他の技術情報

Visual C#

Visual C# について

Visual C# IDE の使用

C# 2.0 言語およびコンパイラの新機能

Visual Studio 2005 のリリースに伴って、C# 言語は Version 2.0 に更新されました。このバージョンでは、次の新機能がサポートされています。

ジェネリック

Version 2.0 では、ジェネリック型が言語に追加されたため、高度なコードの再利用を実現し、コレクション クラスのパフォーマンスを向上できます。ジェネリック型は、アリティのみが異なる場合があります。また、パラメータを特定の型に強制することもできます。詳細については、「ジェネリック型の型パラメータ (C# プログラミング ガイド)」を参照してください。

反復子

反復子を使用すると、**foreach** ループでコレクションのコンテンツを反復処理する方法をより簡単に指定できます。

部分クラス

部分型定義では、クラスなどの 1 つの型を複数のファイルに分割できます。Visual Studio デザイナでは、この機能を使用して、ユーザー コードから生成されたコードを分割します。

null 許容型

Null 許容型を使用すると、未定義の値を変数に格納できます。Null 許容型は、特定の値を含まない要素が存在する場合があるデータベースやその他のデータ構造体を処理するときに役立ちます。

匿名メソッド

Version 2.0 では、コード ブロックをパラメータとして渡すことが可能になりました。デリゲートが必要なところでは、どこでもコード ブロックを使用できるため、新しいメソッドを定義する必要がありません。

名前空間のエイリアス修飾子

名前空間のエイリアス修飾子 (**::**) は、名前空間メンバへのアクセスの制御を向上できます。global :: エイリアスを使用すると、コード内の要素によって隠されることがあるルート名前空間にアクセスできます。

静的クラス

静的クラスは、インスタンス化できない静的メソッドを含むクラスを宣言するのに安全で便利な方法です。C# Version 1.2 では、クラスをインスタンス化できないようにする場合、クラス コンストラクタをプライベートと定義していました。

外部アセンブリのエイリアス

extern キーワードを拡大利用することで、同じアセンブリに含まれる同一コンポーネントのさまざまなバージョンを参照できます。

プロパティ アクセサのアクセシビリティ

get と **set** の 2 つのアクセサのさまざまなレベルのアクセシビリティをプロパティで定義できます。

デリゲートの共変性と反変性

デリゲートに渡されるメソッドで、戻り値の型とパラメータをより柔軟に使用できます。

方法 : デリゲートを宣言し、インスタンス化して使用する

メソッド グループ変換により、デリゲートの宣言で簡素な構文を使用できます。

固定サイズ バッファ

アンセーフ コード ブロックで、配列が埋め込まれた固定サイズの構造体を宣言できます。

フレンド アセンブリ

非パブリック型へのアクセスをアセンブリから別のアセンブリに提供できます。

インライン警告制御

#pragma 警告ディレクティブを使用すると、特定のコンパイラ警告を有効および無効にできます。

volatile

volatile キーワードを IntPtr および UIntPtr に適用できるようになりました。

C# コンパイラでは、今回のリリースで以下の追加および変更が行われています。

/errorreport オプション

内部コンパイラ エラーをインターネット経由で Microsoft に報告できます。

/incremental オプション

削除されました。

/keycontainer オプションと /keyfile オプション

暗号化キーの指定をサポートします。

/langversion オプション

特定のバージョンの C# 言語との互換性を指定できます。

/linkresource オプション

追加オプションがあります。

/moduleassemblyname オプション

既存のアセンブリで .netmodule ファイルを作成し、非パブリック型にアクセスできます。

/pdb オプション

.pdb ファイルの名前と場所を指定します。

/platform オプション

Itanium ファミリ (IPF) アーキテクチャと x64 アーキテクチャを対象にできます。

#pragma 警告

コードで警告を個別に有効または無効にするために使用します。

参照

概念

C# プログラミング ガイド

その他の技術情報

C# 言語仕様

C# リファレンス

Visual C# アプリケーションから Visual Studio 2005 へのアップグレード

Visual Studio の旧バージョンで作成されたプロジェクト ファイルまたはソース ファイルを開くと、アップグレードウィザードが表示されます。ウィザードの手順に従うと、プロジェクトを Visual Studio 2005 に変換できます。アップグレードウィザードでは、さまざまなタスクが実行されます。たとえば、新しいプロパティと属性の作成、旧式のプロパティと属性の削除などが行われますが、エラー チェック機能が強化されたため、旧バージョンのコンパイラでは発生しなかったエラー メッセージや警告メッセージが表示されることもあります。そのため、既存アプリケーションをアップグレードする最終的な手順は、新しいエラーが解決するようにコードを変更することです。

旧バージョンの C# コンパイラで、あるメッセージが表示されていた場合でも、最新バージョンでは異なるメッセージが表示されることがよくあります。これは、一般に、汎用的なメッセージがより具体的なメッセージに置換されたためです。コードの変更は必要ないため、このような差異については説明していません。

エラー チェックが強化されたためにアップグレードウィザードで生成されるようになった、新しいメッセージを次に示します。

新しいエラー メッセージと警告メッセージ

CS0121: 呼び出しを解決することができません。

暗黙の型変換が行われたため、コンパイラではオーバーロードされたメソッドの一方の形式を呼び出すことができませんでした。このエラーは以下の方法で解決できます。

- 暗黙の変換が行われないような方法でメソッド パラメータを指定します。
- メソッドのオーバーロードをすべて削除します。
- メソッドを呼び出す前に適切な型にキャストします。

CS0122: メソッドはアクセスできない保護レベルになっています。

C++ の **/d1PrivateNativeTypes** コンパイラ オプションでコンパイルしたアセンブリで、型を参照するときにこのエラーを受信することがあります。

最新リリースでは、C++ アセンブリで、パブリックとマークされていない型を使用するシグネチャが生成されるため、このエラーが発生します。

/test:AllowBadRetTypeAccess コマンド ライン オプションを使用すると、この問題を回避できます。この機能を修正したら、このコマンド ライン オプションを削除します。

CS0429: 到達できない式コードが検出されました。

このエラーは、制御の渡らない式がコード中に存在する場合に発生します。たとえば、条件 `false && myTest()` はこの基準に適合します。`&&` 演算子の左側は常に `false` であるため、`myTest()` メソッドは評価されないためです。この問題を修正するには、論理テストを再実行して、制御の渡らない式をなくします。

CS0441: クラスに static と sealed の両方を指定することはできません。

すべての静的クラスはシール クラスでもあります。C# 言語仕様では、1 つのクラスに両方の修飾子を指定することを禁止しています。この場合、コンパイラでエラーとレポートされるようになりました。

このエラーを修正するには、クラスから `sealed` を削除します。

CS1699: アセンブリの署名属性の使用に関する警告

署名を指定するアセンブリの属性は、コードからコンパイラ オプションに移動しました。コードに `AssemblyKeyFile` 属性または `AssemblyKeyName` 属性を使用すると、この警告が表示されるようになりました。

これらの属性の代わりに、次のコンパイラ オプションを使用する必要があります。

- `AssemblyKeyFile` 属性の代わりに `/keyfile` (厳密名キー ファイルの指定) (C# コンパイラ オプション) コンパイラ オプションを使用します。また、`AssemblyKeyName` の代わりに `/keycontainer` (厳密名キー コンテナの指定) (C# コンパイラ オプション) を使用します。

コマンド ライン オプションに切り替えないと、フレンド アセンブリを使用するときにコンパイラの診断の妨げになることがあります。

`/warnaserror` (警告のエラーとしての取り扱い) (C# コンパイラ オプション) を使用している場合、**/warnaserror-:1699** をコンパイラ コマンド ラインに追加することにより、これを警告に変換できます。必要であれば、**/nowarn:1699** を使用して警告をシャット オフすることもできます。

インクリメンタル コンパイルの削除

`/incremental` コンパイラ オプションは削除されました。この機能はエディット コンティニュ機能で置き換えられました。

参照

その他の技術情報

C# コンパイラ オプション

初めて C# アプリケーションを作成する場合

C# アプリケーションの作成に必要な時間はわずか 1 分です。次の手順に従って操作すると、ウィンドウを表示してボタン押下に反応するプログラムが作成されます。

プロシージャ

C# アプリケーションを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [Windows アプリケーション] テンプレートが選択されていることを確認し、[プロジェクト名] フィールドに「**MyProject**」と入力して、[OK] をクリックします。

   Windows フォーム デザイナに [Windows フォーム] が表示されます。これがアプリケーションのユーザー インターフェイスです。
3. [表示] メニューの [ツールボックス] をクリックし、表示されるコントロールのリストを作成します。
4. [コモン コントロール] リストを展開し、[Label] コントロールをフォームにドラッグします。
5. [ツールボックス] の [コモン コントロール] リストにあるボタンを、フォームのラベルの近くにドラッグします。
6. 新しいボタンをダブルクリックすると、コード エディタが表示されます。Visual C# によって、`button1_Click` というメソッドが挿入されています。このメソッドはボタンをクリックしたときに実行されます。
7. メソッドを次のように変更します。

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "Hello, World!";
}
```

8. F5 キーを押して、アプリケーションをコンパイルおよび実行します。

   ボタンをクリックすると、テキスト メッセージが表示されます。これで処理は完了しました。初めての C# アプリケーションが完成しました。

参照

その他の技術情報

Visual C#

Visual C# について

Visual C# エディションのプロジェクト テンプレート

新しいプロジェクトを作成するときに、[新しいプロジェクト] ダイアログ ボックスおよび [新しい Web サイト] ダイアログ ボックスに表示されるアイコンは、使用できるプロジェクトの種類とそのテンプレートを表しています。選択したプロジェクト テンプレートによって、そのプロジェクトに使用できる出力の種類およびその他のオプションが決まります。すべてのプロジェクト テンプレートが Visual C# のすべてのエディションで使用できるとは限りません。

メモ :

Visual C# Express Edition または Visual C# Standard Edition で使用できない機能に関するドキュメントは、これらのエディションのドキュメント セットに含まれている場合があります。

Visual C# プロジェクト テンプレート

Visual Studio のさまざまなエディションで使用できる Visual C# プロジェクト テンプレートを次の表に示します。

テンプレート	Microsoft Visual C# Express Edition	Visual Studio 2005 Standard	Visual Studio 2005 Professional 以上
ASP.NET Web サイト		○	○
ASP.NET Web サービス		○	○
ASP.NET Crystal Reports Web サイト			○
クラス ライブラリ	○	○	○
コンソール アプリケーション	○	○	○
Crystal Reports アプリケーション			○
デバイス アプリケーション		○	○
空のプロジェクト	○	○	○
空の Web サイト		○	○
Excel テンプレート			○
Excel ワークブック			○
ムービー コレクション スタート キット	○	○	○
Outlook アドイン			○
パーソナル Web サイト スタート キット		○	○
Pocket PC 2003: クラス ライブラリ		○	○
Pocket PC 2003: クラス ライブラリ (1.0)		○	○
Pocket PC 2003: コンソール アプリケーション		○	○
Pocket PC 2003: コンソール アプリケーション (1.0)		○	○

Pocket PC 2003: コントロール ライブラリ		○	○
Pocket PC 2003: デバイス アプリケーション		○	○
Pocket PC 2003: デバイス アプリケーション (1.0)		○	○
Pocket PC 2003: 空のプロジェクト		○	○
Pocket PC 2003: 空のプロジェクト (1.0)		○	○
スクリーン セーバー スタート キット	○	○	○
SQL Server プロジェクト			○
Smartphone 2003: クラス ライブラリ (1.0)		○	○
Smartphone 2003: コンソール アプリケーション (1.0)		○	○
Smartphone 2003: デバイス アプリケーション (1.0)		○	○
Smartphone 2003: 空のプロジェクト (1.0)		○	○
テスト プロジェクト			○
Web コントロール ライブラリ		○	○
Windows アプリケーション	○	○	○
Windows CE 5.0: クラス ライブラリ		○	○
Windows CE 5.0: コンソール アプリケーション		○	○
Windows CE 5.0: コントロール ライブラリ		○	○
Windows CE 5.0: デバイス アプリケーション		○	○
Windows CE 5.0: 空のプロジェクト		○	○
Windows コントロール ライブラリ		○	○
Windows サービス			○
Word ドキュメント			○
Word テンプレート			○

参照

概念

インストールとセットアップの基本事項

Visual Studio 2005 の新機能

Visual C# 2005 の新機能

その他の技術情報

Visual C# によるアプリケーションの作成

C# スタート キットの使用

スタート キットは、読み込んでビルドすると、完成したアプリケーションとして単体で使用できます。スタート キットにはドキュメントが付属し、プログラム技術の説明やカスタマイズ方法の提案が記載されています。実行できる C# アプリケーションを確認するには、スタート キットが最適です。

Visual C# スタート キットを読み込み、ビルドするには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。

   [新しいプロジェクト] ダイアログ ボックスが表示されます。このダイアログ ボックスには、Visual C# で作成できる既定の各種アプリケーションが一覧表示されます。

2. アプリケーションの種類を [スタート キット] に変更し、[OK] をクリックします。

   Visual C# にスタート キットが読み込まれます。

3. スタート キットプロジェクトをビルドし、実行するには、F5 キーを押します。

参照

その他の技術情報

Visual C#
Visual C# について

その他の関連資料 (Visual C#)

一般的な問題と特殊な問題に対する回答を見つけるのに役立つ Web サイトやニュースグループを次に示します。

Microsoft のリソース

次に示す Microsoft の Web サイトには、C# 開発に関連するトピックの文書とディスカッション グループが掲載されています。

Web 上

Microsoft Help and Support

サポート技術情報 (KB: Knowledge Base) の文書、ダウンロードや更新用のファイル、Support Webcast、およびその他のサービスへのアクセスを提供します。

Microsoft Visual C# Developer Center

コード サンプル、アップグレード情報、および技術的な情報が用意されています。

MSDN Discussion Groups

世界中の専門家がいるコミュニテイにアクセスする方法を紹介します。

フォーラム

Microsoft Technical Forums

C#、.NET Framework など多数の Microsoft テクノロジに関する Web ベースのディスカッション フォーラム。

ニュースグループ

microsoft.public.dotnet.languages.csharp

Visual C# に関する質問や一般的な議論のためのフォーラムが用意されています。

microsoft.public.vsnet.general

Visual Studio に関する質問や議論のためのフォーラムが用意されています。

microsoft.public.vsnet.ide

Visual Studio 環境での作業について質問したり、意見を交わしたりするフォーラムを提供します。

microsoft.public.vsnet.documentation

Visual C# のドキュメントに関する質問や議論のためのフォーラムが用意されています。

サード パーティのリソース

MSDN のウェブ サイトでは、現在のサード パーティの情報および主要なニュースグループが用意されています。使用できるリソースの最新の一覧については、MSDN コミュニティ Web サイトを参照してください。

参照

その他の技術情報

Visual C#
Visual C# について
Visual Studio のヘルプの使い方
他の開発者との連携
製品のサポートとユーザー補助

C# での操作方法

「操作方法」のページには、C# のプログラミングおよびアプリケーション開発に関連した主要なタスク ベースのトピックが紹介されています。ここでは、C# を使用して実行できる機能の主なカテゴリを紹介します。このリンクから、重要な手順が記載されたヘルプ ページを参照できます。

C# 言語

C# 言語仕様、スレッド処理、ジェネリック、コード例のスニペット、サンプルなどについて説明します。

.NET Framework

ファイル I/O、文字列、コレクション、シリアル化、コンポーネント、アセンブリ、アプリケーション ドメインなどについて説明します。

Windows アプリケーション

Windows アプリケーションのビルド、コントロール、Windows フォーム、描画などについて説明します。

Web ページと Web サービス

ASP.NET Web ページ、XML Web サービスなどについて説明します。

デバッグ

VS デバッガの使い方、.NET Framework Trace クラス、SQL トランザクションのデバッグなどについて説明します。

データ アクセス

データ ソースへの接続、SQL Server、データ バインディングなどについて説明します。

クラスのデザイン

クラス デザイナ、クラスおよびその他の型の処理、型のメンバの作成と変更、クラス ライブラリ デザイン ガイドラインなどについて説明します。

セキュリティ

コード アクセス セキュリティ、セキュリティ ポリシーのベスト プラクティス、アクセス許可セットなどについて説明します。

Office のプログラミング

Office のプログラミング、コントロール、Word、Excel などについて説明します。

スマート デバイス

スマート デバイス プロジェクトの新機能、スマート デバイスのプログラミング、スマート デバイスのデバッグなどについて説明します。

配置

ClickOnce、Windows インストーラ

その他のリソース

以下のサイトにアクセスするには、インターネット接続が必要です。

Visual Studio 2005 Developer Center

Visual Studio 2005 を使用したアプリケーション開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Visual C# Developer Center

C# アプリケーションの開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Microsoft .NET Framework Developer Center

.NET Framework アプリケーションの開発やデバッグに関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

参照

その他の技術情報

Visual C# について

C# 言語 (C# での操作方法)

このページでは、よく使用する C# 言語タスクに関するヘルプへのリンクを紹介します。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「C# での操作方法」を参照してください。

C# 言語

C# 2.0 言語およびコンパイラの新機能

ジェネリック、反復子、匿名メソッド、部分型など、新機能に関する情報を紹介します。

C# スタート キットの使用

Visual C# スタート キットを読み込み、ビルドする方法について説明します。

C# 言語仕様

Microsoft Word 形式で記載された最新バージョンの仕様へのリンクを紹介します。

コマンド ライン

Main() とコマンド ライン引数 (C# プログラミング ガイド)

オブジェクトを作成し、他のメソッドを呼び出す、プログラムのエントリ ポイントである `Main` メソッドについて説明します。C# プログラムでは、エントリ ポイントは 1 つだけに限られます。

方法 : foreach を使用してコマンド ライン引数にアクセスする (C# プログラミング ガイド)

コマンド ライン パラメータにアクセスするコード例を紹介します。

方法 : コマンド ライン引数を表示する (C# プログラミング ガイド)

args 文字列配列を使用して、コマンド ライン引数に表示する方法について説明します。

Main() の戻り値 (C# プログラミング ガイド)

main メソッドで可能な戻り値について説明します。

クラスと継承

base (C# リファレンス)

派生クラスのインスタンスを作成するときに呼び出される、基本クラスのコンストラクタを指定する方法について説明します。

方法 : メソッドに構造体を渡すこととクラス参照を渡すことの違いを理解する (C# プログラミング ガイド)

構造体がメソッドに渡されるときは構造体のコピーが渡されますが、クラス インスタンスが渡されるときは参照が渡されます。その例を紹介します。

インスタンス コンストラクタ (C# プログラミング ガイド)

クラスのコンストラクタと継承について説明します。

方法 : コピー コンストラクタを記述する (C# プログラミング ガイド)

コード例を使用して、クラスのコンストラクタの引数として別のオブジェクトを指定する方法を説明します。

方法 : 構造体間にユーザー定義の変換を実装する (C# プログラミング ガイド)

コード例を使用して、2 つの構造体を定義し、一方からもう一方に変換する方法を説明します。

データ型

ボックス化変換 (C# プログラミング ガイド)

例を使用して、値型とボックス化されたオブジェクトに異なった値を格納できることを示します。

ボックス化解除変換 (C# プログラミング ガイド)

コード例を使用して、ボックス化が無効な場合にエラー メッセージを表示する方法を説明します。

配列

オブジェクトとしての配列 (C# プログラミング ガイド)

コード例を使用して、配列の次元数を表示する方法を説明します。

ジャグ配列 (C# プログラミング ガイド)

コード例を使用して、配列を要素として持つ配列の構築方法を説明します。

パラメータとしての配列の受け渡し (C# プログラミング ガイド)

コード例を使用して、文字配列を初期化し、**PrintArray** メソッドのパラメータとして渡して、その要素を表示する方法を説明します。

ref と out を使用した配列の引き渡し (C# プログラミング ガイド)

コード例を使用して、メソッドに配列を渡すときに使用される **out** と **ref** の違いを説明します。

プロパティ

方法 : 読み取り/書き込みプロパティを宣言および使用する (C# プログラミング ガイド)

読み取り/書き込みプロパティの宣言方法と使用方法がわかる例を紹介します。

方法 : 抽象プロパティを定義する (C# プログラミング ガイド)

抽象プロパティを定義するコード例を紹介します。

メソッド

値型のパラメータの引き渡し (C# プログラミング ガイド)

コード例を使用して、値型を渡すさまざまな方法を説明します。

参照型のパラメータの引き渡し (C# プログラミング ガイド)

コード例を使用して、参照型を渡すさまざまな方法を説明します。

イベント

方法 : イベント サブスクリプションとサブスクリプションの解除 (C# プログラミング ガイド)

フォーム、ボタン、リスト ボックスなど、他のクラスによって公開されたイベントにサブスクライブする方法を説明します。

方法 : .NET Framework ガイドラインに準拠したイベントを発行する (C# プログラミング ガイド)

EventHandler および EventHandler<T> に基づいてイベントを作成する方法を説明します。

方法 : インターフェイス イベントを実装する (C# プログラミング ガイド)

インターフェイス内に宣言されたイベントの実装方法を説明します。

方法 : ディクショナリを使用してイベント インスタンスを格納する (C# プログラミング ガイド)

ハッシュ テーブルを使用して、イベントのインスタンスを格納する方法について説明します。

方法 : 派生クラスから基本クラス イベントを発生させる (C# プログラミング ガイド)

保護された仮想メソッド内の基本クラス イベントをラップして、派生クラスから呼び出せるようにする方法を説明します。

インターフェイス

方法 : インターフェイス メンバを明示的に実装する (C# プログラミング ガイド)

明示的にインターフェイスを実装するクラスを宣言する方法、およびインターフェイスのインスタンスによってメンバにアクセスする方法について説明します。

方法 : 継承を使用してインターフェイス メンバを明示的に実装する (C# プログラミング ガイド)

ボックスの大きさをメートル法とヤード ポンド法の両方の単位で表示する例を紹介します。

ジェネリック

An Introduction to C# Generics

ジェネリックを使用するとタイプ セーフなコレクション クラスを定義できることを説明します。ジェネリック クラスは、一度実装すると、任意の型に宣言して使用できます。

.NET Framework におけるジェネリック

System.Collections.Generic 名前空間の新しいジェネリック コレクションの機能と使用方法について説明します。

ジェネリック コードの default キーワード (C# プログラミング ガイド)

型パラメータに既定のキーワードを使用するコード例を紹介します。

ジェネリック メソッド (C# プログラミング ガイド)

ジェネリック メソッドを宣言する構文を紹介します。また、アプリケーションでジェネリック メソッドを使用する例を紹介します。

型パラメータの制約 (C# プログラミング ガイド)

ジェネリック クラスのインスタンス化に使用する型のメソッドやプロパティへのアクセスを可能にする型パラメータを制約する方法を紹介します。

汎用デリゲート (C# プログラミング ガイド)

汎用デリゲートを宣言する構文を紹介します。また、汎用デリゲートのインスタンス化および使用に関する重要な注意についても説明し、コード例も紹介します。

名前空間

方法 : 名前空間エイリアス修飾子を使用する (C# プログラミング ガイド)

グローバル名前空間のメンバが同名の別のエンティティによって隠される可能性がある場合の、そのメンバへのアクセス機能について説明します。

反復子

方法 : ジェネリック リストの反復子ブロックを作成する (C# プログラミング ガイド)

整数の配列を使用して SampleCollection リストを構築する例を紹介します。for ループは、コレクションを反復処理して各項目の値を生成します。次に、**foreach** ループを使用してコレクションの項目を表示します。

方法 : ジェネリック リストの反復子ブロックを作成する (C# プログラミング ガイド)

ジェネリック クラス Stack<T> でジェネリック インターフェイス IEnumerator<T> を実装する例を紹介します。Push メソッドを使って、T 型の配列が宣言され、値が割り当てられます。GetEnumerator メソッドでは、発生した return ステートメントを使用して配列の値が戻されます。

デリゲート

方法 : デリゲートを結合する (マルチキャスト デリゲート) (C# プログラミング ガイド)

マルチキャストのデリゲートを構成する例を紹介します。

方法 : デリゲートを宣言し、インスタンス化して使用する (C# プログラミング ガイド)

デリゲートの宣言、インスタンス化、および使用方法を示す例を紹介します。

演算子のオーバーロード

方法 : 演算子のオーバーロードを使用して複素数クラスを作成する (C# プログラミング ガイド)

演算子のオーバーロードを使用して、複素数の加算を定義する複素数クラス Complex を作成する方法について説明します。

相互運用性

方法 : COM 相互運用機能を使用して Word によるスペル チェックを行う (C# プログラミング ガイド)

次の例は、C# アプリケーションで Word のスペル チェック機能を使用する方法を示しています。

方法 : COM 相互運用機能を使用して Excel スプレッドシートを作成する (C# プログラミング ガイド)

.NET Framework の COM 相互運用機能を使用して、既存の Excel スプレッドシートを C# で開く方法について例を挙げて説明します。

方法 : Excel 用のオートメーション アドインとしてマネージ コードを使用する (C# プログラミング ガイド)

Excel のワークシートのセルで、所得税率を計算する C# アドインを作成する方法について例を挙げて説明します。

方法 : プラットフォーム呼び出しを使用して Wave ファイルを再生する (C# プログラミング ガイド)

プラットフォーム呼び出しサービスを利用して、Windows プラットフォームで Wave サウンド ファイルを再生する方法について例を挙げて説明します。

アンセーフ コード

方法 : ポインタを使用してバイトの配列をコピーする (C# プログラミング ガイド)

ポインタを使用して配列間でバイトをコピーする方法について説明します。

方法 : Windows の ReadFile 関数を使用する (C# プログラミング ガイド)

Windows の ReadFile 関数を呼び出す方法について説明します。この関数は、パラメータとして読み取りバッファのポインタを要求するため、unsafe コンテキストを使用する必要があります。

スレッド処理

スレッドの使用とスレッド処理

マネージ スレッドの作成と管理、および意図しない結果を防ぐ方法を説明したトピックへのリンクを紹介します。

方法 : スレッドを作成および終了する (C# プログラミング ガイド)

スレッドの作成と開始の方法、および同一プロセス内で同時実行する 2 つのスレッド間の対話について説明する例を紹介します。

スレッドの作成と開始の方法、および同一プロセス内で同時実行する 2 つのスレッド間の対話について説明する例を紹介します。

方法 : producer スレッドと consumer スレッドを同期する (C# プログラミング ガイド)

C# の lock キーワードと Monitor オブジェクトの Pulse メソッドを使用して同期をとる例を紹介します。

方法 : スレッド プールを使用する (C# プログラミング ガイド)

スレッド プールを使用する例を紹介します。

文字列

方法 : 正規表現を使用して文字列を検索する (C# プログラミング ガイド)

Regex クラスを使用して文字列を検索する方法について説明します。単純な検索から、正規表現を活用した複雑な検索まで、さまざまな検索があります。

方法 : 複数の文字列を連結する (C# プログラミング ガイド)

複数の文字列を結合するコード例を紹介します。

方法 : String のメソッドを使用して文字列を検索する (C# プログラミング ガイド)

String メソッドを使用して、文字列を検索するコード例を紹介します。

方法 : Split メソッドを使用して文字列を解析する (C# プログラミング ガイド)

System.String.Split メソッドを使用して、文字列を解析するコード例を紹介します。

方法 : 文字列の内容を変更する (C# プログラミング ガイド)

コード例を使用して、文字列の内容を抽出して配列化し、この配列の要素の一部に変更を加える方法を説明します。

属性

方法 : 属性を使用して C/C++ の共用体を作成する (C# プログラミング ガイド)

Serializable 属性を使用して、固有の特性をクラスに適用する例を紹介します。

DLL の操作

方法 : C# DLL を作成して使用する (C# プログラミング ガイド)

例を挙げて、DLL の構築方法と使用方法について説明します。

アセンブリ

方法 : ファイルがアセンブリであるかどうかを確認する (C# プログラミング ガイド)

DLL がアセンブリであるかどうかをテストして確認する例を紹介します。

方法 : アセンブリを読み込み、アンロードする (C# プログラミング ガイド)

実行時に、現在のアプリケーション ドメインへ固有のアセンブリを読み込むことができるようにする方法について説明します。

方法 : アセンブリを他のアプリケーションと共有する (C# プログラミング ガイド)

アセンブリを他のアプリケーションと共有する方法について説明します。

アプリケーション ドメイン

別のアプリケーション ドメインでのコードの実行 (C# プログラミング ガイド)

他のアプリケーション ドメインに読み込まれたアセンブリを実行する方法について説明します。

方法 : アプリケーション ドメインを作成し、使用する (C# プログラミング ガイド)

演算子のオーバーロードを使用して、3 つの値を持つ論理型を実装する方法について説明します。

サンプル

Visual C# のサンプル

「Hello World サンプル」から「ジェネリックのサンプル (C#)」まで、サンプル ファイルを開いたりコピーしたりできるリンクを紹介します。

参照

概念

C# での操作方法

.NET Framework (C# での操作方法)

このページでは、よく使用する .NET Framework タスクに関するヘルプへのリンクを紹介します。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「C# での操作方法」を参照してください。

全般

C# 言語と .NET Framework の概要

C# 言語と、.NET Framework クラス ライブラリおよびランタイム実行エンジンの関係について説明します。

.NET Framework の概要

共通言語ランタイム、.NET Framework クラス ライブラリ、言語間の相互運用性などの .NET Framework の主要な機能の概念について説明します。

技術のクイック リファレンス

.NET Framework の主なテクノロジ領域に関するクイック リファレンスです。

ファイル I/O

方法 : ディレクトリー覧を作成する

新しいディレクトリを作成します。

方法 : 新しく作成されたデータ ファイルに対して読み書きする

新規に作成されたデータ ファイルを読み書きする方法です。

方法 : ログ ファイルを開いて情報を追加する

ログ ファイルを開き、追記する方法です。

方法 : ファイルにテキストを書き込む

ファイルにテキストを書き込む方法です。

方法 : ファイルからテキストを読み取る

ファイルのテキストを読み取る方法です。

方法 : 文字列から文字を読み取る

文字列の文字を読み取る方法です。

方法 : 文字列に文字を書き込む

文字列に文字を書き込む方法です。

方法 : アクセス制御リスト エントリを追加または削除する

セキュリティを強化するために、アクセス制御リスト (ACL) エントリを追加または削除する方法です。

文字列

新しい文字列の作成

新しい文字列の作成方法です。

文字のトリムと削除

文字列の先頭または末尾から文字を削除する方法です。

文字列の埋め込み

文字列の先頭または末尾にタブや空白を追加する方法です。

文字列の比較

2 つの文字列が等しいかどうかを比較する方法です。

大文字と小文字の変更

大文字を小文字に、小文字を大文字に変更する方法です。

StringBuilder クラスの使用

効率的な文字列操作の手法です。

方法 : 基本的な文字列操作を使用して文字列操作を実行する

文字列を分割する方法、文字列を別の文字列に追加する方法などについて説明します。

方法 : System.Convert を使用してデータ型を変換する

Convert クラスを使用して、文字列値をブール値に変換する例を紹介します。

方法 : 文字列から無効な文字を取り除く

静的な Regex.Replace メソッドを使用して、文字列から無効な文字を取り除く例を紹介します。

方法 : 文字列が有効な電子メール形式であるかどうかを検証する

静的な Regex.IsMatch メソッドを使用して、文字列が有効な電子メール形式かどうかを検証する例を紹介します。

コレクション

コレクションとデータ構造体

.NET Framework コレクション クラスの概要です。

コレクション クラスの選択

使用するコレクションの型を選択する方法です。

ジェネリック コレクションを使用する状況

ジェネリック以外のコレクション クラスと比べた場合のジェネリック コレクション クラスの利点について説明します。

System.Collections.Generic

ジェネリック コレクション クラスへのポータル ページです。

List

List<T> コレクションに対して項目を追加および削除する方法を示すコード例を紹介します。

SortedDictionary

SortedDictionary<K,V> コレクションに対してキー/値ペアを追加および削除する方法を示すコード例を紹介します。

例外

方法 : catch ブロックで特定の例外を使用する

try/catch ブロックを使用して InvalidCastException をキャッチする例を紹介します。

方法 : Try ブロックと Catch ブロックを使用して例外をキャッチする

try/catch ブロックを使用して例外をキャッチする例を紹介します。

方法 : ユーザー定義の例外を作成する

新しい例外クラス **EmployeeListNotFoundException** が **Exception** から派生する例を紹介します。

方法 : finally ブロックを使用する

try/catch ブロックを使用して **ArgumentOutOfRangeException** 例外をキャッチする例を紹介します。

方法 : 例外を明示的にスローする

try/catch ブロックを使用して **FileNotFoundException** 例外をキャッチする例を紹介します。

イベント

方法 : Windows フォーム アプリケーションでイベントを利用する

Windows フォームでボタン クリック イベントを処理する例を紹介します。

方法 : イベント ハンドラ メソッドをイベントに接続する

イベントにイベント ハンドラ メソッドを追加する例を紹介します。

方法 : イベントを発生させる/処理する

「イベントとデリゲート」および「イベントの発生」に詳細に説明されている概念を使用した例を紹介します。

方法 : イベント プロパティを使用して複数のイベントを処理する

イベント プロパティを使用して、複数イベントを処理する例を紹介します。

方法 : クラスにイベントを実装する

クラスにイベントを実装する手順について説明します。

デバッグ

デバッグ (C# での操作方法) を参照してください。

配置

「セキュリティ (C# での操作方法)」を参照してください。

サービス コンポーネント

方法 : コンペンセート リソース マネージャ (CRM) を作成する

コンペンセート リソース マネージャの作成方法を示すコード例を紹介します。

方法 : サービス コンポーネントを作成する

新しいサービス コンポーネントを作成する手順について説明します。

方法 : アセンブリに Description 属性を適用する

DescriptionAttribute 属性を適用して、アセンブリの説明を設定する方法について説明します。

方法 : SetAbort メソッドと SetComplete メソッドを使用する

ContextUtil クラスの静的な **SetComplete** メソッドと **SetAbort** メソッドの使用方法について説明します。

方法 : アセンブリに ApplicationID 属性を適用する

ApplicationID 属性をアセンブリに適用する方法について説明します。

方法 : プールされるオブジェクトを作成しサイズ制限とタイムアウト制限を設定する

プール オブジェクトを作成して、そのサイズとタイムアウト制限を設定する方法について説明します。

方法 : 自動トランザクションを使用する Web サービス メソッドを作成する

自動的なトランザクションを使用する Web サービス メソッドの作成方法について説明します。

方法 : アプリケーションの SoapRoot プロパティを設定する

SoapVRoot プロパティを "MyVRoot" に設定する方法について説明します。

方法 : トランザクション タイムアウトを設定する

トランザクション タイムアウトを 10 秒に設定する方法について説明します。

方法 : ApplicationName 属性を使用してアプリケーション名を設定する

アセンブリ レベルの **ApplicationName** 属性を使用して、アプリケーション名を指定する方法について説明します。

方法 : COM+ の BYOT (Bring Your Own Transaction) 機能を使用する

ServicedComponent クラスの派生クラスから COM+ の BYOT 機能を使用して、分散トランザクション コーディネータ (DTC) にアクセスする手順について説明します。

方法 : プライベート コンポーネントを作成する

クラスで **PrivateComponentAttribute** 属性を使用する方法について説明します。

方法 : アプリケーションのアクティベーション タイプを設定する

アクティベーションの種類を "server" に設定する方法について説明します。

方法 : クラスのインスタンスで同期を有効にする

TestSync クラスのインスタンスで、同期を有効にする方法について説明します。

方法 : .NET Framework クラスで自動トランザクションを使用する

自動的なトランザクションに参加するクラスを用意する方法について説明します。

方法 : JIT アクティベーションを有効にする

クラス上およびクラス外で、JIT アクティベーションと非アクティベーションを有効にする方法について説明します。

方法 : トランザクションに関与するクラスで AutoComplete 属性を設定する

トランザクション対応クラスに **AutoComplete** 属性を配置する方法について説明します。

方法 : 非同期でメッセージを表示するキュー コンポーネントを実装する

メッセージを非同期に表示する、キューのコンポーネントを実装する方法について説明します。

方法 : 疎結合イベントを実装する

一般的なイベント インターフェイス、およびイベントを発生させる発行側を備えた、イベント クラスとイベント シンクを実装する手順について説明します。

方法 : オブジェクト構築を構成する

オブジェクト構築を構成し、**TestObjectConstruct** クラスの既定の初期化文字列を文字列 "Initial Catalog=Northwind;Data Source=.\\SQLServerInstance;Trusted_Connection=yes" に設定する手順と例について説明します。

アセンブリとアプリケーション ドメイン

方法 : アセンブリから型およびメンバの情報を取得する

アセンブリから型情報とメンバ情報を取得する例を紹介します。

方法: シングルファイル アセンブリをビルドする

コマンド ライン コンパイラを使用して、シングルファイル アセンブリを作成する手順について説明します。

方法 : アプリケーション ドメインを作成する

MyDomain という名前の新しいアプリケーション ドメインを作成し、ホスト ドメイン名と新しく作成された子アプリケーション ドメイン名をコンソールに出力します。

方法 : アセンブリの完全修飾名を特定する

指定したクラスを含むアセンブリの完全修飾名をコンソールに表示する方法について説明します。

方法 : アプリケーション ドメインを構成する

AppDomainSetup クラスのインスタンスを作成し、このクラスを使用して新しいアプリケーション ドメインを作成し、情報をコンソールに書き込んでから、このアプリケーション ドメインをアンロードします。

方法 : アセンブリの内容を表示する

基本の "Hello, World" プログラムから始める例を紹介します。また、Ildasm.exe を使用して、Hello.exe アセンブリを逆アセンブルする方法、およびアセンブリ マニフェストを表示する方法について説明します。

方法 : 厳密な名前のアセンブリを参照する

myAssembly.cs というコード モジュールから、myLibAssembly.dll という厳密な名前付きのアセンブリを参照する、myAssembly.dll というアセンブリを作成します。

方法 : アプリケーション ドメインをアンロードする

MyDomain という新しいアプリケーション ドメインを作成し、所定の情報をコンソールに出力してから、アプリケーション ドメインをアンロードします。

方法 : グローバル アセンブリ キャッシュからアセンブリを削除する

hello.dll というアセンブリをグローバル アセンブリ キャッシュから削除する例を紹介します。

方法 : グローバル アセンブリ キャッシュにアセンブリをインストールする

ファイル名 hello.dll のアセンブリをグローバル アセンブリ キャッシュにインストールする例を紹介します。

方法 : マルチファイル アセンブリをビルドする

マルチファイル アセンブリを作成するために使用する手順について説明し、各手順の内容を示す完全な例を紹介します。

方法 : アプリケーション ドメインにアセンブリを読み込む

現在のアプリケーション ドメインにアセンブリを読み込み、そのアセンブリを実行する例を紹介します。

方法 : 厳密な名前でアセンブリに署名する

キー ファイル sgKey.snk を使用して、厳密な名前でアセンブリ MyAssembly.dll に署名する例を紹介します。

方法 : グローバル アセンブリ キャッシュの内容を表示する

グローバル アセンブリ キャッシュ ツール (Gacutil.exe) を使用して、グローバル アセンブリ キャッシュの内容を表示する方法について説明します。

方法 : 公開キーと秘密キーのキー ペアを作成する

アセンブリに厳密な名前で署名する方法、および厳密名ツール (Sn.exe) を使用してキー ペアを作成する方法について説明します。

相互運用

方法 : タイプ ライブラリを Win32 リソースとして .NET ベースのアプリケーションに埋め込む

タイプ ライブラリを Win32 リソースとして .NET Framework ベースのアプリケーションに埋め込む方法について説明します。

方法 : Tlbimp.exe を使用してプライマリ相互運用機能アセンブリを生成する

Tlbimp.exe を使用して、プライマリ相互運用機能アセンブリを生成する例を紹介します。

方法 : 手動でプライマリ相互運用機能アセンブリを作成する

プライマリ相互運用機能アセンブリを手動で作成する例を紹介します。

方法 : 相互運用機能アセンブリをタイプ ライブラリから生成する

タイプ ライブラリから相互運用機能アセンブリを生成する例を紹介します。

方法 : COM シンクによって処理されるイベントを発生させる

イベント ソースとしてのマネージ サーバーと、イベント シンクとしての COM クライアントの例を紹介します。

方法 : ランタイム呼び出し可能ラッパーをカスタマイズする

IDL ソースを変更するかインポートしたアセンブリを変更して、ランタイム呼び出し可能ラッパーをカスタマイズする方法について説明します。

方法 : 登録を必要としないアクティベーション用の .NET ベースのコンポーネントを構成する

登録を必要としないアクティベーション用の .NET Framework ベースのコンポーネントを構成する方法について説明します。

方法 : コールバック関数を実装する

マネージ アプリケーションが、プラットフォーム呼び出しを使用して、ローカル コンピュータ上の各ウィンドウのハンドル値を出力する方法について説明します。

方法 : HRESULT に例外を割り当てる

NoAccessException という新しい例外クラスを作成し、HRESULT E_ACCESSDENIED にマップする例を紹介します。

方法 : 相互運用機能アセンブリを編集する

マーシャリングの変更内容を Microsoft Intermediate Language (MSIL) で指定する方法について説明します。

方法 : タイプ ライブラリへの参照を追加する

タイプ ライブラリへの参照を追加する手順について説明します。

方法 : COM ソースによって発生したイベントを処理する

Internet Explorer ウィンドウを開いて、**InternetExplorer** オブジェクトによって発生したイベントをマネージ コードで実装されたイベント ハンドラに結び付ける例を紹介します。

方法 : ラッパを手動で作成する

IDL に含まれる **ISATest** インターフェイスおよび **SATest** クラスの例と、C# ソース コードのそれらに対応する型の例を紹介します。

方法 : プライマリ相互運用機能アセンブリを登録する

CompanyA.UtilLib.dll プライマリ相互運用機能アセンブリを登録する例を紹介します。

方法 : 複数のバージョンのタイプ ライブラリをラップする

複数のバージョンのタイプ ライブラリをラップする方法について説明します。

セキュリティ

「セキュリティ (C# での操作方法)」を参照してください。

シリアル化

方法 : オブジェクトを逆シリアル化する

オブジェクトをファイルに逆シリアル化する例を紹介します。

方法 : XML スキーマ定義ツールを使用してクラスおよび XML スキーマ ドキュメントを生成する

XML スキーマ定義ツールを使用して、クラスと XML スキーマ ドキュメントを生成する手順について説明します。

方法 : XML ストリームの代替要素名を指定する

クラスのセットが同じ XML ストリームを複数生成する方法について説明します。

方法 : 派生クラスのシリアル化を制御する

派生クラスのシリアル化を制御する例を紹介します。

方法 : オブジェクトを SOAP エンコード済み XML ストリームとしてシリアル化する

オブジェクトを SOAP でエンコードされた XML ストリームとしてシリアル化する手順と例を紹介します。

方法 : シリアル化されたデータをチャンクする

サーバー側のチャンクとクライアント側の処理を実装する手順と例を紹介します。

方法 : オブジェクトをシリアル化する

オブジェクトをシリアル化する手順について説明します。

方法 : XML 要素および XML 属性名を修飾する

XML ドキュメントで修飾名を作成する手順と例を紹介します。

方法 : エンコード済みの SOAP XML シリアル化をオーバーライドする

オブジェクトのシリアル化を SOAP メッセージとしてオーバーライドする手順と例を紹介します。

エンコーディングとローカリゼーション

方法 : Unicode の数字を解析する

Decimal.Parse メソッドを使用して、複数の筆記文字の数字を指定する Unicode コード値から成る文字列を解析する例を紹介します。

方法: カスタム カルチャを作成する

カスタムのカルチャを定義し、作成する手順について説明します。

詳細なプログラミング方法

方法 : 動的メソッドを定義および実行する

単純な動的メソッド、およびクラスのインスタンスにバインドされた動的メソッドを定義し、実行する方法について説明します。

方法 : リフレクションを使用してジェネリック型をチェックおよびインスタンス化する

ジェネリック型の検出と操作の手順について説明します。

方法 : リフレクション出力を使用してジェネリック メソッドを定義する

リフレクション出力ありのジェネリック メソッドを定義する手順について説明します。

方法 : 完全署名を使用して動的アセンブリに厳密な名前を指定する

完全署名を使用して、動的アセンブリに厳密な名前を指定する方法について説明します。

方法 : リフレクションのみのコンテキストにアセンブリを読み込む

アセンブリをリフレクション専用のコンテキストに読み込む手順とコード例を紹介します。

方法 : リフレクション出力を使用してジェネリック型を定義する

2 つの型パラメータを持つ単純なジェネリック型を作成する方法、その型パラメータに対してクラスの制約、インターフェイスの制約、および特殊な制約を適用する方法、その型パラメータを持つクラスを、パラメータの型および戻り値の型として使用するメンバを作成する方法について説明します。

.NET Framework チュートリアル

チュートリアル : Windows フォーム コンポーネントへのスマート タグの追加

標準の Windows フォームの Label コントロールから派生した ColorLabel という名前の単純なサンプル コントロールのコードを使用して、ス

マート タグを追加する方法について説明します。

チュートリアル : SOAP 拡張機能を使用した SOAP メッセージの変更

SOAP 拡張を構築し、実行する方法について説明します。

チュートリアル : ASP.NET を使用した基本的な XML Web サービスの構築

ASP.NET を使用して、基本的な XML Web サービスを構築する方法について説明します。

チュートリアル : 特定のデバイスを対象とした ASP.NET モバイル Web ページのカスタマイズ

特定のデバイス向けにカスタマイズする方法について説明します。

チュートリアル : サービスの説明とプロキシ クラスの生成のカスタマイズ

サービスの説明とプロキシ クラスの生成をカスタマイズする方法について説明します。

チュートリアル : ClickOnce アプリケーションを手動で配置する

マニフェストの生成および編集ツール (Mage: Manifest Generation and Editing tool) のコマンド ライン バージョンまたは GUI バージョンを使用して、完全な ClickOnce 配置を作成するために必要な手順について説明します。

チュートリアル : ClickOnce 配置 API を使用して必要に応じてアセンブリをダウンロードする

アプリケーション内の特定のアセンブリに "オプション" マークを付ける方法、および共通言語ランタイム (CLR: Common Language Runtime) によって要求されたときに、**System.Deployment.Application** 名前空間にあるクラスを使用して、それらのアセンブリをダウンロードする方法について説明します。

チュートリアル : UI 型エディタの実装

PropertyGrid を使用してカスタム型用に独自の UI 型エディタを作成し、編集用のインターフェイスを表示する方法について説明します。

その他のリソース

Visual Studio 2005 Developer Center

Visual Studio 2005 を使用したアプリケーション開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Visual C# Developer Center

C# アプリケーションの開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Microsoft .NET Framework Developer Center

.NET Framework アプリケーションの開発やデバッグに関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

参照

概念

C# での操作方法

Windows アプリケーション (C# での操作方法)

このページでは、よく使用する Windows アプリケーション タスクに関するヘルプへのリンクを紹介します。よく使用するタスクのその他のカテゴリに関するヘルプについては、「C# での操作方法」を参照してください。

一般

Windows ベース アプリケーションの概要

Microsoft Visual Studio 2005 を使用して作成できる Windows アプリケーションの概要について説明します。

Windows フォームと Web フォーム間の選択

どちらのフォームがアプリケーションに最適であるかを判断するうえで役立つ各テクノロジの機能と特徴について説明します。

フォームの操作

Windows フォーム デザイナ

方法 : Windows アプリケーションでスタートアップ フォームを選択する

Windows アプリケーションのスタートアップ フォームを設定する方法について説明します。

方法 : Windows フォームの 1 つのイベント ハンドラに複数のイベントを関連付ける

Windows フォーム アプリケーションで、C# の [プロパティ] ウィンドウの [イベント] ビューを使用して、複数のイベントを 1 つのイベント ハンドラに接続する方法について説明します。

方法 : Windows フォームでマルチペイン ユーザー インターフェイスを作成する

Microsoft Outlook で使用されるユーザー インターフェイスと同様に、[フォルダ] リスト、[メッセージ] ペイン、および [プレビュー] ペインという複数のペインを備えたマルチペイン ユーザー インターフェイスを作成する方法について説明します。

方法 : Windows フォームに背景イメージを追加する

コントロールまたはフォーム自体に背景イメージを配置する方法について説明します。[プロパティ] ウィンドウを使用すると簡単に実行できます。

方法 : デザイン時に Windows フォームのコントロールにツールヒントを設定する

ToolTip 文字列をコード、または Windows フォーム デザイナで設定する方法について説明します。

方法 : Windows フォームに ActiveX コントロールを追加する

Windows フォームに ActiveX コントロールを配置する方法について説明します。

方法 : Windows フォーム コントロールのアクセス キーを作成する

メニュー、メニュー項目、またはコントロール (ボタンなど) のラベルにアクセス キーを作成する方法について説明します。

実行時の Windows フォームの操作

方法 : コントロールのコレクションに対して実行時にコントロールを追加または削除する

フォーム上にあるコンテナ コントロールで、コントロールの追加やコントロールの削除を行うなど、アプリケーション開発で一般的なタスクについて説明します。

方法 : Windows XP の Visual スタイルを有効にする

Windows フォームのクライアント領域で visual スタイルを作成する方法について説明します。

方法 : スタートアップ Windows フォームを非表示にする

アプリケーションの起動時に、Windows ベースのアプリケーションのメイン フォームを非表示にする方法について説明します。

方法 : Windows フォームを常に一番手前に表示する

デザイン時またはプログラムで、Windows フォーム アプリケーション内で一番手前に表示されるようにフォームを設定する方法について説明します。

方法 : Windows フォームをモーダルおよびモードレスで表示する

フォームをモーダル ダイアログ ボックスまたはモードレス ダイアログ ボックスとして表示する方法について説明します。

コントロール

TextBox コントロール

方法 : Windows フォーム TextBox コントロールでテキストを選択する

Windows フォーム TextBox コントロールを使用して、プログラムによってテキストを選択する方法について説明します。

方法 : 文字列に引用符を挿入する (Windows フォーム)

テキストの文字列へ引用符 (" ") を挿入する方法について説明します。

方法 : 読み取り専用テキスト ボックスを作成する (Windows フォーム)

編集可能な Windows フォーム テキスト ボックスを読み取り専用コントロールに変換する方法について説明します。

方法 : Windows フォームの TextBox コントロールを使用してパスワード テキスト ボックスを作成する

Windows フォームの TextBox コントロールで、パスワードのテキスト ボックスを作成する方法について説明します。

方法 : Windows フォーム TextBox コントロールでのカーソル位置を制御する

TextBox コントロールのカーソル位置を制御する方法について説明します。

方法 : MaskedTextBox コントロールにデータをバインドする

MaskedTextBox コントロールにデータをバインドする方法について説明します。

チュートリアル : MaskedTextBox コントロールの使用

次のタスクの実行方法について説明します。

MaskedTextBox コントロールの初期化

文字がマスクに準拠していないとき、ユーザーに警告

ユーザー入力の値が型と適合しないとき、ユーザーに警告

RichTextBox コントロール

方法 : Windows フォームの RichTextBox コントロールにファイルを読み込む

ファイルを Windows フォームの RichTextBox コントロールに読み込む方法について説明します。このコントロールは、プレーンテキスト、Unicode のプレーンテキスト、またはリッチ テキスト形式 (RTF: Rich Text Format) のファイルを表示できます。

方法 : Windows フォームの RichTextBox コントロールにスクロール バーを表示する

RichTextBox コントロールの ScrollBars プロパティに使用できる 7 つの値を表で説明します。

方法 : Windows フォームの RichTextBox コントロールのフォント属性を設定する

SelectionFont プロパティを使用して、選択した文字を太字や斜体にしたり、下線を付けたりする方法について説明します。

方法 : Windows フォームの RichTextBox コントロールを使用してインデント、ぶら下げインデント、および箇条書き段落を設定する

SelectionBullet プロパティを設定して、選択した段落を箇条書きとして設定する方法について説明します。また、SelectionIndent、SelectionRightIndent、および SelectionHangingIndent の各プロパティを使用して、コントロールの左端または右端、およびテキストの他の行の左端を基準にして、段落のインデントを設定できます。

方法 : Windows フォームの RichTextBox コントロールにおけるドラッグ アンド ドロップ操作を有効にする

Windows フォームの RichTextBox コントロールで、DragEnter イベントと DragDrop イベントを処理して、ドラッグ アンド ドロップ操作を有効にする方法について説明します。

方法 : Windows フォームの RichTextBox コントロールを使用して Web スタイルのリンクを表示する

リンクがクリックされたときにブラウザ ウィンドウを開いて、リンク テキストで指定されている Web サイトを表示するように、コードを記述する方法について説明します。

Button コントロール

方法 : Windows フォームのボタンのクリックに応答する

ボタンをクリックしてコードを実行するという、Windows フォームの Button コントロールの最も基本的な使用法について説明します。

方法 : デザイナを使用して Windows フォームの Button コントロールを承認ボタンとして指定する

Button コントロールを承認ボタン (既定のボタンとも呼ばれます) として指定する方法について説明します。ユーザーが **Enter** キーを押すと、フォームの他のコントロールにフォーカスがある場合でも、既定のボタンがクリックされます。

方法 : デザイナを使用して Windows フォームの Button コントロールをキャンセル ボタンとして指定する

Button コントロールをキャンセル ボタンに指定する方法について説明します。ユーザーが **Esc** キーを押すと、フォームの他のコントロールに

フォーカスがある場合でも、このキャンセル ボタンがクリックされます。このようなボタンは、通常、ユーザーがどのアクションにもコミットせずに、すばやく操作を終了できるようにするために設定されます。

CheckBox コントロール

方法 : Windows フォームの CheckBox コントロールでオプションを設定する

Windows フォームの CheckBox コントロールを使用して、ユーザーが "True/False" または "Yes/No" のオプションを選択できるようにする方法について説明します。選択したオプションにはチェック マークが付きます。

方法 : Windows フォーム CheckBox のクリックに応答する

チェック ボックスの状態に応じてアクションを実行するように、アプリケーションをプログラミングする方法について説明します。

RadioButton コントロール

方法 : セットとして機能する Windows フォーム RadioButton コントロールをグループ化する

Panel コントロール、GroupBox コントロール、フォームなどのコンテナに配置することによって、オプション ボタンをグループ化する方法について説明します。

ListBox、ComboBox、および CheckedListBox コントロール

方法 : Windows フォームの ComboBox または ListBox コントロールをデータにバインドする

ComboBox と ListBox をデータにバインドして、データベースのデータ参照、新しいデータの入力、既存データの編集などのタスクを実行する方法について説明します。

方法 : Windows フォーム ComboBox、ListBox、または CheckedListBox コントロールのルックアップ テーブルを作成する

食品の注文フォームのデータを格納して表示する方法の例を表で説明します。

方法 : Windows フォームの ComboBox、ListBox、または CheckedListBox コントロールに項目を追加または削除する

Windows フォームのコンボ ボックス、リスト ボックス、またはチェックされたリスト ボックスに項目を追加する例を紹介します。ここでは、データ バインディングの必要がない、最も簡単な方法を示します。

方法 : Windows フォーム ComboBox、ListBox、または CheckedListBox コントロールの特定の項目にアクセスする

Windows フォームのコンボ ボックス、リスト ボックス、またはチェックされたリスト ボックス内の特定の項目にアクセスする方法について説明します。アクセスすると、リスト内の特定の場所にある項目をプロクラムによって判断できます。

方法 : Windows フォーム ComboBox、ListBox、または CheckedListBox コントロールを並べ替える

データ ビュー、データ ビュー マネージャ、および並べ替えられた配列という、並べ替えがサポートされているデータ ソースの使用方法について説明します。

CheckedListBox コントロール

方法 : Windows フォーム CheckedListBox コントロールでオンになっている項目を判断する

Windows フォームの CheckedListBox コントロールで、オンの項目を判断する方法について説明します。オンの項目を判断するには、CheckedItems プロパティに格納されたコレクション全体を反復処理するか、GetItemChecked メソッドを使用してリスト全体をステップ実行します。

DataGridView コントロール

方法 : デザイナを使用してデータを Windows フォーム DataGridView コントロールにバインドする

デザイナを使用して DataGridView コントロールを複数種類のデータ ソース (データベース、ビジネス オブジェクト、Web サービスなど) に接続する方法について説明します。

方法 : Windows フォーム DataGridView コントロールのデータを検証する

ユーザーが DataGridView コントロールに入力したデータを検証する方法について説明します。

方法 : Windows フォーム DataGridView コントロールでのデータ入力中に発生したエラーを処理する

DataGridView コントロールを使用して、データ入力エラーをユーザーにレポートする方法について説明します。

方法 : Windows フォーム DataGridView コントロールの新しい行に既定値を指定する

DefaultValuesNeeded イベントを使用して、新しい行の既定値を指定する方法について説明します。

方法 : 連結されていない Windows フォーム DataGridView コントロールを作成する

データ ソースにバインドせずに、プログラムで DataGridView コントロールを作成する方法について説明します。

方法 : データバインドされた Windows フォーム DataGridView コントロールに非バインド列を追加する

マスター/詳細シナリオを実装する場合に、[詳細] ボタンの非バインド列を作成して、親テーブルの特定の行に関連付けられた子テーブルを表示する方法について説明します。

方法 : Windows フォーム DataGridView コントロールのセルにイメージを表示する

埋め込みリソースからアイコンを抽出し、ビットマップに変換してイメージ列の各セルに表示する方法について説明します。

方法 : Windows フォーム DataGridView Cells でコントロールをホストする

カレンダー列を作成する方法について説明します。この列のセルは、通常のテキスト ボックスのセルに日付を表示しますが、ユーザーがセルを編集すると、DateTimePicker コントロールが表示されます。

チュートリアル : Windows フォーム DataGridView コントロールのデータの妥当性検査

Northwind サンプル データベースの Customers テーブルの行を取得し、DataGridView コントロールに表示する方法について説明します。CompanyName 列のセルを編集してセルから移動しようとすると、新しい企業名の文字列が空でないことが確認されます。新しい値が空の文字列の場合、DataGridView では、何か文字列が入力されるまでセルからカーソルを移動できません。

チュートリアル : Windows フォーム DataGridView コントロールでのデータ入力中に発生したエラーの処理

Northwind サンプル データベースの Customers テーブルの行を取得し、DataGridView コントロールに表示する方法について説明します。新しい行を作成するとき、または既存の行を編集したときに CustomerID 値が重複すると、DataError イベントが発生します。これは、例外を示す MessageBox を表示することで処理されます。

チュートリアル : バインドされていない Windows フォーム DataGridView コントロールの作成

DataGridView コントロールを作成し、"非バインド" モード行の追加および削除を管理する方法について説明します。

DataGridView レイアウトと書式

方法 : デザイナを使用して Windows フォームの DataGridView コントロールで列を読み取り専用にする

データを含む列を読み取り専用にする手順について説明します。

方法 : デザイナを使用して Windows フォーム DataGridView コントロールの列の並べ替えを有効にする

ユーザーが列を並べ替えることができるようにする方法について説明します。列の並べ替えを有効にすると、ユーザーは列ヘッダーをマウスでドラッグすることにより列を新しい場所に移動できます。

方法 : デザイナを使用して Windows フォーム DataGridView コントロールの列の順序を変更する

デザイナを使用して、Windows フォームの DataGridView コントロールで列の順序を変更する方法について説明します。

方法 : デザイナを使用して Windows フォーム DataGridView コントロールの列を追加および削除する

デザイナを使用して、Windows フォームの DataGridView コントロールで列の追加または削除を行う方法について説明します。

コントロールでのデータ バインディング

方法 : データ バインドで発生するエラーと例外を処理する

データ バインディング操作時に発生するエラーと例外を処理する方法について説明します。

BindingSource コントロール

方法 : デザイナを使用して Windows フォーム コントロールを BindingSource コンポーネントにバインドする

デザイン時にコントロールをバインドする方法について説明します。

方法 : Windows フォーム BindingSource コンポーネントを使用してルックアップ テーブルを作成する

ComboBox コントロールを使用して、親テーブルから子テーブルへの外部キー リレーションシップを持つフィールドを表示する方法について説明します。

方法 : BindingSource を使用して Windows フォーム コントロール内にデータ ソースの更新を反映させる

ResetBindings メソッドを使用して、バインドされたコントロールにデータ ソース内の更新を通知する方法について説明します。

方法 : Windows フォーム BindingSource コンポーネントで ADO.NET データを並べ替える/フィルタ処理する

BindingSource でデータの並べ替えとフィルタ処理を行う方法について説明します。

方法 : Windows フォーム BindingSource を使用して Web サービスにバインドする

クライアント側プロキシを作成してバインドする方法について説明します。

ナビゲータのバインド

方法 : Windows フォーム BindingNavigator コントロールを使用してデータ間を移動する

BindingNavigator コントロールをセットアップする方法について説明します。

方法 : Windows フォームの BindingNavigator コントロールを使用して DataSet を移動する

Binding Navigator コントロールを使用して、データベースのクエリ結果を移動する方法について説明します。

ListView

方法 : Windows フォーム ListView コントロールで項目を追加および削除する

項目を Windows フォームの ListView コントロールに追加および削除する手順について説明します。リスト項目の追加または削除は、いつでも実行できます。

方法 : ListView コントロールに検索機能を追加する

短期間で本格的な Windows フォーム アプリケーションを作成する方法について説明します。

方法 : Windows フォーム ListView コントロール内の項目を選択する

Windows フォームの ListView コントロール内の項目をプログラムで選択する方法について説明します。

方法 : Windows フォーム ListView コントロールのアイコンを表示する

リスト ビューにイメージを表示する方法について説明します。

方法 : Windows フォーム ListView コントロールの列にサブ項目を表示する

リスト項目にサブ項目を追加する方法について説明します。

TreeView

方法 : Windows フォーム TreeView コントロールのアイコンを設定する

ツリー ビューにイメージを表示する方法について説明します。

方法 : Windows フォーム TreeView コントロールでノードを追加および削除する

ツリー ビューで、プログラムによってノードを追加および削除する方法について説明します。

方法 : クリックされた TreeView ノード (Windows フォーム) を判別する

クリックされた TreeView ノードを判別する方法について説明します。

コンテナ コントロール

方法 : ウィンドウを水平方向に分割する

SplitContainer コントロールを分割するスプリッタを水平にする方法について説明します。

方法 : Windows フォームでマルチペイン ユーザー インターフェイスを作成する

Microsoft Outlook で使用されるユーザー インターフェイスと同様に、[フォルダ] リスト、[メッセージ] ペイン、および [プレビュー] ペインという複数のペインを備えたマルチペイン ユーザー インターフェイスを作成する方法について説明します。

方法 : TableLayoutPanel コントロールの行と列を拡大する

隣接する行および列を対象に、TableLayoutPanel コントロールを制御をする方法について説明します。

チュートリアル : TableLayoutPanel を使用した Windows フォーム上のコントロールの配置

次のタスクの実行方法について説明します。

Windows フォーム プロジェクトの作成

行と列のコントロール配置

行と列のプロパティ設定

制御範囲を行および列に指定

オーバーフローの自動処理

ツールボックスでダブルクリックしてコントロールの挿入

外枠を描画してコントロールの挿入

既存コントロールを異なる親コントロールに再割り当て

チュートリアル : FlowLayoutPanel を使用した Windows フォーム上のコントロールの配置

次のタスクの実行方法について説明します。

Windows フォーム プロジェクトの作成

コントロールの水平配置と垂直配置

フロー方向の変更

フロー区切りの挿入

埋め込みとマージンを使用してコントロール配置

ツールボックスでダブルクリックしてコントロールの挿入

外枠を描画してコントロールの挿入

キャレットを使用してコントロールの挿入

既存コントロールを異なる親コントロールに再割り当て

ピクチャおよび Image コントロール

方法 : デザイナを使用してピクチャを読み込む (Windows フォーム)

デザイン時のフォームで、Image プロパティを有効なピクチャに設定して、ピクチャの読み込みと表示を行う方法について説明します。

方法 : 実行時にピクチャを設定する (Windows フォーム)

Windows フォームの PictureBox コントロールで表示されるイメージをプログラムで設定する方法について説明します。

方法 : 実行時にピクチャのサイズまたは配置を変更する (Windows フォーム)

Windows フォームの PictureBox コントロールの SizeMode プロパティを異なる値に設定する方法について説明します。

DateTimePicker

方法 : Windows フォームの DateTimePicker コントロールを使用して日付を設定および取得する

コントロールを表示する前に Value プロパティを設定して、コントロールで最初に選択される日付を決定する方法について説明します。

方法 : Windows フォームの DateTimePicker コントロールを使用してカスタム形式で日付を表示する

カスタム書式を表示し、CustomFormat プロパティを書式指定文字列に設定する方法について説明します。

MonthCalendar

方法 : Windows フォームの MonthCalendar コントロールで日付の範囲を選択する

MonthCalendar コントロールのプロパティを使用して、日付の範囲を設定したり、ユーザーが設定した選択範囲を取得する方法について説明します。

方法 : Windows フォームの MonthCalendar コントロールを使用して特定の日付を太字で表示する

日付を太字または標準のフォントで表示する方法について説明します。

方法 : Windows フォームの MonthCalendar コントロールにおいて複数の月を表示する

Windows フォームの MonthCalendar コントロールで、複数の月を表示する方法について説明します。

方法 : Windows フォームの MonthCalendar コントロールの外観を変更する

月間予定表の配色を変更する方法、現在の日付をコントロールの下部に表示する方法、および週番号を表示する方法について説明します。

データ アクセス (Windows フォームの場合)

チュートリアル : Windows アプリケーションのフォーム間でのデータの受け渡し

最初のフォームのデータを 2 番目のフォームのメソッドに渡す手順について詳細に説明します。

チュートリアル : Windows アプリケーションのフォームでのデータの表示

単一のテーブルのデータを、複数の各コントロールに表示する簡単なフォームを作成します。

チュートリアル : Windows アプリケーションのデータ検索フォームの作成

データを検索する Windows フォームを作成する方法について説明します。

ToolStrip

方法 : ToolStrip に ToolStripItem を配置する

ToolStrip の左側または右側に、ToolStripItem を移動または追加する方法について説明します。

方法 : デザイナを使用して ToolStripMenuItems を無効にする

デザイン時にメニュー項目を無効にする方法について説明します。

方法 : ToolStripMenuItems を移動する

トップレベル メニュー全体およびそのメニュー項目を MenuStrip 上の別の位置に移動する方法について説明します。個々のメニュー項目が所属するトップレベル メニューを変更したり、同じメニュー内でメニュー項目の位置を変更することもできます。

方法 : Windows フォーム内の ToolStrip テキストとイメージの外観を変更する

テキストおよびイメージを ToolStripItem に表示するかどうかを制御する方法、およびテキスト、イメージ、および ToolStrip に合わせて整列する方法について説明します。

コンテキスト メニュー

方法 : ショートカット メニューを Windows フォーム NotifyIcon コンポーネントに関連付ける

ショートカット メニューを Windows フォーム NotifyIcon コンポーネントに関連付ける方法について説明します。

方法 : Windows フォーム ContextMenu コンポーネントのメニュー項目を追加および削除する

Windows フォームのショートカット メニューの項目を追加する方法および削除する方法を説明します。

印刷

方法 : 標準の Windows フォーム印刷ジョブを作成する

PrintPage イベントを処理するコードを記述して、印刷する内容および印刷方法を指定する方法について説明します。

方法 : Windows フォームの印刷ジョブを完了する

PrintDocument コンポーネントの EndPrint イベントを処理して、印刷ジョブを完了する方法について説明します。

方法 : Windows フォームで複数ページのテキスト ファイルを印刷する

オブジェクト (グラフィックスやテキスト) をデバイス (画面やプリンタ) に出力するメソッドを使用して、Windows フォームのテキストを印刷する方法について説明します。

方法 : Windows フォームでユーザーのコンピュータに接続されたプリンタを選択する

プリンタを選択してファイルを印刷する方法について説明します。

方法 : 実行時に PrintDialog のユーザー入力をキャプチャする

実行時の印刷オプションを変更する方法について説明します。オプションの変更には、PrintDialog コンポーネントと PrinterSettings クラスを使用します。

ユーザー コントロールとカスタム コントロール

ユーザー コントロールへのコントロールの追加

ユーザー コントロールにコントロールを追加する方法について説明します。

ユーザー コントロールへのコードの追加

ユーザー コントロールにコードを追加する方法について説明します。

マルチ ドキュメント インターフェイス (MDI) アプリケーション

方法 : MDI 親フォームを作成する

デザイン時に、MDI 親フォームを作成する方法について説明します。

方法 : MDI 子フォームを作成する

多くのワードプロセッシング アプリケーションで実装されている、RichTextBox コントロールを表示する MDI 子フォームを作成する方法について説明します。

方法 : MDI 子フォームを配置する

子フォームを重ねて表示する方法、水平方向または垂直方向に並べて表示する方法、または MDI フォームの下部に子フォームのアイコンとして配置する方法について説明します。

方法 : アクティブな MDI 子フォームを特定する

アクティブな MDI 子フォームを判断してテキストをクリップボードにコピーする方法について説明します。

方法 : アクティブな MDI 子フォームにデータを送信する

クリップボードからアクティブな MDI 子ウィンドウへデータを送信する方法について説明します。

グラフィックス

方法 : 形状のアウトラインを描画する

フォーム上で楕円および四角形のアウトラインを描画する方法について説明します。

方法 : 線形グラデーションを作成する

直線、楕円、および四角形を水平方向の線形グラデーション ブラシで塗りつぶす方法について説明します。

方法 : パス グラデーションを作成する

図形を塗りつぶすときの段階的な色の変化をカスタマイズする方法について説明します。

方法 : 直線、曲線、および形状から図形を作成する

1 つまたは複数の図形を持つパスを作成する方法について説明します。

方法 : 描画する Graphics オブジェクトを作成する

グラフィック オブジェクトを作成して描画する方法について説明します。

方法 : サムネイル イメージを作成する

ビットマップ ファイルからイメージ オブジェクトを構築する方法について説明します。

方法 : 垂直方向のテキストを作成する

StringFormat オブジェクトを使用して、テキストを水平方向ではなく垂直方向に描画するように指定する方法について説明します。

方法 : 描画テキストを配置する

四角形にテキストを描画する方法について説明します。テキストの各行は中央揃えで配置され、テキスト ブロック全体は四角形内の中央に揃えて配置されます。

方法 : Windows フォームに直線を描画する

フォーム上で直線を描画する方法について説明します。

方法 : イメージを回転、反転、および傾斜させる

元のイメージの左上隅、右上隅、および左下隅に対して、それぞれ変換後の点を指定して、イメージを回転、反転、および傾斜させる方法について説明します。

方法 : Windows フォームにテキストを描画する

グラフィックスの DrawString メソッドを使用して、フォームにテキストを描画する方法について説明します。

方法 : ビットマップを読み込んで表示する

ファイルからビットマップを読み込んで、画面に表示する方法について説明します。

方法 : メタファイルを読み込んで表示する

Metafile クラスのメソッドを使用して、ベクター イメージの記録、表示、および検証を行う方法について説明します。

Windows フォームのローカライズとグローバライズ

チュートリアル : Windows フォームのローカリゼーション

Windows アプリケーション プロジェクトのローカライズ処理を行う方法について説明します。

方法 : AutoSize と TableLayoutPanel コントロールを使用して Windows フォームのローカリゼーションをサポートする

文字列の長さに合わせて調整するレイアウトを有効にする方法について説明します。

方法 : Windows フォームのグローバリゼーション用のカルチャおよび UI カルチャを設定する

特定の地域に適した表示形式オプションを設定する方法について説明します。

方法 : グローバリゼーション用に Windows フォームで右から左の方向でテキストを表示する

右から左の方向でテキストを表示する方法について説明します。

その他のリソース

Visual Studio 2005 Developer Center

Visual Studio 2005 を使用したアプリケーション開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Visual C# Developer Center

C# アプリケーションの開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Microsoft .NET Framework Developer Center

.NET Framework アプリケーションの開発やデバッグに関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

参照
概念
C# での操作方法

Web ページと Web サービス (C# での操作方法)

このページでは、よく使用する Web アプリケーション タスクに関するヘルプへのリンクを紹介します。よく使用するタスクのその他のカテゴリに関するヘルプについては、「C# での操作方法」を参照してください。

Web ページ

Visual Studio の Web 開発の新機能

ASP.NET Web ページ作成用の開発ツールである Visual Web Developer を紹介します。

ASP.NET Web ページの概要

Web アプリケーションにおける ASP.NET Web ページの動作の基本的な特性の概要について説明します。

Web サービス

XML ドキュメントと XML データ

XML ドキュメントと XML データを読み書きする方法に関する文書へのリンクを示します。

マネージ コードを使用した XML Web サービス

XML Web サービスの作成方法と配置方法、およびマネージ コードを使用した XML Web サービスへのアクセス方法に関する文書へのリンクを示します。

Getting Started with XML Web Services in Visual Basic and Visual C#

Visual Studio と XML Web サービスで、単純で柔軟性の高い標準ベースのモデルを実現する方法について説明します。このモデルを使って開発することで、プラットフォーム、プログラミング言語、またはオブジェクト モデルに関係なく、アプリケーションをアセンブルできます。

方法 : マネージ コードを使用して XML Web サービスにアクセスする

Web サービスにアクセスする方法について説明します。

方法 : マネージ コードを使用して XML Web サービスに非同期にアクセスする

Web サービスに非同期的にアクセスする方法について説明します。

方法 : ブラウザから XML Web サービスにアクセスする

ブラウザから XML Web サービスにアクセスする方法について説明します。

方法 : XML Web サービス内容を探索する

XML Web サービスのコンテンツを参照する方法について説明します。

方法 : ASP.NET Web サービス プロジェクトを作成する

ASP.NET Web サービス プロジェクトの作成方法について説明します。

方法 : WebService 属性を使用する

WebService 属性を使用して、XML Web サービスの名前空間と説明テキストを指定する方法について説明します。

方法 : WebService クラスを継承する

WebService クラスを継承する方法について説明します。

方法 : XML Web サービス メソッドを作成する

XML Web サービス メソッドの作成方法について説明します。

方法 : WebMethod 属性を使用する

WebMethod 属性を使用して、XML Web サービスの一部としてメソッドを公開することを示す方法について説明します。

方法 : マネージ コードを使用して XML Web サービスをデバッグする

マネージ コードを使用して XML Web サービスをデバッグする方法について説明します。

方法 : マネージ コードを使用して XML Web サービスを配置する

マネージ コードを使用して XML Web サービスを配置する方法について説明します。

方法 : XML Web サービス プロキシを生成する

XML Web サービス プロキシの生成方法について説明します。

方法 : コンソール アプリケーション クライアントを作成する

コンソール アプリケーション Web サービス クライアントの作成方法について説明します。

方法 : Web サービス メソッドによってスローされた例外を処理する

Web サービス メソッドによってスローされた例外を処理する方法について説明します。

方法 : コールバック手法を使用して非同期 Web サービス クライアントを実装する

コールバック技術を使用して非同期の Web サービス クライアントを実装する方法について説明します。

方法 : 待機手法を使用して非同期 Web サービス クライアントを実装する

待機技術を使用して非同期の Web サービス クライアントを実装する方法について説明します。

チュートリアル : インストール時にアプリケーションを別の XML Web サービスにリダイレクトする

URL Behavior プロパティ、Installer クラス、および Web セットアップ プロジェクトを使用して、異なる XML Web サービスにリダイレクトできる Web アプリケーションを作成する方法を示します。

ASP.NET を使用して作成した XML Web サービスのセキュリティ

ASP.NET を使用して構築された Web サービスに使用できる、認証と承認のオプションについて概要を説明します。

その他のリソース

以下のサイトにアクセスするには、インターネット接続が必要です。

Web Services Developer Center

Web サービスの開発および接続に関する文書、コード例、およびその他のさまざまなリソースを提供します。

MSDN XML Developer Center

XML データの操作に関する文書、コード例、およびその他のさまざまなリソースを提供します。

Microsoft ASP.NET Developer Center

ASP.NET Web ページの作成に関する文書、コード例、およびその他のさまざまなリソースを提供します。

参照
概念
C# での操作方法

デバッグ (C# での操作方法)

このページでは、よく使用するデバッグ タスクに関するヘルプへのリンクを紹介します。よく使用するタスクのその他のカテゴリに関するヘルプについては、「C# での操作方法」を参照してください。

Visual Studio デバッガの使用方法

Visual Studio でのビルド

ビルドしながら継続的にアプリケーションをテストおよびデバッグするツールについて説明します。

Visual Studio でのデバッグ

Visual Studio デバッガの使用方法の基本について説明します。

デバッガのロードマップ

デバッグの基本タスクおよびデバッガの機能に関する文書へのリンクを示します。

.NET Framework のトレース機能

方法 : アプリケーション コードにトレース ステートメントを追加する

アプリケーションのトレースに、Write、WriteIf、WriteLine、WriteLineIf、Assert、および Fail の各メソッドを使用する方法について説明します。

方法 : トレース リスナを作成し初期化する

トレース リスナの作成方法と初期化方法について説明します。

方法 : TraceSource とフィルタをトレース リスナと共に使用する

TraceSource と一緒にアプリケーション構成ファイルを使用する方法について説明します。

方法 : トレース スイッチを作成し初期化する

トレース スイッチの作成方法と初期化方法について説明します。

方法 : トレースとデバッグを指定して条件付きコンパイルを実行する

アプリケーションのコンパイラ設定を指定する方法をいくつか説明します。

方法 : トレース ソースを作成し初期化する

構成ファイルを使用して、実行時にトレース ソースによって作成されるトレースの再構成を容易にする方法について説明します。

デバッガ表示属性によるデバッグ機能の拡張

デバッガの表示属性を使用して、デバッグを強化する方法について説明します。

方法 : アプリケーションのコードをトレースする

アプリケーションに Trace クラスをインストルメントする方法について説明します。

方法 : トレース スイッチを設定する

.config ファイルを使用して、スイッチを設定する方法について説明します。

Web サービスのデバッグ

チュートリアル : XML Web サービスのデバッグ

Web サービスのデバッグ手順について説明します。

Windows フォームのデバッグ

チュートリアル : Windows フォームのデバッグ

Windows フォーム アプリケーションのデバッグについて説明します。

SQL アプリケーションのデバッグ

チュートリアル : SQL CLR のユーザー定義のテーブル値関数のデバッグ

SQL/CLR のユーザー定義のテーブル値関数 (UDF: User Defined Table-Valued Function) をデバッグする方法について説明します。

チュートリアル : SQL CLR トリガのデバッグ

SQL/CLR トリガのデバッグ方法について説明します。ここでは、SQL Server 2005 と共にインストールされたデータベースの 1 つである **AdventureWorks** サンプル データベースの Contact テーブルを使用します。このサンプルでは、Contact テーブルに新しい insert の CLR トリガが作成され、ステップ インされます。

チュートリアル : SQL CLR のユーザー定義型のデバッグ

SQL/CLR のユーザー定義型のデバッグ方法について説明します。ここでは、**Adventureworks** サンプル データベースに新しい SQL/CLR 型を作成します。この型は、テーブル定義、INSERT ステートメント、さらに SELECT ステートメントで使用されます。

チュートリアル : SQL CLR のユーザー定義スカラ関数のデバッグ

SQL/CLR のユーザー定義関数 (UDF: User Defined Function) をデバッグする方法について説明します。ここでは、**Adventureworks** サンプル データベースに新しい SQL/CLR ユーザー定義関数を作成します。

チュートリアル : SQL CLR のユーザー定義集計のデバッグ

CLR SQL のユーザー定義集計のデバッグ方法について説明します。この例により、**Adventureworks** サンプル データベースに Concatenate という、新しい CLR SQL 集計関数が作成されます。SQL ステートメントでこの関数を呼び出すと、入力パラメータで指定した列の値がすべて連結されます。

T-SQL データベースのデバッグ

必要なセットアップ手順について説明し、多階層のアプリケーションをデバッグする例を紹介します。

チュートリアル : T-SQL トリガのデバッグ

UPDATE トリガを含む Sales.Currency テーブルが格納されている、**AdventureWorks** データベースを使用する例について説明します。このサンプルには、テーブルの行を更新してトリガを発生させるストアド プロシージャが含まれます。トリガにブレークポイントを設定し、さまざまなパラメータを指定してストアド プロシージャを実行すると、トリガの異なる実行パスをたどることができます。

チュートリアル : T-SQL のユーザー定義関数のデバッグ

AdventureWorks データベースで、ufnGetStock という既存のユーザー定義関数 (UDF: User Defined Function) を使用する例について説明します。この関数は、指定した ProductID のストックに含まれるアイテム数を返します。

チュートリアル : T-SQL ストアド プロシージャのデバッグ

ダイレクト データベース デバッグによって (サーバー エクスプローラでステップ インすることによって) T-SQL ストアド プロシージャを作成する方法とデバッグする方法について説明します。また、ブレークポイントの設定、データ項目の表示など、さまざまなデバッグ技術についても説明します。

その他のリソース

このサイトにアクセスするには、インターネット接続が必要です。

Visual Studio 2005 Developer Center

アプリケーションの開発およびデバッグに関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

参照
概念
C# での操作方法

データ アクセス (C# での操作方法)

このページでは、よく使用するデータ アクセス タスクに関するヘルプへのリンクを紹介します。よく使用するタスクのその他のカテゴリに関するヘルプについては、「C# での操作方法」を参照してください。

一般

方法 : サンプル データベースをインストールする

Northwind サンプル データベース、SQL Server Express (SSE)、MSDE、または Access 版の Northwind など、サンプル データベースのインストール手順について説明します。

チュートリアル : 単純なデータ アプリケーションの作成

データ アプリケーションを作成する詳しい手順について説明します。

Visual Studio でのデータへの接続

Visual Studio でのデータへの接続の概要

データベース、Web サービス、オブジェクトなど、さまざまなソースのデータにアプリケーションを接続する方法について説明します。

チュートリアル : データベース内のデータへの接続

データ ソース構成ウィザードを使用して、アプリケーションを Visual Studio のデータに接続する手順について説明します。

チュートリアル : Web サービスのデータへの接続

データ ソース構成ウィザードを使用して、アプリケーションを Web サービスのデータに接続する手順について説明します。

チュートリアル : Access データベース内のデータへの接続

データ ソース構成ウィザードを使用して、アプリケーションを Access データベースのデータに接続する手順について説明します。

型指定されたデータセットの作成とデザイン

方法 : 型指定されたデータセットを作成する

データ ソース構成ウィザードまたはデータセット デザイナを使用して、型指定されたデータセットを作成する方法について説明します。

チュートリアル : データセット デザイナでのデータセットの作成

データセット デザイナを使用して、データセットを作成する手順について説明します。

チュートリアル : データセット デザイナでの DataTable の作成

データセット デザイナを使用して、DataTable を作成する手順について説明します。

チュートリアル : データ テーブル間のリレーションシップの作成

データセット デザイナを使用し、TableAdapter がない 2 つのデータ テーブルとその間のリレーションシップを作成する方法について説明します。

TableAdapter

TableAdapter の概要

アプリケーションとデータベースとの通信を実現する TableAdapter の概要を説明します。

チュートリアル : 複数のクエリによる TableAdapter の作成

データ ソース構成ウィザードを使用して、データセットに TableAdapter を作成する手順について説明します。また、データセット デザイナ内の TableAdapter クエリの構成ウィザードを使用して、TableAdapter 内に 2 つ目のクエリを作成する手順についても説明します。

データセットへの読み込みとクエリの実行

データセットへの読み込みとデータのクエリの概要

TableAdapter またはコマンド オブジェクトを使用して、データ ソースに対して SQL ステートメントまたはストアド プロシージャを実行する方法について説明します。

チュートリアル : データセットへのデータの読み込み

1 つのデータ テーブルを持つデータセットを作成し、Northwind サンプル データベースの Customers テーブルからデータを入力する方法について説明します。

チュートリアル : データセットへの XML データの読み込み

データセットに XML データを読み込む Windows アプリケーションを作成する方法について説明します。

Windows フォームでのデータの表示

データの表示の概要

データ バインド Windows アプリケーションの開発に関連するタスク、オブジェクト、およびダイアログ ボックスの概要を説明します。

チュートリアル : Windows アプリケーションのフォームでのデータの表示

単一のテーブルのデータを、複数の各コントロールに表示する簡単なフォームを作成する手順について説明します。

チュートリアル : Windows アプリケーションのフォームでの関連データの表示

複数のテーブルから取得したデータ、特に、互いに関連し合うテーブルから取得したデータを操作する手順について説明します。

チュートリアル : Windows アプリケーションのデータ検索フォームの作成

特定の都市にいる顧客を返すクエリを作成する方法、およびユーザー インターフェイスを変更して、ユーザーが都市の名前を入力してクエリを実行するボタンを押すことができるようにする方法について説明します。

チュートリアル : ルックアップ テーブルの作成

あるテーブルの外部キー フィールドの値に基づいて、別のテーブルの情報を表示する手順について説明します。

データのバインド

チュートリアル : 単純データ バインディングをサポートするユーザー コントロールの作成

DefaultBindingPropertyAttribute を実装するコントロールの作成方法について説明します。TextBox や CheckBox と同様に、このコントロールには、データにバインドできる 1 つのプロパティを含めることができます。

チュートリアル : 複合データ バインディングをサポートするユーザー コントロールの作成

ComplexBindingPropertiesAttribute を実装するコントロールの作成方法について説明します。DataGridView や ListBox と同様に、このコントロールには、データにバインドできる DataSource プロパティと DataMember プロパティが含まれます。

チュートリアル : 検索データ バインドをサポートするユーザー コントロールの作成

LookupBindingPropertiesAttribute を実装するコントロールの作成方法について説明します。ComboBox と同様に、このコントロールには、データにバインドできる 3 つのプロパティが含まれます。

Visual Studio におけるオブジェクトのバインド

(データセットや Web サービスとは対照的に) カスタム オブジェクトをアプリケーションでデータ ソースとして操作するときに使用するデザイン時のツールについて説明します。

データセットのデータの編集 (DataTable)

データセットのデータの編集の概要

データセットのデータ編集とデータ クエリに関する一般的なタスクへのリンクを含む表です。

データの検証

データの妥当性検査の概要

データ オブジェクトに入力する値が、データセットのスキーマ内の制約、およびアプリケーションに対して設定されている規則に従っていることを確認する、データ検証の概要を説明します。

チュートリアル : データセットへの検証の追加

ColumnChanging イベントを使用して、有効な値がレコードに入力されていることを検証する方法について説明します。

データの保存

データ保存の概要

情報を元のデータ ソースに記述する処理を、データセットのデータを変更する処理と分離する方法について説明します。

ADO.NET における同時実行制御

同時実行制御の一般的な方法、および同時実行エラーを処理する固有の ADO.NET 機能について説明します。

チュートリアル : TableAdapter DBDirect メソッドを使用してデータを保存する

TableAdapter の DbDirect メソッドを使用して、SQL ステートメントをデータベースに対して直接実行する詳細な手順について説明します。

チュートリアル : 同時実行例外の処理

DBConcurrency 例外をキャッチし、エラーが発生した行を見つけて、対応方法の 1 つを示す、Windows アプリケーションを作成する手順について説明します。

データ リソース
データ ユーザー インターフェイス要素

アプリケーションでデータ アクセスをデザインするときに使用するすべてのダイアログ ボックスとウィザードについて説明します。

ADO.NET データアダプタ

ADO.NET データアダプタ オブジェクトについて、および Visual Studio でそれを使用する方法について説明します。

マネージ コードでの SQL Server 2005 オブジェクトの作成
SQL Server プロジェクト

ストアド プロシージャやトリガなどのデータベース オブジェクトの作成、および Microsoft SQL Server 2005 データベースのデータの取得や更新に、Transact-SQL プログラミング言語以外に .NET 言語を使用する方法について説明します。

チュートリアル : マネージ コードでのストアド プロシージャの作成

次の操作手順を説明します。

マネージ コードでのストアド プロシージャの作成

SQL Server 2005 データベースへのストアド プロシージャの配置

データベースでストアド プロシージャをテストするためのスクリプトの作成

ストアド プロシージャが正しく実行されたことを確認するためのデータベースのデータの照会

その他のリソース

次のサイトにアクセスするには、インターネット接続が必要です。

Visual Studio 2005 Developer Center

Visual Studio 2005 を使用したアプリケーション開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Visual C# Developer Center

C# アプリケーションの開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Microsoft .NET Framework Developer Center

.NET Framework アプリケーションの開発やデバッグに関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Microsoft Universal Data Access

Microsoft のデータ アクセス テクノロジをアプリケーションで使用する方法に関する多数の文書やリソースを提供します。

SQL Server Developer Center

SQL Server の使用方法に関する多数の文書やリソースを提供します。

参照
概念
C# での操作方法

クラスのデザイン (C# での操作方法)

このページでは、よく使用する C# クラス デザイナ タスクに関するヘルプへのリンクを紹介します。よく使用するタスクのその他のカテゴリに関するヘルプについては、「C# での操作方法」を参照してください。

クラス デザイナ

方法 : クラス ダイアグラムに型を作成する

クラス、列挙型、インターフェイス、構造体、デリゲートなど、新しい型を作成する方法について説明します。

型のメンバの作成および構成

型のメンバの操作方法について説明します。

方法 : ジェネリック型を継承する

クラスがジェネリックから継承する場合に関係を確立する方法について説明します。

方法 : 型の間の継承を定義する

クラス デザイナを使用して、クラス ダイアグラムに現在表示されている 2 つの型の継承関係を定義する方法について説明します。

方法 : 型の間の関連付けを定義する

関連付けを定義する方法について説明します。クラス デザイナの関連行は、ダイアグラムのクラスの関係を示します。

方法 : クラス ダイアグラムから型シェイプおよび関連付けられているコードを削除する

図形、または図形とコードの両方をクラス ダイアグラムから削除する方法について説明します。

方法 : 型または型のメンバにカスタム属性を適用する

型または型のメンバにカスタム属性を適用する方法について説明します。

クラスおよびその他の型の処理

方法 : 型の間の継承を表示する

選択した型と基本型の間に継承関係がある場合に、その選択した型の基本型を表示する方法について説明します。

方法 : 派生型を表示する

選択した型から派生する型を表示します。型とその基本クラスまたはインターフェイスとの間に継承関係が存在することを想定しています。

方法 : クラス ダイアグラムから型シェイプを削除する

ダイアグラムから図形を削除する手順について説明します。

方法 : 型シェイプのコンパートメントを表示する

コンパートメントの表示/非表示を切り替える手順について説明します。

方法 : 型の詳細を表示する

コンパートメントの詳細を表示する手順について説明します。

方法 : メンバ表記と関連付け表記の間で変更する

メンバの表記と関連付けを変更する手順について説明します。

方法 : 型のメンバを表示する

型のメンバの表示/非表示を切り替える手順について説明します。

方法 : プロジェクトにクラス ダイアグラムを追加する

クラス ダイアグラムをプロジェクトに追加する手順について説明します。

方法 : 既存の型を表示する

デザイン サーフェイスに既存の型を表示する手順について説明します。

方法 : プロジェクトにクラス ダイアグラムを追加する

クラス ダイアグラムをプロジェクトに追加する手順について説明します。

自身で記述していないコードについて

Visual Studio クラス デザイナをツールとして使用して、他のユーザーが記述したクラスと型を理解する方法について説明します。ツールでは、コードのグラフィカル表示が使用されます。この表示は必要に応じてカスタマイズできます。

方法 : 型のメンバをグループ化する

種類、アクセス修飾子、またはどのようにアルファベット順に並べ替えるかによって、メンバをグループ化する手順について説明します。

方法 : クラス ダイアグラムにコメントを追加する

コメント図形を使用してクラス ダイアグラムに注釈を付ける手順について説明します。

クラス ダイアグラムのカスタマイズ

クラス ダイアグラムでプロジェクト情報を表示する方法を変更する手順について説明します。

方法 : Microsoft Office ドキュメントにクラス ダイアグラムの要素をコピーする

1 つ、複数、またはすべての図形をクラス ダイアグラムから他のドキュメントにコピーする手順について説明します。

方法 : クラス ダイアグラムを印刷する

Visual Studio の印刷機能を使用して、クラス ダイアグラムを印刷する手順について説明します。

方法 : 型のメンバをオーバーライドする

クラス デザイナを使用して、子クラスのメンバで、基本クラスから継承したメンバをオーバーライド (新しい実装を提供) する手順について説明します。

方法 : 型と型のメンバの名前を変更する

クラス デザイナ、[クラスの詳細] ウィンドウ、または [プロパティ] ウィンドウを使用して、型または型のメンバの名前を変更する手順について説明します。

方法 : 型のメンバをある型から別の型に移動する

現在のクラス ダイアグラムに両方の型が表示されている場合に、ある型から別の型に型のメンバを移動する手順について説明します。

方法 : インターフェイスを実装する

クラス デザイナを使用して、インターフェイスの作成、実装、および削除を行う方法について説明します。

方法 : 抽象クラスを実装する

クラス デザイナを使用して、抽象クラスを実装する手順について説明します。

方法 : インターフェイスに抽出する (C# のみ)

1 つ以上のパブリック メンバをある型から新しいインターフェイスに抽出する手順について説明します。

方法 : パラメータの順序を変更する (C# のみ)

クラス デザイナに表示されている型で、メソッドのパラメータを並べ替える手順について説明します。

型のメンバの作成と変更

方法 : [クラスの詳細] ウィンドウを開く

型のメンバを構成できるウィンドウである [クラスの詳細] について説明します。

[クラスの詳細] ウィンドウの要素

[クラスの詳細] ウィンドウに表示される行の内容について説明します。

方法 : メンバを作成する

クラス デザイナ、[クラスの詳細] ウィンドウのツール バー、または [クラスの詳細] ウィンドウのいずれかを使用して、メンバを作成する方法について説明します。

方法 : メソッドにパラメータを追加する

[クラスの詳細] ウィンドウを使用して、メソッドにパラメータを追加する方法について説明します。

方法 : 型のメンバを変更する

[クラスの詳細] ウィンドウを使用して、クラス デザイナで作成した型のメンバを変更する方法について説明します。

[クラスの詳細] ウィンドウの使用上の注意

[クラスの詳細] ウィンドウを使用する場合のヒントです。

読み取り専用情報の表示

クラス デザイナと [クラスの詳細] ウィンドウを使用して、プロジェクト、またはプロジェクトから参照されるプロジェクトやアセンブリについて、型 (および型のメンバ) を表示する方法について説明します。

クラス ライブラリ デザイン ガイド

方法 : コントロール用デザイナを実装する

HelpLabel 拡張プロバイダの制御に、デザイナ (HelpLabelDesigner) を実装する方法について説明します。

方法 : デザイン モードでコンポーネントを作成および設定する

デザイナ サービスを使用して、カスタム デザイナのコンポーネントを作成および初期化する方法について説明します。

方法 : Windows フォームでデザイン時サポートにアクセスする

.NET Framework に用意されている、デザイン時のサポートにアクセスする手順について説明します。

方法 : HelpLabel 拡張プロバイダを実装する

HelpLabel コントロールを作成し、拡張プロバイダを構築する方法について説明します。

方法 : デザイン時サービスにアクセスする

豊富な .NET Framework サービスにアクセスして、コンポーネントおよびコントロールをデザイン環境に統合する方法について説明します。

方法 : 標準の型のコレクションを DesignerSerializationVisibilityAttribute でシリアル化する

DesignerSerializationVisibilityAttribute クラスを使用して、デザイン時にコレクションをシリアル化する方法について説明します。

方法 : デザイン モードでコントロールのカスタム初期化を行う

デザイン環境によるコントロールの作成時に、そのコントロールを初期化する方法について説明します。

方法 : 型コンバータを実装する

型コンバータを使用して値のデータ型を変換する方法について説明します。また、テキストから値への変換や、値を選択するためのドロップダウン リストを使用して、デザイン時のプロパティ構成を簡単に行う方法についても説明します。

方法 : UI 型エディタを実装する

Windows フォームのカスタム UI 型エディタを実装する方法について説明します。

方法 : デザイン モードでコントロールの外観と動作を拡張する

ユーザー インターフェイス (UI) を拡張して、カスタム コントロールをデザインするカスタム デザイナの作成方法について説明します。

方法 : デザイン時機能を活用した Windows フォーム コントロールを作成する

カスタム コントロールおよび関連するカスタム デザイナを作成する方法について説明します。このライブラリが構築されると、フォームで実行するカスタム MarqueeControl 実装を構築できます。

方法 : Windows フォーム コンポーネントにスマート タグを追加する

コンポーネントとカスタム コントロールに、スマート タグのサポートを追加する方法について説明します。

方法 : デザイン モードでコンポーネントの属性、イベント、およびプロパティを調整する

コンポーネントの属性、イベント、およびプロパティを調整するカスタム デザイナの作成方法について説明します。

その他のリソース

次のサイトにアクセスするには、インターネット接続が必要です。

Visual Studio 2005 Developer Center

Visual Studio 2005 を使用したアプリケーション開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Visual C# Developer Center

C# アプリケーションの開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Microsoft .NET Framework Developer Center

.NET Framework アプリケーションの開発やデバッグに関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Microsoft Patterns and Practices Developer Center

アーキテクチャとして Microsoft .NET プラットフォームに適したアプリケーションをデザイン、開発、配置、および操作する方法について説明する、シナリオ固有の推奨事項を示します。

参照
概念
C# での操作方法

セキュリティ (C# での操作方法)

このページでは、よく使用するセキュリティ タスクと配置タスクに関するヘルプへのリンクを紹介します。よく使用するタスクのその他のカテゴリに関するヘルプについては、「C# での操作方法」を参照してください。

一般

Visual Studio におけるセキュリティ

安全なコーディング技法について説明します。

コード アクセス セキュリティの基礎

共通言語ランタイムを対象にした効果的なアプリケーションを記述するために必要な、コード アクセス セキュリティの概念について説明します。

Microsoft Security Developer Center

安全なコードを開発するときに役立つ最新のセキュリティ問題について説明します。

New Technologies Help You Make Your Web Services More Secure

適切なセキュリティの重要性と Web サービスへの影響について説明します。

Determining When to Use Windows Installer Versus XCOPY

DOS XCOPY コマンドと、Microsoft Windows Installer テクノロジという、Microsoft .NET アプリケーションを配置する 2 つの手法について、検証と比較を行います。

セキュリティ ポリシーの実施

コード アクセス セキュリティ ポリシーを管理するうえで基本となる概念と、いくつかの最適な実施方法について説明します。

コード アクセスおよびアクセス許可セット

方法 : データ保護を使用する

データ保護を使用して、インメモリ データ、ファイル、またはストリームを、暗号化または復号化する手順について説明します。

方法 : セキュリティ ポリシーにカスタム アクセス許可を追加する

セキュリティ ポリシーにカスタム アクセス許可を追加する手順について説明します。

方法 : マネージ実行に対する Internet Explorer のセキュリティ設定を有効にする

Internet Explorer のセキュリティ設定を有効にする手順について説明します。

方法 : RequestMinimum フラグを使用して最小のアクセス許可を要求する

RequestMinimum フラグを使用して、FileIOPermission を要求する例を紹介します。

方法 : GenericPrincipal オブジェクトと GenericIdentity オブジェクトを作成する

GenericIdentity クラスを GenericPrincipal クラスと組み合わせて使用して、Windows NT ドメインや Windows 2000 ドメインから独立して存在する承認スキームを作成する例を紹介します。

方法 : WindowsPrincipal プロジェクトを作成する

WindowsPrincipal オブジェクトを作成する方法を 2 つ紹介します。コードがロール ベース検証を繰り返し実行する必要があるのか、または 1 回だけ実行するだけでよいのかによって、作成方法は決まります。

方法 : 強制セキュリティ チェックを実行する

強制的なチェックを使用して、GenericPrincipal が PrincipalPermission オブジェクトと一致することを確認する例を紹介します。

方法 : RequestRefuse フラグを使用することにより、アクセス許可を拒否する

RequestRefuse を使用して、共通言語ランタイムのセキュリティ システムから FileIOPermission が与えられることを拒否する例を紹介します。

方法 : アンマネージ コードへのアクセス許可を要求する

アンマネージ コードへのアクセス許可を要求する例を紹介します。

方法 : 名前付きアクセス許可セットに対するアクセス許可を要求する

名前付きアクセス許可セットについてアクセス許可要求を行う構文の例を紹介します。

方法 : RequestOptional フラグを使用してオプションのアクセス許可を要求する

SecurityAction.RequestOptional フラグを使用して、間接的にその他のアクセス許可をすべて拒否する、FileIOPermission を要求する例を紹介します。

方法 : キー コンテナに非対称キーを格納する

非対称キーを作成し、それをキー コンテナへ格納し、後でキーを取得し、最後にキー コンテナからキーを削除する方法について説明します。

方法 : Caspol.exe を使用してアセンブリをセキュリティ ポリシーに追加する

カスタム セキュリティ オブジェクトが実装されたアセンブリを、完全信頼アセンブリー覧に追加する方法について説明します。

方法 : Caspol.exe を使用してコード グループを表示する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、あるポリシー レベルに属するコード グループの簡単な一覧や、コード グループの名前と説明の一覧を表示する方法について説明します。

方法 : アクセス許可セットのアクセス許可を変更する

.NET Framework 構成ツール (Mscorcfg.msc) を使用して、アクセス許可セットのアクセス許可を変更する方法について説明します。

方法 : アクセス許可セットにアクセス許可を追加する

.NET Framework 構成ツール (Mscorcfg.msc) を使用して、アクセス許可セットにアクセス許可を追加する方法について説明します。

方法 : Caspol.exe を使用してポリシーの変更の警告を非表示にする

Caspol.exe を使用してポリシーの変更の警告を非表示にする方法について説明します。

方法 : コード グループのメンバシップ条件を変更する

Mscorcfg.msc を使用して、コード グループに関するメンバシップ条件を変更する方法について説明します。

方法 : Caspol.exe を使用してコード グループとアクセス許可セットを表示する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、アセンブリが所属するコード グループすべてを一覧表示する方法について説明します。

方法 : Caspol.exe を使用して、既定以外のユーザーに対するセキュリティ ポリシーを管理する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、現在のユーザー以外のユーザーに対するユーザー ポリシーを管理する方法について説明します。

方法 : 既存のコード グループに関連付けられているアクセス許可セットを変更する

Mscorcfg.msc を使用して、アクセス許可セットを変更する方法について説明します。

方法 : Caspol.exe を使用してアセンブリのアクセス許可の問題を分析する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、アセンブリが実行できない、保護されているリソースにアセンブリがアクセスする、動作してはいけないときに動作してしまうなどの症状の原因となる可能性のある問題をトラブルシューティングする方法について説明します。

方法 : Caspol.exe を使用してアクセス許可セットを表示する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、すべてまたは 1 つのポリシー レベルに属するアクセス許可セットの一覧を表示する方法について説明します。

方法 : Caspol.exe を使用してポリシーの変更を元に戻す

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、変更される直前のコンピュータ、ユーザー、エンタープライズの各ポリシーを回復する方法について説明します。

方法 : XML ファイルを使用してアクセ許可をインポートする

このようなアクセス許可の情報を XML ファイルに記述する方法の例を紹介します。

方法 : Caspol.exe を使用して既定のセキュリティ ポリシー設定に戻す

Caspol.exe を使用して、既定のセキュリティ ポリシー設定に戻す方法について説明します。

方法 : Caspol.exe を使用してコード グループを追加する

Caspol.exe を使用して、コード グループを追加する方法について説明します。

方法 : Caspol.exe の自己保護機構をオーバーライドする

自己保護機構を、必要に応じてオーバーライドする方法について説明します。

方法 : コード グループを作成する

Mscorcfg.msc を使用してコード グループを作成する方法について説明します。

方法 : 同時実行ガベージ コレクションを無効にする

ランタイムによるガベージ コレクションの実行方法を指定するときに、<gcConcurrent> 要素を使用する方法について説明します。

方法 : XML ファイルを使用してコード グループをインポートする

XML ファイル内に記述された、コード グループとそれに関連付けられているメンバシップ条件およびアクセス許可セットの情報の例を紹介します。

方法 : 発行者ポリシーを作成する

myAssembly のあるバージョンを別のバージョンにリダイレクトする、発行者ポリシー ファイルの例を紹介します。

方法 : Caspol.exe を使用してコード グループを削除する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、コード グループ階層構造からコード グループを削除する方法について説明します。

方法 : 構成ファイルでチャネル テンプレートを作成する

構成ファイルでチャネル テンプレートを作成する方法の例を紹介します。

方法 : Caspol.exe を使用してアクセス許可セットを変更する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、XML ファイルで指定した新しいセットで元のアクセス許可セットを置き換える方法について説明します。

方法 : アクセス許可セットを削除する

.NET Framework 構成ツール (Mscorcfg.msc) を使用して、特定のレベルのアクセス許可セットを削除する方法について説明します。

方法 : アクセス許可セットを作成する

.NET Framework 構成ツール (Mscorcfg.msc) を使用して、特定のレベルに対してアクセス許可セットを作成し、それを新しいコード グループまたは既存のコード グループに関連付ける方法について説明します。

方法 : コード グループを exclusive または final レベルに設定する

Mscorcfg.msc を使用して、新しいコード グループを exclusive または level final に設定する方法について説明します。

方法 : ポリシー アセンブリリストにアセンブリを追加する

.NET Framework 構成ツール (Mscorcfg.msc) を使用して、アセンブリを完全信頼のアセンブリリストに追加する方法について説明します。

方法 : XML ファイルを使用してアクセス許可セットをインポートする

XML ファイル内のアクセス許可セットおよびアクセス許可の例を紹介します。

方法 : DEVPATH を使用してアセンブリを指定する

DEVPATH 環境変数で指定されたディレクトリでランタイムがアセンブリを検索するように指定する例を紹介します。

方法 : ホスト アプリケーション ドメインのサーバー側でアクティブ化されるオブジェクトとクライアント側でアクティブ化されるオブジェクトを登録する

ホスト アプリケーション ドメインのサーバー側でアクティブ化されるオブジェクトとクライアント側でアクティブ化されるオブジェクトを登録する方法の例を紹介します。

方法 : Caspol.exe を使用してセキュリティ ポリシーを表示する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、セキュリティ ポリシー (コード グループ階層構造) を表示したり、すべてまたは 1 つのポリシー レベルの既知のアクセス許可セットの一覧を表示したりする方法について説明します。

方法 : Caspol.exe を使用してアクセス許可セットを追加する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、アクセス許可セットをコード グループに追加する方法について説明します。

方法 : Caspol.exe を使用してコード グループを変更する

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) の **–chggroup** オプションを使用して、コード グループの名前、メンバシップ条件、アクセス許可セット、フラグまたは説明を変更する方法について説明します。

方法 : チャネルを設定する

"http" とは異なる名前の HttpChannel を構築し、サーバー アプリケーションに使用する例を紹介します。

方法 : Caspol.exe を使用してセキュリティのオンとオフを切り替える

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して、セキュリティのオンとオフを切り替える方法について説明します。

方法 : アクセス許可セットからアクセス許可を削除する

.NET Framework 構成ツール (Mscorcfg.msc) を使用して、アクセス許可セットのアクセス許可を削除する方法について説明します。

方法 : .NET Framework 構成ツール (Mscorcfg.msc) を使用して一般的なセキュリティ タスクを実行する

.NET Framework 構成ツール (Mscorcfg.msc) を使用して、要件に応じたセキュリティ ポリシーを設定する方法について説明します。

その他のリソース

Microsoft Security Developer Center

安全なアプリケーションの開発に関する多数の文書やリソースを提供します。

Visual Studio 2005 Developer Center

Visual Studio 2005 を使用したアプリケーション開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Visual C# Developer Center

C# アプリケーションの開発に関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

Microsoft .NET Framework Developer Center

.NET Framework アプリケーションの開発やデバッグに関する多数の文書やリソースを提供します。このサイトは、定期的に新しい内容に更新されます。

参照

概念

C# での操作方法

Office のプログラミング (C# での操作方法)

このページでは、よく使用する Office プログラミングのタスクに関するヘルプへのリンクを紹介します。よく使用するタスクのその他のカテゴリに関するヘルプについては、「C# での操作方法」を参照してください。

一般

方法 : ドキュメント レベルのプロジェクトをアップグレードする

Microsoft Visual Studio 2005 Tools for Office へのアップグレードを手動で完了するために必要な手順について説明します。

Excel を使用したチュートリアル

Microsoft Office Excel 2003 の自動化、データ分析の実行、およびコントロールの操作という、3 種類の基本的なタスクについて説明します。

Word を使用したチュートリアル

Microsoft Office 2003 のツールを使用して Microsoft Office Word 2003 プロジェクトを自動化する方法について説明します。

方法 : プライマリ相互運用機能アセンブリを使用して Office アプリケーションを自動化する

既存のプロジェクトで追加の操作を実行し、Visual Basic または C# を使用したアンマネージ コードの呼び出しを可能にする手順について説明します。

方法 : Office ドキュメントにコントロールを追加する

デザイン時、または実行時に、Office ドキュメントへコントロールを追加する方法について説明します。

Word アプリケーションと Excel アプリケーション

方法 : Excel の計算をプログラムで実行する

アプリケーションの一部、または全体について、プログラムで計算を実行する方法について説明します。

方法 : Office メニューをプログラムで作成する

Microsoft Office Excel 2003 のメニュー バーに、"New Menu" というメニューを作成する例を紹介します。

方法 : Office ツール バーをプログラムで作成する

"Test" という名前のツール バーを Microsoft Office Word 2003 に作成する例を紹介します。このツール バーはドキュメントの中央付近に表示され、ボタンが 2 つあります。ボタンをクリックすると、メッセージ ボックスが表示されます。

方法 : Office プロジェクトのエラーを処理する

デバッガが共通言語ランタイム例外で中断されるように設定する方法について説明します。

参照

概念

C# での操作方法

スマート デバイス

このページでは、よく使用するスマート デバイスのプログラミング タスクに関するヘルプへのリンクを紹介します。よく使用するタスクのその他のカテゴリに関するヘルプについては、「C# での操作方法」を参照してください。

スマート デバイス入門

スマート デバイス プロジェクトの新機能

Visual Studio 2005 で使用できるようになった新機能と拡張された機能について説明します。

デバイス機能と必要な開発ツール

さまざまなスマート デバイス ハードウェア、ハードウェア機能、および開発ツールを表で紹介します。

デバイス プロジェクトのリモート ツール

embedded Visual C++ 4.0 で使用できたリモート ツールの一覧です。このリモート ツールは Visual Studio 2005 と同梱され、デバイス アプリケーションの開発とデバッグに役立っていました。

方法 : スマート デバイス開発用のヘルプを最適化する

インストールしたヘルプを確認する方法、モバイルおよび Embedded 開発 ヘルプを追加する方法、およびヘルプをフィルタ処理する方法について説明します。

方法 : Visual Studio でデバイス エミュレータを起動する

[デバイスへの接続] ダイアログ ボックスまたはデバイス エミュレータ マネージャを使用して、デバイス エミュレータを起動する方法について説明します。

以前のツールで作成されたプロジェクトの更新

Visual Studio 2005 開発環境の改良点について説明します。

開発言語の選択

スマート デバイスに配置するアプリケーション、コントロール、またはライブラリを開発するときに、使用できるプログラミング言語のオプションについて説明します。

チュートリアル : デバイス対応の Windows フォーム アプリケーションの作成

デスクトップのプログラミングと、デバイスのプログラミング、つまりデバイスを対象にする場合のプログラミングでの主な違いについて説明します。このチュートリアルでは、デバイスとは Pocket PC 2003 の組み込みエミュレータです。

Visual Basic と Visual C# を使用したスマート デバイスのプログラミング

.NET Compact Framework を使用したデバイスのプログラミング

Visual Basic 言語または Visual C# 言語、および .NET Compact Framework を使用してスマート デバイス アプリケーションを開発するための情報について説明します。

デバイス プロジェクト用 .NET Compact Framework リファレンス

.NET Framework クラス ライブラリのサブセットとしての .NET Compact Framework に関する情報について説明します。

方法 : Visual Basic または Visual C# を使用してデバイス アプリケーションを作成する

デバイス アプリケーションの作成方法と、このプロセスとデスクトップ アプリケーション開発との相違点について説明します。

方法 : プラットフォーム間でソース コードを共有する (デバイス)

コンパイラ定数を使用してさまざまなプラットフォーム間で同じソース コードを共有する方法について説明します。

方法 : デバイス プロジェクトのプラットフォームを変更する

同じプロジェクト内でプラットフォームを切り替える方法について説明します。

方法 : プロジェクトを新しいバージョンの .NET Compact Framework にアップグレードする

新しいバージョンのプラットフォームがインストールされた場合に、既存のプロジェクトのプラットフォームをアップグレードする方法について説明します。

デバイス プロジェクト内のコード スニペットの管理

デバイス プロジェクトだけに関係するスニペットの使用方法について説明します。

方法 : デバイス プロジェクトのコードがプラットフォームでサポートされているか検査する

コードが対象プラットフォームでサポートされているかどうかを確認する方法について説明します。

方法 : HardwareButton イベントを処理する (デバイス)

Pocket PC 上のアプリケーション キーをオーバーライドする方法について説明します。

方法 : フォームの向きおよび解像度を変更する (デバイス)

既定値が不正または欠けている場合に、フォームの向きと解像度を変更する方法について説明します。

方法 : 既定のデバイスを変更する (マネージ プロジェクト)

プロジェクトの開発中に対象デバイスを変更する方法について説明します。

方法 : スマート デバイス開発用のヘルプを最適化する

スマート デバイス ヘルプのフィルタを使用して、デバイス アプリケーション開発でサポートされている .NET Framework 要素のみを表示する方法について説明します。

デバイスの接続

方法 : Virtual PC セッションからデバイス エミュレータに接続する

TCP/IP を使用せずに接続する方法について説明します。

方法 : Smartphone エミュレータ ファイル システムにアクセスする

独自のファイル ビューアを持たない、Smartphone エミュレータのファイル システムにアクセスする方法について説明します。

方法 : Bluetooth を使用して接続する

Bluetooth を使用して接続する方法について説明します。

方法 : IR を使用して接続する

赤外線を使用して接続する方法について説明します。

方法 : ActiveSync を使用せずに Windows CE デバイスに接続する

ActiveSync サービスを使用できない場合にデバイスに接続する方法について説明します。

方法 : エミュレータから開発用コンピュータのファイルにアクセスする

共有フォルダを使用して、エミュレータから開発用コンピュータのファイルにアクセスする方法について説明します。

方法 : 接続オプションの設定 (デバイス)

接続オプションを設定するためのコモン ダイアログ ボックスの場所について説明します。

スマート デバイスのデバッグ

デバイスのデバッグとデスクトップのデバッグとの違い

デバイスのデバッガとデスクトップのデバッガの違いについて説明します。

方法 : マネージ デバイスのプロセスに接続する

マネージ デバイス プロセスにアタッチする方法について説明します。

方法 : デバイスのレジストリ設定を変更する

リモート レジストリ エディタを使用してデバイスのレジストリ設定を変更する方法について説明します。

チュートリアル : マネージ コードとネイティブ コードの両方を含むソリューションのデバッグ

これらの混合ソリューションをデバッグする方法について説明します。

マネージ デバイス プロジェクトのデータ

方法 : SqlCeResultSet コードを生成する (デバイス)

データセットの代わりに結果セットを生成する方法について説明します。

方法 : デザイン時の接続文字列を変更する (デバイス)

Visual Studio がデザイン時に SQL Server Mobile データベースへの接続に使用する文字列を変更する方法について説明します。

方法 : 実行時の接続文字列を変更する (デバイス)

アプリケーションが実行時に SQL Server Mobile データベースへの接続に使用する文字列を変更する方法について説明します。

方法 : ナビゲーション ボタンを追加する (デバイス)

.NET Compact Framework ではサポートされていない **DataNavigator** クラスの代替手法について説明します。

方法 : データベースのデータの変更を永続化する (デバイス)

データセットの変更内容を再びデータベースに適用する方法について説明します。

方法 : データベースを作成する (デバイス)

Visual Studio 環境を使用してプロジェクトの内部または外部に SQL Mobile データベースを作成する方法について説明します。

方法 : デバイス プロジェクトにデータベースを追加する

サーバー エクスプローラで使用できる SQL Mobile データベースを Visual Basic または Visual C# プロジェクトのデータ ソースとして追加する方法について説明します。

方法 : SQL Server データベースをデータ ソースとして追加する (デバイス)

SQL Server データベースを Visual Basic または Visual C# プロジェクトのデータ ソースとして追加する方法について説明します。

方法 : ビジネス オブジェクトをデータ ソースとして追加する (デバイス)

ビジネス オブジェクトを Visual Basic または Visual C# プロジェクトのデータ ソースとして追加する方法について説明します。

方法 : Web サービスをデータ ソースとして追加する (デバイス)

Web サービスを Visual Basic または Visual C# プロジェクトのデータ ソースとして追加する方法について説明します。

方法 : データベースのテーブルを管理する (デバイス)

テーブルを追加および削除する方法と、既存のテーブル スキーマを編集する方法について説明します。

方法 : データベースの列を管理する (デバイス)

列を追加および削除する方法と、列のプロパティを編集する方法について説明します。

方法 : データベースのインデックスを管理する (デバイス)

インデックスを追加および削除する方法と、インデックスの並べ替え順序のプロパティを変更する方法について説明します。

方法 : データベースのパスワードを管理する (デバイス)

新しい SQL Mobile データベースのパスワードを設定する方法と、既存のデータベースのパスワードを変更する方法について説明します。

方法 : データベースを縮小および修復する (デバイス)

SQL Mobile データベースを縮小および修復する方法について説明します。

方法 : パラメータ付きクエリを作成する (デバイス)

パラメータ クエリの作成方法について説明します。

チュートリアル : パラメータ クエリ アプリケーション

パラメータ クエリの作成を含む、完全なプロジェクトの詳細な手順について説明します。

方法 : マスター/詳細アプリケーションを作成する (デバイス)

マスター/詳細リレーションシップの実装方法について説明します。

チュートリアル : データベース マスター/詳細アプリケーション

マスター/詳細アプリケーションの作成と実行を含む、完全なプロジェクトの詳細な手順について説明します。

方法 : データベース内のデータをプレビューする (デバイス)

データベース内のデータを表示するためのいくつかのオプションについて説明します。

方法 : データ アプリケーション用の概要ビューと編集ビューを生成する (デバイス)

データ フォームを使用して、データ グリッドの単一データ行を表示および編集する方法について説明します。

デバイス ソリューションのパッケージ化と配置

チュートリアル : 配置用のスマート デバイス ソリューションのパッケージ化

アプリケーションとそのリソースをパッケージ化する詳細な手順を示します。

デバイス プロジェクトにおけるセキュリティ

方法 : デバイス プロジェクトで証明書をインポートおよび適用する

デバイス プロジェクトの署名に、[証明書の選択] ダイアログ ボックスを効率的に使用する方法について説明します。

方法 : Signtool.exe をビルド後のイベントとして起動する (デバイス)

ビルド後のイベントにより元のバイナリが変更される際に、プロジェクトに署名する方法について説明します。

方法 : デバイスのセキュリティ モデルを照会する

デバイス証明書ストアに既にインストールされている証明書を調べる方法について説明します。

方法 : Visual Basic アプリケーションまたは Visual C# アプリケーションに署名する (デバイス)

.NET Compact Framework に対して記述されたアプリケーションに署名する手順を示します。

方法 : Visual Basic アセンブリまたは Visual C# アセンブリに署名する (デバイス)

プロジェクト アセンブリに署名する手順を示します。

方法 : CAB ファイルに署名する (デバイス)

デバイス CAB プロジェクトに署名する手順を示します。

方法 : Visual Basic プロジェクトまたは Visual C# プロジェクトでデバイスを用意する

マネージ プロジェクトのデバイス ストアにデジタル証明書を追加する手順を示します。

方法 : セキュリティ モデルを使用したデバイスを用意する

RapiConfig.exe を使用して、セキュリティ モデルによってデバイスを用意する方法について説明します。

参照

概念

C# での操作方法

配置 (C# での操作方法)

このページでは、よく使用する配置タスクに関するヘルプへのリンクを紹介します。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「C# での操作方法」を参照してください。

ClickOnce

方法 : ClickOnce アプリケーションを発行する

ClickOnce アプリケーションを Web サーバー、ファイル共有、またはリムーバブル メディアに発行して、ユーザーが ClickOnce アプリケーションを利用できるようにする方法について説明します。

方法 : 発行場所を指定する

アプリケーション ファイルとマニフェストを配置する場所を指定する方法について説明します。

方法 : インストール URL を指定する

インストールの URL プロパティを使用して、ユーザーがアプリケーションをダウンロードする Web サーバーを指定する方法について説明します。

方法 : サポート URL を指定する

Support URL プロパティを使用して、ユーザーがアプリケーションに関する情報を取得できる Web ページまたはファイル共有を指定する方法について説明します。

方法 : ClickOnce のインストール モードを指定する

アプリケーションをオフラインまたはオンラインのどちらで利用できるようにするかを指定する、インストール モードの設定方法について説明します。

方法 : CD インストールの自動開始を有効にする

メディアを挿入したときに ClickOnce アプリケーションが自動的に起動するように、自動開始を有効にする方法について説明します。

方法 : ClickOnce の発行バージョンを設定する

発行するアプリケーションを更新として扱うかどうかを指定する Publish Version プロパティの設定方法について説明します。

方法 : ClickOnce の発行バージョンを自動的にインクリメントする

アプリケーションが更新として発行されるように、Publish Version プロパティを変更する方法について説明します。

方法 : ClickOnce で発行されるファイルを指定する

ファイルを除外する方法、ファイルをデータ ファイルや必須コンポーネントとしてマークする方法、および条件付きでインストールするファイルのグループを作成する方法について説明します。

方法 : ClickOnce アプリケーションと共に必須コンポーネントをインストールする

アプリケーションと共にパッケージ化される必須コンポーネントを指定する方法について説明します。

方法 : ClickOnce アプリケーションの更新を管理する

更新チェックを実行する時期と方法、更新が必須かどうか、およびアプリケーションの更新をチェックする場所を指定する方法について説明します。

方法 : ClickOnce アプリケーションにデータ ファイルを含める

任意の種類のデータ ファイルを ClickOnce アプリケーションに追加するための手順について説明します。

方法 : Systems Management Server を使用して .NET Framework を配置する

Systems Management Server を実行するサーバーで、実行する必要のあるタスクについて説明します。

方法 : ClickOnce アプリケーション用の信頼された発行者をクライアント コンピュータに追加する

コマンド ライン ツールの CertMgr.exe を使用して、クライアント コンピュータの信頼された発行者ストアに発行者の証明書を追加する方法について説明します。

方法 : 配置の更新用に別の場所を指定する

配置マニフェストで、初期インストール後に Web サイトからアプリケーションを更新できるように、更新用の別の場所を指定する方法について説明します。

方法 : Active Directory を使用して .NET Framework を配置する

Active Directory を使用して、.NET Framework を配置する手順について説明します。

方法 : ClickOnce アプリケーションでクエリ文字列を取得する

ClickOnce アプリケーションを使用して、クエリ文字列の内容を取得する手順について説明します。また、ClickOnce アプリケーションで短いコードを使用して、アプリケーションの最初の起動時にクエリ文字列の値を読み取る方法についても説明します。

方法 : ClickOnce のセキュリティ設定を有効にする

配置時にセキュリティ設定を一時的に無効にする方法について説明します。

方法 : ClickOnce アプリケーションのセキュリティ ゾーンを設定する

セキュリティ ゾーンを設定し、アプリケーション テーブルで必要なアクセス許可を作成する方法について説明します。

方法 : ClickOnce アプリケーションのカスタム アクセス許可を設定する

アプリケーションが正しく動作するために必要な特定のアクセス許可だけを付与する方法について説明します。

方法 : ClickOnce アプリケーションのアクセス許可を調べる

アクセス許可の検出ツールを実行してアプリケーションを分析し、必要なアクセス許可を調べる方法について説明します。

方法 : アクセス許可が制限された ClickOnce アプリケーションをデバッグする

エンド ユーザーと同じアクセス許可を使用してアプリケーションをデバッグする方法について説明します。

Windows インストーラ
Windows インストーラ配置

Windows インストーラの配置を使用して、ユーザーに配布するインストーラ パッケージを作成する方法に関する文書へのリンクを示します。

チュートリアル : Windows ベースのアプリケーションの配置

メモ帳を起動する Windows アプリケーションのインストーラの作成手順について説明します。

チュートリアル : マージ モジュールを使用した共有コンポーネントのインストール

マージ モジュールを使用して共有コンポーネントをパッケージ化および配布することで、一貫性のある配置を実現する方法について説明します。

チュートリアル : カスタム動作の作成

インストール終了時にユーザーを Web ページへ導く DLL カスタム動作の作成手順について説明します。

チュートリアル : カスタム動作を使用した、インストール時のメッセージの表示

カスタム動作を使用してユーザー入力を受け取り、インストール中に表示するメッセージ ボックスにユーザー入力を渡す方法について説明します。

チュートリアル : カスタム動作を使用した、インストール時のアセンブリのプリコンパイル

インストール時にアセンブリをネイティブ コードにプリコンパイルするために、DLL のパス名を CustomActionData プロパティに渡す方法について説明します。

チュートリアル : カスタム動作を使用して、インストール時にデータベースを作成する

カスタム動作と CustomActionData プロパティを使用して、インストール時にデータベースとデータベース テーブルを作成する方法について説明します。

チュートリアル : インストール時にアプリケーションを別の XML Web サービスにリダイレクトする

URL Behavior プロパティ、Installer クラス、および Web セットアップ プロジェクトを使用して、異なる XML Web サービスにリダイレクトできる Web アプリケーションを作成する方法を示します。

方法 : Windows インストーラ配置で必須コンポーネントをインストールする

インストール時にコンポーネントの有無を自動的に検出し、事前に定義された一連の必須コンポーネントをインストールする方法 (ブートストラップと呼ばれるプロセス) について説明します。

方法 : 配置プロジェクトを作成または追加する

開発時および開発後にソリューションを配置する場所と方法を指定する方法について説明します。

方法 : セットアップ プロジェクトを作成または登録する

Windows インストーラ (.msi) ファイルの作成方法について説明します。このファイルは、別のコンピュータや Web サーバーにアプリケーションを配布してインストールするために使用します。

方法 : マージ モジュール プロジェクトの作成または登録を行う

マージ モジュール プロジェクトを作成し、複数のアプリケーション間で共有されるファイルやコンポーネントをパッケージ化する方法について説明します。

方法 : Cab プロジェクトを作成または登録する

CAB プロジェクトを作成し、Web ブラウザにコンポーネントをダウンロードするときに使用できるキャビネット (.cab) ファイルを作成する方法について説明します。

方法 : 配置プロジェクトのプロパティを設定する

[配置プロパティ] ダイアログ ボックスを使用して、構成依存プロパティを設定する方法について説明します。

方法 : 配置プロジェクトに項目を追加する

インストーラに含める必要がある項目を指定し、それを対象コンピュータのどの場所にインストールするかを指定する方法について説明します。

方法 : 配置プロジェクトにマージ モジュールを登録する

マージ モジュール (.msm ファイル) を使用して、複数の配置プロジェクト間でコンポーネントを共有する方法について説明します。

方法 : アイコンを追加および削除する

対象コンピュータにアプリケーションをインストールする際に、アイコンをインストールしてアプリケーションと関連付ける方法について説明します。

方法 : 配置プロジェクトから項目を除外する

配置プロジェクトから特定のファイルを除外する方法について説明します。

方法 : オペレーティング システムのバージョンに基づく条件付きインストールを設定する

Condition プロパティを設定して条件ロジックをインストーラに追加し、たとえば、オペレーティング システムのバージョンごとに異なるファイルをインストールしたり、異なるレジストリ値を設定したりする方法について説明します。

参照
概念
C# での操作方法
C# アプリケーションの配置

Visual C# IDE の使用

ここでは、Visual C# の統合開発環境 (IDE) について紹介します。また、プロジェクトの設定から、完成したアプリケーションをエンド ユーザーに配布するまで、開発サイクルの各段階で IDE を使用する方法について説明します。

このセクションの内容

IDE の概要 (Visual C#)

Visual C# 統合開発環境を構築するエディタとウィンドウの使い方について説明します。

プロジェクトの作成 (Visual C#)

作成するアプリケーションの種類に適したプロジェクトを設定する方法について説明します。

ユーザー インターフェイスのデザイン (Visual C#)

デザイナを使用してアプリケーションにコントロールを追加する方法について説明します。

コードの編集 (Visual C#)

コード エディタを使用してソース コードを入力する方法、および IntelliSense、コード スニペット、リファクタリングなどのツールの使用方法について説明します。

移動と検索 (Visual C#)

プロジェクト ファイル内の移動方法、すばやく検索する方法について説明します。

ビルドとデバッグ (Visual C#)

プロジェクト ファイルから実行可能アセンブリを作成する方法、および Visual Studio デバッガで実行する方法について説明します。

コードのモデリングと解析 (Visual C#)

クラス デザイナでクラスの関係を表示する方法、オブジェクト テスト ベンチでオブジェクトをテストする方法、およびコード解析ツールを実行する方法について説明します。

リソースの追加と編集 (Visual C#)

アイコン、文字列、テーブルなどの種類のデータ ファイルをプロジェクトに追加する方法について説明します。

ヘルプの表示 (Visual C#)

必要なドキュメントを検索する方法について説明します。

C# アプリケーションの配置

エンド ユーザーにアプリケーションを配布する方法について説明します。

Visual C# のコード エディタの機能

コード スニペットやリファクタリングなど、Visual C# に固有な IDE 機能に関する参考資料のすべてを紹介します。

Visual C# IDE の設定

既定の Visual C# 設定を変更する方法について説明します。

Visual C# のショートカット キー

Visual C# の操作を高速にする方法について説明します。

参照

その他の技術情報

Visual C#
Visual C# について
C# リファレンス
Visual Studio の統合開発環境

IDE の概要 (Visual C#)

Visual C# の統合開発環境 (IDE: Integrated Development Environment) は、共通のユーザー インターフェイスで公開される開発ツールのコレクションです。ツールの一部は他の Visual Studio 言語と共有されますが、C# コンパイラなど、Visual C# に固有のものもあります。ここでは、開発プロセスのさまざまな段階において IDE で作業するときに、重要な Visual C# ツールの使用方法について概要を説明します。

メモ：

ASP.NET 2.0 Web アプリケーションを開発する場合、Visual Web Developer IDE を使用します。この IDE は、Visual Studio 2005 と完全に統合されています。ただし、分離コード ページが Visual C# の場合は、Visual Web Developer 内で Visual C# コード エディタを使用します。そのため、「ユーザー インターフェイスのデザイン (Visual C#)」など、このセクションの一部のトピックは、Web アプリケーションにそのまま適用できないこともあります。

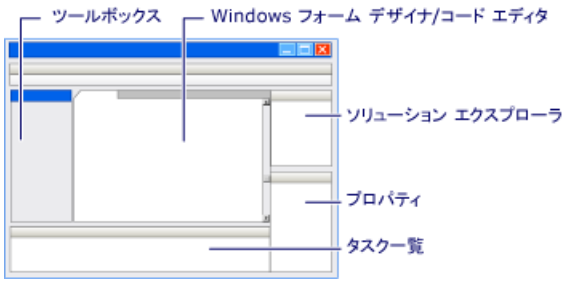
Visual C# ツール

次に、Visual C# で重要なツールとウィンドウを示します。ほとんどのツールのウィンドウは、[表示] メニューから開くことができます。

- コード エディタ。ソース コードを記述します。
- C# コンパイラ。C# ソース コードを実行可能プログラムに変換します。
- Visual Studio デバッガ。プログラムをテストします。
- ツールボックスとデザイナ。マウスを使用して、ユーザー インターフェイスを簡単に開発できます。
- ソリューション エクスプローラ。プロジェクト ファイルと設定を表示および管理します。
- プロジェクト デザイナ。コンパイラ オプション、配置パス、リソースなどを設定します。
- クラス ビュー。ファイル別ではなく、型別にソース コード全体をナビゲーションします。
- プロパティ ウィンドウ。ユーザー インターフェイスのコントロールに関するプロパティとイベントを設定します。
- オブジェクト ブラウザ。.NET Framework アセンブリや COM オブジェクトなど、ダイナミック リンク ライブラリで使用できるメソッドとクラスを表示します。
- Document Explorer。ローカル コンピュータまたはインターネット上にある製品ドキュメントの表示と検索を行います。

IDE でツールを表示する方法

IDE では、ウィンドウ、メニュー、プロパティ ページ、およびウィザード経由でツールとやり取りします。基本的な IDE は次のようになります。



Ctrl キーを押しながら Tab キーを押すと、開かれている任意のツール ウィンドウやファイルにすばやくアクセスできます。詳細については、「移動と検索 (Visual C#)」を参照してください。

エディタと Windows フォーム デザイナのウィンドウ

コード エディタと Windows フォーム デザイナのどちらも、大きなメイン ウィンドウを使用します。コード ビューとデザイン ビューを切り替えるには、F7 キーを押します。または、[表示] メニューの [コード] か [デザイナ] をクリックします。デザイン ビューでは、ツールボックスからウィンドウにコントロールをドラッグして配置できます。コントロールは、左側のマージンにある [ツールボックス] タブをクリックすると表示されます。コード エディタの詳細については、「コードの編集 (Visual C#)」を参照してください。Windows フォーム デザイナの詳細については、「Windows フォーム デザイナ」を参照してください。

右下の [プロパティ] ウィンドウはデザイン ビューでのみ表示されます。このウィンドウで、ボタン、テキスト ボックスなどのユーザー インターフェイス コントロールについて、プロパティを設定し、イベントをフックします。このウィンドウを [自動的に隠す] に設定すると、[コード ビュー] に切り替えるたびに、右側のマージンに最小化表示されます。[プロパティ] ウィンドウとデザイナの詳細については、「ユーザー インターフェイスのデザイン (Visual C#)」を参照してください。

ソリューション エクスプローラとプロジェクト デザイナ

右上のウィンドウはソリューション エクスプローラです。プロジェクトのすべてのファイルが階層的なツリー ビューで表示されます。[プロジェクト] メニューを使用し、新しいファイルをプロジェクトに追加すると、ソリューション エクスプローラに表示されます。ソリューション エクスプローラには、ファイルだけでなく、プロジェクト設定や、アプリケーションに必要な外部ライブラリへの参照も含まれます。

[プロジェクト デザイナ] プロパティ ページにアクセスするには、ソリューション エクスプローラの [プロパティ] ノードを右クリックし、[開く] をクリックします。ビルド オプション、セキュリティ要件、配置の詳細などのプロジェクト プロパティを変更するときに、このページを使用します。ソリューション エクスプローラとプロジェクト デザイナの詳細については、「プロジェクトの作成 (Visual C#)」を参照してください。

コンパイラ、デバッガ、およびエラー一覧のウィンドウ

C# コンパイラは対話形式のツールではないため、ウィンドウはありませんが、[プロジェクト デザイナ] でコンパイラ オプションを設定できます。[ビルド] メニューの [ビルド] をクリックすると、IDE から C# コンパイラが呼び出されます。ビルドが成功すると、ステータス ペインにビルド正常終了メッセージが表示されます。ビルド エラーがあると、エディタ/デザイナ ウィンドウの下部にエラー一覧が記載された [エラー一覧] ウィンドウが表示されます。エラーをダブルクリックすると、ソース コードの該当する行にジャンプします。F1 キーを押すと、強調表示されているエラーのヘルプ ドキュメントが表示されます。

デバッガにはさまざまなウィンドウがあり、実行しているアプリケーションの変数値や型に関する情報が表示されます。デバッグ時には、[コード エディタ] ウィンドウを使用して、デバッガの実行を一時停止する行を指定し、一度に 1 行ずつコードをステップ実行できます。詳細については、「ビルドとデバッグ (Visual C#)」を参照してください。

IDE のカスタマイズ

Visual C# のウィンドウはいずれも、ドッキング可能またはフローティングにしたり、非表示/表示を切り替えたり、新しい位置へ移動したりできます。ウィンドウの動作を変更するには、タイトル バーの下向きの矢印かプッシュピン アイコンをクリックし、使用できるオプションから選択します。ドッキング ウィンドウを新しいドッキング位置に移動するには、ウィンドウ ドロッパー アイコンが表示されるところまでタイトル バーをドラッグします。マウスの左ボタンを押したままで、新しい位置のアイコン上にマウス ポインタを移動します。ウィンドウをドッキングする側 (左側、右側、上部、または下部) のアイコン上にポインタを移動します。中間のアイコンにポインタを移動すると、ウィンドウがタブ付きウィンドウになります。ポインタを移動するときに、青色の半透明の四角形が表示されます。これは、ウィンドウをドッキングする新しい位置を示します。

[ツール] メニューの [オプション] をクリックすると、IDE のさまざまな機能をカスタマイズできます。詳細については、「[オプション] ダイアログ ボックス (Visual Studio)」を参照してください。

参照

概念

ユーザー インターフェイスのデザイン (Visual C#)
プロジェクトの作成 (Visual C#)
コードの編集 (Visual C#)
ビルドとデバッグ (Visual C#)

その他の技術情報

Visual C#
Visual C# IDE の使用
Visual Studio の統合開発環境
Visual Web Developer
Visual Web Developer のユーザー インターフェイス要素

プロジェクトの作成 (Visual C#)

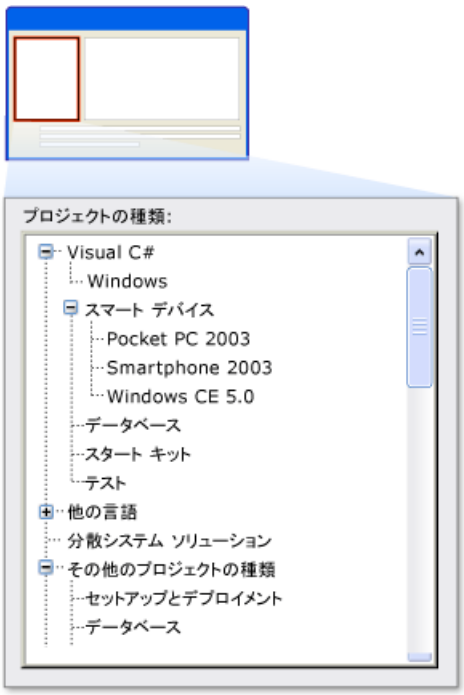
コーディング作成を始める場合、第 1 段階はプロジェクトを設定することです。プロジェクトには、アプリケーションの素材のすべてが含まれています。たとえば、ソース コード ファイルだけでなく、アイコンなどのリソース ファイル、プログラムが依存する外部ファイルへの参照、およびコンパイラの設定などの構成データがあります。プロジェクトをビルドすると、Visual C# からは C# コンパイラなどの内部ツールが呼び出され、プロジェクト内のファイルを基にして実行可能なアセンブリが作成されます。

新規プロジェクトの作成

新しいプロジェクトを作成するには、[ファイル] メニューをクリックし、[新規作成] をポイントして、[プロジェクト] をクリックします。

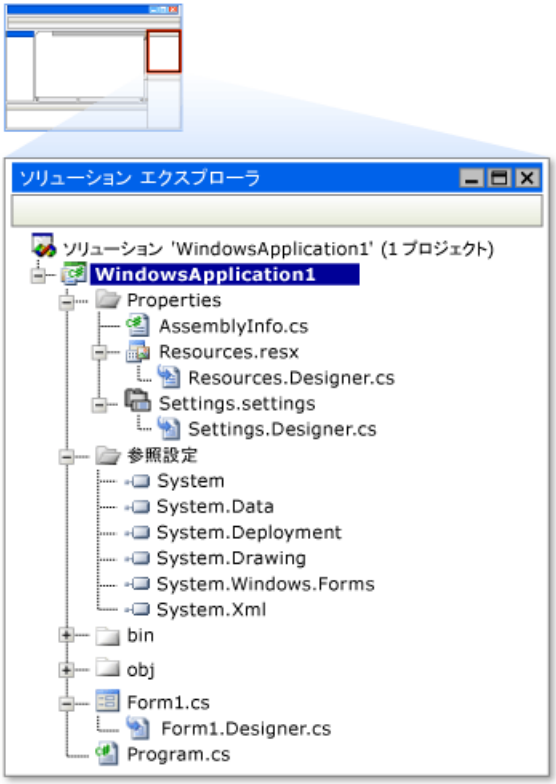
メモ：
[プロジェクト] ではなく [Web サイト] を選択すると、Visual Web Developer の統合開発環境 (IDE) が開きます。IDE は、ASP.NET Web アプリケーションを作成する Visual Studio 内の環境とは区別されます。Visual Web Developer IDE では、C# の分離コード ファイルを編集するときに、Visual C# コード エディタを使用します。Web アプリケーションを作成する場合、Visual Web Developer のドキュメントを主に使用しますが、C# エディタの詳細については、「コードの編集 (Visual C#)」を参照してください。

次に、[新しいプロジェクト] ダイアログ ボックスの図を示します。ウィンドウの左側では、既定で、[Visual C#] が選択されています。右側には、6 個以上のプロジェクト テンプレートがあり、この中から選択できます。左側の [スマート デバイス] ノードまたは [その他のプロジェクトの種類] ノードを展開すると、右側に異なるプロジェクトの種類が表示されます。



スタート キットは、プロジェクト テンプレートの一種です。スタート キットをインストールすると、[新しいプロジェクト] ダイアログ ボックスに表示されます。詳細については、「スタート キット」を参照してください。

プロジェクト テンプレートを選択し、[OK] をクリックすると、プロジェクトが作成され、コーディングを始めることができます。プロジェクト ファイル、参照、設定、リソースは、[ソリューション エクスプローラ] ウィンドウの右側に表示されます。



プロジェクトに含まれるもの

プロパティ

[プロパティ] ノードは、プロジェクト全体に適用される構成の設定を表し、ソリューション フォルダの .csproj ファイルに格納されます。たとえば、コンパイル オプション、セキュリティ設定、配置設定などが含まれます。プロジェクトを修正するには、プロジェクト デザイナを使用します。[プロパティ] を右クリックし、[開く] をクリックすると、プロジェクト デザイナに一連の [プロパティ ページ] が表示されます。詳細については、「プロジェクト プロパティの変更 (Visual C#)」を参照してください。

参照

プロジェクトのコンテキストでは、参照とは、単にアプリケーションを実行するときに必要なバイナリ ファイルです。一般に、参照によって、.NET Framework クラス ライブラリ ファイルなどの DLL ファイルが識別されます。また、.NET アセンブリ ("shim" と呼ばれます) も参照できます。これによって、COM オブジェクトやネイティブ Win32 DLL でアプリケーションからメソッドを呼び出すことができるようになります。プログラムで、他のアセンブリで定義されたクラスのインスタンスを作成する場合、プロジェクトをコンパイルする前に、そのアセンブリ ファイルへの参照をプロジェクトに追加する必要があります。参照を追加するには、[プロジェクト] メニューの [参照の追加] をクリックします。既定で、C# プロジェクトには必ず mscorlib.dll への参照が含まれます。この DLL には、コア .NET Framework クラスが含まれます。その他の .NET Framework DLL や他のファイルへの参照を追加するには、[プロジェクト] メニューをクリックし、[参照の追加] を選択します。

メモ :

プロジェクトの参照の概念と、C# などのプログラミング言語で使用される参照型の概念とを混同しないでください。前者はファイルとディスク上の格納位置を参照します。後者は、**class** キーワードを使用して宣言された C# の型を参照します。

リソース

リソースはアプリケーションに含まれるデータですが、他のソース コードとは独立して変更できる方法で格納されます。たとえば、文字列は、ソース コードにハードコーディングせずに、すべてリソースとして格納できます。文字列は、後で別の言語に翻訳してから、アプリケーション フォルダに追加できます。これで、アセンブリを再コンパイルしなくても顧客にリリースできます。Visual C# では、文字列、イメージ、アイコン、オーディオ、およびファイルという 5 種類のリソースが定義されています。リソース デザイナを使用して、リソースの追加、削除、または編集を行います。リソース デザイナにアクセスするには、[プロジェクト デザイナ] の [リソース] タブをクリックします。

フォーム

Windows フォーム プロジェクトを作成すると、Visual C# では、既定で Form1 という 1 つのフォームがプロジェクトに追加されます。フォームを表す 2 つのファイルは、Form1.cs と Form1.designer.cs という名前です。Form1.cs にはコードを記述します。designer.cs は、ツールボックスからコントロールをドラッグ アンド ドロップしたときに、そのコントロールの実行コードが、Windows フォーム デザイナによって自動的に記述されるファイルです。

[プロジェクト] メニュー項目をクリックし、[Windows フォームの追加] を選択して、新しいフォームを追加します。各フォームには、2 つのファイルが関連付けられています。Form1.cs (名前を変更することもできます) には、フォームとコントロール (リスト ボックスやテキスト ボックスなど) を構成するソース コード、およびボタンのクリックやキーストロークなどのイベントに応答するソース コードが含まれます。単純な Windows フォーム プロ

ジェクトであれば、このファイルでコーディングの大部分またはすべてを処理できます。

Designer.cs ファイルには、フォームにコントロールをドラッグしたときや、[プロパティ] ウィンドウでプロパティを設定したときなどに、デザイナで自動的に記述されたソース コードが含まれます。一般に、このファイルは手動で編集しないことをお勧めします。

メモ：
当然のことですが、コンソール アプリケーション プロジェクトを作成する場合、Windows フォームにソース コード ファイルは含まれません。

その他のソース コード ファイル

プロジェクトには、特定の Windows フォームに関連付けられているかどうかにかかわらず、その他にも .cs ファイルが含まれます。前述の [ソリューション エクスプローラ] の図には、program.cs にアプリケーションのエントリ ポイントが含まれます。1 つの .cs ファイルに定義されるクラスと構造体の数は、任意です。[プロジェクト] メニューの [新しい項目の追加] または [既存項目の追加] をクリックして、プロジェクトに新規または既存のファイルやクラスを追加できます。

参照

処理手順

方法 : ソリューションとプロジェクトのビルド構成を作成する
方法 : Windows アプリケーション プロジェクトを作成する

概念

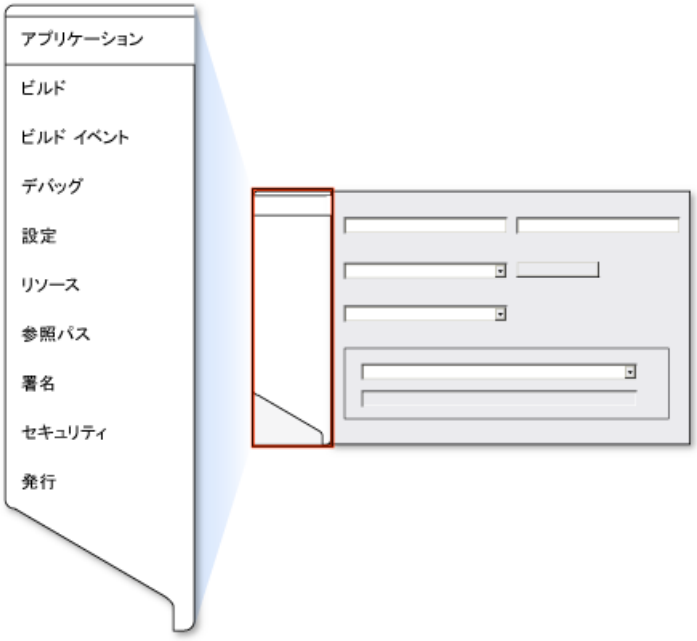
ソリューション、プロジェクト、および項目の概要
ソリューション エクスプローラの使用
ソリューション エクスプローラで非表示のプロジェクト ファイル
プロジェクトとソリューションの制御

その他の技術情報

Visual C#
Visual C# IDE の使用

プロジェクト プロパティの変更 (Visual C#)

プロジェクトを作成した後は、プロジェクト デザイナを使用して、実行可能ファイルの名前変更、ビルド プロセスのカスタマイズ、DLL への参照の追加、セキュリティ設定の強化などのタスクを実行できます。[プロパティ] をクリックするか、[ソリューション エクスプローラ] の [プロパティ] 項目を右クリックして、[プロジェクト] メニューのプロジェクト デザイナにアクセスします。プロジェクト デザイナは、次の図のように、エディタ/デザイナ ウィンドウに表示されます。



プロジェクトのプロパティは、プロジェクト デザイナで 10 ページにグループ化されています。プロジェクト デザイナのプロパティ ページは、Windows フォーム デザイナとコード エディタで使用するのと同じ中間のペインにあります。

メモ :
Visual Studio Team System には、コード分析のために追加のプロパティ ページがあります。

上の図では、[アプリケーション] プロパティ ページが表示されています。左のタブにあるラベル ([ビルド]、[ビルド イベント]、[デバッグ] など) をクリックすると、対応するプロパティ ページにアクセスできます。ここに入力したプロジェクト固有の情報は、.csproj ファイルに格納されます。.csproj ファイルはソリューション エクスプローラに表示されませんが、ドライブのプロジェクト フォルダにあります。Visual C# で作業している場合、ページにマウス カーソルを移動して **F1** キーを押すと、プロパティ ページのヘルプにアクセスできます。

プロジェクト デザイナの各ページの簡単な説明を次の表に示します。

プロパティ ページ	説明
アプリケーション	アセンブリの名前、プロジェクトの種類、バージョン番号などのアセンブリ情報、その他のリソース オプションを変更します。詳細については、「[アプリケーション] ページ (プロジェクト デザイナ) (C#)」を参照してください。
ビルド	コンパイルしたアセンブリを格納する位置、条件付きコンパイルのオプション、エラーと警告の処理方法などの設定を変更します。詳細については、「[ビルド] ページ (プロジェクト デザイナ) (C#)」を参照してください。
ビルド イベント	カスタム ビルド ステップの作成と変更を行います。詳細については、「[ビルド イベント] ページ (プロジェクト デザイナ) (C#、J#)」を参照してください。
デバッグ	デバッガで実行するときのコマンド ライン引数やその他の設定を指定します。詳細については、「[デバッグ] ページ (プロジェクト デザイナ)」を参照してください。
リソース	文字列、アイコン、イメージなどの種類のファイルを、リソースとしてプロジェクトに追加します。詳細については、「[リソース] ページ (プロジェクト デザイナ)」を参照してください。

設定	特定のユーザーが使用するデータベースの接続文字列や配色などの設定を格納します。この設定は、実行時に動的に取得できます。詳細については、「[設定] ページ (プロジェクト デザイナ)」を参照してください。
参照パス	プロジェクトで参照されるアセンブリの位置を示すパスを指定します。詳細については、「[参照パス] ページ (プロジェクト デザイナ) (C#、J#)」を参照してください。
署名	ClickOnce の証明書のオプションを指定し、アセンブリに厳密な名前を与えます。詳細については、「[署名] ページ (プロジェクト デザイナ)」および「ClickOnce の配置の概要」を参照してください。
セキュリティ	アプリケーションを実行するときに必要なセキュリティ設定を指定します。詳細については、「[セキュリティ] ページ (プロジェクト デザイナ)」を参照してください。
発行	アプリケーションを Web サイト、ftp サーバーまたはフォルダに配布するときのオプションを指定します。詳細については、「[発行] ページ (プロジェクト デザイナ)」を参照してください。
コード分析 (Visual Studio Team Systemのみ)	セキュリティ上の問題が発生する可能性、.NET Framework のデザイン ガイドラインの順守などについて、ソース コードを解析するツールのオプション。詳細については、「コード分析 (プロジェクト デザイナ)」を参照してください。

参照

概念

プロジェクト デザイナの概要

その他の技術情報

Visual C# IDE の使用

Visual Studio の統合開発環境

ユーザー インターフェイスのデザイン (Visual C#)

Visual C# でユーザー インターフェイス (UI) を実装するには、Windows フォーム デザイナとツールボックスで視覚的に作成するのが最も簡単で便利な方法です。どのようなユーザー インターフェイスを作成する場合でも、3 つの基本的な手順があります。

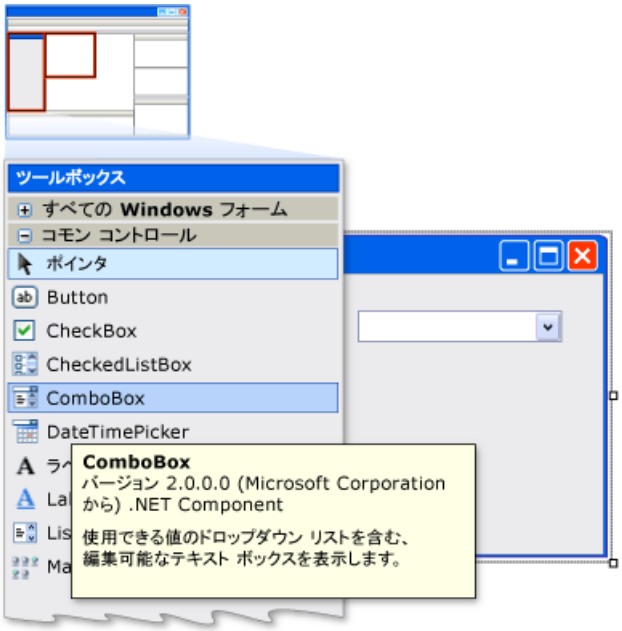
- コントロールをデザイン サーフェイスに追加します。
- コントロールの初期プロパティを設定します。
- 指定したイベントのハンドラを記述します。

自分でコードを記述して UI を作成することもできますが、デザイナを使用すると、手動でのコーディングよりもはるかに速く UI を作成できます。

メモ：
また、Visual C# を使用して、単純なテキスト ベースの UI を持つコンソール アプリケーションを作成することもできます。詳細については、「コンソール アプリケーションの作成 (Visual C#)」を参照してください。

コントロールの追加

デザイナで、フォームを表すデザイン サーフェイスに、ボタンやテキスト ボックスなどを、マウスを使用してドラッグします。次の図に、[ツールボックス] ウィンドウから Windows フォーム デザイナへドラッグしたコンボ ボックスを示します。



このアクションは、C# ソース コードに変換され、<name>.designer.cs というプロジェクト ファイルに書き込まれます (ここで、<name> はフォームに付けた名前です)。アプリケーションを実行すると、デザイン サーフェイスで配置したとおりの位置とサイズで UI 要素が表示されます。詳細については、「Windows フォーム デザイナ」を参照してください。

プロパティの設定

フォームにコントロールを追加した後は、[プロパティ] ウィンドウを使用して、背景色や既定のテキストなど、コントロールのプロパティを設定できます。[プロパティ] ウィンドウで指定した値は、実行時にコントロールが作成されるときに割り当てられるプロパティの初期値でしかありません。多くの場合、この初期値は、実行時にプログラムでアクセスまたは変更できます。このとき、アプリケーションで、コントロール クラスのインスタンスに関するプロパティを取得または設定するだけです。[プロパティ] は、デザイン時に便利なウィンドウです。コントロールでサポートされているプロパティ、イベント、およびメソッドのすべてを参照できます。詳細については、「[プロパティ] ウィンドウ」を参照してください。

イベントの処理

グラフィカル ユーザー インターフェイスのあるプログラムは、まずイベント ドリブン型です。つまり、テキスト ボックスへのテキスト入力、ボタンのクリック、リスト ボックスの選択変更など、ユーザーが何かを実行するまで待機しています。ユーザー操作が発生すると、.NET Framework クラスの単なるインスタンスであるコントロールから、アプリケーションにイベントが送信されます。この場合、イベントを受信したときにアプリケーションで呼び出す特殊なメソッドを記述して、イベントを処理する、という選択肢があります。

[プロパティ] ウィンドウを使用して、コードで処理するイベントを指定します。イベントを確認するには、デザイナでコントロールを選択し、[プロパ

ティ] ウィンドウのツール バーにある稲妻の印が付いた [イベント] ボタンをクリックします。イベントのボタンを次の図に示します。

[プロパティ] ウィンドウでイベント ハンドラを追加すると、本文が空であるメソッドが自動的に記述されます。メソッドにコードを記述して何かに利用することもできますが、そのままにしてもかまいません。コントロールでは多数のイベントが生成されることが多いのですが、ほとんどの場合、処理が必要なイベントは数個で、1 つのこともあります。たとえば、ボタンの **Click** イベントを処理する場合でも、ボタンの外観を詳細にカスタマイズしないのであれば、**Paint** イベントを処理する必要はありません。

次に行う作業

Windows フォーム ユーザー インターフェイスの詳細については、以下のトピックを参照してください。

- Windows ベースのアプリケーションの作成
- チュートリアル : 簡単な Windows フォームの作成
- Windows フォーム デザイナ ユーザー インターフェイス要素

.NET Framework クラス ライブラリの System.Windows.Forms および関連する名前空間には、Windows フォーム開発で使用されるクラスが含まれます。

参照

その他の技術情報

Visual C#
Visual C# IDE の使用

コードの編集 (Visual C#)

Visual C# コード エディタは、ソース コードを記述するためのワード プロセッサです。文章、段落、文法に対する広範囲のサポート機能が Microsoft Word にあるように、C# コード エディタにも C# 構文や .NET Framework のサポート機能があります。このサポート機能は、主に 5 つのカテゴリにグループ化できます。

- IntelliSense: エディタに入力するたびに、関連する .NET Framework のクラスとメソッドの基本ドキュメントが表示されます。また、自動的なコード生成も行われます。
- リファクタリング: 開発プロジェクトのカーソル上で呼び出して、コード ベースを簡単に再構築します。
- コード スニペット: よく繰り返されるコード パターンを含むライブラリを検索できます。
- 波線: 入力時に、スペルミスのある単語、誤った構文、および警告を表すビジュアル表示です。
- 読みやすさの補助: アウトライン機能とカラー表示です。

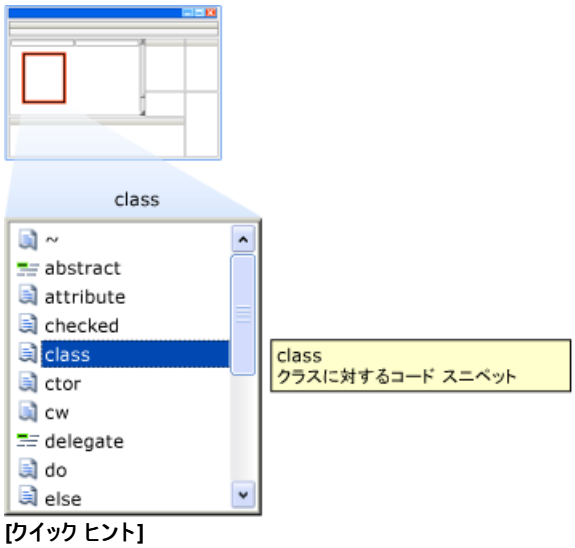
IntelliSense

IntelliSense は、ヘルプを検索する時間を最小限に抑え、コード入力を正確に効率よく行うことができるようにデザインされた、関連する機能セットに付けられた名前です。これらの機能ではいずれも、エディタに言語のキーワード、.NET Framework の型、およびメソッドのシグネチャを入力するたびに、関連する基本情報を表示します。この情報は、ツールヒント、リスト ボックス、およびスマート タグに表示されます。

メモ：
IntelliSense の機能の多くは、他の Visual Studio 言語と共有されています。また、MSDN ライブラリの「コーディング補助機能」の図で説明されています。次に、IntelliSense の概要と詳細なドキュメントへのリンクを説明します。

[入力候補一覧]

エディタでソース コードを入力するたびに、すべての C# キーワードと .NET Framework クラスを含むリスト ボックスが IntelliSense によって表示されます。入力した名前と一致する項目がリスト ボックスにあるときは、それが選択されます。選択した項目を適用する場合、Tab キーを押すだけで、名前またはキーワードの入力が完了します。詳細については、「C# でのコンプリート リスト」を参照してください。



[クイック ヒント]

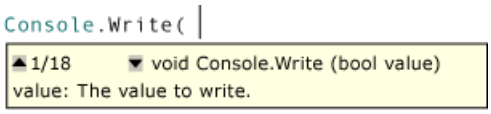
.NET Framework の型にマウス カーソルを移動すると、IntelliSense によって [クイック ヒント] のツールヒントが表示されます。このヒントにはその型に関する基本的なドキュメントが表示されます。詳細については、「クイック ヒント」を参照してください。

[メンバの一覧]

.NET Framework の型をコード エディタに入力し、ドット演算子 (.) を入力すると、その型のメンバを含むリスト ボックスが IntelliSense によって表示されます。選択して Tab キーを押すと、そのメンバ名が入力されます。詳細については、「メンバの一覧」を参照してください。

[パラメータ ヒント]

コード エディタでメソッド名を入力し、左かっこを入力すると、[パラメータ ヒント] のツールヒントが IntelliSense によって表示されます。このヒントには、そのメソッドのパラメータの順序と型が表示されます。メソッドをオーバーロードする場合、オーバーロードされるシグネチャのすべてをスクロール ダウンできます。詳細については、「[パラメータ ヒント]」を参照してください。



[Using を追加]

適切な修飾名がない場合でも、.NET Framework クラスのインスタンスを作成することがあります。このようなとき、IntelliSense によって、未解決の識別子の後にスマート タグが表示されます。スマート タグをクリックすると、IntelliSense によって、識別子を解決できる **using** ディレクティブのリストが表示されます。リストから選択すると、ソース コード ファイルの先頭にそのディレクティブが追加され、現在の位置でコーディングを続けることができます。詳細については、「using の追加」を参照してください。

リファクタリング

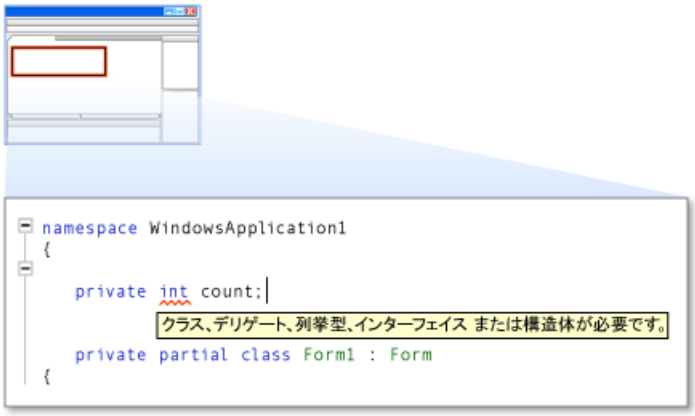
開発プロジェクトの過程でコード ベースが大きくなり発展してくると、読みやすいように、またはコンパクトになるように変更するのが望ましいこともあります。たとえば、メソッドの一部を小さなメソッドに分割したり、メソッドのパラメータを変更したり、識別子の名前を変更したりすることがあります。リファクタリング機能では、検索と置換のような従来のツールよりも便利に、知的に、また完璧に、このような操作のすべてを実行できます。リファクタリングにはコード エディタで右クリックしてアクセスします。詳細については、「リファクタリング」を参照してください。

コード スニペット

コード スニペットは、一般的に使用される、小さな単位の C# ソース コードです。数回のキーストロークで、ソース コードをすばやく正確に入力できます。コード スニペット メニューには、コード エディタで右クリックしてアクセスします。Visual C# に備えられている多数のスニペットから検索するだけでなく、自分でスニペットを作成することもできます。詳細については、「コード スニペット (C#)」を参照してください。

波線

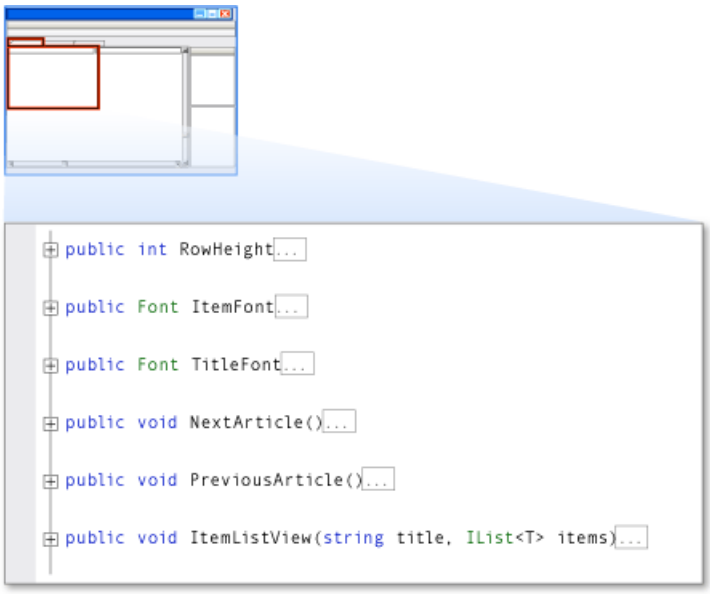
波線は、入力時にコードのエラーに関する簡単なフィードバックを示します。赤い波線は、セミコロンがない場合やかっこが合っていない場合など、構文のエラーを示します。緑の波線は、コンパイラで警告される可能性があることを示します。青い波線は「エディット コンティニュ」の問題を示します。次の図に赤い波線の例を示します。



読みやすさの補助

[アウトライン]

コード エディタでは、自動的に、名前空間、クラス、およびメソッドを折りたたむことができる領域として扱います。折りたたむと、ソース コード ファイルの他の部分が、検索しやすくなり、読みやすくなります。折りたたむことができる独自の領域を作成するには、コードを **#region** ディレクティブと **#endregion** ディレクティブで囲みます。



カラー表示

C# ソース コード ファイルの多様なカテゴリの識別子は、それぞれ異なる色で表示されます。詳細については、「コードの色づけ」を参照してください。

参照

その他の技術情報

Visual C# IDE の使用

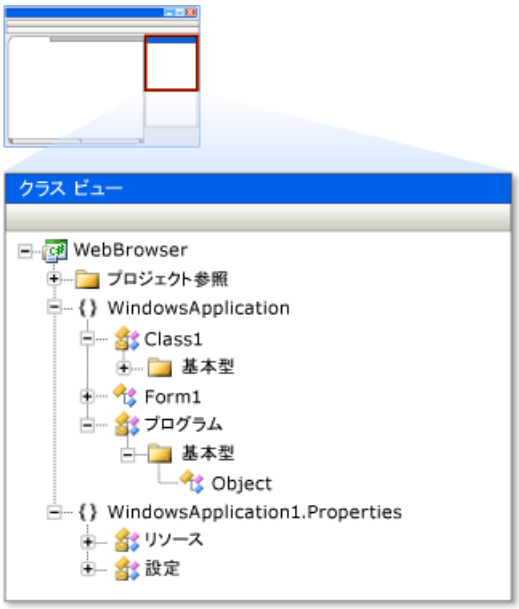
移動と検索 (Visual C#)

Visual C# には、ソース コード、プロジェクト ファイル、および開いているウィンドウで移動および検索を行うときに役立つツールがあります。

- クラス ビュー
- ナビゲーション バー
- CTRL-TAB ナビゲーション
- フォルダを指定して検索

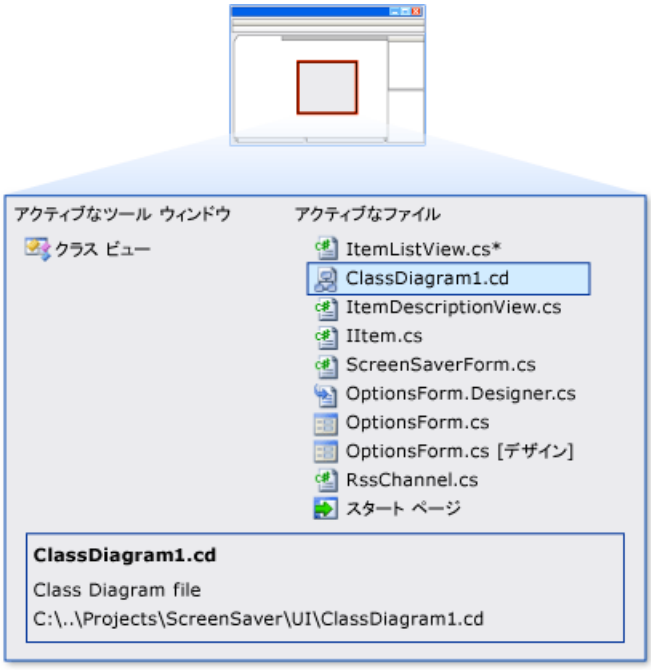
クラス ビュー

[クラス ビュー] ウィンドウには、ソリューション エクスプローラと同様に、ファイルではなくクラスに基づいてプロジェクトが表示されます。[クラス ビュー] を使用すると、プロジェクトに含まれるクラスまたはクラスのメンバにすばやく移動できます。[クラス ビュー] にアクセスするには、[表示] メニューの [クラス ビュー] をクリックします。



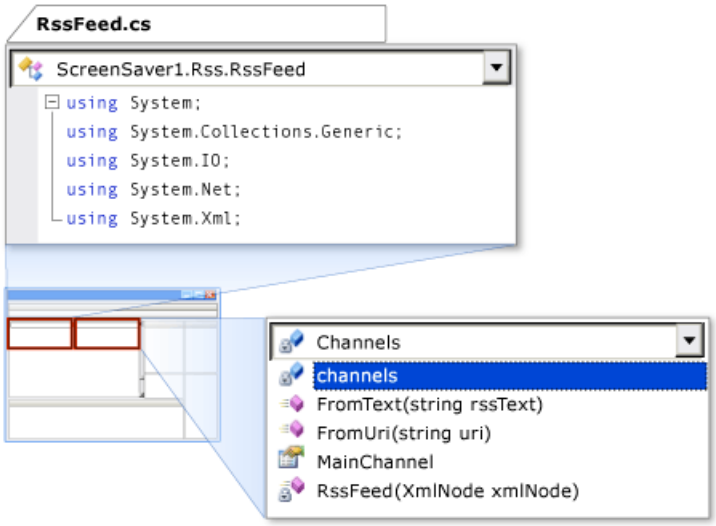
CTRL-TAB ナビゲーション

場合によっては、Visual C# プロジェクトで複数のウィンドウがアクティブになることがあります。すばやくウィンドウを切り替えるには、Ctrl キーを押しながら Tab キーを押します。アクティブなツールとソース コードのウィンドウのすべてが列挙されたウィンドウが表示されます。Ctrl キーを押しながら方向キーを押すと、表示するウィンドウを選択できます。



ナビゲーション バー

すべてのコード エディタ ウィンドウの上部に、2 つのリスト ボックスで構成されたナビゲーション バーがあります。左側のリスト ボックスには、現在のファイルで定義されているすべてのクラスが表示されます。右側のリスト ボックスには、左のリスト ボックスで選択されたクラスのすべてのメンバが表示されます。右側のリスト ボックスをクリックすると、メソッドに直接移動できます。



フォルダを指定して検索

Ctrl キーと Shift キーを押しながら F キーを押すと、[フォルダを指定して検索] ダイアログ ボックスが表示されます。このダイアログ ボックスでは、プロジェクト全体の操作を検索し、置換できます。

メモ :
メソッドや型の名前を変更したり、メソッドのパラメータを変更したりするには、リファクタリング機能を使用します。検索と置換よりも完璧に実行され、また高度な機能があります。詳細については、「リファクタリング」を参照してください。

参照項目

- 方法 : コード間またはテキスト間を移動する
- 方法 : コードをアウトライン表示する/非表示にする
- 方法 : ドキュメントのインクリメンタル検索を実行する

参照

その他の技術情報

Visual C#

Visual C# IDE の使用

ビルドとデバッグ (Visual C#)

Visual C# で実行可能アプリケーションをビルドするには、[ビルド] メニューの [ビルド] をクリックします (または Ctrl キーと Shift キーを押しながら B キーを押します)。アプリケーションのビルドと起動を 1 回の操作で行うには、F5 キーを押すか、[デバッグ] メニューの [実行] をクリックします。

ビルド処理では、プロジェクト ファイルが C# コンパイラに設定され、ソース コードが Microsoft Intermediate Language (MSIL) に変換されます。また、メタデータ、リソース、マニフェストなどのモジュールがある場合は、その MSIL と結合され、アセンブリが作成されます。アセンブリは、一般に、.exe または .dll という拡張子を持つ実行可能ファイルです。実行内容をテストして確認するには、アプリケーションを開発するときにデバッグ バージョンをビルドします。最終的に、正しく動作することが確認できれば、顧客に配布するリリース バージョンを作成します。

アセンブリの詳細については、「アセンブリの概要」を参照してください。

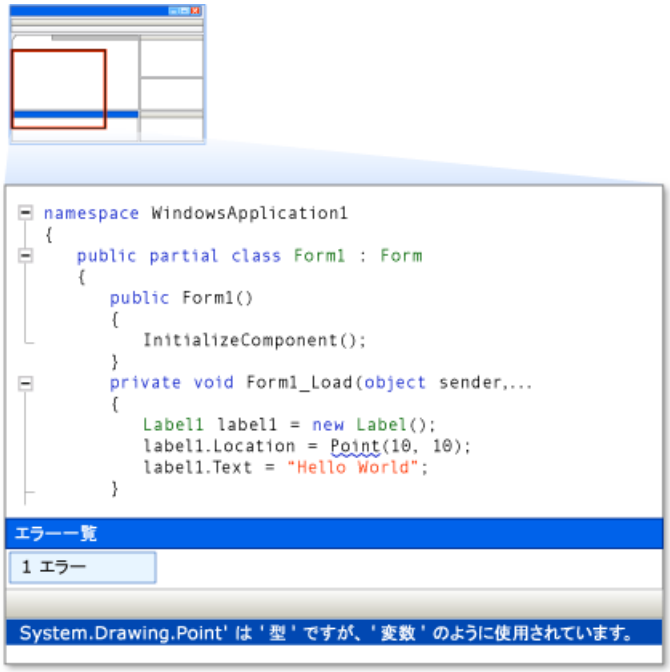
ビルド設定

さまざまなビルド設定を指定するには、ソリューション エクスプローラのプロジェクト項目を右クリックし、[プロジェクト デザイナ] の [ビルド] ペインを選択します。詳細については、「プロジェクト デザイナの概要」および「C# コンパイラ オプション」を参照してください。

Visual Studio では MSBuild ツールを使用してアセンブリを作成します。MSBuild はコマンド ラインからも実行でき、さまざまな方法でカスタマイズできます。詳細については、「MSBuild」を参照してください。

ビルド エラー

C# 構文にエラーがある場合、または既知の型やメンバへ解決できない識別子がある場合は、ビルドは成功しません。また、[エラー一覧] ウィンドウにエラー一覧が表示されます。この一覧は既定でコード エディタの真下に表示されます。エラー メッセージをダブルクリックすると、エラーが発生したコード行が表示されます。



C# コンパイラのエラー メッセージは、通常は理解しやすいのですが、問題の原因がわからない場合は、エラー リストでエラー メッセージを選択して F1 キーを押すと、そのメッセージのヘルプ ページが表示されます。ヘルプ ページには、役に立つ追加情報が説明されています。それでも問題を解決できない場合、C# のフォーラムかニュースグループに質問してみる方法があります。フォーラムにアクセスするには、[コミュニティ] メニューの [質問] をクリックします。

メモ :

コンパイラ エラーのヘルプ ページを読んでも特定のエラーに役立たなかった場合は、問題の説明を Microsoft に送信してください。ドキュメントの改善に活用させていただきます。電子メールを送信するには、エラーについて説明されたヘルプ ページの下部にあるリンクをクリックします。

リリース構成とデバッグ構成

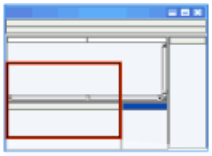
プロジェクトで作業中であるときは、アプリケーションはデバッグ構成でビルドするのが一般的です。ビルド構成では、デバッガで変数値を確認し、実行を制御できるためです。また、リリース構成でビルドを作成してテストすることで、リリース構成またはデバッグ構成のビルドでのみ出現するバグがないことも確認できます。.NET Framework プログラミングでは、このようなバグはまれですが、発生する可能性はあります。

アプリケーションをエンド ユーザーに配布する準備が整ったら、リリース ビルドを作成します。リリース ビルドは、同じソースのデバッグ ビルドよりもサ

イズがはるかに小さく、一般にパフォーマンスも格段に優れています。プロジェクト デザイナの [ビルド] ペイン、または [ビルド] ツール バーで、ビルド構成を設定できます。詳細については、「ビルド構成」を参照してください。

デバッグ

コード エディタで作業しているときは、F9 キーを押すだけでコード行にブレークポイントを設定できます。F5 キーを押して Visual Studio デバッガ内でアプリケーションを実行すると、アプリケーションはその行で停止し、その時点での変数値を確認できます。また、F10 キーを押すか追加のブレークポイントを設定すると、コードが 1 行ずつステップ実行されるため、ループを中断する方法とタイミングを確認できます。

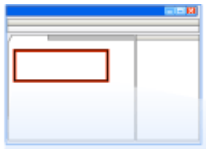


```
private void Form1_Load(object sender, EventArgs e)
        {
            Label label1 = new Label();
            Point point = new Point(10, 10);
            label1.Location = point;
            label1.Text = "Hello World";
        }
```

名前	値
this	{WindowsApplication1.Form1, Text: Form1}
sender	{WindowsApplication1.Form1, Text: Form1}
e	{System.EventArgs}
label1	{System.Windows.Forms.Label, Text: }
point	{X = 10 Y = 10}
IsEmpty	false
X	10
Y	10
Static メンバ	Static メンバ
パブリックでないメンバ	パブリックでないメンバ

また、条件付きブレークポイントを設定して、指定した条件に適合するときにのみ実行を停止することもできます。トレースポイントはブレークポイントに似た機能ですが、実行は停止されず、出力ウィンドウに指定した変数値が表示されるのみであるという点が異なります。詳細については、「ブレークポイントとトレースポイント」を参照してください。

実行がブレークポイントで停止したときに、スコープ内の変数にマウス カーソルを移動すると、その変数に関する情報が表示されます。次の図に、デバッガのデータのヒントを示します。



```
public Font TitleFont
{
    get
    {
        // Choose a font for the title text.
        // This font will be twice as big as the ItemFont
        float titleFontHeight = (float) (percentOfArti...
        if (titleFont == null || titleFont.Size !=title...
        {
            titleFont = new Font("Microsoft Sans Serif",...
                titleFont {Name = "Microsoft Sans...
        }
        return titleFont;
    }
}
```

デバッガの実行がブレークポイントで停止した後に、F10 キーを押すと、コードを 1 行ずつステップ実行できます。このとき、アプリケーションを停止して再コンパイルしなくても、コードのエラーをある程度修正してデバッグを続けることができます。

Visual Studio デバッガは強力なツールです。「エディット コンティニュ」、「デバッガでのデータ表示」、「ビジュアライザ」、および「Just-In-Time デバッグ」などのさまざまな概念を理解するために、ドキュメントを読んで活用することをお勧めします。

参照

処理手順

方法 : デバッグ構成とリリース構成を設定する
方法 : エディタでコードをデバッグする

関連項目

System.Diagnostics

その他の技術情報

Visual C#

Visual C# IDE の使用

デバッグの準備 : C#、J#、および Visual Basic のプロジェクト

デバッグの設定と準備

コードのモデリングと解析 (Visual C#)

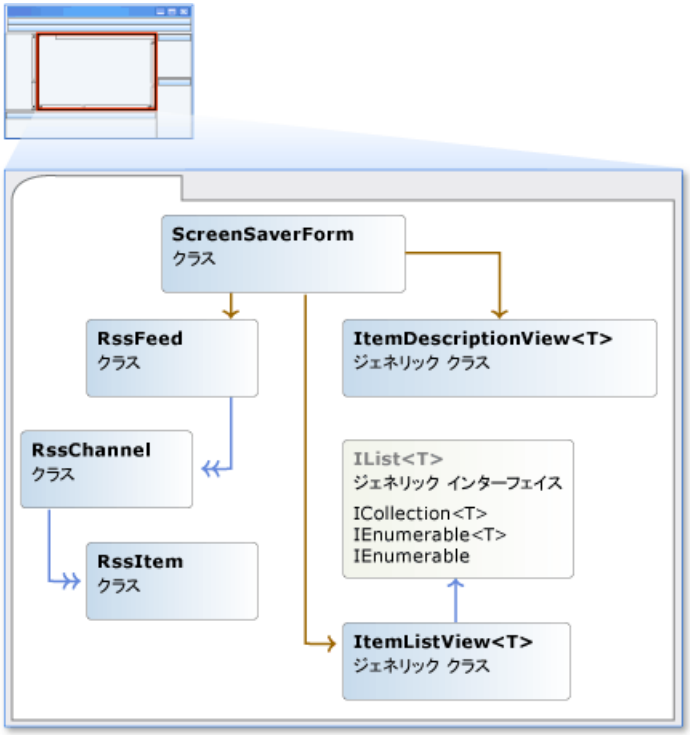
ソフトウェア開発者にとって、基本アーキテクチャがよくわからないソース コードで作業することは珍しくありません。たとえば、他人が記述したソース コードの場合や、かなり以前に記述されたために、元の作成者でも機能を完全に思い出すことができない場合などです。また、バイナリ形式でのみ使用できるライブラリの内容を理解する必要がある、という場合もよくあります。Visual C# には、ソース コードとバイナリのアセンブリに含まれる型と型の関係のモデリング、解析、および理解に役立つツールがあります。

- クラス デザイナ。型の継承と関連付けの関係を視覚的に表します。
- オブジェクト ブラウザ。.NET Framework アセンブリでエクスポートされた型、メソッド、およびイベントと、COM オブジェクトなどのネイティブ DLL を確認します。
- メタデータをソース形式で表示する。マネージ アセンブリの型情報を、自分で作成したプロジェクトのソース コードのように表示します。

上記のツール以外に、コードに含まれるさまざまな問題を検査するマネージ コードのコード分析ツールがVisual Studio Team Systemにあります。

クラス デザイナ

クラス デザイナは、ソフトウェア アプリケーションまたはコンポーネントに含まれる型どうしの関係を、視覚的にモデリングするグラフィカル ツールです。新しい型をデザインするとき、既存の型をリファクタリングしたり削除したりするときにも使用できます。単純なクラス デザインを次の図に示します。

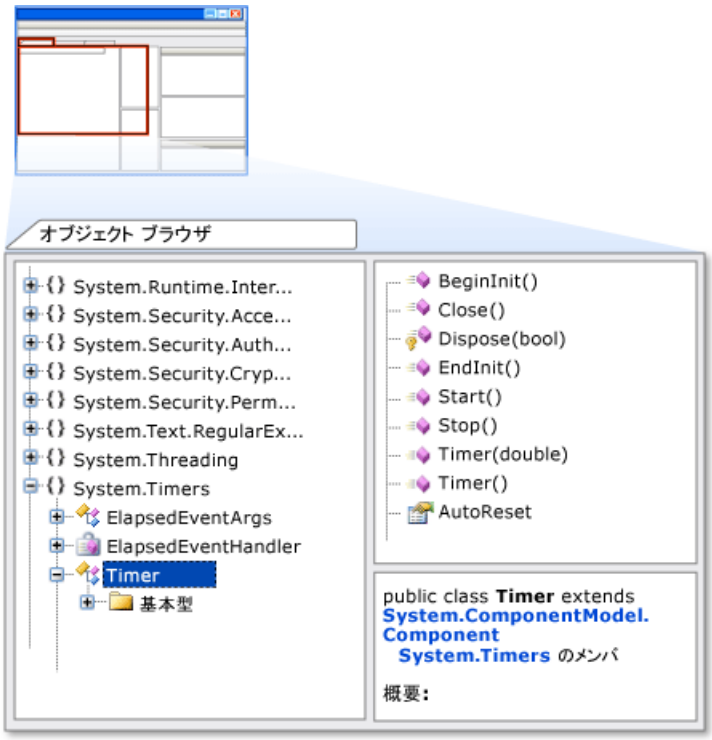


プロジェクトにクラス ダイアグラムを追加するには、[プロジェクト] メニューの [新しい項目の追加] をクリックし、[クラス ダイアグラムの追加] をクリックします。

詳細については、「クラスと型のデザインおよび表示」を参照してください。

オブジェクト ブラウザ

オブジェクト ブラウザを使用すると、ネイティブ DLL とマネージ DLL (COM オブジェクトなど) の型情報を表示できます。オブジェクト ブラウザに表示される情報は、[クラス ビュー] で表示されるものと似ていますが、オブジェクト ブラウザでは、プロジェクトで参照されている DLL だけでなく、システム上にある任意の DLL を検査できます。さらに、オブジェクト ブラウザでは、選択した型の XML ドキュメントのコメントも表示されます。バイナリ ファイルの型情報が表示されているオブジェクト ブラウザを次の図に示します。



詳細については、「オブジェクト ブラウザ」を参照してください。

メタデータをソース形式で表示する

メタデータをソース形式で表示する機能を使用すると、マネージ アセンブリのクラスでも、自分で作成したプロジェクトのソース コードであるかのように型情報を表示できます。この機能は、実際のソース コードにアクセスできないときに、クラスのすべてのパブリック メソッドについて、シグネチャをすぐに表示するときに便利です。

たとえば、コード エディタでステートメント `System.Console.WriteLine()` を入力し、`Console` にカーソルを移動し、右クリックして [定義へ移動] を選択すると、Console クラスの宣言を含むソース コード ファイルと同様の内容が表示されます。この宣言は、リフレクション を使用してアセンブリのメタデータから構築されます。また、どのメソッドの実装も公開されませんが、XML ドキュメントのコメントがあれば、表示されます。

メタデータをソース形式で表示する機能は、オブジェクト ブラウザでマネージ型を選択し、[表示] メニューの [コード定義ウィンドウ] をクリックしても使用できます。

詳細と図については、「ソースとして使用するメタデータ」を参照してください。

マネージ コードのコード分析

マネージ コードのコード分析ツールでは、マネージ アセンブリが解析され、セキュリティ問題が発生する可能性などがレポートされます。また、Microsoft .NET Framework デザイン ガイドラインのプログラミング規則とデザイン規則に違反している部分について説明されます。この情報は警告として表示されます。プロジェクト デザイナのツールにアクセスするには、ソリューション エクスプローラの [プロパティ] を右クリックし、[開く] を選択します。

詳細については、「コード分析 (プロジェクト デザイナ)」および「マネージ コードに対するコード分析の概要」を参照してください。

参照

概念

コードの編集 (Visual C#)
リフレクション (C# プログラミング ガイド)

その他の技術情報

Visual C# IDE の使用
クラス ライブラリ開発のデザイン ガイドライン
例外のデザインのガイドライン
メンバのデザインのガイドライン
型のデザインのガイドライン

リソースの追加と編集 (Visual C#)

Visual C# アプリケーションには、ソース コード以外のデータが含まれることがよくあります。このようなデータはプロジェクト リソースと呼ばれ、バイナリ データ、テキスト ファイル、オーディオ ファイル、ビデオ ファイル、文字列テーブル、アイコン、イメージ、XML ファイルなど、アプリケーションに必要な種類のデータが含まれます。プロジェクト リソース データは、.resx ファイル (既定では Resources.resx という名前) に XML 形式で格納されます。このファイルはソリューション エクスプローラで開くことができます。プロジェクト リソースの詳細については、「リソース ファイルの操作」を参照してください。

プロジェクトへのリソース追加

プロジェクトにリソースを追加するには、[ソリューション エクスプローラ] ウィンドウで、プロジェクトの [プロパティ] ノードを右クリックし、[開く] をクリックします。次に、[プロジェクト デザイナ] ウィンドウの [リソース] ページの [リソースの追加] をクリックします。

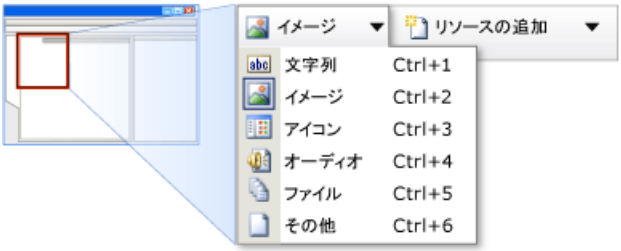
また、リソースはリンク リソース (外部ファイル) または埋め込みリソース (.resx ファイルに直接埋め込まれるリソース) としてプロジェクトに追加できます。

- リンク リソースを追加した場合、プロジェクトのリソース情報を格納する .resx ファイルには、ディスク上のリソース ファイルへの相対パスのみが含まれます。イメージ、ビデオ、またはその他の複雑なファイルをリンク リソースとして追加した場合は、リソース デザイナで該当するファイルの種類に関連付けた既定のエディタを使用して、それらのファイルを編集できます。
- 埋め込みリソースを追加した場合は、データがプロジェクトのリソース (.resx) ファイルに直接格納されます。埋め込みリソースに格納できるのは文字列のみです。

詳細については、「リンク リソースと埋め込みリソース」および「.Resx ファイル形式のリソース」を参照してください。

リソースの編集

リソース デザイナで各プロジェクト リソースを編集する既定のアプリケーションを割り当てると、開発時にプロジェクト リソースを追加し、変更できます。リソース デザイナにアクセスするには、[ソリューション エクスプローラ] ウィンドウの [プロパティ] を右クリックし、[開く] をクリックして、[プロジェクト デザイナ] ウィンドウの [リソース] タブをクリックします。詳細については、「[リソース] ページ (プロジェクト デザイナ)」を参照してください。リソース デザイナのメニュー オプションを次の図に示します。



埋め込みリソースを編集するには、.resx ファイルを使用して、個々の文字やバイトを直接操作する必要があります。そのため、複雑な種類のファイルは、開発時にリンク リソースとして格納する方が便利です。バイナリ エディタを使用すると、.resx ファイルを含むリソース ファイルを、16 進形式または ASCII 形式のバイナリ レベルで編集できます。イメージ エディタを使用すると、アイコンとカーソルの他に、リンク リソースとして格納された .jpeg ファイルや GIF ファイルも編集できます。また、これらのファイルの種類では、エディタとして他のアプリケーションを選択することもできます。詳細については、「リソース エディタでのリソースの表示と編集」を参照してください。

リソースのアセンブリへのコンパイル

アプリケーションをビルドするときに、Visual Studio では resgen.exe ツールが呼び出され、アプリケーション リソースが `Resources` という内部クラスに変換されます。このクラスは、Resources.Designer.cs ファイルに含まれます。このファイルは、ソリューション エクスプローラでは Resources.resx ファイルの下に入れ子になっています。実行時に厳密に型指定されたリソースになるように、すべてのプロジェクト リソースは、`Resources` クラスによって静的で読み取り専用の **get** プロパティにカプセル化されます。Visual C# IDE を通じてビルドする場合は、.resx ファイルに埋め込まれたリソースとリンク ファイルの両方を含む、カプセル化されたすべてのリソース データがアプリケーション アセンブリ (.exe ファイルまたは .dll ファイル) に直接コンパイルされます。つまり、Visual C# IDE は常に /resource コンパイラ オプションを使用します。コマンド ラインからビルドする場合は、/linkresource コンパイラ オプションを指定して、メイン アプリケーション アセンブリとは別個のファイルにリソースを配置できます。これは高度なシナリオであり、必要になるのはまれです。メイン アプリケーション アセンブリとは別個にリソースを配置するシナリオとしては、次に説明するサテライト アセンブリを使用する方が一般的です。

実行時のリソースへのアクセス

実行時にリソースにアクセスするには、他のクラス メンバにアクセスするときと同様に参照するだけです。Image01 という名前を付けたビットマップ リソースを取得する方法を次の例に示します。リソース クラスが名前空間 *<projectName>*`.Properties` 内にある点に注意してください。個々のリソースの完全修飾名を使用するか、リソース クラスにアクセスするソース ファイル内に適切な using ディレクティブを追加する必要がありま

す。

```
System.Drawing.Bitmap bitmap1 = myProject.Properties.Resources.Image01;
```

内部的には、get プロパティによって ResourceManager クラスが使用され、オブジェクトの新しいインスタンスが作成されます。

詳細については、「アプリケーションのリソース」および「リソース ファイル ジェネレータ (Resgen.exe)」を参照してください。

サテライト アセンブリに含まれるリソース

複数言語へのローカライズ (翻訳) が行われる予定のアプリケーションを作成する場合、カルチャ固有の各文字列セットを、固有のサテライト アセンブリのリソースとして格納できます。アプリケーションを配布するときに、メインのアプリケーション アセンブリの他に、適切なサテライト アセンブリを含めます。メインのアプリケーション アセンブリを再コンパイルしなくても、サテライト アセンブリを新たに追加したり、既存のサテライト アセンブリを変更したりできます。詳細については、「サテライト アセンブリの作成」および「固有カルチャのリソースの検索と使用」を参照してください。

参照

概念

プロジェクト デザイナの概要

その他の技術情報

Visual C#

Visual C# について

共通言語ランタイムのアセンブリ

アプリケーションのグローバライズとローカライズ

ヘルプの表示 (Visual C#)

Visual Studio のヘルプ ドキュメントは、MSDN ライブラリに含まれます。コンピュータまたはネットワークにローカルにインストールできます。また、インターネット (http://msdn.microsoft.com/library/ja) からでも使用できます。ローカル バージョンのライブラリは、.hxs 形式で圧縮された HTML ファイルのコレクションで構成されています。ライブラリのすべてまたは一部をコンピュータにインストールできます。MSDN 全体をインストールすると 2 GB のサイズになり、多数の Microsoft テクノロジに関するドキュメントが含まれています。MSDN ドキュメントは、Microsoft Document Explorer という Visual Studio ヘルプのブラウザを使用して、ローカルに、またはオンラインで、参照できます。

Visual C# で作業しているときにヘルプを表示するには、6 種類の方法があります。

- F1 キーによる検索
- 検索
- キーワード
- 目次
- カテゴリから検索
- ダイナミック ヘルプ

オンライン ヘルプとローカル ヘルプ

[オプション] メニューの [ヘルプ] オプション プロパティ ページで、次のように、F1 検索など、検索動作に関するオプションを指定できます。

- 最初にオンラインの MSDN ライブラリを検索し、一致するものがなければローカル ドキュメントを検索します。
- 最初にローカルの MSDN ライブラリを検索し、一致するものがなければオンライン ドキュメントを検索します。
- ローカルの MSDN ライブラリのみを検索します。

このオプションは、検索を初めて呼び出したときにも表示されます。オンラインの MSDN ドキュメントには、ローカル ドキュメントよりも新しい情報が掲載されていることがあります。そのため、Visual C# で作業するときにインターネット接続できる場合には、検索オプションで MSDN ライブラリを最初に検索するように設定することをお勧めします。アップデートをダウンロードしてローカル ドキュメントを更新できることもあります。ドキュメント更新の詳細については、Visual Studio Developer Centerを参照してください。

F1 キーによる検索

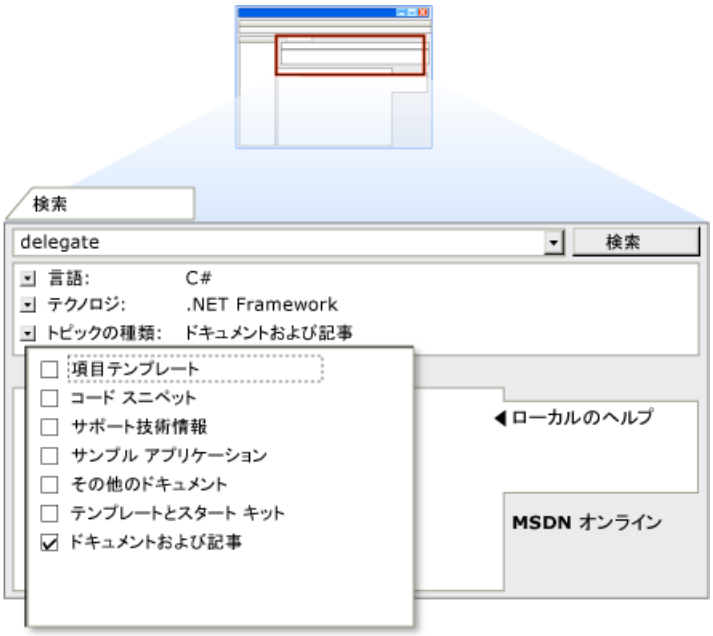
F1 キーには、状況依存の検索機能があります。コード エディタでは、C# キーワードや .NET Framework クラス メンバの途中またはその直後にカーソルを置いて F1 キーを押すと、関連するヘルプ ドキュメントにアクセスできます。ダイアログ ボックスや他のウィンドウにフォーカスがある場合、F1 キーを押すとそのウィンドウのヘルプが表示されます。

F1 検索で検索されるのは、1 ページのみです。一致するものがなければ、トラブルシューティングのヒントになる情報ページが表示されます。

検索

[検索] インターフェイスを使用すると、指定した 1 つ以上の用語に一致するすべてのドキュメントが検索されます。

[検索] のインターフェイスは次のようになります。



また、[オプション] メニューの [ヘルプ] オプション メニューを使用して、MSDN ライブラリとは別に Codezone Web サイトも検索するかどうかを指定できます。Codezone は、Microsoft のパートナーが運営しているサイトで、C# と .NET Framework に関して有効な情報が掲載されています。Codezone のコンテンツは、オンラインでのみ使用できます。

[検索] と F1 検索のどちらの場合も、オンライン検索とローカル検索に同じオプションが適用されます。

[検索] インターフェイスでは、含まれるドキュメントの種類を指定することで検索範囲を狭くしたり広くしたりできます。[言語]、[テクノロジ]、および [コンテンツの種類] という 3 つのオプションがあります。通常、現在の開発シナリオに適用するオプションのみをオンにすることで、最適の検索結果が得られます。

キーワード

キーワード検索を使用すると、ローカルの MSDN ライブラリにあるドキュメントを簡単に検索できます。フルテキスト検索ではなく、各ドキュメントに割り当てられたインデックスのキーワードのみが検索されます。キーワード検索は、フルテキスト検索よりも一般に高速であり、より適切なドキュメントが検索されます。キーワード検索ボックスで指定したキーワードを含むドキュメントが複数ある場合、絞り込まれたウィンドウが開き、複数の選択肢から選択できます。

キーワードのウィンドウは、既定で、Document Explorer の左側に表示されます。Visual C# の [ヘルプ] メニューからアクセスできます。

目次

MSDN ライブラリの目次には、ライブラリのすべてのトピックが階層構造のツリー ビューで表示されます。目次は、ドキュメント全体を参照してライブラリに含まれている概念を確認するときや、キーワードや検索で見つからなかったドキュメントを探すときに役立ちます。F1、キーワード、または検索でドキュメントを検索したときに、目次のどこに位置しているかを確認すると、そのトピックに関連する他のトピックを確認できるため便利です。Document Explorer のツール バーにある [目次と同期] ボタンをクリックすると、現在表示されているページが MSDN ライブラリのどこに位置しているかを確認できます。

カテゴリから検索

[カテゴリから検索] は、MSDN ライブラリにフィルタをかけた表示です。通常、特定のタスクを実行する方法を示す、「方法」や「チュートリアル」などのドキュメントが含まれます。[カテゴリから検索] ヘルプには、[Document Explorer] ツール バー、[ヘルプ] メニュー、または [開始] ページからアクセスします。Visual Studio の言語ごとに固有の [カテゴリから検索] ページがあります。また、表示されるページは、アクティブなプロジェクトの種類によって変わります。

ダイナミック ヘルプ

[ダイナミック ヘルプ] ウィンドウには、コード エディタのカーソル位置に基づいて、.NET Framework と C# 言語の関連ドキュメントへのリンクが表示されます。詳細については、「方法 : [ダイナミック ヘルプ] をカスタマイズする」を参照してください。

参照

その他の技術情報

Visual C# IDE の使用

C# アプリケーションの配置

配置とは、完成したアプリケーションやコンポーネントを他のコンピュータにインストールできるように配布するためのプロセスです。コンソール アプリケーション、または Windows フォームに基づいたスマート クライアント アプリケーションの場合、ClickOnce と Windows インストーラという 2 つの配置オプションがあります。

ClickOnce の配置

ClickOnce 配置では、Windows アプリケーションを Web サーバーまたはネットワーク ファイル共有に発行して、簡単にインストールできるようになります。ほとんどの場合、ClickOnce で配置することが推奨されます。最小限のユーザー操作でインストールと実行を行うことができる、Windows ベースの自動更新アプリケーションを作成できるためです。

ClickOnce 配置のプロパティを構成するには、発行ウィザード ([ビルド] メニューから使用できます) を使用するか、プロジェクト デザイナの [発行] ページを使用します。詳細については、「[発行] ページ (プロジェクト デザイナ)」を参照してください。ClickOnce の詳細については、「ClickOnce の配置」を参照してください。

Windows インストーラ

Windows インストーラ配置を使用すると、ユーザーに配布するインストーラ パッケージを作成できます。ユーザーはセットアップ ファイルを実行し、ウィザードの手順を実行することによって、アプリケーションをインストールできます。これは、ソリューションにセットアップ プロジェクトを追加することによって実行されます。ビルドすると、ユーザーに配布するセットアップ ファイルが作成されます。ユーザーはセットアップ ファイルを実行し、ウィザードの手順を実行することによって、アプリケーションをインストールできます。

Windows インストーラの詳細については、「Windows インストーラ配置」を参照してください。

参照

処理手順

方法 : ClickOnce アプリケーションを発行する

概念

配置の代替手段
ClickOnce の配置の概要
Windows Installer を使用したランタイム アプリケーションの配置
セットアップ プロジェクト

その他の技術情報

Visual C#

方法：C# プロジェクトにアプリケーション構成ファイルを追加する

C# プロジェクトにアプリケーション構成ファイル (app.config ファイル) を追加することで、共通言語ランタイムがアセンブリ ファイルを見つけて読み込む方法をカスタマイズできます。アプリケーション構成ファイルの詳細については、「ランタイムがアセンブリを検索する方法」を参照してください。

プロジェクトをビルドすると、開発環境は自動的に app.config ファイルのコピーを作成し、そのファイル名を実行可能ファイルと同じファイル名に変更し、新しい .config ファイルを bin ディレクトリに移動します。

C# プロジェクトにアプリケーション構成ファイルを追加するには

1. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

   [新しい項目の追加] ダイアログ ボックスが表示されます。

2. [アプリケーション構成ファイル] テンプレートを選択し、[追加] をクリックします。

   app.config という名前のファイルがプロジェクトに追加されます。

参照

処理手順

方法 : アプリケーション構成ファイルを使用して対象とする .NET Framework のバージョンを指定する

Visual C# のコード エディタの機能

Visual C# には、コードの編集とナビゲートを支援するツールが用意されています。

このセクションの内容

リファクタリング

アプリケーションの動作を変更することなくコードを変更できる、リファクタリング操作の一覧があります。

コード スニペット (C#)

自動的に一般的なコード コンストラクタをアプリケーションに追加する、Visual C# のコード スニペットを使用する概要を説明します。

コードの色づけ

コード エディタがコード コンストラクタを色づけする方法を説明します。

ソースとして使用するメタデータ

IDE でメタデータをソース コードとして表示する方法を説明します。

関連するセクション

テキスト、コード、およびマークアップの編集

コード エディタを使用してコードを作成および書式設定する方法を説明します。

コード内での移動

[検索と置換] ウィンドウ、ブックマーク、およびコードの行を特定するためのタスク リストとエラー リストを使用する手順へのリンクがあります。

Visual C# IDE の設定

C# 開発者向けに Visual Studio 設定の概要を説明します。

リファクタリング

リファクタリングとは、コードの外部動作は変更せずに、コードの内部構造を変更することで、コードを作成した後で改良するためのプロセスです。

Visual C# には、[リファクタ] メニューに次のリファクタリング コマンドがあります。

- メソッドの展開
- 名前の変更
- フィールドのカプセル化
- インターフェイスの展開
- ローカル変数をパラメータへ昇格
- パラメータの削除
- パラメータ順序の再変更

複数のプロジェクトのリファクタリング

Visual Studio では、複数のプロジェクトのリファクタリングをサポートしています。ファイル間の参照を修正するすべてのリファクタリング操作では、同じ言語を使用するすべてのプロジェクト間で、この種の参照が修正されます。これは、プロジェクト対プロジェクトのすべての参照に適用されます。たとえば、クラス ライブラリを参照するコンソール アプリケーションがある場合、[名前の変更] リファクタリング操作を使用してクラス ライブラリ型の名前を変更すると、コンソール アプリケーションでのこのクラス ライブラリ型への参照も更新されます。

[変更のプレビュー] ダイアログ ボックス

多くのリファクタリング操作には、リファクタリング操作によってコードで実行される参照の変更すべてを、コミットする前にレビューする機能があります。このようなリファクタリング操作では、リファクタリング ダイアログ ボックスに [参照の変更のプレビュー] オプションが表示されます。このオプションを選択し、リファクタリング操作を受け入れると、[変更のプレビュー] ダイアログ ボックスが表示されます。[変更のプレビュー] ダイアログ ボックスには、2 つのビューがあります。下部ビューには、リファクタリング操作によって参照がすべて更新された状態で、コードが表示されます。[変更のプレビュー] ダイアログ ボックスの [キャンセル] をクリックすると、リファクタリング操作は中断し、コードは変更されません。

エラーを許容するリファクタリング

リファクタリングでは、エラーが許容されます。つまり、ビルドできないプロジェクトでもリファクタリングを実行できます。ただし、このような場合、あいまいな参照がリファクタリング処理によって正しく更新されないことがあります。

参照

処理手順

方法 : C# リファクタリング スニペットを復元する

その他の技術情報

Visual C# のコード エディタの機能

メソッドの展開

[メソッドの展開] は、既存のメンバのコード片から新規メソッドを簡単に作成できるリファクタリング操作です。

[メソッドの展開] を使用すると、既存メンバのコード ブロック内からコードの選択部分を抽出することで、新規メソッドを作成できます。選択したコードを含む新規メソッドが作成され、既存メンバの選択したコード部分がこの新規メソッドへの呼び出しに置き換えられます。コード片を独立したメソッドにすることで、コードをすばやく正確に再構成でき、再利用性と読みやすさが向上します。

[メソッドの展開] には、次の利点があります。

- メソッドの独立性と再利用可能性を重視することで、コーディングの推奨手順を遵守できる。
- 適切に構成することで、コードを自己文書化できる。説明的な名前を使用すると、高水準のメソッドも一連のコメントのように読みやすくなる。
- オーバーライドを単純にするために、詳細なメソッドの作成が促進される。
- コードの重複を減らすことができる。

解説

[メソッドの展開] を使用すると、新規メソッドが同じクラスのソース メンバの後に挿入されます。

部分型

クラスが部分型である場合は、**[メソッドの展開]** によって新規メソッドがソース メンバの直後に作成されます。**[メソッドの展開]** では、新規メソッドのシグネチャを判断し、新規メソッド内のコードによってインスタンス データが参照されていないときは、静的メソッドを作成します。

ジェネリック型パラメータ

制約のないジェネリック型パラメータを持つメソッドを抽出する場合、生成されるコードでは、そのパラメータに値が割り当てられない限り、"ref" 修飾子が追加されません。抽出されたメソッドでジェネリック型引数として参照型をサポートしている場合は、メソッド シグネチャ内のパラメータに手動で "ref" を追加する必要があります。

匿名メソッド

ローカル変数への参照を含む匿名メソッドについて、そのローカル変数が匿名メソッドの外部で宣言または参照されている場合に、その匿名メソッドの一部を抽出しようとすると、セマンティクスが変更される可能性があるという警告が表示されます。具体的には、ローカル変数の値が匿名メソッドに渡されるタイミングが異なります。

匿名メソッドでローカル変数の値を使用する場合、その値は匿名メソッドの実行時に取得されます。匿名メソッドが別のメソッドに抽出される場合、ローカル変数の値は、抽出されたメソッドの呼び出し時に取得されます。

このセマンティクスの変更について、次に例を示します。このコードを実行すると、コンソールには "**11**" が出力されます。[メソッドの展開] を使用して、コード コメントでマークされているコードの一部を独自メソッドに抽出し、その後リファクタリングされたコードを実行すると、コンソールには "**10**" が出力されます。

```
class Program
{
    delegate void D();
    D d;
    static void Main(string[] args)
    {
        Program p = new Program();
        int i = 10;
        /*begin extraction*/
            p.d = delegate { Console.WriteLine(i++); };
        /*end extraction*/
        i++;
        p.d();
    }
}
```

この問題を回避するには、匿名メソッドで使用されているローカル変数をクラスのフィールドにします。

参照

処理手順

方法 : [メソッドの展開] でコードをリファクタリングする

[メソッドの展開] ダイアログ ボックス

このダイアログ ボックスは、[メソッドの展開] で生成される新規メソッドの名前を指定したり、[メソッドの展開] で作成される新規メソッド シグネチャが意図したとおりであることを確認したりするときに使用します。

このダイアログ ボックスを開くには、[リファクタ] メニューの [メソッドの展開] をクリックします。このダイアログ ボックスは、コードの選択範囲からメソッドを抽出できる場合だけ利用できます。

[新しいメソッド名]

[メソッドの展開] で生成される新規メソッドの名前を入力します。

[メソッド シグネチャのプレビュー]

新規メソッド シグネチャのプレビューを表示します。シグネチャは、選択したコードに基づくリファクタリング エンジンによって自動的に決定され、このテキスト ボックスでは変更できません。

参照

処理手順

方法 : [メソッドの展開] でコードをリファクタリングする

関連項目

メソッドの展開

方法：[メソッドの展開] でコードをリファクタリングする

次の手順では、既存のメンバのコード片から新規メソッドを作成する方法を説明します。ここで説明する手順を使用して、[メソッドの展開] リファクタリング操作を実行します。

[メソッドの展開] を使用するには

1. 後述の例で説明する方法で、コンソール アプリケーションを作成します。

   詳細については、「コンソール アプリケーション」を参照してください。

2. コード エディタで、抽出するコード片を選択します。

   ```
   double area = PI * radius * radius.
   ```

3. [リファクタ] メニューの [メソッドの展開] をクリックします。[メソッドの展開] ダイアログ ボックスが表示されます。

   キーボード ショートカットとして Ctrl キーを押しながら R キーを押し、次に Ctrl キーを押しながら M キーを押すことでも、[メソッドの展開] ダイアログ ボックスを表示できます。

   [メソッドの展開] ダイアログ ボックスを表示するには、選択したコードを右クリックし、コンテキスト メニューの [リファクタ] をポイントし、[メソッドの展開] をクリックする方法もあります。

4. [新しいメソッド名] ボックスに新規メソッドの名前 (たとえば「**CircleArea**」) を入力します。新規メソッド シグネチャのプレビューが [メソッド シグネチャのプレビュー] に表示されます。

5. [OK] をクリックします。

使用例

この例をセットアップするために、`ExtractMethod` という名前のコンソール アプリケーションを作成し、`Class1` を次のコードで置き換えます。詳細については、「コンソール アプリケーション」を参照してください。

```
class A
{
    const double PI = 3.141592;

    double CalculatePaintNeeded(double paintPerUnit, double radius)
    {
        // Select any of the following:
        // 1. The entire next line of code.
        // 2. The right-hand side of the next line of code.
        // 3. Just "PI *" of the right-hand side of the next line
        //    of code (to see the prompt for selection expansion).
        // 4.  All code within the method body.
        // ...Then invoke Extract Method.

        double area = PI * radius * radius;

        return area / paintPerUnit;
    }
}
```

参照

関連項目

メソッドの展開

概念

リファクタリング

名前の変更

[名前の変更] は、フィールド、ローカル変数、メソッド、名前空間、プロパティ、型など、コード シンボルの識別子の名前を簡単に変更できるリファクタリング操作です。[名前の変更] を使用すると、識別子の宣言と呼び出し内だけでなく、コメントや文字列内の名前も変更できます。

メモ：
Visual Studio のソース管理 を使用するときは、名前の変更リファクタリングを使用する前に最新版のソースを取得します。

[名前の変更] リファクタリングは、次の Visual Studio 機能で利用できます。

機能	**開発環境におけるリファクタリングの動作**
コード エディタ	コード エディタでは、カーソルをコード シンボル宣言上に移動すると、名前の変更リファクタリングを利用できます。カーソルがこの位置にあると、キーボード ショートカットにより、またはスマート タグ、コンテキスト メニュー、[リファクタ] メニューの [名前の変更] をクリックして、[名前の変更] を呼び出すことができます。[名前の変更] をクリックすると、[名前の変更] ダイアログ ボックスが表示されます。詳細については、「[名前の変更] ダイアログ ボックス」および「方法 : 識別子の名前を変更する」を参照してください。
クラス ビュー	クラス ビューで識別子を選択した場合、コンテキスト メニューまたは [リファクタ] メニューから名前の変更リファクタリングを利用できます。
オブジェクト ブラウザ	オブジェクト ブラウザで識別子を選択した場合、[リファクタ] メニューだけから名前の変更リファクタリングを利用できます。
Windows フォーム デザイナのプロパティ グリッド	Windows フォーム デザイナの [プロパティ グリッド] でコントロールの名前を変更すると、そのコントロールの名前の変更操作が開始されます。[名前の変更] ダイアログ ボックスは表示されません。
ソリューション エクスプローラ	ソリューション エクスプローラでは、コンテキスト メニューから [名前の変更] を利用できます。選択したソース ファイルにクラス名がそのファイル名と同じクラスがある場合は、このコマンドを使用して、ソース ファイルの名前の変更と、名前の変更リファクタリングを同時に実行できます。 たとえば、既定の Windows アプリケーションを作成してから Form1.cs を TestForm.cs に名前を変更すると、ソース ファイル名 Form1.cs は TestForm.cs に変更され、Form1 クラスとこのクラスへのすべての参照は TestForm に名前が変更されます。 **メモ：** [元に戻す] (Ctrl + Z) を実行しても、コード内の名前の変更リファクタリングだけが元に戻り、ファイル名は元の名前に戻りません。 選択したソース ファイルにクラス名がそのファイル名と同じクラスがない場合は、ソリューション エクスプローラの [名前の変更] を実行すると、ソース ファイルの名前の変更だけが実行され、名前の変更リファクタリングは実行されません。

名前の変更操作

[名前の変更] を実行すると、リファクタリング エンジンは、次の表に示すように、各コード シンボルに基づいて名前の変更操作を実行します。

コード シンボル	**名前の変更操作**
フィールド	当該フィールドの宣言と使用箇所が、新しい名前に変更されます。
ローカル変数	当該変数の宣言と使用箇所が、新しい名前に変更されます。

メソッド	当該メソッドの名前と、当該メソッドへのすべての参照が、新しい名前に変更されます。
名前空間	宣言、すべての使用ステートメント、および完全修飾名内で、名前空間の名前を新しい名前に変更します。 **メモ：** 名前空間の名前を変更すると、Visual Studio により、[プロジェクト デザイナ] ウィンドウの [アプリケーション] ページ にある既定の名前空間プロパティが更新されます。このプロパティは、[編集] メニューの [元に戻す] をクリックしてもリセットできません。既定の名前空間プロパティの値をリセットするには、[プロジェクト デザイナ] ウィンドウでプロパティを編集します。
プロパティ	当該プロパティの宣言と使用箇所が、新しい名前に変更されます。
型	型のすべての宣言とすべての使用箇所 (コンストラクタとデストラクタを含む) を新しい名前に変更します。部分型の場合、名前の変更操作はすべての部分に反映されます。

解説

他の型のメンバを実装またはオーバーライドするメンバ、あるいは他の型のメンバによって実装またはオーバーライドされるメンバの名前を変更しようとすると、Visual Studio により、カスケード更新を行うことを通知するダイアログ ボックスが表示されます。[続行] をクリックすると、リファクタリング エンジンにより、名前を変更するメンバと実装関係またはオーバーライド関係にある基本型および派生型のすべてのメンバが再帰的に検出され、その名前が変更されます。

次のコード例に、実装関係またはオーバーライド関係にあるメンバを示します。

C#

```
interface IBase
{
    void Method();
}
public class Base
{
    public void Method()
    { }
    public virtual void Method(int i)
    { }
}
public class Derived : Base, IBase
{
    public new void Method()
    { }
    public override void Method(int i)
    { }
}
public class C : IBase
{
    public void Method()
    { }
}
```

この例の `C.Method()` は `Ibase.Method()` を実装しています。このため、`C.Method()` の名前を変更すると、`Ibase.Method()` の名前も変更されます。その後、リファクタリング エンジンは、`Derived.Method()` が `Ibase.Method()` を実装していることを再帰的に確認し、`Derived.Method()` の名前を変更します。`Base.Method()` の名前は変更しません。これは、`Derived.Method()` が `Base.Method()` をオーバーライドしていないからです。ユーザーが [名前の変更] ダイアログ ボックスで [オーバーロードの名前を変更する] チェック ボックスをオンにしていない限り、リファクタリング エンジンはここで停止します。

[オーバーロードの名前を変更する] チェック ボックスがオンになっている場合、リファクタリング エンジンは、`Derived.Method()` をオーバーロードする `Derived.Method(int i)`、`Derived.Method(int i)` によってオーバーライドされる `Base.Method(int i)`、および `Base.Method(int i)` のオーバーロードである `Base.Method()` の名前を変更します。

メモ：

参照アセンブリに定義されたメンバの名前を変更しようとすると、ビルド エラーが発生することを通知するダイアログ ボックスが表示されます。

参照

処理手順

方法 : 識別子の名前を変更する

概念

リファクタリング

[名前の変更] ダイアログ ボックス

[名前の変更] を使用して、コード内でシンボル (フィールド、ローカル変数、メソッド、名前空間、プロパティ、型など) に付けられた識別子の名前を変更します。

[新しい名前]

名前を変更するコード要素である識別子に対して、新しい名前を指定します。

[場所]

名前の変更操作を実行するときに検索する名前空間を指定します。

[参照の変更のプレビュー]

コードが変更される前に、[変更のプレビュー - 名前の変更] ダイアログ ボックスで変更をプレビューすることを指定します。

[コメント内の検索]

チェック ボックスがオンのときは、コメント内を検索することを指定します。

[文字列内の検索]

チェック ボックスがオンのときは、文字列内を検索することを指定します。

[オーバーロードの名前を変更する]

リファクタリング操作に、メソッドのオーバーロードを含めることを指定します。オンにすると、リファクタリング エンジンにより、基本型や継承型ではなく、その型の同名のメソッドの名前が変更されます。

メモ :
このオプションは、メソッドに対してだけ使用できます。

解説

[名前の変更] リファクタリング操作でコメントや文字列が検索されると、テキストがグローバル検索および置換操作での単純文字列一致に基づいて変更されます。[コメント内の検索] または [文字列内の検索] を選択した場合は、[参照の変更のプレビュー] を選択してください。

名前空間の名前を変更すると、Visual Studio により、[プロジェクト デザイナ] ウィンドウの [アプリケーション] ページにある既定の名前空間プロパティが更新されます。このプロパティは、[編集] メニューの [元に戻す] をクリックしてもリセットできません。既定の名前空間プロパティの値をリセットするには、[プロジェクト デザイナ] ウィンドウでプロパティを編集します。

参照

処理手順

方法 : 識別子の名前を変更する

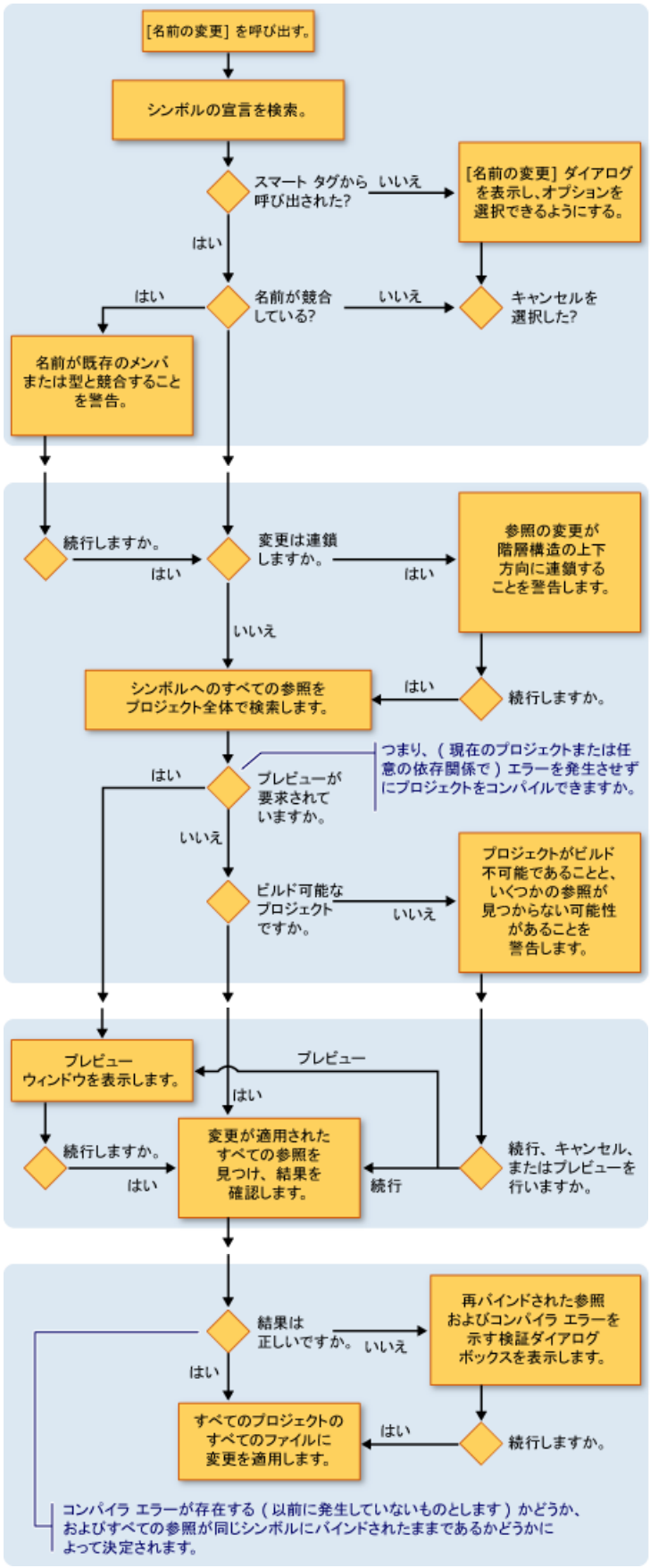
関連項目

名前の変更

[変更のプレビュー] ダイアログ ボックス

名前の変更リファクタリングのフロー図

次の図では、名前の変更リファクタリング操作を完了する手順を示します。



参照

関連項目

名前の変更

概念

リファクタリング

方法：識別子の名前を変更する

次の手順では、コードで識別子の名前を変更する方法を説明します。ここで説明する手順を使用して、名前の変更 リファクタリング操作を実行します。

識別子の名前を変更するには

1. 次の「例」セクションで説明する方法で、コンソール アプリケーションを作成します。

   詳細については、「コンソール アプリケーション」を参照してください。

2. メソッド宣言またはメソッド呼び出しで、`MethodB` にカーソルを移動します。

3. [リファクタ] メニューの [名前の変更] をクリックします。[名前の変更] ダイアログ ボックスが表示されます。

   [名前の変更] ダイアログ ボックスは、キーボード ショートカットの F2 キーを押して表示することもできます。

   [名前の変更] ダイアログ ボックスを表示するには、カーソルを右クリックし、コンテキスト メニューの [リファクタ] をポイントし、[名前の変更] をクリックする方法もあります。

4. [新しい名前] フィールドに「**MethodC**」と入力します。

5. [コメント内の検索] チェック ボックスをオンにします。

6. [OK] をクリックします。

7. [変更のプレビュー] ダイアログ ボックスの [適用] をクリックします。

スマート タグを使用して識別子の名前を変更するには

1. 次の「例」セクションで説明する方法で、コンソール アプリケーションを作成します。

   詳細については、「コンソール アプリケーション」を参照してください。

2. `MethodB` の宣言で、メソッド識別子を入力するか BackSpace キーを押します。その識別子の下に、スマート タグのプロンプトが表示されます。

メモ：
識別子の宣言でスマート タグを使用して呼び出すことができるのは、名前の変更リファクタリングだけです。

3. Shift キーと Alt キーを押しながら F10 キーを押し、次に↓キーを押して、スマート タグ メニューを表示します。

   または

   マウス ポインタをスマート タグ プロンプト上に移動して、スマート タグを表示します。マウス ポインタをスマート タグ上に移動し、↓をクリックして、スマート タグ メニューを表示します。

4. コードの変更をプレビューせずに、名前の変更リファクタリングを呼び出すには、[名前を '**<identifer1>**' から '**<identifier2>**' に変更] をクリックします。**<identifer1>** に対するすべての参照が自動的に **<identifier2>** に更新されます。

   または

   コードの変更をプレビューし、名前の変更リファクタリングを呼び出すには、[プレビューで名前の変更] をクリックします。[変更のプレビュー] ダイアログ ボックスが表示されます。

使用例

この例をセットアップするために、`RenameIdentifier` という名前のコンソール アプリケーションを作成し、`Class1` を次のコードで置き換えます。詳細については、「コンソール アプリケーション」を参照してください。

```
class ProtoClassA
    {
        // Invoke on 'MethodB'.
        public void MethodB(int i, bool b) { }
    }
```

```
class ProtoClassC
{
    void D()
    {
        ProtoClassA MyClassA = new ProtoClassA();

        // Invoke on 'MethodB'.
        MyClassA.MethodB(0, false);
    }
}
```

参照

関連項目

名前の変更

概念

リファクタリング

フィールドのカプセル化

[フィールドのカプセル化] リファクタリング操作により、既存のフィールドからプロパティをすばやく作成し、コードを新規プロパティへの参照に合わせて更新できます。

フィールドが public (C# リファレンス) である場合、他のオブジェクトは、そのフィールドを所有するオブジェクトに検知されずに、そのフィールドに直接アクセスして変更できます。プロパティ (C# プログラミング ガイド) を使用してこのようなフィールドをカプセル化することにより、フィールドへの直接アクセスを禁止できます。

新規プロパティを作成するために、[フィールドのカプセル化] によって、カプセル化するフィールドのアクセス修飾子が private (C# リファレンス) に変更され、そのフィールドの **get** アクセサおよび **set** アクセサが生成されます。フィールドが読み取り専用で宣言されている場合など、**get** アクセサだけが生成されることもあります。

リファクタリング エンジンでは、[フィールドのカプセル化] ダイアログ ボックスの [参照の更新] セクションで指定した領域について、新規プロパティへの参照に基づいてコードが更新されます。

解説

[フィールドのカプセル化] は、カーソルがフィールド宣言と同じ行にある場合だけ利用できます。

複数のフィールドが宣言されている宣言では、[フィールドのカプセル化] によってコンマがフィールド間の区切り文字として使用され、カーソルと同じ行のカーソルに最も近いフィールドでリファクタリングが開始されます。宣言でフィールドの名前を選択することで、カプセル化するフィールドを指定することもできます。

このリファクタリング操作によって生成されるコードは、[フィールドのカプセル化] コード スニペットでモデル化されています。コード スニペットは変更できます。詳細については、「方法 : コード スニペットを管理する」を参照してください。

フィールドとプロパティが使用されるタイミングの詳細については、「プロパティ プロシージャとフィールド」を参照してください。

参照

処理手順

方法 : [フィールドのカプセル化] でコードをリファクタリングする

概念

リファクタリング

コード スニペット (C#)

[フィールドのカプセル化] ダイアログ ボックス

このダイアログ ボックスを使用して、フィールドのカプセル化 リファクタリング操作の設定を指定します。

[フィールド名]

新規プロパティが生成されるフィールドの現在の名前を指定します。

[プロパティ名]

[フィールドのカプセル化] で生成される新規プロパティの名前を指定します。リファクタリング操作によって、一意のプロパティ名が自動的に生成されます。ただし、この名前は任意の有効な識別子に変更できます。

メモ :
入力した名前が無効な識別子または既存の名前と競合する名前の場合、エラーが表示され、リファクタリングは実行されません。

[参照の更新:]

リファクタリング エンジンによって、コードが新規プロパティへの参照に基づいて自動的に更新される箇所を指定します。

オプション	**説明**
External	外側の型の外部にあるフィールドへの各参照が、新規プロパティへの参照に置き換えられることを指定します。外側の型の内部でフィールドを使用している箇所は、変更されません。
All	フィールドへのすべての参照が、新規プロパティへの参照に置き換えられることを指定します。 **メモ :** [フィールドのカプセル化] では、コンストラクタ内部のフィールド参照は更新されません。

[参照の変更のプレビュー]

実際の変更前に、コードに対する変更が [参照の変更のプレビュー - フィールドのカプセル化] ダイアログ ボックスに表示されることを指定します。

[コメント内の検索]

リファクタリング エンジンによって、更新する既存のフィールドへの参照がコードのコメント内で検索されることを指定します。

[文字列内の検索]

リファクタリング エンジンによって、更新する既存のフィールドへの参照が文字列値内で検索されることを指定します。

解説

[フィールドのカプセル化] リファクタリング操作でコメントや文字列が検索されると、テキストがグローバル検索および置換操作での単純文字列一致に基づいて変更されます。[コメント内の検索] または [文字列内の検索] を選択した場合は、エラーを避けるために [参照の変更のプレビュー] を選択してください。

参照

処理手順

方法 : [フィールドのカプセル化] でコードをリファクタリングする

関連項目

[変更のプレビュー] ダイアログ ボックス

方法：[フィールドのカプセル化] でコードをリファクタリングする

次の手順では、既存のフィールドからプロパティを作成し、新規プロパティへの参照でコードを更新する方法を説明します。ここで説明する手順を使用して、フィールドのカプセル化 リファクタリング操作を実行します。

フィールドからプロパティを作成するには

1. 後述の例で説明する方法で、コンソール アプリケーションを作成します。

   詳細については、「コンソール アプリケーション テンプレート」を参照してください。

2. コード エディタとテキスト エディタで、宣言内のカプセル化するフィールドの名前にカーソルを移動します。次の例では、カーソルを `width` という語に移動します。

   ```
       public int width, height;
   ```

3. [リファクタ] メニューの [フィールドのカプセル化] をクリックします。[フィールドのカプセル化] ダイアログ ボックスが表示されます。

   キーボード ショートカットとして Ctrl キーを押しながら R キーを押し、次に Ctrl キーを押しながら F キーを押すことでも、[フィールドのカプセル化] ダイアログ ボックスを表示できます。

   [フィールドのカプセル化] ダイアログ ボックスを表示するには、カーソルを右クリックし、コンテキスト メニューの [リファクタ] をポイントし、[フィールドのカプセル化] をクリックする方法もあります。

4. 設定を指定します。

5. Enter キーを押すか、[OK] をクリックします。

6. [参照の変更のプレビュー] を選択した場合は、[参照の変更のプレビュー] ウィンドウが表示されます。[適用] をクリックします。

   次の **get** アクセサ コードおよび **set** アクセサ コードがソース ファイルに表示されます。

   ```
           public int Width
           {
               get
               {
                   return width;
               }

               set
               {
                   width = value;
               }
           }
   ```

   `Main` メソッドのコードも、新しい `Width` プロパティ名で更新されます。

   ```
   Square mySquare = new Square();
   mySquare.Width = 110;
   mySquare.height = 150;
   // Output values for width and height.
   Console.WriteLine("width = {0}", mySquare.Width);
   ```

使用例

この例をセットアップするために、`EncapsulateFieldExample` という名前のコンソール アプリケーションを作成し、`Class1` を次のコードで置き換えます。詳細については、「コンソール アプリケーション」を参照してください。

```
class Square
```

```
{
    // Select the word 'width' then use Encapsulate Field.
    public int width, height;
}
class MainClass
{
    public static void Main()
    {
        Square mySquare = new Square();
        mySquare.width = 110;
        mySquare.height = 150;
        // Output values for width and height.
        Console.WriteLine("width = {0}", mySquare.width);
        Console.WriteLine("height = {0}", mySquare.height);
    }
}
```

参照

関連項目

フィールドのカプセル化

概念

リファクタリング

インターフェイスの展開

[インターフェイスの展開] は、既存のクラス、構造体、またはインターフェイスに基づくメンバを使用して新規インターフェイスを簡単に作成できるリファクタリング操作です。

複数のクライアントで使用するクラス、構造体、またはインターフェイスのメンバのサブセットが同じ場合、または複数のクラス、構造体、またはインターフェイスがメンバのサブセットを共有する場合は、インターフェイスにメンバのサブセットを組み入れることが便利なことがあります。インターフェイスの使用の詳細については、「インターフェイス (C# プログラミング ガイド)」を参照してください。

[インターフェイスの展開] では、新規ファイル内にインターフェイスが生成され、その新規ファイルの先頭にカーソルが移動します。新規インターフェイスに対して抽出するメンバ、新規インターフェイスの名前、および生成されるファイルの名前は、[インターフェイスの展開] ダイアログ ボックスを使用して指定できます。

解説

この機能は、抽出するメンバが含まれるクラス、構造体、またはインターフェイスにカーソルが位置しているときだけ利用できます。カーソルがこの位置に置かれているときに、[インターフェイスの展開] リファクタリング操作を呼び出してください。

クラスまたは構造体でインターフェイスの抽出を呼び出す場合、新規のインターフェイス名が含まれるように、ベースおよびインターフェイスの一覧が変更されます。インターフェイスでインターフェイスの抽出を呼び出す場合、ベースおよびインターフェイスの一覧は変更されません。

参照

処理手順

方法 : [インターフェイスの展開] でコードをリファクタリングする

概念

リファクタリング

[インターフェイスの展開] ダイアログ ボックス

このダイアログ ボックスを使用して、[インターフェイスの展開] リファクタリング操作の設定を指定します。このダイアログ ボックスは [リファクタ] メニューで開くことができます。

[新しいインターフェイス名]

生成されるインターフェイスの名前を指定します。

[生成された名前]

インターフェイスの抽出元になる型 (クラスまたは構造体) のベースおよびインターフェイスの一覧に追加される名前を表示します。

[新しいファイル名]

新規インターフェイスが含まれる新規ファイルの名前を指定します。

[インターフェイスからパブリック メンバを選択してください]

新規インターフェイスを設定するために使用できるすべての有効なメンバを一覧表示します。メソッド、インデクサ、プロパティ、イベントなどがあります。

新規インターフェイスに対して抽出する各メンバを選択します。右側にある [すべて選択] または [すべて解除] を使用して、フォームを形成するメンバをすべて選択または選択解除できます。

参照

処理手順

方法 : [インターフェイスの展開] でコードをリファクタリングする

概念

リファクタリング

方法：[インターフェイスの展開] でコードをリファクタリングする

ここで説明する手順を使用して、[インターフェイスの展開] リファクタリング操作を実行します。

[インターフェイスの展開] を使用するには

1. 後述の例で説明する方法で、コンソール アプリケーションを作成します。

   詳細については、「コンソール アプリケーション」を参照してください。

2. カーソルを MethodB に移動し、[リファクタ] メニューの [インターフェイスの展開] をクリックします。[インターフェイスの展開] ダイアログ ボックスが表示されます。

   キーボード ショートカットとして Ctrl キーを押しながら R キーを押し、次に Ctrl キーを押しながら I キーを押すことでも、[インターフェイスの展開] ダイアログ ボックスを表示できます。

   [インターフェイスの展開] ダイアログ ボックスを表示するには、カーソルを右クリックし、コンテキスト メニューの [リファクタ] をポイントし、[インターフェイスの展開] をクリックする方法もあります。

3. [すべて選択] をクリックします。

4. [OK] をクリックします。

   新規ファイル IProtoA.cs と次のコードが表示されます。

```
using System;
namespace TopThreeRefactorings
{
    interface IProtoA
    {
        void MethodB(string s);
    }
}
```

使用例

この例をセットアップするために、`ExtractInterface` という名前のコンソール アプリケーションを作成し、`Class1` を次のコードで置き換えます。詳細については、「コンソール アプリケーション」を参照してください。

```
// Invoke Extract Interface on ProtoA.
// Note:  the extracted interface will be created in a new file.
class ProtoA
{
    public void MethodB(string s) { }
}
```

参照

関連項目

インターフェイスの展開

概念

リファクタリング

ローカル変数をパラメータへ昇格

[ローカル変数をパラメータへ昇格] は、変数をローカルでの使用からメソッド、インデクサ、またはコンストラクタのパラメータに簡単に移し、同時に呼び出し側を正しく更新できる Visual C# リファクタリング操作です。

[ローカル変数をパラメータへ昇格] を実行する前に、上位変換する変数にカーソルを移動します。変数を宣言するステートメントでは、変数への値や式の割り当ても必要です。カーソルを移動したら、キーボード ショートカットを使用するか、またはショートカット メニューのコマンドをクリックして、[ローカル変数をパラメータへ昇格] を呼び出します。

[ローカル変数をパラメータへ昇格] を呼び出すと、変数がメンバのパラメータ リストの末尾に追加されます。変更されたメンバに対する呼び出しは直ちに更新され、変数に割り当てられていた式として新しいパラメータが使用されます。コードは変更されないため、変数の上位変換前と同様に機能します。詳細については、「方法 : ローカル変数をパラメータへ上位変換する」を参照してください。

解説

このリファクタリングは、上位変換される変数が定数値に割り当てられている場合に最適です。変数は宣言だけ、または割り当てだけではなく、宣言して初期化する必要があります。

参照

処理手順

方法 : ローカル変数をパラメータへ上位変換する

概念

リファクタリング

方法：ローカル変数をパラメータへ上位変換する

ここで説明する手順を使用して、[ローカル変数をパラメータへ昇格] リファクタリング操作を実行します。詳細については、「ローカル変数をパラメータへ昇格」を参照してください。

ローカル変数をパラメータへ上位変換するには

1. コンソール アプリケーションを作成し、以下の例に従ってセットアップします。詳細については、「コンソール アプリケーション テンプレート」を参照してください。
2. `MethodB` の `i` にカーソルを置きます。
3. [リファクタ] メニューの [ローカル変数をパラメータへ昇格] をクリックします。

   キーボード ショートカットとして Ctrl キーを押しながら R キーを押し、次に Ctrl キーを押しながら P キーを押すことでも、このリファクタリング操作を実行できます。

   このリファクタリング操作を実行するには、カーソルを右クリックし、コンテキスト メニューの [リファクタ] をポイントし、[ローカル変数をパラメータへ昇格] をクリックする方法もあります。

これにより、`MethodB` にパラメータ `int i` が用意され、 `ProtoA.MethodB` 呼び出しによって `0` が値として渡されます。

使用例

この例をセットアップするために、`PromoteLocal` という名前のコンソール アプリケーションを作成し、`Class1` を次のコードで置き換えます。詳細については、「コンソール アプリケーション テンプレート」を参照してください。

```
class ProtoA
{
    public static void MethodB()
    {
        // Invoke on 'i'
        int i = 0;
    }
}

class ProtoC
{
    void MethodD()
    {
        ProtoA.MethodB();
    }
}
```

参照

概念

リファクタリング

パラメータの削除

[パラメータの削除] は、メソッド、インデクサ、またはデリゲートのパラメータを簡単に削除できるリファクタリング操作です。**[パラメータの削除]** を実行すると、宣言が削除され、メンバが呼び出されるすべての場所で、新規の宣言を反映するようにパラメータが削除されます。

[パラメータの削除] は、メソッド、インデクサ、またはデリゲートにカーソルを移動してから実行します。カーソルを移動したら、[リファクタ] メニュー、キーボード ショートカット、またはコンテキスト メニューのコマンドを使用して、**[パラメータの削除]** を呼び出します。

[パラメータの削除] を呼び出すと、[パラメータの削除] ダイアログ ボックスが表示されます。詳細については、「[パラメータの削除] ダイアログ ボックス」または「方法 : パラメータを削除する」を参照してください。

解説

メソッド宣言またはメソッド呼び出しからパラメータを削除できます。カーソルをメソッド宣言またはデリゲート名に移動し、パラメータの削除を呼び出します。

注意
パラメータの削除を使用すると、メンバの本体内で参照されているパラメータを削除できますが、メソッド本体内にあるそのパラメータへの参照は削除されません。このため、コードでビルド エラーが発生することがあります。ただし、リファクタリング操作を実行する前に、[変更のプレビュー] ダイアログ ボックスを使用してコードをレビューできます。

削除されるパラメータがメソッドの呼び出し中に変更される場合、パラメータを削除すると、その変更も削除されます。たとえば、次のメソッド呼び出しを考えます。

```
MyMethod(param1++, param2);
```

これは、リファクタリング操作によって、次のように変更されます。

```
MyMethod(param2);
```

この場合、**param1** の値は増加しません。

参照

処理手順

方法 : パラメータを削除する

概念

リファクタリング

[パラメータの削除] ダイアログ ボックス

このダイアログ ボックスを使用して、パラメータの削除 リファクタリング操作で削除するパラメータを指定します。

[パラメータ]

一覧からパラメータを削除できます。

[メソッド シグネチャのプレビュー]

パラメータが削除された状態の新規メソッド シグネチャが表示されます。

[参照の変更のプレビュー]

このチェック ボックスがオンの場合、[変更のプレビュー - パラメータの削除] ダイアログ ボックスが表示されます。

解説

削除されるパラメータがメソッドの呼び出し中に変更される場合、パラメータを削除すると、その変更も削除されます。たとえば、次のメソッド呼び出しを考えます。

```
MyMethod(param1++, param2);
```

変更後は次のようになります。

```
MyMethod(param2);
```

この場合、**param1** の値は増加しません。参照の変更をプレビューすることをお勧めします。

参照

処理手順

方法 : パラメータを削除する

関連項目

パラメータの削除

[変更のプレビュー] ダイアログ ボックス

方法：パラメータを削除する

ここで説明する手順を使用して、[パラメータの削除] リファクタリング操作を実行します。詳細については、「パラメータの削除」を参照してください。

パラメータを削除するには

1. コンソール アプリケーションを作成し、次の例をセット アップします。

   詳細については、「コンソール アプリケーション テンプレート」を参照してください。

2. メソッド宣言またはメソッド呼び出しで、`A` メソッドにカーソルを移動します。

3. [パラメータの削除] ダイアログ ボックスを表示するには、[リファクタ] メニューの [パラメータの削除] を選択します。

   キーボード ショートカットとして Ctrl キーを押しながら R キーを押し、次に Ctrl キーを押しながら V キーを押すことでも、[パラメータの削除] ダイアログ ボックスを表示できます。

   [パラメータの削除] ダイアログ ボックスを表示するには、カーソルを右クリックし、コンテキスト メニューの [リファクタ] をポイントし、[パラメータの削除] をクリックする方法もあります。

4. [パラメータ] フィールドを使用して、カーソルを `int i` に移動し、[削除] をクリックします。

5. [OK] をクリックします。

6. [変更のプレビュー - パラメータの削除] ダイアログ ボックスの [適用] をクリックします。

使用例

この例をセットアップするために、`RemoveParameters` という名前のコンソール アプリケーションを作成し、`Class1` を次のコードで置き換えます。詳細については、「コンソール アプリケーション テンプレート」を参照してください。

```
class A
{
    // Invoke on 'A'.
    public A(string s, int i) { }
}

class B
{
    void C()
    {
        // Invoke on 'A'.
        A a = new A("a", 2);
    }
}
```

参照

関連項目

パラメータの削除

概念

リファクタリング

パラメータ順序の再変更

[パラメータ順序の再変更] は、メソッド、インデクサ、およびデリゲートのパラメータの順序を簡単に変更できる Visual C# リファクタリング操作です。[パラメータ順序の再変更] によって宣言が変更され、メンバが呼び出されるすべての場所で、パラメータが新しい順序を反映するように再配置されます。

[パラメータ順序の再変更] は、メソッド、インデクサ、またはデリゲートにカーソルを移動してから実行します。カーソルを移動したら、キーボード ショートカットを使用するか、またはコンテキスト メニューのコマンドをクリックして、[パラメータ順序の再変更] を呼び出します。

[パラメータ順序の再変更] を呼び出すと、[パラメータ順序の再変更] ダイアログ ボックスが表示されます。詳細については、「[パラメータ順序の再変更] ダイアログ ボックス」および「方法 : パラメータを並べ替える」を参照してください。

解説

メソッド宣言またはメソッド呼び出しからパラメータの順序を変更できます。カーソルはメソッドまたはデリゲートの宣言に移動し、コードの本体には移動しないでください。

参照

処理手順

方法 : パラメータを並べ替える

概念

リファクタリング

[パラメータ順序の再変更] ダイアログ ボックス

このダイアログ ボックスを使用して、[パラメータ順序の再変更] リファクタリング操作でのパラメータの順序を指定します。

[パラメータ]

矢印ボタンを使用して、一覧の中でパラメータを上下に移動できます。

[メソッド シグネチャのプレビュー]

パラメータを並べ替えた状態のメソッド シグネチャの複製が表示されます。

[参照の変更のプレビュー]

このチェック ボックスがオンの場合、[変更のプレビュー - パラメータの順番の再変更] ダイアログ ボックスが表示されます。

参照

処理手順

方法 : パラメータを並べ替える

関連項目

パラメータ順序の再変更

[変更のプレビュー] ダイアログ ボックス

方法：パラメータを並べ替える

[パラメータ順序の再変更] リファクタリング操作を使用して、メソッド、インデクサ、コンストラクタ、およびデリゲートのパラメータの順序を変更し、自動的にそれらの呼び出しサイトを更新できます。

パラメータを並べ替えるには

1. クラス ライブラリを作成し、下の「例」セクションで説明するようにセット アップします。
2. メソッド宣言またはメソッド呼び出しで、`MethodB` にカーソルを移動します。
3. [リファクタ] メニューの [パラメータ順序の再変更] をクリックします。

   または

   キーボード ショートカットで Ctrl キーを押しながら R キーを押し、次に Ctrl キーを押しながら O キーを押して、[パラメータ順序の再変更] ダイアログ ボックスを表示します。

   または

   カーソルを右クリックし、コンテキスト メニューの [リファクタ] をポイントし、[パラメータの削除] をクリックして、[パラメータ順序の再変更] ダイアログ ボックスを表示します。

   [パラメータ順序の再変更] ダイアログ ボックスが表示されます。
4. [パラメータ順序の再変更] ダイアログ ボックスで、[パラメータ] リストの [`int i`] を選択します。次に、下矢印ボタンをクリックします。

   または

   [`int i`] を [パラメータ] リストの [`bool b`] の下にドラッグします。
5. [パラメータ順序の再変更] ダイアログ ボックスで、[OK] をクリックします。

   [パラメータ順序の再変更] ダイアログ ボックスの [参照の変更のプレビュー] チェック ボックスがオンの場合は、[参照の変更のプレビュー] ダイアログ ボックスが表示されます。メソッド シグネチャとメソッド呼び出しの両方における `MethodB` のパラメータ リストの変更がプレビューされます。

   a. [変更のプレビュー - パラメータの順番の再変更] ダイアログ ボックスが表示された場合は、[適用] をクリックします。

この例では、`MethodB` のメソッド宣言とすべてのメソッド呼び出し側が更新されます。

使用例

この例をセットアップするために、`ReorderParameters` という名前のクラス ライブラリを作成し、`Class1` を次のコードで置き換えます。

```
class ProtoClassA
{
    // Invoke on 'MethodB'.
    public void MethodB(int i, bool b) { }
}

class ProtoClassC
{
    void D()
    {
        ProtoClassA MyClassA = new ProtoClassA();

        // Invoke on 'MethodB'.
        MyClassA.MethodB(0, false);
    }
}
```

参照

関連項目

パラメータ順序の再変更

概念

リファクタリング

[変更のプレビュー] ダイアログ ボックス

[変更のプレビュー] ダイアログ ボックスを使用すると、リファクタリング操作を実際に実行する前に、そのリファクタリング操作によってコードに加えられる参照の変更をすべて確認できます。

参照の変更をプレビューすることは、省略可能なリファクタリング プロセスです。参照の変更をプレビューするには、[参照の変更のプレビュー] チェック ボックスをオンにします。このチェック ボックスは、次のダイアログ ボックスにあります。

- [名前の変更] ダイアログ ボックス
- [フィールドのカプセル化] ダイアログ ボックス
- [パラメータの削除] ダイアログ ボックス
- [パラメータ順序の再変更] ダイアログ ボックス

メモ：
Windows フォーム デザイナのプロパティ グリッドからの名前の変更リファクタリングでは、コードが直接変更されます。[名前の変更] ダイアログ ボックスと [変更のプレビュー] ダイアログ ボックスは、どちらも表示されません。詳細については、名前の変更 リファクタリングを参照してください。

[<Refactor> : <Code> を <RefactoredCode> にしました]

プロジェクトのコード内で、変更が加えられる位置を示します。コード ステートメントは、ソース ファイル ノードの下にノードとして表示されます。コード ステートメント ノードを選択すると、[コード変更のプレビュー] ボックスでは、対応するリファクタリングされた参照が強調表示された状態で表示されます。

[コード変更のプレビュー]

リファクタリング操作による参照の変更がすべてプログラムに取り込まれた状態で、プログラムを表示します。

参照

処理手順

方法 : 識別子の名前を変更する
方法 : [フィールドのカプセル化] でコードをリファクタリングする
方法 : パラメータを削除する
方法 : パラメータを並べ替える

[リファクタリングの警告] ダイアログ ボックス

この警告ダイアログ ボックスは、コンパイラがプログラムを完全に理解できなかったこと、およびリファクタリング エンジンがすべての当該参照を更新できなかった可能性があることを示します。また、この警告ダイアログ ボックスには、変更をコミットする前に、[変更のプレビュー] ダイアログ ボックスでコードをプレビューするための機能も用意されています。

メモ：
メソッドに構文エラー (IDE で赤い波線で示されている部分) がある場合、リファクタリング エンジンでは、そのメソッドの要素への参照は更新されません。この動作を次の例に示します。

既定では、参照の変更をプレビューせずにリファクタリング操作を実行し、プログラムでコンパイル エラーが検出されると、開発環境によってこの警告ダイアログ ボックスが表示されます。

[参照の変更のプレビュー] が有効であるリファクタリング操作を実行し、プログラムでコンパイル エラーが検出されると、[リファクタリングの警告] ダイアログ ボックスが表示される代わりに、[変更のプレビュー] ダイアログ ボックスの下部に次の警告メッセージが表示されます。

プロジェクトまたはその依存関係の 1 つが現在ビルドしません。参照が更新されない場合があります。

このリファクタリング警告は、[参照の変更のプレビュー] が用意されているリファクタリング操作だけで表示されます。具体的には、次のリファクタリング ダイアログ ボックスのリファクタリング操作です。

- [名前の変更] ダイアログ ボックス
- [フィールドのカプセル化] ダイアログ ボックス
- [パラメータの削除] ダイアログ ボックス
- [パラメータ順序の再変更] ダイアログ ボックス

[毎回このダイアログを表示する]

このオプションの既定値はオンです。オンにした場合、リファクタリング操作中にコンパイル エラーが検出されると、[リファクタリングの警告] ダイアログ ボックスが表示され続けます。

このチェック ボックスをオフにした場合、その後のリファクタリング操作でこの警告ダイアログ ボックスは無効になります。このチェック ボックスをオフにしてから、その後のリファクタリング操作でこの警告ダイアログ ボックスをもう一度有効にするには、[詳細] ([オプション] ダイアログ ボックス - [テキスト エディタ] - [C#]/[J#])の [リファクタリングのときにビルド エラーが存在する場合は警告する] チェック ボックスをオンにします。

[続行]

参照の変更をプレビューせずに、現在のリファクタリング操作を継続します。

[プレビュー]

コードをプレビューするために、[変更のプレビュー] ダイアログ ボックスを開きます。

[キャンセル]

現在のリファクタリング操作をキャンセルします。コードは変更されません。

例

リファクタリング エンジンによって参照が更新されない場合のコード例を次に示します。リファクタリングを使用して **example** を別の名前に変更すると、**ContainsSyntaxError** 内の参照は更新されません。その他の 2 つの参照は更新されます。

```
public class Class1
{
    static int example;

    static void ContainsSyntaxError()
    {
        example = 20
    }

    static void ContainsSemanticError()
```

```
        {
            example = "Three";
        }

        static void ContainsNoError()
        {
            example = 1;
        }
    }
```

参照

概念

リファクタリング

[検証結果] ダイアログ ボックス

このダイアログ ボックスは、リファクタリングの検証プロセス中に、リファクタリング エンジンがコンパイル エラーまたは再バインディングの問題を検出したときに表示されます。

[検証結果]

リファクタリング操作の結果としてコンパイル エラーまたは再バインディングの問題が含まれるステートメントを識別します。

[プレビューに関する問題]

プログラムでコンパイル エラーまたは再バインディング上の問題の原因になる参照の変更 (またはリファクタリングされたコード) のプレビューを表示します。

解説

このダイアログ ボックスは、次の場合は表示されません。

- リファクタリング操作を開始する前に未解決のコンパイル エラーが生成され、リファクタリング操作の結果として再バインディングの問題が発生しない場合。
- [変更のプレビュー] ダイアログ ボックスですべての参照を消去した場合。
- 警告ダイアログ ボックスで、名前付けの競合が存在し、リファクタリング操作で検証プロセスがスキップされることが示され、[はい] をクリックした場合。

再バインディングの問題

再バインディングの問題は、リファクタリング操作によって、コード参照が当初バインドされていた対象とは異なる対象に意図せずバインドされた場合に発生します。[検証結果] ダイアログ ボックスでは、2 種類の再バインディングの問題が区別されます。

参照の定義が名前変更されたシンボルではなくなる場合

この種類の再バインディングの問題は、参照が名前変更されたシンボルを参照しなくなった場合に発生します。次に例を示します。

```
class Example
{
private int a;
    public Example(int b)
    {
        a = b;
    }
}
```

リファクタリングを使用して `a` を `b` に名前変更すると、このダイアログ ボックスが表示されます。名前変更された変数 `a` に対する参照は、フィールドにバインドされず、コンストラクタに渡されるパラメータにバインドされるようになります。

参照の定義が名前変更されたシンボルになる場合

この種類の再バインディングの問題は、名前変更されたシンボルを参照していなかった参照が、名前変更されたシンボルを参照するようになった場合に発生します。次に例を示します。

```
class Example
{
    private static void Method(object a)
    {
    }
    private static void OtherMethod(int a)
    {
    }
    static void Main(string[] args)
    {
        Method(5);
    }
}
```

リファクタリングを使用して `OtherMethod` を `Method` に名前変更すると、このダイアログ ボックスが表示されます。`Main` 内の参照は、**object** パラメータを受け入れるオーバーロードされたメソッドではなく、**int** パラメータを受け入れるオーバーロードされたメソッドを参照するようになります。

参照

関連項目

[変更のプレビュー] ダイアログ ボックス

概念

リファクタリング

コード スニペット (C#)

Visual Studio には、コード スニペットと呼ばれる新機能が用意されています。コード スニペットを使用すると、短いエイリアスを入力し、それを一般的なプログラミング構成要素に展開できます。たとえば、**for** コード スニペットからは空の **for** ループが作成されます。一部のコード スニペットは、ブロックの挿入コード スニペットです。このコード スニペットでは、コードの行を選択してからコード スニペットを選択すると、コードの選択した行が取り込まれます。たとえば、コードの行を選択してから **for** コード スニペットをアクティブにすると、これらのコード行をループ ブロック内に含む **for** ループが作成されます。コード スニペットを使用すると、プログラム コードを短時間で簡単に作成でき、その信頼性を向上できます。

コード スニペットの使用

通常、コード スニペットは、コード エディタでエイリアスの短い名前 (コード スニペット ショートカット) を入力し、Tab キーを押して使用します。[IntelliSense] メニューにも [コード スニペットの挿入] メニュー コマンドが用意されており、コード エディタに挿入して使用できるコード スニペットの一覧が表示されます。コード スニペットの一覧は、Ctrl キーを押しながら K キーを押し、次に X キーを押してアクティブにできます。詳細については、「方法 : コード スニペットを使用する (C#)」および「方法 : ブロックの挿入コード スニペットを使用する」を参照してください。

コード スニペットを選択すると、コード スニペットのテキストが自動的にカーソル位置に挿入されます。このとき、コード スニペット内の編集できるすべてのフィールドは黄色で強調表示され、1 番目の編集できるフィールドが自動的に選択されます。現在選択されているフィールドは、赤い四角で囲まれています。たとえば **for** コード スニペットの場合、編集できるフィールドは、初期化子変数 (既定では `i`) および長さ式 (既定では `length`) です。

フィールドが選択されている場合、ユーザーはそのフィールドの新しい値を入力できます。Tab キーを押すと、コード スニペットの編集できるフィールド間を移動できます。Shift キーを押しながら Tab キーを押すと、逆順に移動できます。フィールドをクリックすると、カーソルがそのフィールドに移動し、フィールドをダブルクリックすると、そのフィールドが選択されます。フィールドが強調表示されたとき、フィールドの説明を示すツールヒントが表示されることがあります。

各フィールドの 1 番目のインスタンスだけを編集できます。そのフィールドが強調表示されると、そのフィールドの他のインスタンスはアウトライン化されます。編集できるフィールドの値を変更すると、そのフィールドは、コード スニペット内で使用されているすべての場所で変更されます。

Enter キーまたは Esc キーを押すと、フィールドの編集はキャンセルされ、コード エディタは通常の状態に戻ります。

編集できるコード スニペット フィールドの既定の色を変更するには、[オプション] ダイアログ ボックスの [フォントおよび色] ペインの [コード スニペット フィールド] 設定を変更します。詳細については、「方法 : エディタで使用するフォントのフォント フェイス、サイズ、色を変更する」を参照してください。

コード スニペットの作成

既定で Visual Studio に含まれているコード スニペットだけでなく、カスタム コード スニペットを作成して利用できます。カスタム コード スニペットの作成の詳細については、「コード スニペットの作成」を参照してください。

メモ :

C# コード スニペットの場合、[<ショートカット>] フィールドを指定するのに使用できる文字は、英数字、シャープ記号 (#)、ティルダ (~)、アンダースコア (_)、および半角ダッシュ (-) です。

既定で Visual C# に含まれるコード スニペットの詳細については、「既定のコード スニペット」を参照してください。

参照

関連項目

コード スニペット ピッカー

既定のコード スニペット

コード スニペット インサータでは、カーソル位置にコード スニペットを挿入したり、現在選択されているコードの周りにブロックの挿入コード スニペットを挿入したりします。コード スニペット インサータは、[IntelliSense] メニューの [コード スニペットの挿入] または [ブロックの挿入] から呼び出すか、キーボード ショートカットとして Ctrl キーを押しながら K キーを押し、次に X キーを押すか、Ctrl キーを押しながら K キーを押して、次に S キーを押して呼び出します。

コード スニペット インサータでは、利用できるすべてのコード スニペットのコード スニペット名が表示されます。コード スニペット インサータには、コード スニペットの名前または名前の一部を入力できる入力ダイアログ ボックスもあります。コード スニペット名と最もよく一致する項目が強調表示されます。Tab キーを押すと、いつでもコード スニペット インサータが閉じ、現在選択されているコード スニペットが挿入されます。Esc キーを押すか、コード エディタでマウスをクリックすると、コード スニペットが挿入されないまま、コード スニペット インサータが閉じます。

既定のコード スニペット

既定で Visual Studio に含まれているコード スニペットは次のとおりです。

名前 (またはショートカット)	説明	スニペットを挿入できる位置
#if	#if ディレクティブおよび #endif ディレクティブを作成します。	任意。
#region	#region ディレクティブおよび #endregion ディレクティブを作成します。	任意。
~	含んでいるクラスのデストラクタを作成します。	クラスの内部。
attribute	Attribute から派生したクラスの宣言を作成します。	名前空間 (グローバル名前空間を含む)、クラス、または構造体の内部。
checked	checked ブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
class	クラス宣言を作成します。	名前空間 (グローバル名前空間を含む)、クラス、または構造体の内部。
ctor	含んでいるクラスのコンストラクタを作成します。	クラスの内部。
cw	WriteLine の呼び出しを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
do	do while ループを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
else	else ブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
enum	列挙型宣言を作成します。	名前空間 (グローバル名前空間を含む)、クラス、または構造体の内部。
equals	Object クラスで定義された Equals メソッドをオーバーライドするメソッド宣言を作成します。	クラスまたは構造体の内部。
exception	例外 (既定では Exception) から派生したクラスの宣言を作成します。	名前空間 (グローバル名前空間を含む)、クラス、または構造体の内部。
for	for ループを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。

foreach	foreach ループを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
forr	各反復後にループ変数の値をデクリメントする for ループを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
if	if ブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
indexer	インデクサ宣言を作成します。	クラスまたは構造体の内部。
interface	インターフェイス宣言を作成します。	名前空間 (グローバル名前空間を含む)、クラス、または構造体の内部。
invoke	イベントのセキュリティ保護された呼び出しを行うブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
iterator	反復子を作成します。	クラスまたは構造体の内部。
iterindex	入れ子にされたクラスを使用して、反復子とインデクサの "名前付き" ペアを作成します。	クラスまたは構造体の内部。
lock	lock ブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
mbox	System.Windows.Forms.MessageBox.Show の呼び出しを作成します。System.Windows.Forms.dll への参照を追加する必要がある場合があります。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
namespace	名前空間宣言を作成します。	名前空間 (グローバル名前空間を含む) の内部。
prop	プロパティ宣言およびバッキング フィールドを作成します。	クラスまたは構造体の内部。
propg	"get" アクセサだけとバッキング フィールドを持つプロパティ宣言を作成します。	クラスまたは構造体の内部。
sim	static int Main メソッド宣言を作成します。	クラスまたは構造体の内部。
struct	構造体宣言を作成します。	名前空間 (グローバル名前空間を含む)、クラス、または構造体の内部。
svm	static void Main メソッド宣言を作成します。	クラスまたは構造体の内部。
switch	switch ブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
try	try-catch ブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
tryf	try-finally ブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
unchecked	unchecked ブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。

unsafe	unsafe ブロックを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。
using	using ディレクティブを作成します。	名前空間 (グローバル名前空間を含む) の内部。
while	while ループを作成します。	メソッド、インデクサ、プロパティ アクセサ、またはイベント アクセサの内部。

解説

ショートカットを使用すると、メニューを使用しなくても、IntelliSense によって自動的にコード スニペットがコード エディタに入力されます。詳細については、「方法 : コード スニペットを使用する (C#)」を参照してください。

参照

処理手順

方法 : ブロックの挿入コード スニペットを使用する

関連項目

コード スニペット ピッカー

概念

コード スニペット (C#)

方法：コード スニペットを使用する (C#)

次の手順では、コード スニペットを使用する方法を説明します。コード スニペットは、キーボード ショートカット、IntelliSense オート コンプリート、IntelliSense コンプリート単語リスト、[編集] メニュー、およびコンテキスト メニューという 5 つの方法で利用できます。

キーボード ショートカットからコード スニペットを使用するには

1. Visual Studio IDE で、編集するファイルを開きます。
2. コード エディタで、コード スニペットを挿入する位置にカーソルを移動します。
3. Ctrl キーを押しながら K キーを押し、次に Ctrl キーを押しながら X キーを押します。
4. コード スニペット インサータからコード スニペットを選択し、Tab キーまたは Enter キーを押します。

   または、コード スニペットの名前を入力し、Tab キーまたは Enter キーを押すこともできます。

IntelliSense オート コンプリートでコード スニペットを使用するには

1. Visual Studio IDE で、編集するファイルを開きます。
2. コード エディタで、コード スニペットを挿入する位置にカーソルを移動します。
3. コードに追加するコード スニペットのショートカットを入力します。
4. Tab キーを押し、もう一度 Tab キーを押して、コード スニペットを呼び出します。

IntelliSense コンプリート単語リストからコード スニペットを使用するには

1. Visual Studio IDE で、編集するファイルを開きます。
2. コード エディタで、コード スニペットを挿入する位置にカーソルを移動します。
3. コードに追加するコード スニペットのショートカットを 1 文字ずつ入力していきます。オート コンプリートがオンの場合、IntelliSense コンプリート単語リストが表示されます。表示されない場合は、Ctrl キーを押しながら Space キーを押してアクティブにします。
4. コンプリート単語リストからコード スニペットを選択します。
5. Tab キーを押し、もう一度 Tab キーを押して、コード スニペットを呼び出します。

[編集] メニューからコード スニペットを使用するには

1. Visual Studio IDE で、編集するファイルを開きます。
2. コード エディタで、コード スニペットを挿入する位置にカーソルを移動します。
3. [編集] メニューの [IntelliSense] をポイントし、[スニペットの挿入] をクリックします。
4. コード スニペット インサータからコード スニペットを選択し、Tab キーまたは Enter キーを押します。

   または、コード スニペットの名前を入力し、Tab キーまたは Enter キーを押すこともできます。

コンテキスト メニューからコード スニペットを使用するには

1. Visual Studio IDE で、編集するファイルを開きます。
2. コード エディタで、コード スニペットを挿入する位置にカーソルを移動します。
3. カーソルを右クリックし、コンテキスト メニューの [スニペットの挿入] をクリックします。
4. コード スニペット インサータからコード スニペットを選択し、Tab キーまたは Enter キーを押します。

   または、コード スニペットの名前を入力し、Tab キーまたは Enter キーを押すこともできます。

参照

処理手順

方法 : ブロックの挿入コード スニペットを使用する

関連項目

既定のコード スニペット
コード スニペット ピッカー
概念
コード スニペット (C#)

方法：ブロックの挿入コード スニペットを使用する

次の手順では、ブロックの挿入コード スニペットを使用する方法を説明します。ブロックの挿入コード スニペットは、キーボード ショートカット、[編集] メニュー、およびコンテキスト メニューの 3 種類の方法で利用できます。

キーボード ショートカットからブロックの挿入コード スニペットを使用するには

1. Visual Studio IDE で、編集するファイルを開きます。
2. コード エディタで、囲む対象のテキストを選択します。
3. Ctrl キーを押しながら K キーを押し、次に Ctrl キーを押しながら S キーを押します。
4. マウスを使用するか、またはコード スニペット名を入力してから Tab キーまたは Enter キーを押して、コード スニペット リストからコード スニペットを選択します。

[編集] メニューからブロックの挿入コード スニペットを使用するには

1. Visual Studio IDE で、編集するファイルを開きます。
2. コード エディタで、囲む対象のテキストを選択します。
3. [編集] メニューの [IntelliSense] をポイントし、[ブロックの挿入] をクリックします。
4. コード スニペット インサータからコード スニペットを選択し、Tab キーまたは Enter キーを押します。

   または、コード スニペットの名前を入力し、Tab キーまたは Enter キーを押すこともできます。

コンテキスト メニューからブロックの挿入コード スニペットを使用するには

1. Visual Studio IDE で、編集するファイルを開きます。
2. コード エディタで、囲む対象のテキストを選択します。
3. 選択したテキストを右クリックし、コンテキスト メニューの [ブロックの挿入] をクリックします。
4. コード スニペット インサータからコード スニペットを選択し、Tab キーまたは Enter キーを押します。

   または、コード スニペットの名前を入力し、Tab キーまたは Enter キーを押すこともできます。

参照

処理手順

方法 : コード スニペットを使用する (C#)

関連項目

既定のコード スニペット

コード スニペット ピッカー

概念

コード スニペット (C#)

方法：C# リファクタリング スニペットを復元する

C# リファクタリング操作は、次のディレクトリにあるコード スニペットに依存しています。

Installation directory\Microsoft Visual Studio 8\VC#\Snippets*language ID*\Refactoring

この Refactoring ディレクトリまたはこのディレクトリ内のファイルを削除または破損すると、IDE では、C# リファクタリング操作が機能しない場合があります。次の手順は、C# リファクタリング コード スニペットを復元する場合に役立ちます。

C# リファクタリング スニペットをコード スニペット マネージャで使用できることを確認するには

1. [ツール] メニューの [コード スニペット マネージャ] を選択します。
2. [コード スニペット マネージャ] ダイアログ ボックスで、[言語] ボックスの一覧の [Visual C#] を選択します。

   [リファクタリング] フォルダがツリー ビューのフォルダの一覧に表示されます。

コード スニペット マネージャで、リファクタリング スニペットを復元するには

1. [ツール] メニューの [コード スニペット マネージャ] を選択します。
2. [コード スニペット マネージャ] ダイアログ ボックスで、[言語] ボックスの一覧の [Visual C#] を選択します。
3. [追加] をクリックします。[コード スニペット ディレクトリ] ダイアログ ボックスが表示されます。このダイアログ ボックスを使用して、コード スニペット マネージャに追加するディレクトリを検索および指定します。
4. ディレクトリ パスが次のいずれかである Refactoring フォルダを検索します。

   Installation directory\Microsoft Visual Studio 8\VC#\Snippets*language ID*\Refactoring
5. [コード スニペット ディレクトリ] ダイアログ ボックスの [開く] をクリックし、次にコード スニペット マネージャの [OK] をクリックします。

リファクタリング コード スニペットのディレクトリを復元するには

1. [コード スニペット マネージャ] ダイアログ ボックスで [オンライン検索] をクリックします。
2. 「リファクタリング」と入力し、[検索] をクリックします。

   検索結果には、Refactoring フォルダを再インストールする際に使用する .vsi ファイルをダウンロードできる Web サイトが含まれています。

参照

関連項目

コード スニペット マネージャ

概念

リファクタリング

コードの色づけ

コード エディタでは、コード エディタでコードの内容の認識や識別を容易にするために、トークンおよびコード コンストラクタが解析されます。コード エディタでコードが解析されると、コード コンストラクタが適切に色づけされます。

[トークン]

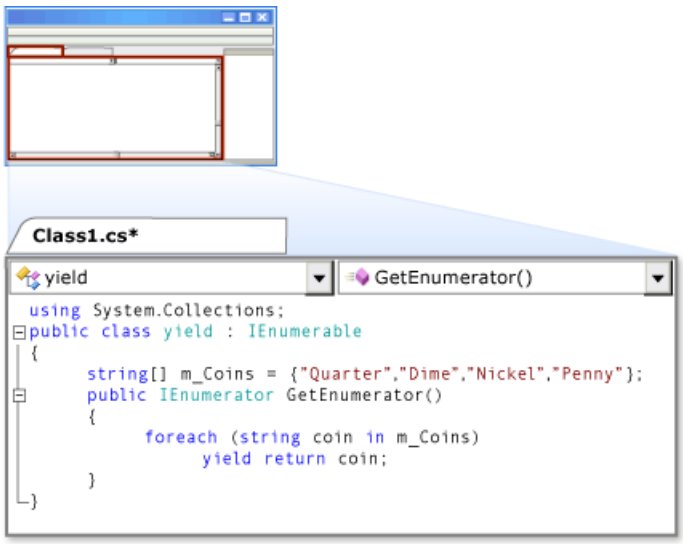
コード エディタでは、次のトークン型を色づけします。

- コメント
- 除外されたコード
- 識別子
- キーワード
- 数字
- 演算子
- プリプロセッサ キーワード
- 文字列
- 文字列 (C# @ Verbatim)
- ユーザー型
- ユーザー型 (値型)
- ユーザー型 (列挙型)
- ユーザー型 (デリゲート)
- XML CData セクション
- XML Doc 属性
- XML Doc コメント
- XML Doc タグ

既定の色づけ設定は、[フォントおよび色] ([オプション] ダイアログ ボックス - [環境]) を使用して変更できます。

コンテキスト キーワード

コード エディタでは、コンテキスト キーワードが適切に色づけされます。次の例では、**yield** 型を濃い青緑色に、**yield** キーワードを青に色づけします。



かっこの対応を示す色づけ

コード エディタでは、かっこが対応していることを示すために、太字による色づけまたは強調表示による色づけを使用します。

太字の色づけ

次のコード コンストラクタ ペアを編集するとき、文字列またはコード コンストラクタのペアは、互いに対応していることを示すために短時間太字で表示されます。

" "	文字列
@" "	逐語的文字列
#if, **#endif**	条件セクションのプリプロセッサ ディレクティブ
#region, **#endregion**	条件セクションのプリプロセッサ ディレクティブ
case, **break**	制御ステートメントのキーワード
default, **break**	制御ステートメントのキーワード
for, **break**	評価式のキーワード
for, **continue**	評価式のキーワード
foreach, **break**	評価式のキーワード
foreach, **continue**	評価式のキーワード
while, **break**	評価式のキーワード
while, **continue**	評価式のキーワード

[全般] ([オプション] ダイアログ ボックス - [テキスト エディタ]) の [区切り記号を自動強調表示する] プロパティをオフにすることで、この機能を無効にできます。

強調表示による色づけ

カーソルが開始区切り文字の直前に置かれた場合、または終了区切り文字の直後に置かれた場合、開始区切り文字と終了区切り文字が互いに対応していることを示すために、強調表示のための灰色の四角形が表示されます。この機能は、次のペアで利用できます。

{ }	中かっこ
[ ]	角かっこ
( )	かっこ

例

かっこの対応を示す色づけを確認するには、コード エディタに次のコードを入力してください (このコードをコピーして貼り付けないでください)。

```
class A
{
     public A()
     {
          if(true)
               int x =0;
          else
               int x =1;
     }
}
```

色分けの設定

色分けの設定は、Visual Studio の設定 で永続的です。

参照

関連項目

自動一致機能

ソースとして使用するメタデータ

メタデータをソースに指定すると、読み取り専用バッファで C# ソース コードとして扱われるメタデータを表示できます。これにより、(実装を含まない) 型およびメンバの宣言を表示できます。メタデータをソースとして表示するには、プロジェクトまたはソリューションから使用できないソース コードを持つ型またはメンバに対して [定義へ移動] コマンドを実行します。

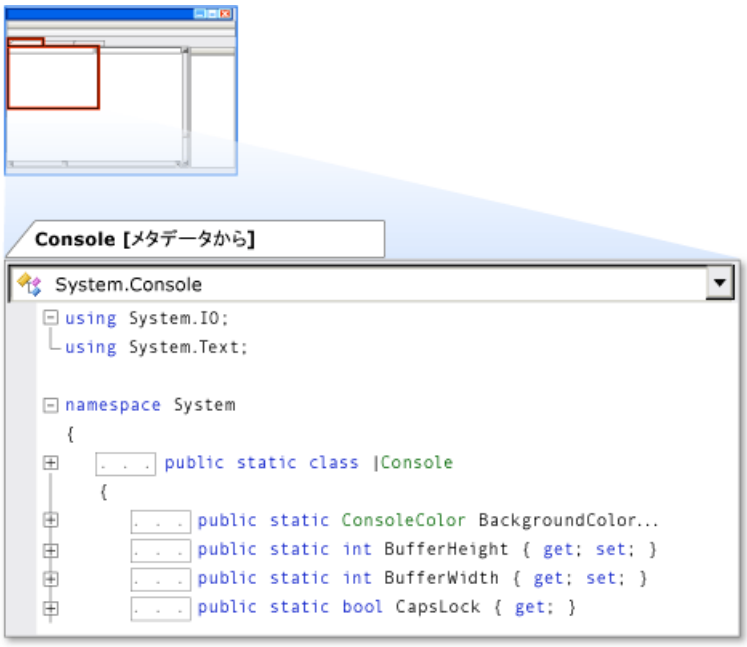
メモ :
internal として宣言されている型またはメンバに対して [定義へ移動] コマンドを実行すると、統合開発環境 (IDE: Integrated Development Environment) では、参照先のアセンブリがフレンド アセンブリかどうかに関係なく、型またはメンバのメタデータがソースとして表示されません。

メタデータは、コード エディタまたは [コード定義] ウィンドウでソースとして表示できます。

コード エディタでメタデータをソースとして表示する方法

ソース コードを使用できない項目に対して [定義へ移動] コマンドを実行すると、その項目のメタデータのビューを含むタブ付きドキュメントは、コード エディタでソースとして表示されます。[[メタデータから]] の前にある型名は、ドキュメントのタブに表示されます。

たとえば、Console に対して [定義へ移動] コマンドを実行した場合、**Console** のメタデータは、コード エディタで、その宣言のように C# ソース コードとして表示されますが、実装を含んでいません。



[コード定義] ウィンドウでメタデータをソースとして表示する方法

[コード定義] ウィンドウがアクティブな場合や表示されている場合、IDE では、コード エディタでカーソルの下にある項目、および [クラス ビュー] または [オブジェクト ブラウザ] で選択されている項目に対して自動的に [定義へ移動] コマンドが実行されます。その項目のソース コードが使用できない場合、IDE では、その項目のメタデータがソースとして [コード定義] ウィンドウに表示されます。

たとえば、コード エディタで **Console** という単語にカーソルを配置した場合、**Console** のメタデータが [コード定義] ウィンドウにソースとして表示されます。このソースは、**Console** の宣言に似ていますが、実装を含んでいません。

[コード定義] ウィンドウに表示される項目の宣言を表示する場合、[コード定義] ウィンドウは 1 レベル下のウィンドウなので、[定義へ移動] コマンドを明示的に使用する必要があります。

参照

関連項目

[コード定義ウィンドウ]
[シンボルの検索結果] ウィンドウ

Visual C# のショートカット キー

Visual C# には、マウスまたはメニューを使用せずにアクションを実行する場合に使用できる多くのショートカット キーが用意されています。

このセクションの内容

参照

その他の技術情報

Visual C#
Visual C# IDE の使用

Visual C# IDE の設定

Visual C# の設定とは、ツール ウィンドウ、メニュー、およびキーボード ショートカットの定義済みの構成です。これらの設定は、Visual Studio の設定 機能の一部であり、作業上の習慣に合わせてカスタマイズできます。

ウィンドウとビュー

機能	既定での表示	メモ
クラス ビュー	×	• クラス ビューは [表示] メニューで利用できます。 • フィルタ処理を有効にします。
コマンド ウィンドウ	×	
[ダイナミック ヘルプ] ウィンドウ	×	F1 キーを押しても、ダイナミック ヘルプ ウィンドウは表示されません。 ダイナミック ヘルプ ウィンドウの詳細については、 「方法 : [ダイナミック ヘルプ] をカスタマイズする」または 「方法 : [ダイナミック ヘルプ] ウィンドウを制御する」を参照してください。
オブジェクト ブラウザ	×	• 既定では、継承されたメンバは表示されません。
[出力] ウィンドウ	×	
ソリューション エクスプローラ	○	ソリューション エクスプローラは、IDE の右側にドッキングされて表示されます。
スタート ページ	IDE を起動した場合は○	[スタート ページ] には、Visual C# に関する MSDN RSS フィードの文書が表示されます。
タスク一覧 (Visual Studio)	×	
ツールボックス	Windows フォーム アプリケーションを作成するときは○	ツールボックスは、IDE の左側にドッキングされ、折りたたまれたウィンドウとして表示されます。

キーボード

機能	動作
ショートカット キー	Visual C# では、次のショートカット キー設定をサポートしています。 • Visual C# 2005 の既定のショートカット キー • Brief の既定のショートカット キー • Emacs の既定のショートカット キー • Visual C++ 2.0 の既定のショートカット キー • Visual Studio 6.0 の既定のショートカット キー

参照

その他の技術情報

Visual C#
Visual C# IDE の使用

Visual C# 2005 の既定のショートカット キー

統合開発環境 (IDE) には、定義済みのキーボード バインディング スキームが複数用意されています。Visual C# 2005 のキーボード マップ スキームに切り替えるには、[ツール] メニューの [オプション] をクリックし、[環境] を展開し、[キーボード] をクリックします。

Visual C# 2005 のキーボード マップ スキームは、[設定のインポートとエクスポート ウィザード] ウィンドウの [Visual C# 開発設定] をクリックしたときに選択される既定のキーボード マップ スキームでもあります。詳細については、「方法 : 選択した設定を変更する」を参照してください。

メモ :
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

次のトピックでは、Visual C# 2005 のキーボード マップ スキームで使用できる既定のショートカット キーを紹介します。

- グローバル ショートカット キー (Visual C# 2005 スキーム)
- HTML デザイナのショートカット キー (Visual C# 2005 スキーム)
- XML デザイナのショートカット キー (Visual C# 2005 スキーム)
- コントロール操作のショートカット キー (Visual C# 2005 スキーム)
- デバッグのショートカット キー (Visual C# 2005 スキーム)
- 検索と置換のショートカット キー (Visual C# 2005 スキーム)
- データのショートカット キー (Visual C# 2005 スキーム)
- テキスト移動のショートカット キー (Visual C# 2005 スキーム)
- テキスト選択のショートカット キー (Visual C# 2005 スキーム)
- テキスト操作のショートカット キー (Visual C# 2005 スキーム)
- ウィンドウ管理のショートカット キー (Visual C# 2005 スキーム)
- 統合ヘルプのショートカット キー (Visual C# 2005 スキーム)
- オブジェクト ブラウザのショートカット キー (Visual C# 2005 スキーム)
- マクロのショートカット キー (Visual C# 2005 スキーム)
- ツール ウィンドウのショートカット キー (Visual C# 2005 スキーム)
- プロジェクトのショートカット キー (Visual C# 2005 スキーム)
- イメージ エディタのショートカット キー (Visual C# 2005 スキーム)
- ダイアログ エディタのショートカット キー (Visual C# 2005 スキーム)
- リファクタリングのショートカット キー (Visual C# 2005 スキーム)
- マネージ リソース エディタのショートカット キー (Visual C# 2005 スキーム)
- コード スニペットのショートカット キー (Visual C# 2005 スキーム)
- クラス ダイアグラムのショートカット キー (Visual C# 2005 スキーム)
- ブックマーク ウィンドウのショートカット キー (Visual C# 2005 スキーム)
- アクセラレータ エディタおよびストリング エディタのショートカット キー (Visual C# 2005 スキーム)

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

その他の技術情報

ショートカット キー

グローバル ショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、統合開発環境 (IDE: Integrated Development Environment) 内のさまざまな場所で使用できます。

メモ :

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Diagram.Properties	**Alt** + **Enter**	ダイアグラムから [プロパティ] ウィンドウにフォーカスを切り替えます。
Edit.Copy	Ctrl + C	選択したアイテムをクリップボードにコピーします。
Edit.Cut	Ctrl + X	選択したアイテムをファイルから削除し、クリップボードにコピーします。
Edit.CycleClipboardRing	Ctrl + Shift + V	クリップボード リングのアイテムをファイル内のカーソル位置に貼り付け、貼り付けたアイテムを自動的に選択します。ショートカット キーを繰り返し押すと、クリップボード リングのアイテムを 1 つずつ確認できます。
Edit.Delete	Delete	カーソル位置の右側にある 1 文字を削除します。
Edit.OpenFile	**Ctrl** + **Shift** + **G**	[ファイルを開く] ダイアログ ボックスを表示します。このダイアログ ボックスでは、ファイルを選択して開くことができます。
Edit.Paste	Ctrl + V	カーソル位置に、クリップボードの内容を挿入します。
Edit.Redo	**Ctrl** + **Y**	前回取り消した操作を再度実行します。
Edit.Undo	Ctrl + Z	最後に行った編集操作を元に戻します。
File.Print	Ctrl + P	[印刷] ダイアログ ボックスを表示します。このダイアログ ボックスでは、プリンタの設定を選択できます。
File.SaveAll	**Ctrl** + **Shift** + **S**	現在のソリューションのドキュメントと外部ファイル プロジェクトのファイルをすべて保存します。
File.SaveSelectedItems	**Ctrl** + **S**	選択したアイテムを現在のプロジェクトに保存します。
Tools.GoToCommandLine	**Ctrl** + **/**	[標準] ツール バーの [検索/コマンド] ボックスにポインタを配置します。
View.Backward	Alt + ←	履歴の表示で前のページを表示します。[Web ブラウザ] ウィンドウでだけ使用できます。
View.EditLabel	F2	[ソリューション エクスプローラ] ウィンドウで選択されたアイテムの名前を変更できます。
View.Forward	Alt + →	履歴の表示で次のページを表示します。[Web ブラウザ] ウィンドウでだけ使用できます。
View.ViewCode	F7	選択したアイテムをエディタのコード ビューに表示します。

View.ViewDesigner	Shift + F7	選択したアイテムをデザイナ ビューに表示します。

参照

概念

Visual C# 2005 の既定のショートカット キー

その他の技術情報

ショートカット キー

HTML デザイナのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、HTML デザイナでファイルを編集しているときにだけ使用できます。デザイナの特定のビューでしか使用できないショートカット キーもあります。HTML デザイナで使用できるその他のショートカット キーとして は、テキスト移動のショートカット キー (全般的な開発設定)、テキスト選択のショートカット キー (全般的な開発設定)、テキスト操作のショートカット キー (全般的な開発設定) などがあります。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Format.Bold	Ctrl + B	選択したテキストを太字と標準の間で切り替えます。デザイン ビューだけで使用できます。
Format.ConvertToHyperlink	Ctrl + L	テキストが選択されたときに、[ハイパーリンク] ダイアログ ボックスを表示します。デザイン ビューだけで使用できます。
Format.InsertBookmark	**Ctrl** + **Shift** + **L**	[ブックマーク] ダイアログ ボックスを表示します。デザイン ビューだけで使用できます。
Format.Italic	**Ctrl** + **I**	選択したテキストを斜体と標準の間で切り替えます。デザイン ビューだけで使用できます。
Format.Underline	Ctrl + U	選択したテキストを下線付きと標準の間で切り替えます。デザイン ビューだけで使用できます。
Layout.InsertColumntotheLeft	**Ctrl** + **Alt** + ←	テーブルの現在の列の左に列を 1 列追加します。デザイン ビューだけで使用できます。
Layout.InsertColumntotheRight	**Ctrl** + **Alt** + →	テーブルの現在の列の右に列を 1 列追加します。デザイン ビューだけで使用できます。
Layout.InsertRowAbove	**Ctrl** + **Alt** + ↑	テーブルの現在の行の上に行を 1 行追加します。デザイン ビューだけで使用できます。
Layout.InsertRowBelow	Ctrl + Alt + ↓	テーブルの現在の行の下に行を 1 行追加します。デザイン ビューだけで使用できます。
Project.AddContentPage	Ctrl + M、Ctrl + C	新しい *.aspx ファイルを Web サイトに追加し、このファイルを HTML デザイナで開きます。デザイン ビューだけで使用できます。
View.AutoCloseTagOverride	Ctrl + Shift + ピリオド (.)	現在のタグについて、終了タグの既定の動作を一時的にオーバーライドします。詳細については、「[タグ指定オプション]」を参照してください。ソース ビューだけで使用できます。
View.Details	**Ctrl** + **Shift** + **Q**	コメント、スクリプト、絶対座標で配置された要素のアンカーなど、ビジュアルな表示を持たない HTML 要素のアイコンを表示します。デザイン ビューだけで使用できます。
View.EditMaster	**Ctrl** + **M**、**Ctrl** + **M**	ソース ビューで *.master ファイルを開きます。デザイン ビューだけで使用できます。
View.NextView	Ctrl + PageDown	現在のドキュメントのデザイン ビュー、ソース ビュー、およびサーバー コード ビューを切り替えます。すべてのビューで使用できます。
View.NonVisualControls	**Ctrl** + **Alt** + **Q**	div、span、form、script など、表示されない要素についてシンボルを表示します。デザイン ビューだけで使用できます。
View.ShowSmartTag	Shift + Alt + F10	Web サーバー コントロールの一般的なコマンドについて、スマート タグのメニューを表示します。デザイン ビューだけで使用できます。
View.ViewDesigner	Shift + F7	現在のドキュメントをデザイン ビューに切り替えます。ソース ビューだけで使用できます。
View.ViewMarkup	Shift + F7	現在のドキュメントをソース ビューに切り替えます。デザイン ビューだけで使用できます。
View.VisibleBorders	Ctrl + Q	ゼロに設定された BORDER 属性をサポートする HTML 要素の周囲に 1 ピクセルの境界線を表示します。これらの HTML 要素には、テーブル、テーブル セル、区分などがあります。デザイン ビューだけで使用できます。

Window.PreviousTab	Ctrl + PageUp	現在のドキュメントのデザイン ビュー、ソース ビュー、およびサーバー コード ビューを切り替えます。すべてのビューで使用できます。

参照

関連項目

HTML デザイナ

概念

Visual C# 2005 の既定のショートカット キー

XML デザイナのショートカット キー (Visual C# 2005 スキーム)

XML デザイナでは、以下のショートカット キーを使用できます。

メモ :
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Schema.Collapse	Ctrl + 負符号 (-)	入れ子になった要素を折りたたみます。XML デザイナのスキーマ ビューでだけ使用できます。
Schema.Expand	Ctrl + 等号 (=)	入れ子になった要素を展開します。XML デザイナのスキーマ ビューでだけ使用できます。

参照

概念

Visual C# 2005 の既定のショートカット キー

コントロール操作のショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、デザイン画面のコントロールの移動、選択、およびサイズを変更するときに使用できます。

メモ :

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Edit.MoveControlDown	↓	デザイン サーフェイスで、コントロールを下に 1 ピクセルずつ移動します。
Edit.MoveControlDownGrid	**Ctrl** + ↓	デザイン サーフェイスで、コントロールを下に 8 ピクセルずつ移動します。
Edit.MoveControlLeft	←	デザイン サーフェイスで、コントロールを左に 1 ピクセルずつ移動します。
Edit.MoveControlLeftGrid	Ctrl + ←	デザイン サーフェイスで、コントロールを左に 8 ピクセルずつ移動します。
Edit.MoveControlRight	→	デザイン サーフェイスで、コントロールを右に 1 ピクセルずつ移動します。
Edit.MoveControlRightGrid	Ctrl + →	デザイン サーフェイスで、コントロールを右に 8 ピクセルずつ移動します。
Edit.MoveControlUp	↑	デザイン サーフェイスで、コントロールを上に 1 ピクセルずつ移動します。
Edit.MoveControlUpGrid	**Ctrl** + ↑	デザイン サーフェイスで、コントロールを上に 8 ピクセルずつ移動します。
Edit.SelectNextControl	Tab	コントロールの Tabindex プロパティに基づいて、ページ上の次のコントロールに移動します。
Edit.SelectPreviousControl	**Shift** + **Tab**	ページで前に選択したコントロールに戻ります。
Edit.ShowTileGrid	Enter	デザイン領域にグリッドを表示します。
Edit.SizeControlDown	Shift + ↓	デザイン サーフェイスで、コントロールの高さを 1 ピクセルずつ増加します。
Edit.SizeControlDownGrid	**Ctrl** + **Shift** + ↓	デザイン サーフェイスで、コントロールの高さを 8 ピクセルずつ増加します。
Edit.SizeControlLeft	Shift + ←	デザイン サーフェイスで、コントロールの幅を 1 ピクセルずつ減少します。
Edit.SizeControlLeftGrid	**Ctrl** + **Shift** + ←	デザイン サーフェイスで、コントロールの幅を 8 ピクセルずつ減少します。
Edit.SizeControlRight	Shift + →	デザイン サーフェイスで、コントロールの幅を 1 ピクセルずつ増加します。
Edit.SizeControlRightGrid	**Ctrl** + **Shift** + →	デザイン サーフェイスで、コントロールの幅を 8 ピクセルずつ増加します。
Edit.SizeControlUp	Shift + ↑	デザイン サーフェイスで、コントロールの高さを 1 ピクセルずつ減少します。
Edit.SizeControlUpGrid	**Ctrl** + **Shift** + ↑	デザイン サーフェイスで、コントロールの高さを 8 ピクセルずつ減少します。

参照

概念

Visual C# 2005 の既定のショートカット キー

デバッグのショートカット キー (Visual C# 2005 スキーム)

次のショートカット キーは、コードのデバッグ時に使用できます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Debug.ApplyCodeChanges	Alt + F10	エディット コンティニュ機能を使用してデバッグ対象のコードに変更を適用できるビルドを開始します。
Debug.Autos	Ctrl + D、Ctrl + A	[自動変数] ウィンドウを表示して、現在のプロシージャで現在実行している行のスコープに現在存在する変数の値を表示します。
Debug.BreakAll	**Ctrl + Alt + Break**	デバッグ セッションのすべてのプロセスの実行を一時的に停止します。実行モードでだけ使用できます。
Debug.BreakAtFunction	Ctrl + D、Ctrl + N	[ブレークポイントの作成] ダイアログ ボックスを表示します。
Debug.Breakpoints	Ctrl + D、Ctrl + B	[ブレークポイント] ダイアログ ボックスを表示します。このダイアログ ボックスでは、ブレークポイントの追加と変更ができます。
Debug.CallStack	Ctrl + D、Ctrl + C	[呼び出し履歴] ウィンドウに、すべてのアクティブ プロシージャの一覧、または現在の実行スレッドのスタック フレームの一覧を表示します。実行モードでだけ使用できます。
Debug.DeleteAllBreakpoints	Ctrl + Shift + F9	プロジェクトのブレークポイントをすべてクリアします。
Debug.Disassembly	**Ctrl + Alt + D**	[逆アセンブリ] ウィンドウを表示します。
Debug.EnableBreakpoint	Ctrl + F9	ブレークポイントの有効と無効を切り替えます。
Debug.Exceptions	Ctrl + D、Ctrl + E	[例外] ダイアログ ボックスを表示します。
Debug.Immediate	Ctrl + D、Ctrl + I	[イミディエイト] ウィンドウを表示します。このウィンドウでは、式を評価し、各コマンドを実行できます。
Debug.Locals	Ctrl + D、Ctrl + L	[ローカル] ウィンドウを表示して、現在のスタック フレームの各プロシージャの変数およびその値を表示します。

Debug.Memory1	Ctrl + Alt + M、1	[メモリ 1] ウィンドウを表示して、大きなバッファや文字列など、[ウォッチ] ウィンドウや [変数] ウィンドウで明瞭に表示されないデータを表示します。
Debug.Memory2	Ctrl + Alt + M、2	[メモリ 2] ウィンドウを表示して、大きなバッファや文字列など、[ウォッチ] ウィンドウや [変数] ウィンドウで明瞭に表示されないデータを表示します。
Debug.Memory3	Ctrl + Alt + M、3	[メモリ 3] ウィンドウを表示して、大きなバッファや文字列など、[ウォッチ] ウィンドウや [変数] ウィンドウで明瞭に表示されないデータを表示します。
Debug.Memory4	Ctrl + Alt + M、4	[メモリ 4] ウィンドウを表示して、大きなバッファや文字列など、[ウォッチ] ウィンドウや [変数] ウィンドウで明瞭に表示されないデータを表示します。
Debug.Modules	Ctrl + D、Ctrl + M	[モジュール] ウィンドウを表示します。このウィンドウでは、プログラムで使用されている .dll ファイルまたは .exe ファイルを表示できます。マルチプロセス デバッグでは、右クリックして [すべてのプログラムのモジュールを表示] をクリックします。
Debug.Processes	Ctrl + D、Ctrl + P	[プロセス] ウィンドウを表示します。実行モードで使用できます。
Debug.QuickWatch	Ctrl + D、Ctrl + Q	[クイック ウォッチ] ダイアログ ボックスに、選択した式の現在の値を表示します。中断モードだけで使用できます。このコマンドを使用して、ウォッチ式を定義していない変数、プロパティ、またはその他の式の現在の値をチェックします。
Debug.Registers	Ctrl + D、Ctrl + R	[レジスタ] ウィンドウを表示します。このウィンドウには、ネイティブ コード アプリケーションをデバッグするためのレジスタの内容が表示されます。
Debug.Restart	Ctrl + Shift + F5	デバッグ セッションを終了し、リビルドして、最初からもう一度アプリケーションを実行します。中断モードと実行モードで使用できます。
Debug.RunToCursor	Ctrl + F10	中断モードでは、現在のステートメントから選択したステートメントへコードの実行を再開します。実行中の行を示すマージン インジケータがマージン インジケータ バーに表示されます。デザイン モードでは、デバッガを起動し、カーソル位置までコードを実行します。
Debug.ScriptExplorer	**Ctrl + Alt + N**	[スクリプト エクスプローラ] ウィンドウを表示します。このウィンドウには、デバッグ処理中のドキュメント セットが一覧表示されます。実行モードで使用できます。
Debug.SetNextStatement	Ctrl + Shift + F10	選択したコード行から実行されるように設定します。
Debug.ShowNextStatement	Alt + NUM *	次に実行されるステートメントが強調表示されます。
Debug.Start	F5	自動的にデバッガがアタッチされ、[<プロジェクト名> プロパティ] ダイアログ ボックスで指定したスタートアップ プロジェクトからアプリケーションが実行されます。中断モードの場合は、続行に変更します。
Debug.StartWithoutDebugging	Ctrl + F5	デバッガを起動せずにコードを実行します。
Debug.StepInto	F11	関数呼び出しの実行後、一度に 1 ステートメントずつコードを実行します。
Debug.StepIntoCurrentProcess	**Ctrl + Alt + F11**	[プロセス] ウィンドウから使用できます。

Debug.StepOut	Shift + F11	現在の実行ポイントがある関数の、残りの行を実行します。
Debug.StepOutCurrentProcess	Ctrl + Shift + Alt + F11	[プロセス] ウィンドウから使用できます。
Debug.StepOver	F10	コードの次の行を実行しますが、関数呼び出しによる実行は行いません。
Debug.SetpOverCurrentProcess	**Ctrl** + **Alt** + **F10**	[プロセス] ウィンドウから使用できます。
Debug.StopDebugging	Shift + F5	プログラムの現在のアプリケーションの実行を停止します。中断モードと実行モードで使用できます。
Debug.Threads	Ctrl + D、Ctrl + T	[スレッド] ウィンドウに、現在のプロセスのすべてのスレッドと、その情報を表示します。
Debug.ToggleBreakpoint	F9	現在の行のブレークポイントを設定または削除します。
Debug.ToggleDisassembly	Ctrl + D、Ctrl + D	現在のソース ファイルの逆アセンブリ情報を表示します。中断モードだけで使用できます。
Debug.Watch	**Ctrl** + **Alt** + **W**、**1**	[ウォッチ 1] ウィンドウに、選択した変数またはウォッチ式の値を表示します。
Debug.Watch2	**Ctrl** + **Alt** + **W**、**2**	[ウォッチ 2] ウィンドウに、選択した変数またはウォッチ式の値を表示します。
Debug.Watch3	**Ctrl** + **Alt** + **W**、**3**	[ウォッチ 3] ウィンドウに、選択した変数またはウォッチ式の値を表示します。
Debug.Watch4	**Ctrl** + **Alt** + **W**、**4**	[ウォッチ 4] ウィンドウに、選択した変数またはウォッチ式の値を表示します。
DebuggerContextMenus.BreakpointsWindow.Delete	Alt + F9、D	選択したブレークポイントを削除します。[ブレークポイント] ウィンドウだけで使用できます。
DebuggerContextMenus.BreakpointsWindow.GoToDisassembly	Alt + F9、A	[逆アセンブリ] ウィンドウを表示します。[ブレークポイント] ウィンドウだけで使用できます。
DebuggerContextMenus.BreakpointsWindow.GoToSourceCode	**Alt** + **F9**、**S**	コード ファイル内の選択されたブレークポイントの位置に移動します。[ブレークポイント] ウィンドウだけで使用できます。
Tools.AttachToProcess	**Ctrl** + **Alt** + **P**	[プロセスにアタッチ] ダイアログ ボックスを表示します。このダイアログ ボックスでは、1 つのソリューションで複数のプログラムを同時にデバッグできます。

参照

概念

Visual C# 2005 の既定のショートカット キー

データのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、統合開発環境 (IDE: Integrated Development Environment) でデータを扱う場合に使用できます。

メモ :

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Data.Column	Ctrl + L	データ セットの最後に新しい列を追加します。データセット エディタでだけ使用できます。
Data.Execute	**Ctrl** + **Alt** + **F5**	現在アクティブになっているデータベース オブジェクトを実行します。
Data.InsertColumn	Insert	データセットで選択された列の上に新しい列を追加します。データセット エディタでだけ使用できます。
Data.RunSelection	Ctrl + Q	SQL エディタの現在の選択項目を実行します。
Data.ShowDataSources	Shift + Alt + D	[データ ソース] ウィンドウを表示します。
Data.StepInto	Alt + F5	現在アクティブなデータベース オブジェクトについてデバッグ モードに入ります。
QueryDesigner.CancelRetrievingData	Ctrl + T	現在実行中のクエリをキャンセルまたは中断します。クエリおよびビュー デザイナでだけ使用できます。
QueryDesigner.Criteria	Ctrl + 2	クエリおよびビュー デザイナの抽出条件ペインを表示します。クエリおよびビュー デザイナでだけ使用できます。
QueryDesigner.Diagram	Ctrl + 1	クエリおよびビュー デザイナのダイアグラム ペインを表示します。クエリおよびビュー デザイナでだけ使用できます。
QueryDesigner.ExecuteSQL	Ctrl + R	クエリを実行します。クエリおよびビュー デザイナでだけ使用できます。
QueryDesigner.GoToRow	Ctrl + G	結果ペインで、デザイナの一番下にドッキングされているツール ストリップにフォーカスを移動します。クエリおよびビュー デザイナでだけ使用できます。
QueryDesigner.JoinMode	Ctrl + Shift + J	JOIN モードを有効にします。クエリおよびビュー デザイナでだけ使用できます。
QueryDesigner.Results	Ctrl + 4	クエリおよびビュー デザイナの結果ペインを表示します。クエリおよびビュー デザイナでだけ使用できます。
QueryDesigner.SQL	Ctrl + 3	クエリおよびビュー デザイナの SQL ペインを表示します。クエリおよびビュー デザイナでだけ使用できます。
View.Datasets	**Ctrl** + **Alt** + **D**	レポート デザイナの [データセットのレポート] ウィンドウを表示します。

参照

概念

Visual C# 2005 の既定のショートカット キー

テキスト移動のショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、テキスト エディタで開いているドキュメント内を移動するときに使用できます。

メモ :
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Edit.CharLeft	←	カーソルを 1 文字左に移動します。
Edit.CharRight	→	カーソルを 1 文字右に移動します。
Edit.ClearBookmarks	Ctrl + B、Ctrl + C	現在のドキュメントで、名前のないブックマークをすべて削除します。
Edit.DocumentEnd	**Ctrl** + **End**	カーソルをドキュメントの最後の行に移動します。
Edit.DocumentStart	**Ctrl** + **Home**	カーソルをドキュメントの最初の行に移動します。
Edit.GoTo	Ctrl + G	[指定行へジャンプ] ダイアログ ボックスを表示します。
Edit.GoToBrace	Ctrl +]	カーソルをドキュメント内の次の中かっこ ({}) に移動します。
Edit.LineDown	↓	カーソルを 1 行下へ移動します。
Edit.LineEnd	END	カーソルを行末に移動します。
Edit.LineStart	Home	カーソルを行頭に移動します。
Edit.LineUp	↑	カーソルを 1 行上へ移動します。
Edit.NextBookmark	Ctrl + B、Ctrl + N	カーソルを次のブックマーク位置に移動します。
Edit.NextError	Ctrl + Shift + F12	[エラー一覧] ウィンドウ内の次のエラー エントリに移動します。ウィンドウが自動的にスクロールされ、エディタ内のテキストの影響を受けるセクションが表示されます。
Edit.PageDown	PageDown	エディタ ウィンドウを 1 画面下にスクロールします。
Edit.PageUp	PageUp	エディタ ウィンドウを 1 画面上にスクロールします。
Edit.PreviousBookmark	Ctrl + B、Ctrl + P	カーソルを前のブックマーク位置に移動します。
Edit.QuickInfo	**Ctrl** + **K**、**Ctrl** + **I**	現在の言語に基づいて、クイック ヒントを表示します。
Edit.ScrollLineDown	**Ctrl** + ↓	テキストを 1 行下にスクロールします。テキスト エディタでだけ使用できます。

Edit.ScrollLineUp	**Ctrl** + ↑	テキストを 1 行上にスクロールします。テキスト エディタでだけ使用できます。
Edit.ToggleBookmark	**Ctrl** + **K**、**Ctrl** + **K** または Ctrl + B、Ctrl + T	現在の行のブックマークを設定または削除します。
Edit.ViewBottom	Ctrl + PageDown	アクティブなウィンドウの最後の可視行に移動します。
Edit.ViewTop	Ctrl + PageUp	アクティブなウィンドウの最初の可視行に移動します。
Edit.WordNext	Ctrl + →	カーソルを 1 単語右に移動します。
Edit.WordPrevious	Ctrl + ←	カーソルを 1 単語左に移動します。
View.BrowseNext	Ctrl + Shift + 1	項目の定義、宣言、または参照のいずれかのうち、直後にあるものに移動します。[オブジェクト ブラウザおよびクラス ビュー] ウィンドウで使用できます。
View.BrowsePrevious	Ctrl + Shift + 2	項目の定義、宣言、または参照のいずれかのうちで、直前にあるものに移動します。[オブジェクト ブラウザとクラス ビュー] ウィンドウで使用できます。
View.NavigateBackward	Ctrl + 負符号 (-)	前に参照したコード行に移動します。
View.NavigateForward	Ctrl + Shift + 負符号 (-)	次に参照したコード行に移動します。
View.PopBrowseContext	Ctrl + Shift + 8	現在のファイルのコード内で前に呼び出された項目に移動します。
View.ForwardBrowseContext	Ctrl + Shift + 7	現在のファイルのコード内で次に呼び出された項目に移動します。

参照

概念

Visual C# 2005 の既定のショートカット キー

テキスト選択のショートカット キー (Visual C# 2005 スキーム)

次のショートカット キーは、テキスト エディタで開いているドキュメント内のテキストを選択するときに使用できます。

メモ :

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Edit.CharLeftExtend	Shift + ←	カーソルを 1 文字左に移動し、選択範囲を拡張します。
Edit.CharLeftExtendColumn	**Shift** + **Alt** + ←	カーソルを 1 文字左に移動し、列選択範囲を拡張します。
Edit.CharRightExtend	Shift + →	カーソルを 1 文字右に移動し、選択範囲を拡張します。
Edit.CharRightExtendColumn	**Shift** + **Alt** + →	カーソルを 1 文字右に移動し、列選択範囲を拡張します。
Edit.DocumentEndExtend	**Ctrl** + **Shift** + **End**	カーソル位置からドキュメントの最後の行までの文字列を選択します。
Edit.DocumentStartExtend	**Ctrl** + **Shift** + **Home**	カーソル位置からドキュメントの最初の行までの文字列を選択します。
Edit.GoToBraceExtend	**Ctrl** + **Shift** + 右角かっこ (**]**)	カーソルを次の中かっこ ({}) に移動して、選択範囲を拡張します。
Edit.LineDownExtend	Shift + ↓	カーソル位置から、テキスト選択範囲を 1 行下に拡張します。
Edit.LineDownExtendColumn	**Shift** + **Alt** + ↓	ポインタを 1 行下に移動して、列の選択範囲を拡張します。
Edit.LineEndExtend	Shift + End	カーソル位置から現在行の末尾までを選択します。
Edit.LineEndExtendColumn	**Shift** + **Alt** + **End**	カーソルを行末に移動して、列の選択範囲を拡張します。
Edit.LineStartExtend	Shift + Home	カーソル位置から行頭までの文字列を選択します。
Edit.LineStartExtendColumn	**Shift** + **Alt** + **Home**	カーソルを行末に移動して、列の選択範囲を拡張します。
Edit.LineUpExtend	Shift + ↑	カーソル位置から始めて、上方向に文字列を 1 行ずつ選択します。
Edit.LineUpExtendColumn	**Shift** + **Alt** + ↑	カーソルを 1 行上に移動して、列の選択範囲を拡張します。
Edit.PageDownExtend	**Shift** + **PageDown**	選択範囲を 1 ページ下に拡張します。
Edit.PageUpExtend	**Shift** + **PageUp**	選択範囲を 1 ページ上に拡張します。
Edit.SelectAll	Ctrl + A	現在のドキュメントの内容をすべて選択します。
Edit.SelectCurrentWord	**Ctrl** + **Shift** + **W**	カーソルを含む語 (カーソルの右の語の手前まで) を選択します。
Edit.SelectToLastGoBack	Ctrl + 等号 (=)	エディタの現在の位置から前の位置までの範囲を選択します。
Edit.ViewBottomExtend	**Ctrl** + **Shift** + **PageDown**	カーソルをビューの最終行に移動して選択範囲を拡張します。

Edit.ViewTopExtend	**Ctrl** + **Shift** + **PageUp**	選択範囲を現在アクティブなウィンドウの一番上まで拡張します。
Edit.WordNextExtend	**Ctrl** + **Shift** + →	選択範囲を右に 1 単語拡張します。
Edit.WordNextExtendColumn	**Ctrl** + **Shift** + **Alt** + →	カーソルを 1 単語右に移動し、列選択範囲を拡張します。
Edit.WordPreviousExtend	**Ctrl** + **Shift** + ←	選択範囲を左に 1 単語拡張します。
Edit.WordPreviousExtendColumn	**Ctrl** + **Shift** + **Alt** + ←	カーソルを 1 単語左に移動し、列選択範囲を拡張します。

参照

概念

Visual C# 2005 の既定のショートカット キー

テキスト操作のショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、テキスト エディタで開いているドキュメント内のテキストを削除、移動、または書式設定するときに使用できます。

メモ :

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Edit.BreakLine	Enter	改行を挿入します。 **メモ :** HTML デザイナのデザイン ビューなど、一部のエディタでは、コンテキストによって、Enter キーを押したときの動作が異なります。詳細については、使用しているエディタのドキュメントを参照してください。
Edit.CharTranspose	Ctrl + T	カーソルの両側にある文字を入れ替えます。たとえば、AC|BD の場合は、AB|CD になります。テキスト エディタでだけ使用できます。
Edit.CollapseTag	Ctrl + M、Ctrl + T	選択された HTML タグを非表示にし、代わりに省略記号 ([...]) を表示します。省略記号 ([...]) の上にマウス ポインタを置くと、ツールヒントとして完全なタグが表示されます。
Edit.CollapseToDefinitions	**Ctrl** + **M**、**Ctrl** + **O**	プロシージャなどの、コード内に領域を作成するための論理境界を自動的に判断し、その境界を非表示にします。
Edit.CommentSelection	Ctrl + E、Ctrl + C	プログラミング言語に適したコメント構文を使用して、現在のコード行をコメントとしてマークします。
Edit.CompleteWord	Ctrl + K、Ctrl + W	現在の言語に基づいて、単語の入力候補を表示します。
Edit.CopyParameterTip	Ctrl + Shift + Alt + C	IntelliSense によって表示されたパラメータ情報をクリップボードにコピーします。
Edit.DeleteBackwards	BackSpace	カーソルの左側にある 1 文字を削除します。
Edit.DeleteHorizontalWhitespace	Ctrl + E、Ctrl + \	選択範囲に含まれている空白を折りたたみます。範囲が選択されていない場合は、カーソルに隣接した空白を削除します。
Edit.FormatDocument	Ctrl + E、Ctrl + D	[オプション] ダイアログ ボックスの [テキスト エディタ] で、言語の [書式設定] ペインの指定に従って言語のインデントおよび空白の書式化を適用します。
Edit.FormatSelection	Ctrl + E、Ctrl + F	選択したコード行を周囲のコード行に基づいて適切にインデントします。
Edit.GenerateMethodSub	Ctrl + K、Ctrl + M	カーソルが置かれているメソッド呼び出しに対して新しいメソッド宣言を作成します。 詳細については、「[メソッド スタブの生成]」を参照してください。
Edit.HideSelection	**Ctrl** + **M**、**Ctrl** + **H**	選択したテキストを非表示にします。ファイル内の隠し文字列の位置は、シグナル アイコンによって表されます。

Edit.InsertTab	Tab	指定された数の空白を挿入することによってテキスト行にインデントを設定します。
Edit.InsertSnippet	Ctrl + K、Ctrl + X	Insert Code Snippet. 詳細については、「コード スニペット (C#)」を参照してください。
Edit.LineCut	Ctrl + L	選択したすべての行を切り取ってクリップボードに移動します。何も選択されていない場合は、現在の行をクリップボードに移動します。
Edit.LineDelete	**Ctrl** + **Shift** + **L**	選択したすべての行を削除します。何も選択されていない場合は、現在の行を削除します。
Edit.LineOpenAbove	Ctrl + Enter	カーソルの上に空白行を挿入します。
Edit.LineOpenBelow	**Ctrl** + **Shift** + **Enter**	カーソルの下に空白行を挿入します。
Edit.LineTranspose	**Shift** + **Alt** + **T**	カーソルがある行を次の行の下へ移動します。
Edit.ListMembers	Ctrl + J	コードの編集時に、ステートメント入力候補として現在のクラスのメンバー一覧を表示します。
Edit.MakeLowercase	Ctrl + U	選択したテキストを小文字に変更します。
Edit.MakeUppercase	**Ctrl** + **Shift** + **U**	選択したテキストを大文字に変更します。
Edit.OverTypeMode	Insert	挿入モードと上書き入力モードを切り替えます。テキスト エディタで作業しているときにだけ使用できます。
Edit.ParameterInfo	**Ctrl** + **Shift** + **Space**	現在の言語に基づいて、現在のパラメータに関するツールヒントを表示します。HTML デザイナのソース ビューでだけ使用できます。
Edit.PasteParameterTip	**Ctrl** + **Shift** + **Alt** + **P**	以前に IntelliSense からコピーしたパラメータ情報をカーソルが示す位置に貼り付けます。
Edit.StopHidingCurrent	**Ctrl** + **M**、**Ctrl** + **U**	現在選択されている領域のアウトライン情報を削除します。
Edit.StopOutlining	**Ctrl** + **M**、**Ctrl** + **P**	ドキュメント全体からすべてのアウトライン情報を削除します。
Edit.SwapAnchor	Ctrl + E、Ctrl + A	現在の選択項目のアンカとエンドポイントを入れ替えます。
Edit.TabLeft	**Shift** + **Tab**	選択した行を 1 タブ ストップ分左に移動します。
Edit.ToggleAllOutlining	**Ctrl** + **M**、**Ctrl** + **L**	以前に隠し文字としてマークしたすべてのセクションの表示と非表示を切り替えます。
Edit.ToggleOutliningExpansion	**Ctrl** + **M**、**Ctrl** + **M**	現在選択している隠し文字のセクションの表示と非表示を切り替えます。

Edit.ToggleTaskListShortcut	Ctrl + E、Ctrl + T	現在の行のショートカットを設定または削除します。
Edit.ToggleWordWrap	Ctrl + E、Ctrl + W	エディタのワードラップを有効または無効にします。
Edit.UncommentSelection	Ctrl + E、Ctrl + U	現在のコード行からコメント構文を削除します。
Edit.ViewWhiteSpace	Ctrl + E、Ctrl + S または **Ctrl** + **R**、**Ctrl** + **W**	空白とタブの記号の表示/非表示を切り替えます。
Edit.WordDeleteToEnd	Ctrl + Del	カーソルの右側にある語を削除します。
Edit.WordDeleteToStart	Ctrl + Back Space	カーソルの左側にある語を削除します。
Edit.WordTranspose	**Ctrl** + **Shift** + **T**	カーソル位置の両側の単語を入れ替えます。たとえば、|End Sub は Sub End| になります。

参照

概念

Visual C# 2005 の既定のショートカット キー

ウィンドウ管理のショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、ツールやドキュメントのウィンドウの移動、クローズ、またはウィンドウ内を移動するときに使用できます。

メモ :

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
View.FullScreen	Shift + Alt + Enter	全画面表示モードのオンとオフを切り替えます。
View.NavigateBackward	Ctrl + 負符号 (-)	移動履歴にある前のドキュメントまたはウィンドウに戻ります。
View.NavigateForward	Ctrl + 等号 (=)	移動履歴にある次のドキュメントまたはウィンドウに進みます。
Window.ActivateDocumentWindow	Esc	メニューまたはダイアログ ボックスを閉じ、実行中の操作をキャンセルします。または、現在のドキュメント ウィンドウにフォーカスを置きます。
Window.CloseDocumentWindow	Ctrl + F4	現在の MDI 子ウィンドウを閉じます。
Window.CloseToolWindow	Shift + Esc	現在のツール ウィンドウを閉じます。
Window.MoveToNavigationBar	Ctrl + F2	コード ビューまたはサーバー コード ビューでコード エディタを表示しているときに、ポインタをエディタ上部のドロップダウン バーに移動します。
Window.NextDocumentWindow	Ctrl + F6	MDI 子ウィンドウを一度に 1 ウィンドウずつ順番に参照します。
Window.NextDocumentWindowNav	Ctrl + Tab	最初のドキュメント ウィンドウを選択した状態で IDE ナビゲータを表示します。
Window.NextPane	Alt + F6	次のツール ウィンドウに移動します。
Window.NextTab	Ctrl + Page Down	ドキュメントまたはウィンドウ内の次のタブに移動します。
Window.NextToolWindowNav	Alt + F7	最初のツール ウィンドウを選択した状態で IDE ナビゲータを表示します。
Window.PreviousDocumentWindow	Ctrl + Shift + F6	エディタまたはデザイナの前のドキュメントに移動します。
Window.PreviousDocumentWindowNav	Ctrl + Shift + Tab	前のドキュメント ウィンドウを選択した状態で IDE ナビゲータを表示します。
Window.PreviousPane	Shift + Alt + F6	前に選択していたウィンドウに移動します。

Window.PreviousSplitPane	Shift + F6	分割ペイン ビューのドキュメントの前のペインに移動します。
Window.PreviousTab	Ctrl + Page Up	ドキュメントまたはウィンドウ内の前のタブに移動します。
Window.PreviousToolWindowNav	Shift + Alt + F7	前のツール ウィンドウを選択した状態で IDE ナビゲータを表示します。
Window.ShowEzMDIFileList	Ctrl + Alt + ↓	開いているすべてのドキュメントがポップアップ リストに表示されます。

参照

概念

Visual C# 2005 の既定のショートカット キー

統合ヘルプのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、ヘルプ トピックを表示したり、ヘルプ トピックの間を移動したりするときに使用できます。

メモ :

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Help.Contents	Ctrl + F1、Ctrl + C	MSDN に含まれるドキュメントの [目次] ウィンドウを表示します。
Help.DynamicHelp	Ctrl + F1、Ctrl + D	[ダイナミック ヘルプ] ウィンドウを表示します。
Help.F1Help	F1	現在選択されているユーザー インターフェイスに対応するヘルプ トピックを表示します。
Help.HelpFavorites	Ctrl + F1、Ctrl + F	[ヘルプのお気に入り] を表示します。
Help.HowDoI	Ctrl + F1、Ctrl + h	選択されたユーザー設定に対応する [カテゴリから検索] ページを表示します。
Help.Index	Ctrl + F1、I	MSDN に含まれるドキュメントの [キーワード] ウィンドウを表示します。
Help.IndexResults	Ctrl + F1、Ctrl + T	[キーワード検索の結果] ウィンドウを表示します。
Help.Nexttopic	**Alt** + ↓ または Alt + →	目次内の次のトピックを表示します。ヘルプ (Web ブラウザ) のウィンドウでだけ使用できます。
Help.Previoustopic	**Alt** + ↑ または Alt + ←	目次内の前のトピックを表示します。ヘルプ (Web ブラウザ) のウィンドウでだけ使用できます。
Help.Search	Ctrl + F1、Ctrl + S	Visual Studio のヘルプ ページの [検索] タブを表示します。このページでは、MSDN に含まれるドキュメント内の語や文を検索できます。
Help.Searchresults	Ctrl + F1、Ctrl + R	最近の検索で生成されたトピックの一覧をフォーカスした状態で、Visual Studio のヘルプ ページの [検索] タブを表示します。
Help.WindowHelp	Shift + F1	現在のユーザー インターフェイスに対応するヘルプ トピックを表示します。

参照

概念

Visual C# 2005 の既定のショートカット キー
Visual Studio の設定

オブジェクト ブラウザのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、[オブジェクト ブラウザ] ウィンドウで使用できます。

メモ :
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Edit.FindSymbol	Alt + F12	[検索と置換] ダイアログ ボックスの、シンボルの検索ペインを表示します。
Edit.GoToDeclaration	Ctrl + F12	コードで選択したシンボルの定義を表示します。
Edit.GoToDefinition	F12	コードで選択したシンボルの宣言を表示します。
Edit.QuickFindSymbol	Shift + Alt + F12	選択されたオブジェクトまたはメンバをファイルから検索し、結果を [シンボルの検索結果] ウィンドウに表示します。
View.ObjectBrowser	**Ctrl + Alt + J**	オブジェクト ブラウザを表示します。オブジェクト ブラウザでは、パッケージで利用できるクラス、プロパティ、メソッド、イベント、および定数と、プロジェクトのオブジェクト ライブラリやプロシージャを表示できます。

参照

概念

Visual C# 2005 の既定のショートカット キー

マクロのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、マクロの操作時に使用できます。

メモ :
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
View.MacroExplorer	Alt + F8	[マクロ エクスプローラ] ウィンドウを表示します。このウィンドウには、現在のソリューションで使用できるすべてのマクロの一覧が表示されます。
Tools.MacrosIDE	Alt + F11	マクロ IDE、Visual Studio のマクロを起動します。
Tools.RecordTemporaryMacro	**Ctrl** + **Shift** + **R**	Visual Studio IDE をマクロ記録モードにします。
Tools.RunTemporaryMacro	**Ctrl** + **Shift** + **P**	記録したマクロを再生します。

参照

概念

Visual C# 2005 の既定のショートカット キー

ツール ウィンドウのショートカット キー (Visual C# 2005 スキーム)

次のショートカット キーは、特定のツール ウィンドウを表示するときに使用できます。

メモ :

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Tools.CodeSnippetManager	Ctrl + K、Ctrl + B	コード スニペット マネージャ を表示します。コード スニペット マネージャでは、ファイル内でコード スニペットの検索および挿入を実行できます。
View.BookmarkWindow	Ctrl + W、Ctrl + B	[ブックマーク] ウィンドウを表示します。
View.ClassView	Ctrl + W、Ctrl + C	[クラス ビュー] ウィンドウを表示します。
View.ClassViewGoToSearchCombo	Ctrl + K、Ctrl + V	[クラス ビュー 検索] ボックスにフォーカスを設定します。
View.CodeDefinitionWindow	Ctrl + W、Ctrl + D	[コード定義] ウィンドウを表示します。
View.CommandWindow	Ctrl + W、Ctrl + A	[コマンド] ウィンドウを表示します。このウィンドウでは、統合開発環境 (IDE: Integrated Development Environment) を操作するコマンドを入力できます。
View.DocumentOutline	Ctrl + W、Ctrl + U	[ドキュメント アウトライン] ウィンドウを表示して、現在のドキュメントのアウトラインをフラットまたは階層で表示します。
View.ErrorList	Ctrl + W、Ctrl + E	[エラー一覧] ウィンドウを表示します。
View.FindSymbolResults	Ctrl + W、Ctrl + Q	
View.ObjectBrowser	Ctrl + W、Ctrl + J	
View.Output	Ctrl + W、Ctrl + O	[出力] ウィンドウに、実行時のステータス メッセージを表示します。
View.PendingCheckins	Ctrl + W、Ctrl + G	
View.PropertiesWindow	Ctrl + W、Ctrl + P	[プロパティ] ウィンドウを表示します。このウィンドウには、現在選択されているアイテムのデザイン時のプロパティおよびイベントの一覧が表示されます。
View.PropertyPages	Shift + F4	現在選択しているアイテムのプロパティ ページを表示します。

View.ResourceView	Ctrl + W、Ctrl + R	[リソース ビュー] ウィンドウを表示します。
View.ServerExplorer	Ctrl + W、Ctrl + L	サーバー エクスプローラを表示します。このウィンドウでは、データベース サーバー、イベント ログ、メッセージ キュー、Web サービス、およびその他のオペレーティング システム サービスを表示および操作できます。
View.SolutionExplorer	Ctrl + W、Ctrl + S	現在のソリューション内のプロジェクトとファイルの一覧を示すソリューション エクスプローラを表示します。
View.TaskList	Ctrl + W、Ctrl + T	[タスク一覧] ウィンドウを表示します。このウィンドウでは、タスク、コメント、ショートカット、警告、およびエラー メッセージをカスタマイズ、分類、および管理できます。
View.Toolbox	Ctrl + W、Ctrl + X	コードで挿入または使用できるコントロールやその他のアイテムを含む [ツールボックス] を表示します。
View.WebBrowser	Ctrl + W、Ctrl + W	インターネットのページを表示できる [Web ブラウザ] ウィンドウを表示します。

参照

概念

Visual C# 2005 の既定のショートカット キー

プロジェクトのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、プロジェクトへの新規アイテムの追加、プロジェクトのビルド、ファイルのオープン、またはプロジェクトのオープン時に使用できます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Build.BuildSolution	F6	ソリューションをビルドします。
Build.Cancel	**Ctrl** + **Break**	現在のビルドを中止します。
Build.Compile	Ctrl + F7	選択したファイルのマシン語コード、リンカ ディレクティブ、セクション、外部参照、関数名またはデータ名を含むオブジェクト ファイルを作成します。
File.NewFile	Ctrl + N	[新しいファイル] ダイアログ ボックスを表示します。このダイアログ ボックスでは、新しいファイルを選択して現在のプロジェクトに追加できます。
File.NewProject	**Ctrl** + **Shift** + **N**	[新しいプロジェクト] ダイアログ ボックスを表示します。
File.OpenFile	Ctrl + O	[ファイルを開く] ダイアログ ボックスを表示します。
File.OpenProject	**Ctrl** + **Shift** + **O**	[プロジェクトを開く] ダイアログ ボックスを表示します。このダイアログ ボックスでは、既存のプロジェクトをソリューションに追加できます。
Project.AddClass	Shift + Alt + C	[新しい項目の追加] ダイアログ ボックスを表示し、既定としてクラス テンプレートを選択します。
Project.AddExistingItem	**Shift** + **Alt** + **A**	[既存項目の追加] ダイアログ ボックスを表示します。このダイアログ ボックスでは、現在のプロジェクトに既存のファイルを追加できます。
Project.AddNewItem	**Ctrl** + **Shift** + **A**	[新しい項目の追加] ダイアログ ボックスを表示します。このダイアログ ボックスでは、現在のプロジェクトに新しいファイルを追加できます。
Project.Override	**Ctrl** + **Alt** + **Ins**	派生クラスで基本クラスのメソッドをオーバーライドできます。

参照

概念

Visual C# 2005 の既定のショートカット キー

イメージ エディタのショートカット キー (Visual C# 2005 スキーム)

既定でキーに対応付けられているイメージ エディタ コマンド用のショートカット キーは、以下の表のとおりです。ショートカット キーを変更するには、[ツール] メニューの [オプション] をクリックし、[環境] を展開して [キーボード] をクリックします。詳細については、「方法 : ショートカット キーの組み合わせを操作する」を参照してください。

メモ :

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド	キー	説明
Image.AirBrushTool	Ctrl + A	選択したサイズと色のエアブラシを使用して描画します。
Image.BrushTool	Ctrl + B	選択した形、サイズ、および色のブラシを使用して描画します。
Image.CopyAndOutlineSelection	**Ctrl** + **Shift** + **U**	現在の選択領域のコピーを作成し、その外枠を描画します。現在の選択領域に背景色が含まれる場合に透明を選択していると、背景色は解除されます。
Image.DrawOpaque	Ctrl + J	現在の選択領域を不透明または透明にします。
Image.EllipseTool	Ctrl + P	選択した線の幅と色で楕円を描画します。
Image.EraserTool	**Ctrl** + **Shift** + **I**	イメージの一部を現在の背景色を使用して消去します。
Image.FilledEllipseTool	**Ctrl** + **Shift** + **Alt** + **P**	塗りつぶされた楕円を描画します。
Image.FilledRectangleTool	**Ctrl** + **Shift** + **Alt** + **R**	塗りつぶされた四角形を描画します。
Image.FilledRoundRectangleTool	**Ctrl** + **Shift** + **Alt** + **W**	塗りつぶされた角の丸い四角形を描画します。
Image.FillTool	Ctrl + F	領域を塗りつぶします。
Image.FlipHorizontal	**Ctrl** + **H**	イメージまたは選択領域の上下を反転させます。
Image.FlipVertical	**Shift** + **Alt** + **H**	イメージまたは選択領域の左右を反転させます。
Image.LargerBrush	**Ctrl** + **=**	ブラシのサイズを各方向に 1 ピクセルずつ拡大します。ブラシのサイズを小さくする方法については、この表の「Image.SmallerBrush」を参照してください。
Image.LineTool	Ctrl + L	選択した形、サイズ、および色で直線を描画します。
Image.MagnificationTool	**Ctrl** + **M**	[拡大] ツールに切り替えます。これによって、イメージの特定領域を拡大表示できます。
Image.Magnify	**Ctrl** + **Shift** + **M**	現在の拡大率と 1:1 の間で切り替えます。
Image.NewImageType	Insert	[<Device> イメージ タイプの新規作成] ダイアログ ボックスを開きます。これを使用して、さまざまな種類のイメージを作成できます。

Image.NextColor	Ctrl +] または Ctrl + →	描画の前景色を次のパレット カラーに変更します。
Image.NextRightColor	**Ctrl** + **Shift** + 右角かっこ (**]**) または **Shift** + **Ctrl** + →	描画の背景色を次のパレット カラーに変更します。
Image.OutlinedEllipseTool	**Shift** + **Alt** + **P**	塗りつぶされた楕円を外枠付きで描画します。
Image.OutlinedRectangleTool	**Shift** + **Alt** + **R**	塗りつぶされた四角形を外枠付きで描画します。
Image.OutlinedRoundRectangleTool	**Shift** + **Alt** + **W**	塗りつぶされた角の丸い四角形を外枠付きで描画します。
Image.PencilTool	**Ctrl** + **I**	1 ピクセルのペンシルを使用して描画します。
Image.PreviousColor	**Ctrl** + **[** または Ctrl + ←	描画の前景色を前のパレット カラーに変更します。
Image.PreviousRightColor	**Ctrl** + **Shift** + **[** または **Shift** + **Ctrl** + ←	描画の背景色を前のパレット カラーに変更します。
Image.RectangleSelectionTool	**Shift** + **Alt** + **S**	移動、コピー、または編集するイメージの一部を四角形として選択します。
Image.RectangleTool	**ATL** + **R**	選択した線の幅と色で四角形を描画します。
Image.Rotate90Degrees	**Ctrl** + **Shift** + **H**	イメージまたは選択領域を 90 度回転させます。
Image.RoundedRectangleTool	**Alt** + **W**	選択した線の幅と色で角の丸い四角形を描画します。
Image.ShowGrid	**Ctrl** + **Alt** + **S**	ピクセル グリッドを切り替えます。つまり、[グリッドの設定] ダイアログ ボックスの [ピクセル表示] チェック ボックスをオンまたはオフにします。
Image.ShowTileGrid	**Ctrl** + **Shift** + **Alt** + **S**	タイル グリッドを切り替えます。つまり、[グリッドの設定] ダイアログ ボックスの [タイル表示] チェックボックスをオンまたはオフにします。
Image.SmallBrush	**Ctrl** + .(ピリオド)	ブラシのサイズを 1 ピクセルに縮小します。この表の「Image.LargerBrush」と「Image.SmallerBrush」も参照してください。
Image.SmallerBrush	Ctrl + - (マイナス)	ブラシのサイズを各方向に 1 ピクセルずつ縮小します。ブラシを元のサイズに戻す方法については、この表の「Image.LargerBrush」を参照してください。

Image.TextTool	Ctrl + T	[テキスト ツール] ダイアログ ボックスを開きます。
Image.UseSelectionAsBrush	Ctrl + U	現在の選択項目をブラシとして使用して描画します。
Image.ZoomIn	**Ctrl** + **Shift** + **.**(ピリオド) または **Ctrl** + ↑	現在のビューの拡大率を上げます。
Image.ZoomOut	**Ctrl** + **,** (カンマ) または **Ctrl** + ↓	現在のビューの拡大率を下げます。

マネージ プロジェクトにリソースを追加する方法については、『.NET Framework 開発者ガイド』の「アプリケーションのリソース」を参照してください。マネージ プロジェクトにリソース ファイルを手動で追加する方法、リソースへのアクセス方法、静的なリソースの表示方法、およびリソース文字列をプロパティに割り当てる方法については、「チュートリアル : Windows フォームのローカリゼーション」および「チュートリアル : ASP.NET でのローカリゼーションのためのリソースの使用」を参照してください。

必要条件

なし

参照

関連項目

イメージ エディタ

概念

Visual C# 2005 の既定のショートカット キー

ダイアログ エディタのショートカット キー (Visual C# 2005 スキーム)

次の表に、ダイアログ エディタ コマンドの既定のキーボード ショートカット キーを示します。ショートカット キーを変更するには、[ツール] メニューの [オプション] をクリックし、[環境] を展開して [キーボード] をクリックします。詳細については、「方法 : ショートカット キーの組み合わせを操作する」を参照してください。

メモ :
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド	キー	説明
Format.AlignBottoms	**Ctrl** + **Shift** + ↓	選択したコントロールの下端を最も優先度の高いコントロールに合わせます。
Format.AlignCenters	Shift + F9	選択したコントロールの垂直方向の中央を最も優先度の高いコントロールに合わせます。
Format.AlignLefts	**Ctrl** + **Shift** + ←	選択したコントロールの左端を最も優先度の高いコントロールに合わせます。
Format.AlignMiddles	F9	選択したコントロールの水平方向の中央を最も優先度の高いコントロールに合わせます。
Format.AlignRights	**Ctrl** + **Shift** + →	選択したコントロールの右端を最も優先度の高いコントロールに合わせます。
Format.AlignTops	**Ctrl** + **Shift** + ↑	選択したコントロールの上端を最も優先度の高いコントロールに合わせます。
Format.ButtonBottom	Ctrl + B	選択したボタンをダイアログ ボックスの下部中央に合わせて配置します。
Format.ButtonRight	Ctrl + R	選択したボタンをダイアログ ボックスの右上隅に配置します。
Format.CenterHorizontal	Ctrl + Shift + F9	コントロールをダイアログ ボックス内で水平方向に中央揃えします。
Format.CenterVertical	Ctrl + F9	コントロールをダイアログ ボックス内で垂直方向に中央揃えします。
Format.CheckMnemonics	**Ctrl** + **M**	ニーモニックの一意性をチェックします。
Format.SizeToContent	Shift + F7	選択したコントロールのサイズをキャプションのテキストが収まるように変更します。
Format.SpaceAcross	Alt + →	選択したコントロールを水平方向に等間隔で配置します。
Format.SpaceDown	**Alt** + ↓	選択したコントロールを垂直方向に等間隔で配置します。
Format.TabOrder	Ctrl + D	ダイアログ ボックス内のコントロールの順序を設定します。
Format.TestDialog	Ctrl + T	ダイアログ ボックスを実行し、外観と動作をテストします。
Format.ToggleGuides	Ctrl + G	ダイアログ ボックス編集用に、グリッドなし、ガイドライン、グリッドの間で順番に切り替えます。

マネージ プロジェクトにリソースを追加する方法については、『.NET Framework 開発者ガイド』の「アプリケーションのリソース」を参照してください。マネージ プロジェクトにリソース ファイルを手動で追加する方法、リソースへのアクセス方法、静的なリソースの表示方法、およびリソース文字列をプロパティに割り当てる方法については、「チュートリアル : Windows フォームのローカリゼーション」および「チュートリアル : ASP.NET でのローカリゼーションのためのリソースの使用」を参照してください。

参照

関連項目

ダイアログ エディタ

概念

Visual C# 2005 の既定のショートカット キー

検索と置換のショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーを使用して、単一のファイルまたは複数のファイルからテキストを検索したり、オブジェクトやメンバを検索したりできます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Edit.Find	Ctrl + F	[検索と置換] ダイアログ ボックスの [クイック検索] タブを表示します。
Edit.FindAllReferences	Ctrl + K、Ctrl + R	すべてのシンボル リファレンスの検索場所の一覧を表示します。
Edit.FindInFiles	**Ctrl + Shift + F**	[検索と置換] ダイアログ ボックスの [フォルダを指定して検索] タブを表示します。
Edit.FindNext	F3	前の検索テキストの、次の出現箇所を検索します。
Edit.FindNextSelected	Ctrl + F3	ドキュメントで現在選択しているテキストの、次の出現箇所を検索します。
Edit.FindPrevious	Shift + F3	検索テキストの、前の出現箇所を検索します。
Edit.FindPreviousSelected	Ctrl + Shift + F3	現在選択しているテキストまたはカーソル位置にある単語の、前の出現箇所を検索します。
Edit.GoToFindCombo	**Ctrl + /**	[標準] ツール バーの [検索/コマンド] ボックスに挿入ポインタを配置します。
Edit.IncrementalSearch	**Ctrl + I**	インクリメンタル検索を開始します。インクリメンタル検索の開始時に文字がまだ入力されていない場合は、前のパターンが再度呼び出されます。テキストが見つかった場合は、次の出現箇所を検索します。
Edit.Replace	**Ctrl + H**	[検索と置換] ダイアログ ボックスの [クイック置換] タブに置換オプションを表示します。
Edit.ReplaceInFiles	**Ctrl + Shift + H**	[検索と置換] ダイアログ ボックスの [フォルダを指定して検索] タブに置換オプションを表示します。
Edit.ReverseIncrementalSearch	**Ctrl + Shift + I**	インクリメンタル検索の検索方向を変更して、ファイルの末尾から先頭に向けて検索します。
Edit.StopSearch	**Alt + F3、S**	現在の [ファイル内の検索] 操作を停止します。
View.FindSymbolResults	Ctrl + W、Ctrl + Q	[シンボルの検索結果] ウィンドウを表示します。このウィンドウにシンボルの検索結果が表示されます。

参照

概念

Visual C# 2005 の既定のショートカット キー

リファクタリングのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーで、リファクタリング 操作を実行できます。

メモ：
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Refactor.EncapsulateField	Ctrl + R、Ctrl + E	[フィールドのカプセル化] ダイアログ ボックス を表示します。このダイアログ ボックスでは、既存のフィールドからプロパティを作成し、新規プロパティへの参照でコードを更新できます。
Refactor.ExtractInterface	Ctrl + R、Ctrl + I	[インターフェイスの展開] ダイアログ ボックス を表示します。このダイアログ ボックスでは、既存のクラス、構造体、またはインターフェイスから派生したメンバを使用して新しいインターフェイスを作成できます。
Refactor.ExtractMethod	Ctrl + R、Ctrl + M	[メソッドの展開] ダイアログ ボックス を表示します。このダイアログ ボックスでは、既存のメソッドのコードの一部から新しいメソッドを作成できます。
Refactor.PromoteLocalVariabletoParameter	**Ctrl** + **R**、**Ctrl** + **P**	ローカルの変数をメソッド、インデクサ、またはコンストラクタ パラメータに移動し、呼び出しサイトを正しく更新します。詳細については、「ローカル変数をパラメータへ昇格」を参照してください。
Refactor.RemoveParameters	Ctrl + R、Ctrl + V	[リモート パラメータ] ダイアログ ボックスを表示します。このダイアログ ボックスでは、メンバが呼び出された位置で宣言を変更して、メソッド、インデクサ、またはデリゲートからパラメータを削除できます。詳細については、「パラメータの削除」を参照してください。
Refactor.Rename	F2 または **Ctrl** + **R**、**Ctrl** + **R**	[名前の変更] ダイアログ ボックス を表示します。このダイアログ ボックスでは、コード内でシンボル (フィールド、ローカル変数、メソッド、名前空間、プロパティ、型など) に付けられた識別子の名前を変更できます。
Refactor.ReorderParameters	Ctrl + R、Ctrl + O	[パラメータ順序の再変更] ダイアログ ボックス を表示します。このダイアログ ボックスでは、メソッド、インデクサ、およびデリゲートのパラメータの順序を変更できます。

参照

概念

Visual C# 2005 の既定のショートカット キー

マネージ リソース エディタのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、マネージ リソース エディタで編集作業を行っているときにだけ使用できます。詳細については、「[リソース] ページ (プロジェクト デザイナ)」を参照してください。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Edit.EditCell	F2	[その他] ビューと [文字列] ビューの選択されたセルで編集モードに切り替えます。
Edit.Remove	Delete	[ファイル] ビュー、[イメージ] ビュー、[アイコン] ビュー、および [オーディオ] ビューの選択されたファイルを削除します。
Edit.RemoveRow	Ctrl + Del	[その他] ビューと [文字列] ビューの選択された行を削除します。
Resources.Audio	Ctrl + 4	マネージ リソース エディタを [オーディオ] ビューに切り替え、現在のプロジェクトのサウンド ファイルを表示します。.wav、.wma、.mp3 の各形式を含むサウンド ファイルを表示します。
Resources.Files	Ctrl + 5	マネージ リソース エディタを [ファイル] ビューに切り替え、その他のビューでは見つからないファイルを表示します。
Resources.Icons	Ctrl + 3	マネージ リソース エディタを [アイコン] ビューに切り替え、現在のプロジェクトのアイコン (*.ico) ファイルを表示します。
Resources.Images	Ctrl + 2	マネージ リソース エディタを [イメージ] ビューに切り替え、現在のプロジェクトのイメージ ファイルを表示します。.bmp、.jpg、.gif の各形式を含むイメージ ファイルを表示します。
Resources.Other	Ctrl + 6	マネージ リソース エディタを [その他] ビューに切り替え、文字列のシリアル化をサポートするその他の型を追加するため、設定グリッドを表示します。
Resources.Strings	Ctrl + 1	マネージ リソース エディタを [文字列] ビューに切り替え、グリッド内の文字列と、文字列リソースの名前、値、およびコメントの列を表示します。

参照

概念

Visual C# 2005 の既定のショートカット キー
.Resx ファイル形式のリソース

コード スニペットのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、コード スニペットを操作するときに使用できます。

メモ：
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Edit.InsertSnippet	Ctrl + K、Ctrl + X	コード スニペット ピッカー を表示します。コード スニペット ピッカーでは、IntelliSense を使用してスニペットを選択して、カーソル位置に挿入できます。
Edit.SurroundWith	Ctrl + K、Ctrl + S	コード スニペット ピッカー を表示します。コード スニペット ピッカーでは、IntelliSense を使用してスニペットを選択して、選択されたテキストの周囲のスニペットをラップできます。
Tools.CodeSnippetsManager	Ctrl + K、Ctrl + B	コード スニペット マネージャ を表示します。コード スニペット マネージャでは、ファイル内でコード スニペットの検索および挿入を実行できます。

参照

概念

Visual C# 2005 の既定のショートカット キー

クラス ダイアグラムのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーの組み合わせは、クラス ダイアグラムの操作時に限り使用できます。

メモ：
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
ClassDiagram.Collapse	Num + - (負符号)	[クラスの詳細情報] ウィンドウの展開されたノードを縮小します。または、ダイアグラム内の選択されたシェイプ コンパートメントを縮小します。
ClassDiagram.Expand	Num + + (正符号)	[クラスの詳細情報] ウィンドウの縮小されたノードを展開します。または、ダイアグラム内の選択されたシェイプ コンパートメントを展開します。
Edit.Delete	Ctrl + Del	クラス ダイアグラムから選択された項目を削除します。
Edit.ExpandCollapseBaseTypeList	Shift + Alt + B	選択されたシェイプ コンパートメント内の基本型を展開または縮小します。 たとえば、Interface1 が Interface2、Interface3、および Interface4 を継承する場合、Interface1 のシェイプ コンパートメントには親インターフェイスが一覧表示されます。このコマンドでは、継承されたインターフェイスの一覧を縮小し、概要情報として、Interface1 によって継承された基本インターフェイスの数だけを表示できます。
Edit.NavigateToLollipop	Shift + Alt + L	シェイプ コンパートメントのロリポップ インターフェイスを選択します。ロリポップは、1 つ以上のインターフェイスを実装するシェイプ上に表示されます。
Edit.RemoveFromDiagram	Delete	選択されたシェイプ コンパートメントをダイアグラムから削除します。
View.ViewCode	Enter または F7	選択された項目に対応するファイルを開き、正しい位置に挿入ポイントを表示します。

参照

概念

Visual C# 2005 の既定のショートカット キー

その他の技術情報

クラス ダイアグラムの使用

ブックマーク ウィンドウのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、[ブックマーク] ウィンドウ またはエディタで、ブックマークを操作しているときだけ使用できます。

メモ：
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド名	ショートカット キー	説明
Edit.ClearBookmarks	Ctrl + B、Ctrl + C	ドキュメント内のすべてのブックマークを削除します。
Edit.EnableBookmark	Ctrl + B、Ctrl + E	現在のドキュメント内でブックマークを使用できるようにします。
Edit.NextBookmark	Ctrl + B、Ctrl + N	ドキュメント内の次のブックマークに移動します。
Edit.PreviousBookmark	Ctrl + B、Ctrl + P	前のブックマークに移動します。
Edit.ToggleBoomark	Ctrl + B、Ctrl + T	ドキュメント内の現在の行でブックマークを切り替えます。
View.BookmarkWindow	Ctrl + W、Ctrl + B	[ブックマーク] ウィンドウを表示します。

参照

概念

Visual C# 2005 の既定のショートカット キー

アクセラレータ エディタおよびストリング エディタのショートカット キー (Visual C# 2005 スキーム)

以下のショートカット キーは、アクセラレータ エディタまたはストリング エディタで使用します。

メモ：
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド	ショートカット キー	説明
Edit.NewAccelerator	Insert	キーボード ショートカットの新しいエントリを追加します。アクセラレータ エディタだけで使用できます。
Edit.NewString	Insert	文字列テーブルに新規エントリを追加します。ストリング エディタでだけ使用できます。
Edit.NextKeyTyped	**Ctrl** + **W**	キーボード ショートカットとして使用するキーを押すように求める [次のキーのキャプチャ] メッセージ ボックスを表示します。アクセラレータ エディタだけで使用できます。

参照

概念

Visual C# 2005 の既定のショートカット キー

Visual C# への移行

ここでは、他のプログラミング言語から C# に移行する場合の構文と概念を紹介します。また、Java 言語のソースを C# のソース コードに変換するときに利用できる Java Language Conversion Assistant の参照ドキュメントも紹介します。

このセクションの内容

Java 経験者が C# で開発する場合

C# 言語の構文および構成要素を Java 言語と比較します。

Java アプリケーションの Visual C# への変換

Java Language Conversion Assistant (Java プロジェクトを Visual C# に移植するツール) について説明します。

C++ 経験者が C# で開発する場合

C# 言語の機能を C++ 言語と比較します。

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C#

Visual C# について

Java 経験者が C# で開発する場合

このセクションのトピックでは、C# 言語および .NET Framework の概要について説明します。

このセクションの内容

- Java 開発者のための C# プログラミング言語
- Java 開発者のための C# コード例
- Java 開発者のための C# アプリケーションの種類の説明

参照

概念

Java Language Conversion Assistant

その他の技術情報

Visual C# について

Java 開発者のための C# プログラミング言語

ここでは、C# プログラミング言語と Java プログラミング言語との類似点と相違点について説明します。

このセクションの内容

ソース ファイルの名前付け規則 (C# と Java の比較)

データ型 (C# と Java の比較)

演算子 (C# と Java の比較)

フロー制御 (C# と Java の比較)

ループ ステートメント (C# と Java の比較)

クラスの基本事項 (C# と Java の比較)

Main () およびその他のメソッド (C# と Java の比較)

不特定の数のパラメータの使用 (C# と Java の比較)

プロパティ (C# と Java の比較)

構造体 (C# と Java の比較)

配列

継承と派生クラス (C# と Java の比較)

イベント

演算子のオーバーロード (C# と Java の比較)

例外 (C# と Java の比較)

C# の高度な手法 (C# と Java の比較)

ガベージ コレクション (C# と Java の比較)

安全なコードと安全でないコード (C# と Java の比較)

概要 (C# と Java の比較)

関連するセクション

C# プログラミング ガイド

Visual C#

Visual C# への移行

Java 開発者のための C# プログラミング言語

ソース ファイルの名前付け規則 (C# と Java の比較)

C# のクラスを含むファイルの名前付け規則は、Java とわずかに異なります。Java では、すべてのソース ファイルに .java 拡張子が付きます。各ソース ファイルは、トップレベルのパブリック クラス宣言を 1 つ含み、クラス名とファイル名を一致させる必要があります。言い換えると、たとえば、パブリック スコープで宣言された **Customer** というクラスは、Customer.java という名前のソース ファイルで定義する必要があります。

C# の場合、ソース コードは .cs 拡張子で示されます。Java と違って、ソース ファイルには、トップレベルのパブリック クラス宣言を複数含めることができ、ファイル名とクラス名を一致させる必要がありません。

トップレベルの宣言

Java でも C# でも、ソース コードは、一定の順序で並べられたいくつかのトップレベルの宣言で始まります。Java と C# のプログラムで行われる宣言には、ごくわずかな違いしかありません。

Java でのトップレベルの宣言

Java では、**package** キーワードを使ってクラスをグループ化できます。パッケージ化されたクラスは、ソース ファイルの最初の実行可能な行で **package** キーワードを使用する必要があります。この行の後に、他のパッケージのクラスへのアクセスに必要な import ステートメントとクラス宣言が次のように続きます。

```
package Acme;
import java.io.*;
class Customer
{
    ...
}
```

C# でのトップレベルの宣言

C# では、名前空間という概念を使用して、論理的に関連するクラスを namespace キーワードでグループ化します。これらのクラスは、Java のパッケージと同じように機能し、同じ名前のクラスが 2 つの別々の名前空間に表示されることがあります。現在の名前空間の外部の名前空間で定義されたクラスにアクセスするには、using ディレクティブと名前空間の名前を次のように使用します。

C#

```
using System.IO;

namespace Acme
{
    class Customer
    {
        // ...
    }
}
```

using ディレクティブは、名前空間の宣言の中に配置できますが、その場合、インポートされる名前空間は、それを格納する名前空間の一部になります。

Java では、同じ 1 つのソース ファイルで複数のパッケージを使用できませんが、C# では、次のように 1 つの .cs ファイルで複数の名前空間を使用できます。

C#

```
namespace AcmeAccounting
{
    public class GetDetails
    {
        // ...
    }
}

namespace AcmeFinance
{
    public class ShowDetails
    {
```

```
            // ...
        }
    }
```

完全修飾名と名前空間のエイリアス

Java の場合と同様に、DataSet や前の例の `AcmeAccounting.GetDetails` などのクラスの完全修飾名を指定すると、名前空間の `using` 参照を使わずに .NET Framework 内のクラスにも、ユーザー定義の名前空間のクラスにもアクセスできます。

完全修飾名は非常に長くなることがありますが、そのような場合は、**using** キーワードを使用して短い名前 (エイリアス) を指定すると、コードを読みやすくできます。

次のコードでは、架空の企業で記述されたコードを参照するエイリアスを作成しています。

C#

```
using DataTier = Acme.SQLCode.Client;

class OutputSales
{
    static void Main()
    {
        int sales = DataTier.GetSales("January");
        System.Console.WriteLine("January's Sales: {0}", sales);
    }
}
```

WriteLine の構文で、書式文字列内の {x} の x は、その位置に挿入される値の、引数リスト内での位置を示します。たとえば、`GetSales` メソッドが 500 を返したとすると、このアプリケーションの出力は次のようになります。

```
January's Sales: 500
```

プリプロセッサ ディレクティブ

C や C++ と同様に C# にも、ソース ファイルの特定部分を条件付きで省略したり、エラー状態や警告状態を通知したり、ソース コードの異なる領域を線引きしたりするプリプロセッサ ディレクティブがあります。C# には、別個のプリプロセス手順が存在しないため、C 言語や C++ 言語に合わせて "プリプロセッサ ディレクティブ" という用語を使用しています。詳細については、「C# プリプロセッサ ディレクティブ」を参照してください。

参照

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

データ型 (C# と Java の比較)

ここでは、データの表現、割り当て、およびガベージ コレクションに関して、Java と C# の主な類似点と相違点を説明します。

複合データ型

クラスという概念は Java でも C# でも、フィールド、メソッド、およびイベントを含む複合データ型を意味します (クラスの継承については、別途「継承と派生クラス (C# と Java の比較)」で説明します)。C# では、継承をサポートしないスタック割り当て複合データ型として構造体の概念を導入しています。その他のほとんどの点で、構造体はクラスによく似ています。構造体を使用すると、関連するフィールドとメソッドを負荷の小さい方法でグループ化し、これらを小さなループやパフォーマンスが重視されるその他のシナリオで使用できます。

C# では、クラスのインスタンスのガベージ コレクションを実行する前に呼び出すデストラクタ メソッドを作成できます。Java では、**finalize** メソッドを使用して、オブジェクトのガベージ コレクションを実行する前にリソースをクリーンアップするコードを含めることができます。C# の場合、この機能は、クラス デストラクタによって実行されます。デスクトラクタは、引数を持たない、ティルダ文字 (~) で始まるコンストラクタによく似ています。

組み込みのデータ型

C# では、Java で使用できるすべてのデータ型を使用でき、また符号なしの数値と新しい 128 ビット高精度浮動小数点型も使用できます。

Java の各プリミティブ データ型では、コア クラス ライブラリが、Java オブジェクトとして表現されるラッパー クラスを提供します。たとえば、Int32 クラスは、**int** データ型をラップし、Double クラスは、double データ型をラップします。

一方、C# のプリミティブ データ型はいずれも System 名前空間のオブジェクトです。各データ型には、短い名前 (エイリアス) が与えられます。たとえば、**int** は **System.Int32** の短い名前であり、**double** は **System.Double** の短い名前です。

C# のデータ型とそれぞれのエイリアスの一覧を次の表に示します。最初の 8 つのデータ型は、Java で使用できるプリミティブ型に対応します。ただし、Java の **boolean** は、C# では **bool** と呼ばれます。

短い名前	.NET クラス	型	幅	範囲 (ビット)
byte	Byte	符号なし整数	8	0 ～ 255
sbyte	SByte	符号付き整数	8	-128 ～ 127
int	**Int32**	符号付き整数	32	-2,147,483,648 ～ 2,147,483,647
uint	UInt32	符号なし整数	32	0 ～ 4294967295
short	Int16	符号付き整数	16	-32,768 ～ 32,767
ushort	UInt16	符号なし整数	16	0 ～ 65535
long	Int64	符号付き整数	64	-9223372036854775808 ～ 9223372036854775807
ulong	UInt64	符号なし整数	64	0 ～ 18446744073709551615
float	Single	単精度浮動小数点型	32	-3.402823e38 ～ 3.402823e38
double	**Double**	倍精度浮動小数点型	64	-1.79769313486232e308 ～ 1.79769313486232e308
char	Char	単一 Unicode 文字	16	テキストで使用される Unicode 記号
bool	Boolean	論理ブール型	8	true または false
object	Object	他のすべての型の基本型		

string	String	文字列		
decimal	Decimal	29 の有効桁数で 10 進数を表現できる正確な小数または整数型	128	±1.0 × 10e − 28 ～ ±7.9 × 10e28

C# では、すべてのプリミティブ データ型がオブジェクトとして表現されるので、プリミティブ データ型でオブジェクト メソッドを呼び出すことができます。以下にサンプルを示します。

C#

```
static void Main()
{
    int i = 10;
    object o = i;
    System.Console.WriteLine(o.ToString());
}
```

この処理は、自動ボックス化およびボックス化解除によって達成されます。詳細については、
「ボックス化とボックス化解除 (C# プログラミング ガイド)」を参照してください。

定数

Java でも C# でも変数を宣言できます。変数の値はコンパイル時に指定し、実行時には変更できません。Java では **final** フィールド修飾子を使用してこのような変数を宣言しますが、C# では const キーワードを使用します。**const** に加えて、C# では、readonly キーワードを使用して、実行時に値を 1 回代入できる変数を、宣言ステートメントまたはコンストラクタで宣言することもできます。初期化後、**readonly** 変数の値は変更できません。**readonly** 変数が役立つシナリオとしては、別個にコンパイルされた複数のモジュールでバージョン番号などのデータを共有する必要があるような場合が挙げられます。たとえば、モジュール A を新しいバージョン番号で更新して再コンパイルすると、モジュール B を、再コンパイルの必要なしに同じ新しい定数値で初期化できます。

列挙型

列挙型は、C や C++ で使用する場合と同様に名前付き定数をグループ化するために使用します。列挙型は、Java では使用できません。簡単な `Color` 列挙型を定義する例を以下に示します。

C#

```
public enum Color
{
    Green,   //defaults to 0
    Orange,  //defaults to 1
    Red,     //defaults to 2
    Blue     //defaults to 3
}
```

次の列挙型宣言に示すように、列挙型には整数値も割り当てることができます。

C#

```
public enum Color2
{
    Green = 10,
    Orange = 20,
    Red = 30,
    Blue = 40
}
```

Enum 型の GetNames メソッドを呼び出して、列挙型として使用できる定数を表示するコード例を以下に示します。定数を表示した後、値を列挙型に割り当てて表示します。

C#

```
class TestEnums
{
```

```
    static void Main()
    {
        System.Console.WriteLine("Possible color choices: ");

        //Enum.GetNames returns a string array of named constants for the enum.
        foreach(string s in System.Enum.GetNames(typeof(Color)))
        {
            System.Console.WriteLine(s);
        }

        Color favorite = Color.Blue;

        System.Console.WriteLine("Favorite Color is {0}", favorite);
        System.Console.WriteLine("Favorite Color value is {0}", (int) favorite);
    }
}
```

出力

```
Possible color choices:

Green

Orange

Red

Blue

Favorite Color is Blue

Favorite Color value is 3
```

文字列

Java と C# の文字列型は、多少の違いはありますが、基本的に同じように動作します。文字列型は Java でも C# でも不変であり、文字列を作成した後に文字列の値を変更できません。どちらの言語でも、文字列の内容自体を変更するように見えるメソッドは、実際には返す新しい文字列を作成し、元の文字列は変更しません。文字列値を比較するプロセスは、C# と Java で異なります。Java で文字列を比較する場合、== 演算子は、既定では参照型を比較するので、文字列型で **equals** メソッドを呼び出す必要があります。C# では、== または **!=** 演算子を使用して、文字列値を直接比較できます。文字列は C# では参照型ですが、== 演算子と **!=** 演算子は、参照ではなく文字列値を既定で比較します。

C# で文字列を連結する場合は、Java の場合と同様に、文字列を連結するたびに新しい文字列クラスを作成するというオーバーヘッドを回避するために文字列型を使用しないでください。代わりに、StringBuilder クラスを使用してください。このクラスは、Java の **StringBuffer** クラスと同じように機能します。

リテラル文字列

C# では、文字列定数内でエスケープ シーケンス (タブ記号を表す "\t" や円記号を表す "\" など) の使用を避けることができます。エスケープ シーケンスを使用しない場合は、文字列値を割り当てる前に、@ 記号を使って逐語的文字列を宣言します。エスケープ文字を使用する方法とリテラル文字列を割り当てる方法を次の例に示します。

C#

```
static void Main()
{
    //Using escaped characters:
    string path1 = "\\\\FileShare\\Directory\\file.txt";
    System.Console.WriteLine(path1);

    //Using String Literals:
    string path2 = @"\\FileShare\Directory\file.txt";
    System.Console.WriteLine(path2);
}
```

変換とキャスト

データ型の自動変換とキャストでは、Java も C# も同様の規則に従います。

Java と同様に C# も、暗黙の型変換と明示的な型変換の両方をサポートします。拡大変換の場合は、暗黙の変換です。たとえば、次のような **int** から **long** への変換は、Java の場合と同様に暗黙の変換です。

C#

```
int int1 = 5;
long long1 = int1;  //implicit conversion
```

.NET Framework データ型間の暗黙の型変換の一覧を次に示します。

変換元の型	変換先の型
Byte	short、ushort、int、uint、long、ulong、float、double、decimal
SByte	short、int、long、float、double、decimal
Int	long、float、double、decimal
Uint	long、ulong、float、double、decimal
Short	int、long、float、double、decimal
Ushort	int、uint、long、ulong、float、double、decimal
Long	float、double、decimal
Ulong	float、double、decimal
Float	double
Char	ushort、int、uint、long、ulong、float、double、decimal

明示的に変換する式は、次のように Java と同じ構文を使ってキャストします。

C#

```
long long2 = 5483;
int int2 = (int)long2;  //explicit conversion
```

明示的な変換を次の表に示します。

変換元の型	変換先の型
Byte	sbyte または char
SByte	byte、ushort、uint、ulong、char
Int	sbyte、byte、short、ushort、uint、ulong、char
Uint	sbyte、byte、short、ushort、int、char
Short	sbyte、byte、ushort、uint、ulong、char
Ushort	sbyte、byte、short、char
Long	sbyte、byte、short、ushort、int、uint、ulong、char

Ulong	sbyte、byte、short、ushort、int、uint、long、char
Float	sbyte、byte、short、ushort、int、uint、long、ulong、char、decimal
Double	sbyte、byte、short、ushort、int、uint、long、ulong、char、float、decimal
Char	sbyte、byte、short
Decimal	sbyte、byte、short、ushort、int、uint、long、ulong、char、float、double

値型と参照型

C# では、次の 2 種類の変数型を使用できます。

- 値型

  char、int、float などの組み込みのプリミティブ データ型と、構造体で宣言されるユーザー定義型があります。

- 参照型

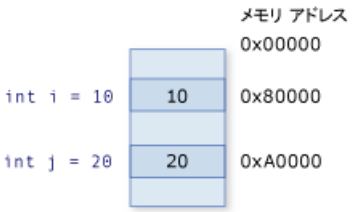
  クラス、およびプリミティブ型から作成されるその他の複合データ型。参照型の変数には、型のインスタンスは含まれず、インスタンスへの参照だけが含まれます。

i と j の 2 つの値型変数を次のように作成した場合、i と j は互いに独立した別個の変数になります。

C#

```
int i = 10;
int j = 20;
```

これらの変数には、それぞれ別々のメモリ位置が割り当てられます。



これら 2 つの変数のうち一方の値を変更しても、もう一方の値には影響しません。たとえば、次のような式が存在する場合でも、変数どうしは結び付けられません。

C#

```
int k = i;
```

つまり、i の値を変更しても、k の値は、i の最初に割り当てられた値のまま変わりません。

C#

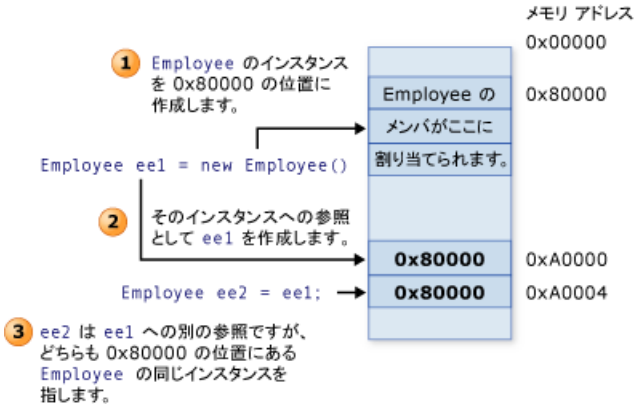
```
i = 30;

System.Console.WriteLine(i.ToString());  // 30
System.Console.WriteLine(k.ToString());  // 10
```

ただし、参照型の動作は異なります。たとえば、2 つの変数を次のように宣言したとします。

C#

```
Employee ee1 = new Employee();
Employee ee2 = ee1;
```

この場合、C# ではクラスは参照型なので、ee1 は、Employee への参照として認識されます。この 2 行のうちの最初の行によって、Employee のインスタンスがメモリに作成され、そのインスタンスへの参照として ee1 が設定されます。そのため、ee2 に ee1 を代入すると、b には、メモリ内にクラスへの参照の複製が格納されます。これら 2 つの変数は、次に示すようにメモリ内の同じオブジェクトを指すため、ee2 でプロパティを変更すると、同じ変更内容が ee1 のプロパティに反映されます。



ボックス化とボックス化解除

値型を参照型に変換する処理をボックス化と呼び、逆に参照型を値型に変換する処理をボックス化解除と呼びます。この 2 つの処理を次のコードに示します。

C#

```
int i = 123;      // a value type
object o = i;     // boxing
int j = (int)o;  // unboxing
```

Java の場合、このような変換は手動で行う必要があります。このようなオブジェクトを作成する (ボックス化) ことで、プリミティブ データ型をラッパー クラスのオブジェクトに変換できます。また、ボックス化解除のオブジェクトで適切なメソッドを呼び出す (ボックス化解除) ことで、ラッパー クラスのオブジェクトからプリミティブ データ型の値を抽出できます。ボックス化とボックス化解除の詳細については、
「ボックス化変換 (C# プログラミング ガイド)」または「ボックス化解除変換 (C# プログラミング ガイド)」を参照してください。

参照

関連項目

データ型 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C#

Java 開発者のための C# プログラミング言語

演算子 (C# と Java の比較)

C# には、Java でサポートされている、適用可能なすべての演算子が用意されています。これらの演算子の一覧を以下に表に示します。表の最後に、C# で使用でき、Java で使用できない新しい演算子を示します。

カテゴリ	記号
単項演算	**++ -- + - ! ~ ()**
乗法演算	*** / %**
加法演算	**+ -**
シフト	**<< >>**
関係演算	**< > <= >= instanceof**
等値演算	**== !=**
論理 AND	**&**
論理 XOR	**^**
論理 OR	**|**
条件 AND	**&&**
条件 OR	**||**
条件	**? :**
代入	**= *= /= %= += -= <<= >>= &= ^= |=**
オペランドの型	**typeof**
オペランドのサイズ	**sizeof**
オーバーフロー チェックの適用	**checked**
オーバーフロー チェックの中止	**unchecked**

Java の演算子の中で C# で使用できないのは、シフト演算子 (>>>) だけです。この演算子が Java に存在するのは、Java には符号なし変数がないためであり、最上位ビットに 1 を挿入するために右シフトが必要な場合に対応するためです。

C# は、符号なし変数をサポートするため、標準の >> 演算子しか必要としません。この演算子の結果は、オペランドが符号付きであるか、符号なしかによって異なります。符号なし数値を右シフトすると最上位ビットに 0 (ゼロ) が挿入され、符号付き数値を右シフトすると、直前の最上位ビットがコピーされます。

checked 演算子と unchecked 演算子

算術演算の結果が、使用中のデータ型に割り当てられているビット数に対して大きすぎるとオーバーフローが発生します。このようなオーバーフローは、特定の整数算術演算で **checked** キーワードや **unchecked** キーワードを使用することによってチェックしたり、無視したりできます。式が、**checked** を使用している定数式である場合は、コンパイル時にエラーが生成されます。

これらの演算子の使い方を示す簡単な例を次のコードに示します。

C#

```
class TestCheckedAndUnchecked
{
    static void Main()
    {
        short a = 10000;
        short b = 10000;

        short c = (short)(a * b);                      // unchecked by default
        short d = unchecked((short)(10000 * 10000));  // unchecked
        short e = checked((short)(a * b));            // checked - run-time error

        System.Console.WriteLine(10000 * 10000);  // 100000000
        System.Console.WriteLine(c);              // -7936
        System.Console.WriteLine(d);              // -7936
        System.Console.WriteLine(e);              // no result
    }
}
```

このコードでは、unchecked 演算子を使用することで、本来ならば次のステートメントによって生成されるコンパイル時のエラーが回避されます。

C#

```
short d = unchecked((short)(10000 * 10000));  // unchecked
```

次の式は既定で unchecked となるので、値が通知なしでオーバーフローします。

C#

```
short c = (short)(a * b);                      // unchecked by default
```

checked 演算子を使用することにより、実行時に式に対してオーバーフロー チェックを適用できます。

C#

```
short e = checked((short)(a * b));            // checked - run-time error
```

最初の 2 つの値を `d` と `c` に割り当てると、プログラムの実行時に -7936 の値が通知なしでオーバーフローしますが、`checked()` を使用して e の値を乗算しようとすると、プログラムは OverflowException をスローします。

メモ：

コードのブロックで算術オーバーフローをチェックするかどうかは、コマンド ライン コンパイラ スイッチ (**/checked**) を使用して制御することも、Visual Studio でプロジェクトごとに直接制御することもできます。

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

フロー制御 (C# と Java の比較)

if else ステートメントや **switch** ステートメントなどのフロー制御ステートメントは、Java と C# で基本的に同じです。

分岐ステートメント

分岐ステートメントは、実行時に一定の条件に従ってプログラムの実行フローを変更します。

if、else、および else if

これらのステートメントは両方の言語で同じです。

switch ステートメント

switch ステートメントは、両方の言語で条件付きの複数分岐処理を実現します。ただし、Java では、case の最後で break ステートメントを使用している場合を除き、1 つの case から "落下 (フォール スルー)" して、次の case を実行できるという点が異なります。C# では、**break** または goto のいずれかのステートメントを各 case の最後で使用する必要があり、いずれも存在しない場合は、次のコンパイル エラーが発生します。

コントロールは 1 つの case ラベルから別のラベルへ落下できません。

case に一致したときに、その case で実行するコードを指定していない場合は、その次の case に制御が落下します。**switch** ステートメントで `goto` を使用しているときは、同じ switch 内の別の case ブロックだけにジャンプできます。既定の case にジャンプする場合は、`goto default.` を使用します。それ以外の場合は、`goto case cond` を使用します。ここでの `cond` は、ジャンプ先の case の一致条件です。また、Java の **switch** との違いとして、Java では、整数型でしか切り替えることができないのに対し、C# では、文字列変数で切り替えることができるという点もあります。

たとえば、次のコードは C# では有効ですが、Java では無効です。

C#

```
static void Main(string[] args)
{
    switch (args[0])
    {
        case "copy":
            //...
            break;

        case "move":
            //...
            goto case "delete";

        case "del":
        case "remove":
        case "delete":
            //...
            break;

        default:
            //...
            break;
    }
}
```

goto の復帰

Java の場合、**goto** は予約済みのキーワードで実装されません。ただし、ラベル付きステートメントで **break** または continue を使用すると、**goto** と同じ目的を達成できます。

C# では、**goto** ステートメントでラベル付きステートメントにジャンプできます。ただし、特定のラベルにジャンプするには、**goto** ステートメントがそのラベルのスコープ内に存在する必要があります。言い換えると、**goto** は、ステートメント ブロックから外にジャンプすることはできますが、ステートメント ブロックの中にはジャンプできません。また、クラスからジャンプすることも、**try...catch** ステートメント内の **finally** ブロックから出ることもできません。**goto** の使用は、オブジェクト指向プログラミングの適切な手法に反するので、できるだけ避けることをお勧めします。

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C#

Java 開発者のための C# プログラミング言語

ループ ステートメント (C# と Java の比較)

ループ ステートメントは、指定されたコード ブロックを、特定の条件が満たされるまで繰り返します。

for ループ

for ループの構文と動作は、次のように C# と Java の両言語で同じです。

C#

```
for (int i = 0; i<=9; i++)
{
    System.Console.WriteLine(i);
}
```

foreach ループ

C# では、foreach ループという新しい種類のループを導入しています。これは、Visual Basic の **For Each** と同じです。**foreach** ループでは、IEnumerable インターフェイスをサポートする、配列などのコンテナ クラスの各項目を反復処理できます。**foreach** ステートメントを使用して配列の内容を出力する方法を次のコード例に示します。

C#

```
static void Main()
{
    string[] arr= new string[] {"Jan", "Feb", "Mar"};

    foreach (string s in arr)
    {
        System.Console.WriteLine(s);
    }
}
```

詳細については、「配列 (C# と Java の比較)」を参照してください。

while ループと do...while ループ

while ステートメントと do**...while** ステートメントの構文と動作は、次のように C# と Java の両言語で同じです。

C#

```
while (condition)
{
    // statements
}
```

C#

```
do
{
    // statements
}
while(condition);  // Don't forget the trailing ; in do...while loops
```

参照

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

クラスの基本事項 (C# と Java の比較)

以下のセクションでは、C# と Java の修飾子を比較します。

アクセス修飾子

C# の修飾子と Java の修飾子にはわずかな違いがありますが、基本的に同じです。クラスの各メンバまたはクラス自体は、許可されるアクセスのスコープを定義するアクセス修飾子を使用して宣言できます。他のクラス内で宣言されていないクラスは、public 修飾子または internal 修飾子のみを指定できます。入れ子になったクラスは、他のクラス メンバと同様に、次の 5 つのアクセス修飾子を指定できます。

- public

  すべての項目からアクセスできます。

- protected

  派生クラスからのみアクセスできます。

- private

  指定したクラス内でのみアクセスできます。

- internal

  同じアセンブリ内でのみアクセスできます。

- **protected internal**

  現在のアセンブリまたは格納しているクラスから派生した型からのみアクセスできます。

public、protected、private の各修飾子

public 修飾子が設定されたメンバは、クラスの内部と外部のどこでも使用できるようになります。**protected** 修飾子の場合は、格納する側のクラスまたはそのクラスから派生したクラス内にアクセスが制限されます。また **private** 修飾子の場合は、格納する側の型の内部だけにアクセスが制限されます。C# の既定のアクセス修飾子は private ですが、Java では、格納しているパッケージの中からどこにでも既定でアクセスできます。

internal 修飾子

internal 項目には、現在のアセンブリ内でのみアクセスできます。.NET Framework のアセンブリは、Java で言えば、JAR ファイルに当たります。アセンブリは、他のプログラムを作成する元になるビルド ブロックを表します。

protected internal 修飾子

protected internal 項目には、現在のアセンブリ、または格納しているクラスから派生した型だけがアクセスできます。

sealed 修飾子

クラス宣言で sealed 修飾子が設定されたクラスは、抽象クラスとは対照的に継承できません。クラスを sealed とマークすると、そのクラスの機能を他のクラスがオーバーライドできなくなります。当然、シール クラスは抽象クラスにすることはできません。また、構造体は暗黙的にシールされるので、継承できません。**sealed** 修飾子は、Java でクラスを **final** キーワードでマークすることと同じです。

readonly 修飾子

C# で定数を定義する場合は、Java の **final** キーワードの代わりに const 修飾子または readonly 修飾子を使用します。C# のこれら 2 つの修飾子は、**const** 項目がコンパイル時に処理され、**readonly** フィールドの値は実行時に指定されるという点で区別されます。これは、**readonly** フィールドへの割り当てを宣言だけでなく、クラス コンストラクタでも行うことができることを意味します。たとえば、次のクラスでは、クラス コンストラクタで初期化される IntegerVariable という **readonly** 変数を宣言しています。

C#

```
public class SampleClass
{
    private readonly int intConstant;

    public SampleClass ()  //constructor
    {
        // You are allowed to set the value of the readonly variable
        // inside the constructor
        intConstant = 5;
```

```
    }


    public int IntegerConstant
    {
        set
        {
            // You are not allowed to set the value of the readonly variable
            // anywhere else but inside the constructor

            // intConstant = value;  // compile-time error
        }
        get
        {
            return intConstant;
        }
    }
}
class TestSampleClass
{
    static void Main()
    {
        SampleClass obj= new SampleClass();

        // You cannot perform this operation on a readonly field.
        obj.IntegerConstant = 100;

        System.Console.WriteLine("intConstant is {0}", obj.IntegerConstant);  // 5
    }
}
```

readonly 修飾子を静的フィールドに適用する場合、このフィールドは、クラスの静的コンストラクタで初期化する必要があります。

参照

関連項目

アクセス修飾子 (C# プログラミング ガイド)

定数 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

Main () およびその他のメソッド (C# と Java の比較)

ここでは、各メソッドと、メソッド パラメータが参照や値によってどのように引き渡されるかについて説明します。

Main () メソッド

C# アプリケーションでは、常に 1 つの **Main** メソッドを使用して、プログラムの実行を開始する位置を指定する必要があります。Java では、小文字で始まる **main** を使用しますが、C# の **Main** は大文字で始まります。

Main は、次のように **int** または **void** のみを返すことができ、コマンド ライン パラメータを表すオプションの文字列配列引数があります。

C#

```
static int Main(string[] args)
{
    //...
    return 0;
}
```

引き渡されたコマンド ライン引数を格納する文字列配列パラメータは Java の場合と同様に機能します。そのため、`args[0]` が最初のコマンド ライン パラメータを指定し、`args[1]` が 2 番目のパラメータを指定します。`args` 配列は、C++ とは違って、EXE ファイルの名前を格納しません。

その他のメソッド

パラメータをメソッドに渡す場合、パラメータは値または参照によって引き渡すことができます。値パラメータは、メソッドで使用する任意の変数の値をただ取得するだけです。そのため、呼び出し元のコード内の変数値は、メソッド内のパラメータに対して実行される処理の影響を受けません。

参照パラメータは、呼び出し元のコードで宣言された変数を指すため、参照による引き渡しの場合、その変数の内容がメソッドによって変更されます。

参照による引き渡し

Java でも C# でも、オブジェクトを参照するメソッド パラメータは常に参照によって引き渡され、プリミティブ データ型パラメータは値によって引き渡されます。

C# の場合、既定ではすべてのパラメータが値によって引き渡されます。参照による引き渡しを行うには、ref または out のいずれかのキーワードを指定する必要があります。これら 2 つのキーワードは、パラメータの初期化の仕方が違います。**ref** パラメータは、使用する前に初期化する必要がありますが、**out** パラメータは、引き渡す前に明示的に初期化する必要がなく、以前の値がすべて無視されます。

ref キーワード

呼び出されたメソッドで、パラメータとして使用されている変数の値を完全に変更する場合は、このキーワードをパラメータに指定します。これにより、呼び出しで使用される変数の値は引き渡されず、変数自体への参照が引き渡されます。この場合、メソッドは参照を処理するため、メソッドの実行時にパラメータに対して行われた変更は、メソッドへのパラメータとして使用された元の変数に反映され、持続されます。

この一例を、`Add` メソッドを使用して次のコードに示します。2 番目の int パラメータは、**ref** キーワードにより参照渡しされます。

C#

```
class TestRef
{
    private static void Add(int i, ref int result)
    {
        result += i;
        return;
    }

    static void Main()
    {
        int total = 20;
        System.Console.WriteLine("Original value of 'total': {0}", total);

        Add(10, ref total);
        System.Console.WriteLine("Value after calling Add(): {0}", total);
```

```
    }
}
```

この簡単なコード例の出力を次に示します。ここでは、結果のパラメータに対して行われた変更が、`Add` メソッドの呼び出しで使用された変数 `total` に反映されます。

```
Original value of 'total': 20
Value after calling Add(): 30
```

このようになるのも、呼び出し元のコード内の total 変数によって占有されている実際のメモリ位置を、結果のパラメータが参照するからです。クラスのプロパティは変数でないので、*ref* パラメータとして直接使用できません。

ref キーワードは、メソッド宣言だけでなく、メソッドを呼び出すときにもパラメータの前に配置する必要があります。

out キーワード

out キーワードの効果は、**ref** キーワードと基本的に同じであり、**out** で宣言されたパラメータに対して行われた変更がメソッドの外部に反映されます。**ref** との違いは 2 つあり、1 つは *out* パラメータの初期値がメソッド内で無視されること、もう 1 つは *out* パラメータはメソッド内に割り当てる必要があることです。この例を次に示します。

C#

```
class TestOut
{
    private static void Add(int i, int j, out int result)
    {
        // The following line would cause a compile error:
        // System.Console.WriteLine("Initial value inside method: {0}", result);

        result = i + j;
        return;
    }

    static void Main()
    {
        int total = 20;
        System.Console.WriteLine("Original value of 'total': {0}", total);

        Add(33, 77, out total);
        System.Console.WriteLine("Value after calling Add(): {0}", total);
    }
}
```

この場合、`Add` メソッドの 3 番目のパラメータは **out** キーワードで宣言され、メソッドの呼び出しでも、そのパラメータに対して **out** キーワードを使用する必要があります。出力は次のようになります。

```
Original value of 'total': 20
Value after calling Add(): 110
```

要約すると、メソッドで既存の変数を変更するときは **ref** キーワードを使用し、メソッド内で生成された値を返すときは **out** キーワードを使用します。呼び出し元のコードに対し、メソッドが複数の結果値を生成するときは、通常、メソッドの戻り値との関連で out キーワードを使用します。

参照

関連項目

Main() とコマンド ライン引数 (C# プログラミング ガイド)
パラメータの引き渡し (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

不特定の数のパラメータの使用 (C# と Java の比較)

C# では、メソッドの宣言時に params キーワードを指定することで、可変数のパラメータをメソッドに送ることができます。引数リストには、標準のパラメータを含めることもできますが、**params** キーワードで宣言されたパラメータを最後に置く必要があります。この *params* パラメータは、可変長配列の形式を取り、メソッドごとに 1 つのみ使用できます。

コンパイラがメソッド呼び出しを解決するときは、呼び出されるメソッドに引数リストが一致するメソッドを検索します。引数リストに一致するメソッド オーバーロードが見つからなくても、適切な型の *params* パラメータを含む一致するバージョンが存在する場合は、そのメソッドが呼び出され、追加のパラメータが配列に置かれます。

この考えを次のコード例に示します。

C#

```
class TestParams
{
    private static void Average(string title, params int[] values)
    {
        int sum = 0;
        System.Console.Write("Average of {0} (", title);

        for (int i = 0; i < values.Length; i++)
        {
            sum += values[i];
            System.Console.Write(values[i] + ", ");
        }
        System.Console.WriteLine("): {0}", (float)sum/values.Length);
    }
    static void Main()
    {
        Average ("List One", 5, 10, 15);
        Average ("List Two", 5, 10, 15, 20, 25, 30);
    }
}
```

上の例では、メソッド `Average` が、型整数配列の *params* パラメータで宣言されているため、任意の数の引数で呼び出すことができます。この出力を次に示します。

```
Average of List One (5, 10, 15, ): 10
Average of List Two (5, 10, 15, 20, 25, 30, ): 17.5
```

型の異なる不特定のパラメータを使用できるようにする場合は、Object 型の *params* パラメータを指定できます。

参照

関連項目

パラメータとしての配列の受け渡し (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

プロパティ (C# と Java の比較)

C# のプロパティは、いわゆる get アクセサ メソッドや set アクセサ メソッドを通じてプライベート フィールドにアクセスできる、クラス、構造体、またはインターフェイスの名前付きメンバです。

`name` というプライベート変数へのアクセスを抽出する、`Animal` クラスの `Species` というプロパティを宣言するコード例を次に示します。

C#

```
public class Animal
{
    private string name;

    public string Species
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
}
```

多くの場合、プロパティは、アクセス先の内部メンバと同じ名前になりますが、上のケースで `Name` のように大文字で始まるか、または内部メンバの名前が下線 (_) で始まります。また、**set** アクセサで使用されている `value` という暗黙のパラメータにも注意してください。このパラメータは、基底のメンバ変数の型を持ちます。

アクセサをサポートしない .NET Framework ベースの言語との互換性を維持するために、アクセサは、実際には `get_X()` メソッドおよび `set_X()` メソッドとして内部で表現されます。プロパティを次のように定義すると、その値を簡単に取得または設定できます。

C#

```
class TestAnimal
{
    static void Main()
    {
        Animal animal = new Animal();
        animal.Species = "Lion";                    // set accessor
        System.Console.WriteLine(animal.Species);   // get accessor

    }
}
```

プロパティが、**get** アクセサのみを持つ場合は読み取り専用プロパティ、**set** アクセサのみを持つ場合は書き込み専用プロパティ、これら両方のアクセサを持つ場合は読み取り/書き込みプロパティになります。

参照

関連項目

プロパティ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

構造体 (C# と Java の比較)

C# は、struct キーワードをサポートします。C で導入されたこのアイテムも Java では使用できません。**struct** は簡易クラスと考えることができます。**structs** には、コンストラクタ、定数、フィールド、メソッド、プロパティ、インデクサ、演算子、および入れ子にされた型を含めることができますが、これらは通常、関連フィールドのグループをカプセル化するためだけに使用されます。構造体は値型なので、クラスよりも若干効果的に割り当てることができます。**structs** は抽象化できず、実装継承をサポートしない点で、クラスとは異なっています。

次の例では、**struct** を new キーワードで初期化し、パラメータを持たない既定のコンストラクタを呼び出してから、インスタンスのメンバを設定します。

C#

```
public struct Customer
{
    public int ID;
    public string Name;

    public Customer(int customerID, string customerName)
    {
        ID = customerID;
        Name = customerName;
    }
}

class TestCustomer
{
    static void Main()
    {
        Customer c1 = new Customer();  //using the default constructor

        System.Console.WriteLine("Struct values before initialization:");
        System.Console.WriteLine("ID = {0}, Name = {1}", c1.ID, c1.Name);
        System.Console.WriteLine();

        c1.ID = 100;
        c1.Name = "Robert";

        System.Console.WriteLine("Struct values after initialization:");
        System.Console.WriteLine("ID = {0}, Name = {1}", c1.ID, c1.Name);
    }
}
```

出力

上のコードをコンパイルして実行した場合、出力を見ると、**struct** 変数が既定で初期化されたことがわかります。次のように int 変数は 0 (ゼロ) に初期化され、string 変数は空の文字列に初期化されます。

```
Struct values before initialization:

ID = 0, Name =

Struct values after initialization:

ID = 100, Name = Robert
```

参照

処理手順

構造体のサンプル

概念

C# プログラミング ガイド

構造体 (C# プログラミング ガイド)

その他の技術情報

Java 開発者のための C# プログラミング言語

配列 (C# と Java の比較)

配列とは同じデータ型の項目が順序付けられたコレクションで、配列名と、配列の先頭から目的の項目までのオフセットを使用してアクセスできます。C# と Java の間には、配列の宣言方法と使用方法に重要な違いがあります。

1 次元配列

1 次元配列では、固定数の項目を直線的に格納し、単一のインデックス値を使って任意の 1 項目を識別します。C# の配列宣言では、角かっこはデータ型の後に配置する必要があり、Java のように変数名の後に配置することはできません。そのため、`integers` 型の配列は、次の構文を使用して宣言します。

C#

```
int[] arr1;
```

C# では、次の宣言は無効です。

C#

```
//int arr2[];  //compile error
```

配列を宣言したら、Java の場合と同様に、**new** キーワードを使って配列のサイズを設定します。次の例のように配列参照を宣言します。

C#

```
int[] arr;
arr = new int[5];  // create a 5 element integer array
```

次に、Java と同じ構文を使用して、1 次元配列内の要素にアクセスします。C# の配列インデックス番号も 0 から始まります。次の構文では、前の配列の最後の要素にアクセスします。

C#

```
System.Console.WriteLine(arr[4]);  // access the 5th element
```

初期化

C# の配列要素は、次のように Java と同じ構文を使用して作成時に初期化できます。

C#

```
int[] arr2Lines;
arr2Lines = new int[5] {1, 2, 3, 4, 5};
```

C# の初期化子の数は、Java と違って、配列のサイズに正確に一致する必要があります。このため、C# の配列は、次のように 1 行で宣言および初期化できます。

C#

```
int[] arr1Line = {1, 2, 3, 4, 5};
```

この構文は、初期化子の数に一致するサイズの配列を作成します。

プログラム ループでの初期化

C# で配列を初期化するには、**for** ループを使用することもできます。配列の各要素をゼロに設定するループを次に示します。

C#

```
int[] TaxRates = new int[5];

for (int i=0; i<TaxRates.Length; i++)
{
    TaxRates[i] = 0;
}
```

ジャグ配列

C# も Java も、行ごとに列の数が異なるジャグ配列 (四角形ではない配列) の作成をサポートします。たとえば、次のジャグ配列には、第 1 行に 4 つのエントリがあり、第 2 行に 3 つのエントリがあります。

C#

```
int[][] jaggedArray = new int[2][];
jaggedArray[0] = new int[4];
jaggedArray[1] = new int[3];
```

多次元配列

C# では、同じ型の値の行列のような、通常の多次元配列を作成できます。Java も C# もジャグ配列をサポートしますが、C# は多次元配列つまり配列の配列もサポートします。

多次元配列は、次の構文を使用して宣言します。

C#

```
int[,] arr2D;      // declare the array reference
float[,,,] arr4D;  // declare the array reference
```

宣言した後は、次のようにメモリを配列に割り当てます。

C#

```
arr2D = new int[5,4];  // allocate space for 5 x 4 integers
```

メモリを割り当てた後、次の構文を使用して配列の要素にアクセスします。

C#

```
arr2D[4,3] = 906;
```

配列は 0 から始まるので、この行は、第 4 行の第 5 列の要素に 906 を設定します。

初期化

多次元配列は、次のいずれかのメソッドによって、1 つのステートメントで作成、設定、および初期化できます。

C#

```
int[,] arr4 = new int [2,3] { {1,2,3}, {4,5,6} };
int[,] arr5 = new int [,]   { {1,2,3}, {4,5,6} };
int[,] arr6 =               { {1,2,3}, {4,5,6} };
```

プログラム ループでの初期化

配列内のすべての要素は、次の例に示すように、入れ子になったループを使って初期化できます。

C#

```
int[,] arr7 = new int[5,4];
```

```
for(int i=0; i<5; i++)
{
    for(int j=0; i<4; j++)
    {
        arr7[i,j] = 0;  // initialize each element to zero
    }
}
```

System.Array クラス

.NET Framework では、配列は Array クラスのインスタンスとして実装されます。このクラスには、Sort や Reverse など役に立つメソッドがいくつかあります。

これらのメソッドが簡単に使用できることを次の例に示します。最初に、**Reverse** メソッドを使用して配列の要素を反転し、次に、**Sort** メソッドによって各要素を並べ替えます。

C#

```
class ArrayMethods
{
    static void Main()
    {
        // Create a string array of size 5:
        string[] employeeNames = new string[5];

        // Read 5 employee names from user:
        System.Console.WriteLine("Enter five employee names:");
        for(int i=0; i<employeeNames.Length; i++)
        {
            employeeNames[i]= System.Console.ReadLine();
        }

        // Print the array in original order:
        System.Console.WriteLine("\nArray in Original Order:");
        foreach(string employeeName in employeeNames)
        {
            System.Console.Write("{0}  ", employeeName);
        }

        // Reverse the array:
        System.Array.Reverse(employeeNames);

        // Print the array in reverse order:
        System.Console.WriteLine("\n\nArray in Reverse Order:");
        foreach(string employeeName in employeeNames)
        {
            System.Console.Write("{0}  ", employeeName);
        }

        // Sort the array:
        System.Array.Sort(employeeNames);

        //  Print the array in sorted order:
        System.Console.WriteLine("\n\nArray in Sorted Order:");
        foreach(string employeeName in employeeNames)
        {
            System.Console.Write("{0}  ", employeeName);
        }
    }
}
```

出力

```
Enter five employee names:

Luca
```

```
Angie

Brian

Kent

Beatriz

Array in Original Order:

Luca Angie Brian Kent Beatriz

Array in Reverse Order:

Beatriz Kent Brian Angie Luca

Array in Sorted Order:

Angie Beatriz Brian Kent Luca
```

参照

概念

C# プログラミング ガイド

配列 (C# プログラミング ガイド)

その他の技術情報

Java 開発者のための C# プログラミング言語

継承と派生クラス (C# と Java の比較)

既存のクラスの機能は、既存のクラスから派生する新しいクラスを作成することによって拡張できます。派生クラスは基本クラスのプロパティを継承します。必要に応じて、メソッドやプロパティを追加したり、オーバーライドしたりできます。

C# では、継承とインターフェイス実装が共に : 演算子によって定義されます。この演算子は、Java の **extends** および **implements** に相当します。クラス宣言では、必ず基本クラスを左端に配置する必要があります。

Java と同様に C# も多重継承をサポートしないため、複数のクラスからクラスを継承できません。ただし、Java と同様に多重継承用のインターフェイスを使用することはできます。

次のコードは、点の位置を表す `x` と `y` の 2 つのプライベート メンバ変数を含む `CoOrds` というクラスを定義しています。これらの変数には、それぞれ `X`、`Y` というプロパティを介してアクセスします。

C#

```
public class CoOrds
{
    private int x, y;

    public CoOrds()  // constructor
    {
        x = 0;
        y = 0;
    }

    public int X
    {
        get { return x; }
        set { x = value; }
    }

    public int Y
    {
        get { return y; }
        set { y = value; }
    }
}
```

`ColorCoOrds` という新しいクラスを `CoOrds` クラスから次のように派生させます。

C#

```
public class ColorCoOrds : CoOrds
```

これで、`ColorCoOrds` は基本クラスのすべてのフィールドとメソッドを継承します。派生クラスには、必要に応じて新しいフィールドやメソッドを追加して機能を拡張できます。次の例では、クラスに色を付けるためにプライベート メンバとアクセサを追加します。

C#

```
public class ColorCoOrds : CoOrds
{
    private System.Drawing.Color screenColor;


    public ColorCoOrds()  // constructor
    {
        screenColor = System.Drawing.Color.Red;
    }

    public System.Drawing.Color ScreenColor
    {
        get { return screenColor; }
        set { screenColor = value; }
```

```
    }
}
```

派生クラスのコンストラクタは、基本クラス (Java の用語ではスーパークラス) のコンストラクタを暗黙的に呼び出します。継承時には、基本クラスのコンストラクタがすべて呼び出された後に、クラス階層で表示される順番で派生クラスのコンストラクタが呼び出されます。

基本クラスへの型キャスト

Java と同様に、C# では、派生型のオブジェクトへの有効な参照が基本クラス参照に含まれている場合でも、基本クラスへの参照を使用して派生クラスのメンバやメソッドにアクセスできません。

派生クラスは、次のように派生型への参照を暗黙的に使用して参照できます。

C#

```
ColorCoOrds color1 = new ColorCoOrds();
CoOrds coords1 = color1;
```

このコードの基本クラス参照 `coords1` には、`color1` 参照のコピーが含まれます。

base キーワード

サブクラス内の基本クラス メンバには、これらのメンバがスーパークラスでオーバーライドされている場合でも、base キーワードを使用してアクセスできます。たとえば、基本クラスと同じシグネチャを持つメソッドを含む派生クラスを作成することもできます。このメソッドの前に new キーワードを配置すると、このメソッドは、派生クラスに属するまったく新しいメソッドになります。その場合でも、base キーワードを使用すると、基本クラス内の元のメソッドにアクセスするためのメソッドを提供できます。

たとえば、`x` 座標と `y` 座標を入れ替える `Invert()` というメソッドが `CoOrds` 基本クラスにあるとします。この場合、次のようなコードを使用して、このメソッドの代替要素を `ColorCoOrds` 派生クラスに提供できます。

C#

```
public new void Invert()
{
    int temp = X;
    X = Y;
    Y = temp;
    screenColor = System.Drawing.Color.Gray;
}
```

このメソッドは `x` と `y` を入れ替えてから、座標の色を灰色に設定します。このメソッドの基本実装にアクセスできるようにする場合は、次のような新しいメソッドを `ColorCoOrds` で作成します。

C#

```
public void BaseInvert()
{
    base.Invert();
}
```

これで、`BaseInvert()` メソッドを呼び出すと、`ColorCoOrds` オブジェクトの基本メソッドが呼び出されます。

C#

```
ColorCoOrds color1 = new ColorCoOrds();
color1.BaseInvert();
```

この場合、次のように `ColorCoOrds` のインスタンスに基本クラスへの参照を割り当て、このオブジェクトのメソッドを呼び出しても同じ結果が得られます。

C#

```
CoOrds coords1 = color1;
```

```
coords1.Invert();
```

コンストラクタの選択

基本クラス オブジェクトは、常に派生クラスより前に作成されます。そのため、基本クラスのコンストラクタは、派生クラスのコンストラクタより前に実行されます。基本クラスに複数のコンストラクタがある場合、派生クラスは、呼び出すコンストラクタを選択できます。たとえば、`CoOrds` クラスを次のように変更すると、2 番目のコンストラクタを追加できます。

C#

```
public class CoOrds
{
    private int x, y;

    public CoOrds()
    {
        x = 0;
        y = 0;
    }

    public CoOrds(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

続いて、base キーワードを使用して `ColorCoOrds` クラスを次のように変更すると、使用できるコンストラクタのうちの特定のものを使用できます。

C#

```
public class ColorCoOrds : CoOrds
{
    public System.Drawing.Color color;

    public ColorCoOrds() : base ()
    {
        color = System.Drawing.Color.Red;
    }

    public ColorCoOrds(int x, int y) : base (x, y)
    {
        color = System.Drawing.Color.Red;
    }
}
```

Java の場合、この機能は **super** キーワードを使用して実装します。

メソッドのオーバーライド

派生クラスでは、宣言したメソッドに新しい実装を提供することで、基本クラスのメソッドをオーバーライドできます。Java と C# の間には、Java のメソッドは既定で仮想メソッドとしてマークされるのに対し、C# のメソッドは virtual 修飾子を使って仮想メソッドとして明示的にマークする必要があるという重要な違いがあります。メソッドに加えてプロパティ アクセサも同じようにオーバーライドできます。

仮想メソッド

派生クラスでオーバーライドされるメソッドは、**virtual** 修飾子を使用して宣言します。派生クラスでは、オーバーライドされたメソッドは、override 修飾子を使用して宣言します。

override 修飾子は、基本クラスで同じ名前とシグネチャを持つメソッドまたプロパティに置き換わる、派生クラスのメソッドまたはプロパティを示します。オーバーライドされる基本メソッドは、**virtual**、abstract、または **override** として宣言する必要があります。非仮想メソッドや静的メソッドをこのようにオーバーライドすることはできません。メソッドまたはプロパティは、オーバーライドする側もオーバーライドされる側も同じアクセス レベルの修飾子を持つ必要があります。

派生クラスでオーバーライドされる、override 修飾子を持つ `StepUp` という仮想メソッドの例を次に示します。

C#

```
public class CountClass
{
    public int count;

    public CountClass(int startValue)  // constructor
    {
        count = startValue;
    }

    public virtual int StepUp()
    {
        return ++count;
    }
}

class Count100Class : CountClass
{
    public Count100Class(int x) : base(x)  // constructor
    {
    }

    public override int StepUp()
    {
        return ((base.count) + 100);
    }
}

class TestCounters
{
    static void Main()
    {
        CountClass counter1 = new CountClass(1);
        CountClass counter100 = new Count100Class(1);

        System.Console.WriteLine("Count in base class = {0}", counter1.StepUp());
        System.Console.WriteLine("Count in derived class = {0}", counter100.StepUp());
    }
}
```

このコードを実行すると、派生クラスのコンストラクタは、基本クラスのメソッド本体を使用します。そのため、このコードを複製せずに count メンバを初期化できます。出力は次のとおりです。

```
Count in base class = 2

Count in derived class = 101
```

抽象クラス

抽象クラスでは、1 つ以上のメソッドやプロパティを抽象項目として宣言します。このようなメソッドには、それらを宣言したクラスで実装が提供されませんが、抽象クラスには、非抽象メソッド (実装が提供されているメソッド) を含めることもできます。抽象クラスは直接初期化できず、派生クラスとしてのみ初期化できます。このような派生クラスでは、派生メンバ自体が抽象項目として宣言されている場合を除き、**override** キーワードを使用して、すべての抽象メソッドおよび抽象プロパティに実装を提供する必要があります。

`Employee` 抽象クラスを宣言する例を次に示します。また、`Employee` クラスで定義された `Show()` 抽象メソッドの実装を提供する `Manager` という派生クラスも作成します。

C#

```
public abstract class Employee
{
    protected string name;

    public Employee(string name)  // constructor
    {
        this.name = name;
```

```
    }

    public abstract void Show();  // abstract show method
}

public class Manager: Employee
{
    public Manager(string name) : base(name) {}  // constructor

    public override void Show()  //override the abstract show method
    {
        System.Console.WriteLine("Name : " + name);
    }
}

class TestEmployeeAndManager
{
    static void Main()
    {
        // Create an instance of Manager and assign it to a Manager reference:
        Manager m1 = new Manager("H. Ackerman");
        m1.Show();

        // Create an instance of Manager and assign it to an Employee reference:
        Employee ee1 = new Manager("M. Knott");
        ee1.Show();  //call the show method of the Manager class
    }
}
```

このコードは、`Manager` クラスによって提供された `Show()` の実装を呼び出し、従業員の名前を画面に出力します。出力は次のとおりです。

```
Name : H. Ackerman

Name : M. Knott
```

インターフェイス

インターフェイスは、メソッドのシグネチャは存在しても、メソッドの実装は存在しない一種のスケルトン クラスです。つまり、インターフェイスは、抽象メソッドのみが存在する抽象クラスのようなものです。C# インターフェイスは Java インターフェイスと基本的に同じであり、同じように機能します。

インターフェイスのすべてのメンバは定義上、パブリックです。そのため、インターフェイスには、定数、フィールド (プライベート データ メンバ)、コンストラクタ、デストラクタ、およびあらゆる種類の静的メンバを含めることができません。インターフェイスのメンバに修飾子が指定されている場合は、コンパイル時にエラーが発生します。

インターフェイスを実装するには、そのインターフェイスからクラスを派生させます。このような派生クラスでは、それ自体が抽象項目として宣言されている場合を除き、インターフェイスのすべてのメソッドに実装を提供する必要があります。

インターフェイスは、Java と同じように宣言します。インターフェイス宣言では、プロパティとしてインターフェイスの型と、インターフェイスが読み取り専用か、書き込み専用か、または読み書き可能かを get と set の 2 つのキーワードのみによって指定します。1 つの読み取り専用プロパティを宣言するインターフェイスの例を次に示します。

C#

```
public interface ICDPlayer
{
    void Play();  // method signature
    void Stop();  // method signature

    int FastForward(float numberOfSeconds);

    int CurrentTrack  // read-only property
    {
        get;
    }
}
```

このインターフェイスからクラスを継承するには、Java の **implements** キーワードの代わりにコロンを使用します。実装するクラスでは、次のように、すべてのメソッドと必要なすべてのプロパティ アクセサの定義を提供する必要があります。

C#

```
public class CDPlayer : ICDPlayer
{
    private int currentTrack = 0;

    // implement methods defined in the interface
    public void Play()
    {
        // code to start CD...
    }

    public void Stop()
    {
        // code to stop CD...
    }

    public int FastForward(float numberOfSeconds)
    {
        // code to fast forward CD using numberOfSeconds...

        return 0;  //return success code
    }

    public int CurrentTrack  // read-only property
    {
        get
        {
            return currentTrack;
        }
    }

    // Add additional methods if required...
}
```

複数インターフェイスの実装

クラスは、次の構文を使用して複数のインターフェイスを実装できます。

C#

```
public class CDAndDVDComboPlayer : ICDPlayer, IDVDPlayer
```

メンバ名があいまいな複数のインターフェイスを実装する場合、クラスは、プロパティ名またはメソッド名の完全な修飾子によって解決されます。言い換えると、派生クラスは、`ICDPlayer.Play()` のようにインターフェイスの所属先を示すメソッドの完全修飾名を使用して競合を解決できます。

参照

関連項目

継承 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

イベント (C# と Java の比較)

イベントは、クラスからオブジェクトのユーザーに対して、そのオブジェクトに関連する何かが発生したときに (グラフィカル ユーザー インターフェイスのコントロールをクリックした場合などに)、通知する 1 つの方法です。この通知をイベントの発生と言います。イベントを発生させるオブジェクトを、イベントのソースまたは送信元と呼びます。

Java でのイベント処理は、通常、カスタム リスナ クラスを実装して行いますが、C# では、イベント処理にデリゲートを使用できます。デリゲートは、メソッドを参照する型です。デリゲートにメソッドを代入すると、デリゲートはそのメソッドとまったく同じように動作します。デリゲートは C++ の関数ポインタに似ていますが、タイプ セーフです。

デリゲート メソッドは、パラメータと戻り値を指定して他のメソッドと同じように使用できます。この例を次に示します。

```
public delegate int ReturnResult(int x, int y);
```

デリゲートの詳細については、「デリゲート (C# プログラミング ガイド)」を参照してください。

イベントには、メソッドと同様に、名前とパラメータ リストを含むシグネチャがあります。このシグネチャは、デリゲート型によって定義されます。この例を次に示します。

```
public delegate void MyEventHandler(object sender, System.EventArgs e);
```

イベントのソースを参照するオブジェクトを最初のパラメータにし、そのイベントに関連するデータを保持するオブジェクトを 2 番目のパラメータにするのが一般的です。ただし、このようなデザインは、C# 言語によって要求も強制もされません。イベントのシグネチャは、void を返す限りは、有効なデリゲート シグネチャと同じにできます。

イベントは、**event** キーワードを使用して、次のように宣言できます。

```
public event MyEventHandler TriggerIt;
```

イベントを発生させるには、イベントの発生時に呼び出すメソッドを次のように定義します。

```
public void Trigger()
{
    TriggerIt();
}
```

イベントを発生させるために、デリゲートを呼び出し、イベントに関連するパラメータを渡します。これで、デリゲートが、イベントに追加されているすべてのハンドラを呼び出します。各イベントには、イベントを受け取る複数のハンドラを割り当てることができます。この場合、イベントは各レシーバを自動的に呼び出します。イベントを発生させるには、レシーバの数には関係なく、イベントを 1 回だけ呼び出します。

クラスでイベントを受け取る場合は、そのイベントをサブスクライブします。イベントをサブスクライブするには、次のように、**+=** 演算子を使用してイベントにデリゲートを追加します。

```
myEvent.TriggerIt += new MyEventHandler(myEvent.MyMethod);
```

イベントのサブスクライブを解除するには、次のように、**-=** 演算子を使用してイベントからデリゲートを削除します。

```
myEvent.TriggerIt -= new MyEventHandler(myEvent.MyMethod);
```

イベントの詳細については、「イベント (C# プログラミング ガイド)」を参照してください。

メモ :

C# 2.0 では、デリゲートは、名前付きメソッドと匿名メソッドの両方をカプセル化できます。匿名メソッドの詳細については、「匿名メソッド (C# プログラミング ガイド)」を参照してください。

サンプル

次の例では、3 つのメソッドが関連付けられたイベントを定義します。イベントがトリガされると、それらのメソッドが実行されます。1 つのメソッドは

イベントから削除され、イベントが改めてトリガされます。

```
// Declare the delegate handler for the event:
public delegate void MyEventHandler();

class TestEvent
{
    // Declare the event implemented by MyEventHandler.
    public event MyEventHandler TriggerIt;

    // Declare a method that triggers the event:
    public void Trigger()
    {
        TriggerIt();
    }
    // Declare the methods that will be associated with the TriggerIt event.
    public void MyMethod1()
    {
        System.Console.WriteLine("Hello!");
    }
    public void MyMethod2()
    {
        System.Console.WriteLine("Hello again!");
    }
    public void MyMethod3()
    {
        System.Console.WriteLine("Good-bye!");
    }

    static void Main()
    {
        // Create an instance of the TestEvent class.
        TestEvent myEvent = new TestEvent();

        // Subscribe to the event by associating the handlers with the events:
        myEvent.TriggerIt += new MyEventHandler(myEvent.MyMethod1);
        myEvent.TriggerIt += new MyEventHandler(myEvent.MyMethod2);
        myEvent.TriggerIt += new MyEventHandler(myEvent.MyMethod3);
        // Trigger the event:
        myEvent.Trigger();

        // Unsuscribe from the the event by removing the handler from the event:
        myEvent.TriggerIt -= new MyEventHandler(myEvent.MyMethod2);
        System.Console.WriteLine("\"Hello again!\" unsubscribed from the event.");

        // Trigger the new event:
        myEvent.Trigger();
    }
}
```

出力

```
Hello!
Hello again!
Good-bye!
"Hello again!" unsubscribed from the event.
Hello!
Good-bye!
```

参照

関連項目

event (C# リファレンス)

delegate (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

Java 経験者が C# で開発する場合

演算子のオーバーロード (C# と Java の比較)

C# では、C++ と同様に、ユーザー定義クラス専用の演算子をオーバーロードできます。これにより、ユーザー定義データ型が基本データ型と同じように違和感なく、論理的に使用できるようになります。たとえば、複素数を表す `ComplexNumber` という新しいデータ型を作成した場合には、標準の算術演算子を使用して複素数に対する数値演算 (+ 演算子を使用して 2 つの複素数を加算するなど) を実行するメソッドを提供できます。

演算子をオーバーロードするには、名前演算子の後にオーバーロード対象の演算子の記号が続く関数を作成します。たとえば、+ 演算子をオーバーロードする場合は、次のようなコードを記述します。

C#

```
public static ComplexNumber operator+(ComplexNumber a, ComplexNumber b)
```

すべての演算子のオーバーロードはクラスの静的なメソッドです。また、等値演算子 (==) をオーバーロードする場合は、非等値演算子 (!=) もオーバーロードする必要があります。< と >、および <= と >= の演算子も、ペアでオーバーロードする必要があります。

オーバーロードが可能な演算子の一覧を以下に示します。

- 単項演算子 : +、-、!、~、++、--、true、false
- 二項演算子 : +、-、*、/、%、&、|、^、<<、>>、==、!=、>、<、>=、<=

+ 演算子と - 演算子をオーバーロードする `ComplexNumber` クラスを作成するコード例を次に示します。

C#

```
public class ComplexNumber
{
    private int real;
    private int imaginary;

    public ComplexNumber() : this(0, 0)  // constructor
    {
    }

    public ComplexNumber(int r, int i)  // constructor
    {
        real = r;
        imaginary = i;
    }

    // Override ToString() to display a complex number in the traditional format:
    public override string ToString()
    {
        return(System.String.Format("{0} + {1}i", real, imaginary));
    }

    // Overloading '+' operator:
    public static ComplexNumber operator+(ComplexNumber a, ComplexNumber b)
    {
        return new ComplexNumber(a.real + b.real, a.imaginary + b.imaginary);
    }

    // Overloading '-' operator:
    public static ComplexNumber operator-(ComplexNumber a, ComplexNumber b)
    {
        return new ComplexNumber(a.real - b.real, a.imaginary - b.imaginary);
    }
}
```

このクラスでは、次のようなコードを使用して 2 つの複素数を作成し、操作できます。

C#

```
class TestComplexNumber
{
    static void Main()
    {
        ComplexNumber a = new ComplexNumber(10, 12);
        ComplexNumber b = new ComplexNumber(8, 9);

        System.Console.WriteLine("Complex Number a = {0}", a.ToString());
        System.Console.WriteLine("Complex Number b = {0}", b.ToString());

        ComplexNumber sum = a + b;
        System.Console.WriteLine("Complex Number sum = {0}", sum.ToString());

        ComplexNumber difference = a - b;
        System.Console.WriteLine("Complex Number difference = {0}", difference.ToString());
    }
}
```

上のプログラムが示すように、`ComplexNumber` クラスに属するオブジェクトには、プラス演算子とマイナス演算子を直観的に使用できます。このプログラムの出力を次に示します。

```
Complex Number a = 10 + 12i

Complex Number b = 8 + 9i

Complex Number sum = 18 + 21i

Complex Number difference = 2 + 3i
```

Java は演算子のオーバーロードをサポートしませんが、文字列連結の際には、内部で + 演算子をオーバーロードします。

参照

処理手順

演算子のオーバーロードのサンプル

関連項目

オーバーロードされた演算子 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

例外 (C# と Java の比較)

例外処理は、C# と Java では基本的に同じです。

プログラムの実行中にエラーが発生すると、.NET Framework 共通言語ランタイム (CLR) は、エラー情報を示す Exception オブジェクトを作成します。.NET Framework では、**Exception** がすべての例外クラスの基本クラスです。**Exception** クラスから派生する例外には、SystemException と ApplicationException の 2 つのカテゴリがあります。System 名前空間のすべての種類は **SystemException** から派生するのに対し、ユーザー定義の例外は、ランタイム エラーとアプリケーション エラーを区別するために **ApplicationException** から派生する必要があります。一般的な **System** 例外は次のとおりです。

- IndexOutOfRangeException : 配列またはコレクションのサイズを超えるインデックスが使用されている。
- NullReferenceException : 参照に有効なインスタンスが設定される前に、その参照のプロパティまたはメソッドが使用されている。
- ArithmeticException : 操作の結果、オーバーフローまたはアンダーフローが発生した。
- FormatException : 引数またはオペランドの書式に誤りがある。

Java の場合と同様に、コードで例外が発生する可能性がある場合は、そのコードを **try** ブロック内に配置します。直後の 1 つ以上の **catch** ブロックでエラー処理を実行しますが、例外がスローされてもされなくても実行するすべてのコードに **finally** ブロックを適用することもできます。詳細については、「try-catch (C# リファレンス)」および「try-catch-finally (C# リファレンス)」を参照してください。

複数の **catch** ブロックを使用したときは、スローされた例外に一致する最初の **catch** ブロックだけが実行されるので、キャッチされた例外を、一般性が低い方から高くなる順に配置する必要があります。C# コンパイラは、これを強制しますが、Java コンパイラは強制しません。

また、C# では、Java と違って **catch** ブロックの引数は必須ではありません。引数が存在しない場合、**catch** ブロックはすべての **Exception** クラスに適用されます。

たとえば、ファイルからの読み取り時には、FileNotFoundException や IOException が発生することがあるので、次のコードに示すように、より限定された **FileNotFoundException** ハンドラを最初に配置するのが適切です。

C#

```
try
{
    // code to open and read a file
}
catch (System.IO.FileNotFoundException e)
{
    // handle the file not found exception first
}
catch (System.IO.IOException e)
{
    // handle any other IO exceptions second
}
catch
{
    // a catch block without a parameter
    // handle all other exceptions last
}
finally
{
    // this is executed whether or not an exception occurs
    // use to release any external resources
}
```

Exception の派生クラスを、独自の例外クラスとして作成できます。たとえば、次のコードは、新しい `Employee` に設定された部署が無効な場合などにスローできる `InvalidDepartmentException` クラスを作成します。このコードでは、ユーザー定義例外のクラス コンストラクタが **base** キーワードを使用して基本クラス コンストラクタを呼び出し、適切なメッセージを送ります。

C#

```
public class InvalidDepartmentException : System.Exception
{
    public InvalidDepartmentException(string department) : base("Invalid Department: " + de
partment)
```

```
        {
        }
    }
```

続いて、次のようなコードを使用して例外をスローできます。

C#

```
class Employee
{
    private string department;

    public Employee(string department)
    {
        if (department == "Sales" || department == "Marketing")
        {
            this.department = department;
        }
        else
        {
            throw new InvalidDepartmentException(department);
        }
    }
}
```

C# では、チェックされた例外をサポートしません。Java では、チェックされた例外を **throws** キーワードで宣言し、呼び出し側のコードで処理する必要がある特定の種類の例外をメソッドがスローできるようにします。

参照

関連項目

例外と例外処理 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Java 開発者のための C# プログラミング言語

C# の高度な手法 (C# と Java の比較)

C# は、インデクサ、属性、デリゲートなどの高度なプログラミング手法を可能にする便利な言語機能を搭載しています。

インデクサ

インデクサを使用すると、配列と同じ方法でクラスや構造体にアクセスできます。たとえば、社内の単一の部署を表すクラスを作成し、このクラスに部署の従業員全員の名前を含めると、次のようにインデクサを使用してこれらの名前にアクセスできます。

C#

```
sales[0] = "Nikki";
sales[1] = "Becky";
```

インデクサを有効にするには、クラス定義などで、次のシグネチャを持つプロパティを定義します。

C#

```
public string this [int index]  //indexer
```

次に、通常のプロパティの場合と同じように get メソッドと set メソッドを指定します。インデクサの使用時に参照先の内部メンバを指定するのがこれらのアクセサです。

次の簡単な例では、`Department` というクラスを作成します。このクラスはインデクサを使用して、内部的には文字列の配列として表現される、該当する部署の従業員にアクセスします。

C#

```
public class Department
{
    private string name;
    private const int MAX_EMPLOYEES = 10;
    private string[] employees = new string[MAX_EMPLOYEES];  //employee array

    public Department(string departmentName)  //constructor
    {
        name = departmentName;
    }

    public string this [int index]  //indexer
    {
        get
        {
            if (index >= 0 && index < MAX_EMPLOYEES)
            {
                return employees[index];
            }
            else
            {
                throw new System.IndexOutOfRangeException();
            }
        }
        set
        {
            if (index >= 0 && index < MAX_EMPLOYEES)
            {
                employees[index] = value;
            }
            else
            {
                throw new System.IndexOutOfRangeException();
            }
        }
    }
```

```
    // code for the rest of the class...
}
```

これで、次のコード例に示すように、このクラスのインスタンスを作成し、アクセスできます。

C#

```
class TestDepartment
{
    static void Main()
    {
        Department sales = new Department("Sales");

        sales[0] = "Nikki";
        sales[1] = "Becky";

        System.Console.WriteLine("The sales team is {0} and {1}", sales[0], sales[1]);
    }
}
```

出力は次のとおりです。

```
The sales team is Nikki and Becky
```

詳細については、「インデクサ (C# プログラミング ガイド)」を参照してください。

属性

C# には、属性と呼ばれる、型の宣言情報を追加する機構があります。属性は、Java の注釈の概念と似ています。型に関する追加情報は、型定義の前の宣言タグの中に配置されます。以下の例は、.NET Framework 属性を使用してクラスやメソッドを装飾する方法を示しています。

次の例では、WebMethodAttribute 属性を追加することにより、`GetTime` メソッドを XML Web サービスとしてマークしています。

C#

```
public class Utilities : System.Web.Services.WebService
{
    [System.Web.Services.WebMethod]  // Attribute
    public string GetTime()
    {
        return System.DateTime.Now.ToShortTimeString();
    }
}
```

WebMethod 属性を追加すると、.NET Framework は、この関数を呼び出すために必要な XML/SOAP 交換を自動的に処理します。この Web サービスを呼び出すと、次の値が取得されます。

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">7:26 PM</string>
```

次の例では、SerializableAttribute 属性を追加して、`Employee` クラスをシリアル化できるクラスとしてマークします。`Salary` フィールドはパブリック フィールドとしてマークされていますが、NonSerializedAttribute 属性によってマークされているため、シリアル化できません。

C#

```
[System.Serializable()]
public class Employee
{
    public int ID;
    public string Name;
    [System.NonSerialized()] public int Salary;
}
```

詳細については、「カスタム属性の作成 (C# プログラミング ガイド)」を参照してください。

デリゲート

C++ や Pascal などの言語は、実行時に呼び出す関数を選択できるようにする関数ポインタの概念をサポートしています。

Java には、関数ポインタの機能を持つ構成要素がありませんが、C# には用意されています。Delegate クラスを使用することによって、デリゲート インスタンスは、呼び出し可能なエンティティであるメソッドをカプセル化します。

インスタンス メソッドの場合、デリゲートは、格納しているクラスのインスタンスとそのインスタンスのメソッドで構成されます。また静的メソッドの場合、呼び出し可能なエンティティは、クラスとそのクラスの静的メソッドで構成されます。そのため、デリゲートは、任意のオブジェクトの関数を呼び出すために使用できます。デリゲートは、オブジェクト指向のタイプ セーフで安全な型です。

デリゲートを定義して使用する際には、次の 3 つのステップがあります。

- 宣言
- インスタンス化
- 呼び出し

デリゲートは、次の構文を使用して宣言します。

C#

```
delegate void Del1();
```

これで、このデリゲートを使用して、void を返し、引数を受け取らないすべての関数を参照できます。

また、文字列パラメータを受け取り、long を返す任意の関数のデリゲートを作成する場合は、次の構文を使用します。

C#

```
delegate long Del2(string s);
```

このデリゲートは、次のようなシグネチャを含むすべてのメソッドに割り当てることができます。

C#

```
Del2 d;                 // declare the delegate variable
d = DoWork;  // set the delegate to refer to the DoWork method
```

ここで、`DoWork` のシグネチャは次のとおりです。

C#

```
public static long DoWork(string name)
```

デリゲートの再割り当て

Delegate オブジェクトは変更不可です。つまり、対応するシグネチャは、いったん設定すると変更できなくなります。ただし、シグネチャが同じであれば、別のメソッドを指すことができます。この例では、`d` を新しいデリゲート オブジェクトに再割り当てし、`d` が `DoMoreWork` メソッドを呼び出すようにします。このような再割り当ては、`DoWork` と `DoMoreWork` が共に同じシグネチャを持つ場合に限られます。

C#

```
Del2 d;                     // declare the delegate variable
d = DoWork;       // set the delegate to refer to the DoWork method
d = DoMoreWork;  // reassign the delegate to refer to the DoMoreWork method
```

デリゲートの呼び出し

デリゲートを呼び出すのは簡単です。メソッド名をデリゲート変数の名前に置き換えるだけです。この場合、11 と 22 の 2 つの値によって `Add` メソッドが呼び出され、変数 `sum` に割り当てられた long 型の結果が返されます。

C#

```
Del operation;                  // declare the delegate variable
operation = Add;      // set the delegate to refer to the Add method
long sum = operation(11, 22);  // invoke the delegate
```

デリゲートの作成、インスタンス化、および呼び出しの例を次に示します。

C#

```
public class MathClass
{
    public static long Add(int i, int j)       // static
    {
        return (i + j);
    }

    public static long Multiply (int i, int j)  // static
    {
        return (i * j);
    }
}

class TestMathClass
{
    delegate long Del(int i, int j);  // declare the delegate type

    static void Main()
    {
        Del operation;  // declare the delegate variable

        operation = MathClass.Add;       // set the delegate to refer to the Add method
        long sum = operation(11, 22);             // use the delegate to call the Add metho
d

        operation = MathClass.Multiply;  // change the delegate to refer to the Multiply me
thod
        long product = operation(30, 40);         // use the delegate to call the Multiply
method

        System.Console.WriteLine("11 + 22 = " + sum);
        System.Console.WriteLine("30 * 40 = " + product);
    }
}
```

出力

```
11 + 22 = 33

30 * 40 = 1200
```

デリゲート インスタンスには、オブジェクト参照を含める必要があります。上の例では、メソッドを静的メソッドとして宣言しているため、オブジェクト参照を指定する必要がありませんが、デリゲートがインスタンス メソッドを参照する場合は、次のようにオブジェクト参照を指定する必要があります。

C#

```
Del operation;                    // declare the delegate variable
MathClass m1 = new MathClass();  // declare the MathClass instance
operation = m1.Add;      // set the delegate to refer to the Add method
```

この例では、`Add` および `Multiply` は `MathClass` のインスタンス メソッドです。`MathClass` のメソッドが静的として宣言されていない場合は、次のように、`MathClass` のインスタンスを使用してデリゲートでメソッドを呼び出します。

C#

```
public class MathClass
{
    public long Add(int i, int j)       // not static
    {
        return (i + j);
    }

    public long Multiply (int i, int j)  // not static
    {
        return (i * j);
    }
}

class TestMathClass
{
    delegate long Del(int i, int j);  // declare the delegate type

    static void Main()
    {
        Del operation;                    // declare the delegate variable
        MathClass m1 = new MathClass();  // declare the MathClass instance

        operation = m1.Add;       // set the delegate to refer to the Add method
        long sum = operation(11, 22);      // use the delegate to call the Add method

        operation = m1.Multiply;  // change the delegate to refer to the Multiply method
        long product = operation(30, 40);  // use the delegate to call the Multiply method

        System.Console.WriteLine("11 + 22 = " + sum);
        System.Console.WriteLine("30 * 40 = " + product);
    }
}
```

出力

この例では、メソッドを静的メソッドとして宣言した前の例と同じ出力が得られます。

```
11 + 22 = 33

30 * 40 = 1200
```

デリゲートとイベント

.NET Framework では、Windows アプリケーションや Web アプリケーションでのボタン クリック イベントなどのイベント処理タスクでもデリゲートを頻繁に使用します。Java でのイベント処理は、通常、カスタム リスナ クラスを実装して行いますが、C# では、イベント処理にデリゲートを利用できます。イベントは、イベント宣言の前にキーワード イベントがある場合を除き、デリゲート型を使用してフィールドのように宣言します。イベントは、通常、パブリックとして宣言しますが、アクセシビリティ修飾子を使用することもできます。**delegate** と **event** の宣言の例を次に示します。

C#

```
// Declare the delegate type:
public delegate void CustomEventHandler(object sender, System.EventArgs e);

// Declare the event variable using the delegate type:
public event CustomEventHandler CustomEvent;
```

イベント デリゲートはマルチキャストなので、複数のイベント処理メソッドへの参照を保持できます。デリゲートは、イベントに対して登録されているイベント ハンドラのリストを管理することで、そのイベントを発生させるクラスのイベント ディスパッチャとして動作します。イベントに複数の関数をサブスクライブする方法を次の例に示します。`EventClass` クラスには、デリゲート、イベント、およびイベントを呼び出すメソッドが含まれます。イベントの呼び出しは、イベントを宣言したクラスの中からしか行うことができないことに注意してください。`TestEvents` クラスは、**+=** 演算子を使用してイベントにサブスクライブでき、**-=** 演算子を使用してアンサブスクライブできます。次の例に示すように、`InvokeEvent` メソッドを呼び出すと、イベントが発生し、それに同期して、イベントにサブスクライブしているすべての関数が起動します。

C#

```
public class EventClass
```

```
{
    // Declare the delegate type:
    public delegate void CustomEventHandler(object sender, System.EventArgs e);

    // Declare the event variable using the delegate type:
    public event CustomEventHandler CustomEvent;

    public void InvokeEvent()
    {
        // Invoke the event from within the class that declared the event:
        CustomEvent(this, System.EventArgs.Empty);
    }
}

class TestEvents
{
    private static void CodeToRun(object sender, System.EventArgs e)
    {
        System.Console.WriteLine("CodeToRun is executing");
    }

    private static void MoreCodeToRun(object sender, System.EventArgs e)
    {
        System.Console.WriteLine("MoreCodeToRun is executing");
    }

    static void Main()
    {
        EventClass ec = new EventClass();

        ec.CustomEvent += new EventClass.CustomEventHandler(CodeToRun);
        ec.CustomEvent += new EventClass.CustomEventHandler(MoreCodeToRun);

        System.Console.WriteLine("First Invocation:");
        ec.InvokeEvent();

        ec.CustomEvent -= new EventClass.CustomEventHandler(MoreCodeToRun);

        System.Console.WriteLine("\nSecond Invocation:");
        ec.InvokeEvent();
    }
}
```

出力

```
First Invocation:

CodeToRun is executing

MoreCodeToRun is executing

Second Invocation:

CodeToRun is executing
```

参照

処理手順

デリゲートのサンプル

概念

C# プログラミング ガイド

デリゲート (C# プログラミング ガイド)

イベント (C# プログラミング ガイド)

その他の技術情報

Java 開発者のための C# プログラミング言語

ガベージ コレクション (C# と Java の比較)

C や C++ の多くのオブジェクトでは、宣言したらリソースを割り当て、安全に使用できるようにする必要があります。また、オブジェクトを使用した後は、リソースを空きメモリ プールに解放する必要もあります。リソースを解放しないと、リソースが次から次へと無駄に消費されるので、メモリ リークと呼ばれる問題がコードで発生します。逆に、リソースの解放が早すぎると、データの損失、他のメモリ領域の破損、null ポインタ例外などが発生する可能性があります。

Java と C# では共に、これらの問題を防ぐために、アプリケーションで使用中のすべてのオブジェクトの有効期間を個別に管理します。

Java では、JVM が、割り当てられたリソースへの参照を追跡して、未使用メモリの解放を管理します。リソースが有効な参照によって参照されなくなったことを JVM が検知すると、リソースのガベージ コレクションが行われます。

C# の場合、ガベージ コレクションは、JVM と同じ機能を持つ共通言語ランタイム (CLR) によって処理されます。CLR のガベージ コレクタは、メモリ ヒープに未参照のオブジェクトが存在しないかどうかを定期的にチェックし、そのようなオブジェクトによってリソースが保持されている場合は解放します。

参照

概念

C# プログラミング ガイド

自動メモリ管理

その他の技術情報

Java 開発者のための C# コード例

安全なコードと安全でないコード (C# と Java の比較)

C# には、タイプ セーフでないコードをサポートするという興味深い特徴があります。通常は、共通言語ランタイム (CLR) が Microsoft Intermediate Language (MSIL) コードの動作を監視し、問題のありそうな動作を回避しますが、Win32 API 呼び出しなどの下位の機能に直接アクセスする必要があるときもあり、そのようなコードが正常に動作することを確信できる場合のみ、これを行うことができます。そのようなコードは、ソース コード内の unsafe ブロックの中に配置する必要があります。

unsafe キーワード

C# では、下位の API 呼び出しを行ったり、ポインタ演算を使用したりするような安全性に劣る操作を実行するコードは、unsafe キーワードでマークされたブロックの中に配置する必要があります。次の要素はいずれも unsafe とマークできます。

- メソッド全体
- 中かっこ内のコード ブロック
- 個別のステートメント

上の 3 つのケースで **unsafe** を使用するコード例を次に示します。

C#

```
class TestUnsafe
{
    unsafe static void PointyMethod()
    {
        int i=10;

        int *p = &i;
        System.Console.WriteLine("*p = " + *p);
        System.Console.WriteLine("Address of p = {0:X2}\n", (int)p);
    }

    static void StillPointy()
    {
        int i=10;

        unsafe
        {
            int *p = &i;
            System.Console.WriteLine("*p = " + *p);
            System.Console.WriteLine("Address of p = {0:X2}\n", (int)p);
        }
    }

    static void Main()
    {
        PointyMethod();
        StillPointy();
    }
}
```

このコードでは、`PointyMethod()` メソッド全体を unsafe とマークしていますが、それは、このメソッドでポインタを宣言し、使用するためです。また、`StillPointy()` メソッドでコードのブロックを unsafe とマークしているのは、このブロックでもポインタを使用するためです。

fixed キーワード

セーフ コードでは、ガベージ コレクタが、その有効期間中にオブジェクトを自由に移動し、フリー リソースを編成して凝縮します。ただし、コードでポインタを使用する場合は、この動作が原因で、予測できない結果が生じやすくなります。そのため、fixed ステートメントを使用して、ガベージ コレクタが特定のオブジェクトを移動しないように指示できます。

fixed キーワードを使用して、`PointyMethod()` メソッドのコード ブロックを実行しているときに配列が移動されないようにする例を次のコードに示します。**fixed** が、アンセーフ コードの中でしか使用されていないことに注意してください。

C#

```
class TestFixed
{
    public static void PointyMethod(char[] array)
    {
        unsafe
        {
            fixed (char *p = array)
            {
                for (int i=0; i<array.Length; i++)
                {
                    System.Console.Write(*(p+i));
                }
            }
        }
    }

    static void Main()
    {
        char[] array = { 'H', 'e', 'l', 'l', 'o' };
        PointyMethod(array);
    }
}
```

参照

処理手順

アンセーフ コードのサンプル

概念

C# プログラミング ガイド

unsafe コードとポインタ (C# プログラミング ガイド)

その他の技術情報

Java 開発者のための C# プログラミング言語

Java 開発者のための C# コード例

C# は、広範なアプリケーションを構築できる、洗練された、簡素でタイプ セーフなオブジェクト指向言語です。Visual C# は、.NET Framework と組み合わせて使用することで、Windows アプリケーション、Web サービス、データベース ツール、コンポーネント、コントロールなどを作成できます。

このセクションの内容

コンソール アプリケーションの開発 (C# と Java の比較)

Java 開発者を対象に、C# によるコンソール アプリケーションについて説明します。

ファイル I/O (C# と Java の比較)

Java 開発者を対象に、C# によるファイル I/O 操作について説明し、XML ファイルの読み込みおよび保存用の XML クラスへのリンクを提供します。

データベース アクセス (C# と Java の比較)

両方の言語でのデータ アクセスを比較します。

ユーザー インターフェイスの開発 (C# と Java の比較)

Java 開発者を対象に、C# による Windows フォーム アプリケーションの作成について説明します。

リソースの管理 (C# と Java の比較)

Java 開発者を対象に、C# を使用した Windows リソース ファイルに関するトピックを紹介します。

Web サービス アプリケーション (C# と Java の比較)

Java 開発者を対象に、C# による XML Web サービス アプリケーション開発について説明します。

モバイル デバイスおよびデータ (C# と Java の比較)

Java 開発者を対象に、C#、.NET Framework Windows フォーム、および SQL Server を使用したデータベースとの対話について説明します。

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C# への移行

Visual C# によるアプリケーションの作成

コンソール アプリケーションの開発 (C# と Java の比較)

コンソール アプリケーションは、グラフィカル ユーザー インターフェイスなしで標準入出力 (I/O) を読み書きします。コンソール アプリケーションの構造は Java と C# で基本的に同じであり、コンソール I/O で同じようなクラスを使用します。

C# と Java では、クラスとメソッド シグネチャは細かな点で異なることもありますが、基本的に同じ概念を使用してコンソール I/O 操作を実行します。C# にも Java にも、コンソール アプリケーションとそれに関連するコンソールの読み取りおよび書き込みメソッドについてメイン エントリ ポイントという概念があります。これは、C# では `Main`、Java では `main` です。

Java の "Hello World" コード例

次の Java のプログラム例では、`static void main()` ルーチンが、アプリケーションの引数への `String` 参照を受け入れます。その後で、`main` ルーチンが行をコンソールに出力します。

```
/*  A Java Hello World Console Application */
public class Hello {
    public static void main (String args[]) {
        System.out.println ("Hello World");
    }
}
```

C# の "Hello World" コード例

次の C# のプログラム例では、`static void Main()` ルーチンが、アプリケーションの引数への `string` 参照を受け入れます。その後で、`Main` ルーチンが行をコンソールに書き込みます。

C#

```
// A C# Hello World Console Application.
public class Hello
{
    static void Main()
    {
        System.Console.WriteLine("Hello World");
    }
}
```

コードのコンパイル

Visual C# を使用する場合、F5 キーを押すだけでコードをコンパイルして実行できます。コマンド ラインを使用し、ファイル名が "Hello.cs" の場合、C# コンパイラを次のように起動します。

csc Hello.cs

参照項目

コンソール アプリケーションの作成の詳細については、「コンソール アプリケーションの作成 (Visual C#)」を参照してください。

.NET Framework コンソール クラスの詳細については、以下のトピックを参照してください。

- Console クラス、WriteLine メソッド、および ReadLine メソッド。
- カテゴリ別の C# コンパイラ オプションの一覧.
- C# 2.0 言語およびコンパイラの新機能

Java から C# への自動変換の詳細については、「Java Language Conversion Assistant の新機能」を参照してください。

参照

関連項目

Main() とコマンド ライン引数 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Java 経験者が C# で開発する場合

ファイル I/O (C# と Java の比較)

C# と Java では、クラスとメソッド シグネチャは細かな点で異なることもありますが、ファイル I/O 操作では基本的に同じ概念を使用します。C# にも Java にも、ファイル クラスという概念と、それに関連するファイルの読み取りおよび書き込みメソッドがあります。XML コンテンツの処理に関しては、同じようなドキュメント オブジェクト モデル (DOM) があります。

Java のファイル操作例

Java では、**File** オブジェクトを使用して、基本ファイル I/O 操作 (ファイルの作成、オープン、クローズ、読み取り、書き込みなど) を実行できます。たとえば、ファイルを作成する場合は、**File** クラスの **createNewFile** メソッドを使用し、ファイルを削除する場合は **delete** メソッドを使用するなど、**File** クラスのメソッドを使用してファイル I/O 操作を実行できます。ファイルの内容の読み取りや書き込みを行うには、**BufferedReader** クラスと **BufferedWriter** クラスを使用します。

新規ファイルを作成する方法、ファイルを削除する方法、ファイルからテキストを読み取る方法、およびファイルに書き込む方法を次のコード例に示します。

```
// Java example code to create a new file
    try
    {
        File file = new File("path and file_name");
        boolean success = file.createNewFile();
    }
    catch (IOException e)    {   }

// Java example code to delete a file.
    try
    {
        File file = new File("path and file_name");
        boolean success = file.delete();
    }
    catch (IOException e)    {    }

// Java example code to read text from a file.
    try
    {
        BufferedReader infile = new BufferedReader(new FileReader("path and file_name "));
        String str;
        while ((str = in.readLine()) != null)
        {
            process(str);
        }
        infile.close();
    }
    catch (IOException e)
    {
        // Exceptions ignored.
    }

// Java example code to writing to a file.
    try
    {
        BufferedWriter outfile =
          new BufferedWriter(new FileWriter("path and file_name "));
        outfile.write("a string");
        outfile.close();
    }
    catch (IOException e)    {    }
```

C# のファイル操作例

C# でファイル I/O 操作を実行する場合は、.NET Framework の同等のクラスとメソッドを使用して作成、オープン、クローズ、読み取り、および書き込みを行うときの基本的な方法をそのまま利用できます。たとえば、.NET Framework の File クラスのメソッドを使用してファイル I/O 操作を実行できます。たとえば、Exists メソッドを使用して、ファイルが存在するかどうかをチェックできます。次のコード例に示すように、Create メソッドを使用してファイルを作成し、必要に応じて既存のファイルを上書きできます。また、FileStream クラスと BufferedStream オブジェクトを使用

して読み取りと書き込みを行うこともできます。

ファイルを削除する方法、ファイルを作成する方法、ファイルに書き込む方法、およびファイルから読み取る方法を次のコード例に示します。

C#

```
// sample C# code for basic file I/O operations
// exceptions ignored for code simplicity

class TestFileIO
{
    static void Main()
    {
        string fileName = "test.txt";  // a sample file name

        // Delete the file if it exists.
        if (System.IO.File.Exists(fileName))
        {
            System.IO.File.Delete(fileName);
        }

        // Create the file.
        using (System.IO.FileStream fs = System.IO.File.Create(fileName, 1024))
        {
            // Add some information to the file.
            byte[] info = new System.Text.UTF8Encoding(true).GetBytes("This is some text in
the file.");
            fs.Write(info, 0, info.Length);
        }

        // Open the file and read it back.
        using (System.IO.StreamReader sr = System.IO.File.OpenText(fileName))
        {
            string s = "";
            while ((s = sr.ReadLine()) != null)
            {
                System.Console.WriteLine(s);
            }
        }
    }
}
```

関連項目

.NET Framework のクラスで、ストリームの作成、読み取り、および書き込みを行う際に役立つのは、StreamReader クラスと StreamWriter クラスです。これら以外に、ファイルの処理に役立つ .NET Framework のクラスは以下のとおりです。

- FileAccess クラスおよび FileAttribute クラス
- Directory クラス、DirectoryInfo クラス、Path クラス、FileInfo クラス、および DriveInfo クラス
- BinaryReader クラスおよび BinaryWriter クラス
- StringReader クラスおよび StringWriter クラス
- TextReader クラスおよび TextWriter クラス
- XmlReader クラスおよび XmlWriter クラス
- ToBase64Transform クラス
- **FileStream** クラス、**BufferedStream** クラス、および MemoryStream クラス

Java から C# への自動変換の詳細については、「Java Language Conversion Assistant 3.0 の新機能」を参照してください。

また、.NET Framework のセキュリティの詳細については、「.NET Security」を参照してください。

参照
概念
C# プログラミング ガイド

データベース アクセス (C# と Java の比較)

C# と Java は、同じような方法でデータベース データにアクセスし、いずれもデータベース ドライバを使用して実際のデータベース操作を実行します。さらに、C# と Java はいずれもデータベース接続、データベース接続に対して実行する SQL クエリ、およびクエリの実行結果セットを必要とします。

データベース ドライバの比較

Java と C# では、JDBC や ODBC などのデータベース ドライバを使用してデータにアクセスできます。JDBC (Java Database Connectivity) ドライバは、Java で記述されたプログラムから使用されます。ODBC (Open Database Connectivity) は、さまざまなプラットフォーム上の各種リレーショナル データベースにアクセスするためのマイクロソフトのデータベース プログラミング インターフェイスです。また、Java プラットフォームの Solaris バージョンにも Windows バージョンにも JDBC-ODBC ブリッジ規格があるので、Java プログラムから ODBC を使用することもできます。

Java では、接続文字列情報が接続ハンドル用のドライバに次のように提供されます。

```
final static private String url = "jdbc:oracle:server,user,pass, …)";
```

C# で .NET Framework を使用すると、データベースにアクセスする際に ODBC ドライバや JDBC ドライバを読み込む必要がありません。データベース接続オブジェクトに接続文字列を次のように設定するだけです。

C#

```
static string connectionString = "Initial Catalog=northwind;Data Source=(local);Integrated
Security=SSPI;";
static SqlConnection cn = new SqlConnection(connectionString);
```

- Oracle データベース用の ODBC ドライバの詳細については、「ODBC Driver for Oracle」を参照してください。DB2 データベース用の OLE DB プロバイダの詳細については、「Microsoft Host Integration Server 2000 Developer's Guide」および「Administration and Management of Data Access Using the OLE DB Provider for DB2」を参照してください。
- Microsoft® SQL Server™ 2000 Driver for JDBC は、Java 対応の任意のアプレット、アプリケーション、またはアプリケーション サーバーから SQL Server 2000 へアクセスできるようにする Type 4 JDBC ドライバです。

Java でのデータベースの読み取り例

Java でデータベースの読み取り操作を実行するには、**Statement** オブジェクトの **executeQuery** メソッドによって作成された **ResultSet** オブジェクトを使用できます。**ResultSet** オブジェクトには、クエリによって返されたデータが含まれます。これにより、**ResultSet** オブジェクトを反復処理してデータにアクセスできます。

データベースから読み取るための Java コードの例を次に示します。

```
Connection c;
try
{
    Class.forName (_driver);
    c = DriverManager.getConnection(url, user, pass);
}
catch (Exception e)
{
    // Handle exceptions for DriverManager
    // and Connection creation:
}
try
{
    Statement stmt = c.createStatement();
    ResultSet results = stmt.executeQuery(
                "SELECT TEXT FROM dba ");
    while(results.next())
    {
        String s = results.getString("ColumnName");
        // Display each ColumnName value in the ResultSet:
    }
    stmt.close();
```

```
}
catch(java.sql.SQLException e)
{
    // Handle exceptions for executeQuery and getString:
}
```

また、データベース書き込み操作を実行する際には、**Connection** オブジェクトから **Statement** オブジェクトが作成されます。**Statement** オブジェクトには、データベースに対して SQL クエリや更新を実行するためのメソッドがあります。更新およびクエリは、**ResultSet** オブジェクトを返すために **Statement** オブジェクトの **executeUpdate** メソッドで使用される書き込み操作の SQL コマンドを含む文字列形式です。

C# でのデータベースの読み取り例

C# で .NET Framework を使用すると、ADO.NET によって提供されるクラスのセットにより、データベースへのアクセスがさらに簡素化されます。ADO.NET は、ODBC ドライバを使用したデータベース アクセスだけでなく、OLE DB プロバイダ経由のデータベース アクセスもサポートします。C# アプリケーションは、.NET Framework の ADO.NET クラスおよび MDAC (Microsoft Data Access Component) を使用して SQL データベースと対話し、データの読み取り、書き込み、検索を実行できます。.NET Framework の System.Data.SqlClient 名前空間およびクラスにより、SQL Server データベースへのアクセスが一段と容易になります。

C# では、データベースの読み取り操作を実行する際に、接続、コマンド、およびデータ テーブルを使用できます。たとえば、**System.Data.SqlClient** 名前空間を使用して SQL Server データベースに接続する場合は、以下を使用できます。

- SqlConnection クラス。
- SqlCommand クラスなどのクエリ。
- DataTable クラスなどの結果セット。

.NET Framework では、DataAdapter を使用して、これら 3 つのオブジェクトを次のように結び付けることができます。

- **SqlConnection** オブジェクトは、**DataAdapter** オブジェクトの接続プロパティを使用して設定します。
- 実行するクエリは、**DataAdapter** の SelectCommand プロパティを使用して指定します。
- **DataTable** オブジェクトは、**DataAdapter** オブジェクトの Fill メソッドを使用して作成します。**DataTable** オブジェクトには、クエリによって返された結果データ セットが含まれます。**DataTable** オブジェクトを反復処理し、Rows コレクションを使用してデータ行にアクセスできます。

コードをコンパイルして実行するには、以下が必要です。以下がない場合、`databaseConnection.Open();` は失敗し、例外がスローされます。

- MDAC (Microsoft Data Access Components) Version 2.7 以降。

  Microsoft Windows XP または Windows Server 2003 を使用している場合、MDAC 2.7 は既に存在します。ただし、Microsoft Windows 2000 を使用している場合は、コンピュータに既にインストールされている MDAC のアップグレードが必要になる場合があります。詳細については、「MDAC Installation」を参照してください。

- SQL Server Northwind データベースへのアクセス、および Northwind サンプル データベースがインストール済みのローカル SQL Server でコードを実行する現在のユーザー名に対する統合セキュリティ特権。

C#

```
// Sample C# code accessing a sample database

// You need:
//   A database connection
//   A command to execute
//   A data adapter that understands SQL databases
//   A table to hold the result set

namespace DataAccess
{
    using System.Data;
    using System.Data.SqlClient;

    class DataAccess
    {
        //This is your database connection:
        static string connectionString = "Initial Catalog=northwind;Data Source=(local);Int
egrated Security=SSPI;";
```

```
        static SqlConnection cn = new SqlConnection(connectionString);

        // This is your command to execute:
        static string sCommand = "SELECT TOP 10 Lastname FROM Employees ORDER BY EmployeeID
 ";

        // This is your data adapter that understands SQL databases:
        static SqlDataAdapter da = new SqlDataAdapter(sCommand, cn);

        // This is your table to hold the result set:
        static DataTable dataTable = new DataTable();

        static void Main()
        {
            try
            {
                cn.Open();

                // Fill the data table with select statement's query results:
                int recordsAffected = da.Fill(dataTable);

                if (recordsAffected > 0)
                {
                    foreach (DataRow dr in dataTable.Rows)
                    {
                        System.Console.WriteLine(dr[0]);
                    }
                }
            }
            catch (SqlException e)
            {
                string msg = "";
                for (int i=0; i < e.Errors.Count; i++)
                {
                    msg += "Error #" + i + " Message: " + e.Errors[i].Message + "\n";
                }
                System.Console.WriteLine(msg);
            }
            finally
            {
                if (cn.State != ConnectionState.Closed)
                {
                    cn.Close();
                }
            }
        }
    }
}
```

ADO.NET の詳細については、以下のトピックを参照してください。

- ADO.NET
- データへのアクセス (Visual Studio)
- ADO.NET の応用例
- ADO.NET for the Java Programmer

.NET Framework データベース アクセス クラスの詳細については、次のトピックを参照してください。

- System.Data.OracleClient
- System.Data.SqlServerCe
- System.Data.Odbc
- System.Data.OleDb

- ADO.NET DataSet
- ADO.NET での DataSet の使用

Java Language Conversion Assistant の詳細については、「Java Language Conversion Assistant 3.0 の新機能」を参照してください。

参照
概念
C# プログラミング ガイド
その他の技術情報
Java 経験者が C# で開発する場合

ユーザー インターフェイスの開発 (C# と Java の比較)

C# では、.NET Framework の豊富な Windows フォーム コンポーネント セットを使用して、クライアント側のフォーム アプリケーションをプログラミングできます。

Java

ほとんどの Java アプリケーションでは、AWT (Abstract Windowing ToolKit) または Swing を使用します。これらは AWT イベント モデルなど、AWT インフラストラクチャを使用してフォーム プログラミングを行います。AWT は、すべての基本的な GUI 機能とクラスを提供します。

Java の例

コンポーネントを追加する際は、通常、フレーム (タイトルと輪郭を持つウィンドウ) を使用します。

```
JFrame aframe = new JFrame();
```

通常、**Component** クラス (グラフィカル表示を持つオブジェクト) は拡張され、次のコードに示す **Shape** コンポーネントの **paint** メソッドのように継承されたメソッドが使用されるか、またはオーバーライドされるのが普通です。

```
     import java.awt.*;
     import javax.swing.*;

     class aShape extends JComponent {
     public void paint(Graphics g) {
     Graphics2D g2d = (Graphics2D)g;

     // Draw the shape.
     }

     public static void main(String[] args) {
     JFrame aframe = new JFrame();
     frame.getContentPane().add(new aShape ());
     int frameWidth = 300;
     int frameHeight = 300;
     frame.setSize(frameWidth, frameHeight);
     frame.setVisible(true);
     }
}
```

イベントを処理するコンポーネントのために、アクション イベントを待機するように登録できます。たとえば、ボタンを押してから離すと、AWT は、そのボタンで **processEvent** を呼び出して、**ActionEvent** のインスタンスをボタンに送ります。ボタンの **processEvent** メソッドは、ボタンのすべてのイベントを受け取り、固有の **processActionEvent** メソッドを呼び出して、アクション イベントを渡します。後者のメソッドは、このボタンによって生成されたアクション イベントへの関与を登録しているすべてのアクション リスナにアクション イベントを渡します。

C#

C# では、.NET Framework の System.Windows.Forms 名前空間およびクラスが、Windows フォームの開発に必要なコンポーネントの包括的なセットを提供します。たとえば、次のコードでは、Label、Button、および MenuStrip を使用しています。

C# の例

単純に Form クラスから次のように派生させます。

C#

```
public partial class Form1 : System.Windows.Forms.Form
```

次に、コンポーネントを追加します。

C#

```
this.button1 = new System.Windows.Forms.Button();
this.Controls.Add(this.button1);
```

ラベル、ボタン、およびメニューをフォームに追加する方法を次のコードに示します。

C#

```
namespace WindowsFormApp
{
    public partial class Form1 : System.Windows.Forms.Form
    {
        private System.ComponentModel.Container components = null;

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.MenuStrip menu1;

        public Form1()
        {
            InitializeComponent();
        }

        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();

            this.label1 = new System.Windows.Forms.Label();
            this.Controls.Add(this.label1);

            this.button1 = new System.Windows.Forms.Button();
            this.Controls.Add(this.button1);

            this.menu1 = new System.Windows.Forms.MenuStrip();
            this.Controls.Add(this.menu1);
        }

        static void Main()
        {
            System.Windows.Forms.Application.Run(new Form1());
        }
    }
}
```

Java と同様に、C# でもコンポーネントのために、イベントを待機するように登録できます。たとえば、ボタンを押してから離すと、ランタイムにより、このボタンの Click イベントに対する関与を登録しているすべてのリスナに **Click** イベントが送られます。

C#

```
private void button1_Click(object sender, System.EventArgs e)
{
}
```

次のコードを使用すると、`button1` という名前の **Button** のインスタンスの **Click** イベントを処理するように `button1_Click` を登録できます。

C#

```
// this code can go in InitializeComponent()
button1.Click += button1_Click;
```

詳細については、「ASP.NET Web アプリケーションの作成 (Visual C#)」を参照してください。

Java から C# への自動変換の詳細については、「Java Language Conversion Assistant 3.0 の新機能」を参照してください。

Forms クラスの詳細については、「Windows フォーム コントロールの機能別一覧」および **System.Windows.Forms** のトピックを参照してください。

参照

概念

C# プログラミング ガイド

ユーザー インターフェイスのデザイン (Visual C#)

その他の技術情報

Java 経験者が C# で開発する場合

リソースの管理 (C# と Java の比較)

Visual Studio を使用する C# では、リソースを簡単に管理できます。

Java

Java アプリケーションは、一般にクラス ファイル、サウンド ファイル、イメージ ファイルなどのアプリケーションのさまざまなリソースと一緒に JAR ファイルにまとめられます。通常は JBuilder または Eclipse を使用します。これらのアプリケーションは Visual Studio がソリューションおよびプロジェクトを管理するのとほぼ同じ方法で JAR ファイルを管理します。

C#

C# プロジェクトでは、リソースを Visual Studio のソリューション エクスプローラから簡単に開くことができます。

また、イメージ エディタとバイナリ エディタを使用して、マネージ プロジェクト内のリソース ファイルを操作することもできます。

マネージ プロジェクトにリソースを追加する方法の詳細については、以下のトピックを参照してください。

- リソースの追加と編集 (Visual C#)
- チュートリアル : Windows フォームのローカリゼーション
- チュートリアル : ASP.NET でのローカリゼーションのためのリソースの使用

これらのリソースは、外部コンテンツまたは埋め込みリソースとしてアプリケーションで読み取ることができます。たとえば、次のコード行では、Assembly などの System.Reflection 名前空間とクラスを使用して、埋め込みリソース ファイルをアセンブリから読み取ります。ここでは、埋め込みリソース ファイルは `assemblyname.file.ext` です。

C#

```
static void Main()
{
    System.Reflection.Assembly asm =
        System.Reflection.Assembly.GetExecutingAssembly();

    System.Drawing.Bitmap tiles = new System.Drawing.Bitmap
        (asm.GetManifestResourceStream("assemblyname.file.ext"));
}
```

- 詳細については、「リフレクション (C# プログラミング ガイド) .」を参照してください。
- アプリケーション リソースの詳細については、「アプリケーション リソースの管理」を参照してください。
- 一般的なリソース エディタの動作については、「リソース エディタ」を参照してください。
- .Resx 形式のリソース ファイルを編集する方法の詳細については、「アプリケーションのリソース」を参照してください。
- XML および SAX2 (Simplified API for XML) の処理の詳細については、「SAX2 Developer Guide」および「XML Developer Center」を参照してください。

参照

概念

C# プログラミング ガイド

リソースの追加と編集 (Visual C#)

その他の技術情報

Java 経験者が C# で開発する場合

Web サービス アプリケーション (C# と Java の比較)

.NET Framework では、Web サービス間での相互運用性が幅広くサポートされています。C# では、.NET Framework、Visual Studio、および ASP.NET を使用することで、Web サービス プロジェクトを作成し、公開するパブリック メソッドに **WebMethod** 属性を追加するだけで簡単に Web サービスを作成できます。

Java

Java では、Web サービス パッケージを使用して、Java Web Services Developer Pack や Apache SOAP などのアプリケーションを実装できます。たとえば、Java では、次の手順に従って Web サービスと Apache SOAP を作成できます。

Apache SOAP を使用して、Java で Web サービスを作成するには

1. Web サービス メソッドを次のように記述します。

```
public class HelloWorld
{

    public String sayHelloWorld()
    {
        return "HelloWorld ";
    }
}
```

2. Apache SOAP 配置記述子を作成します。この記述子の例を次に示します。

```
<dd:service xmlns:dd="http://xml.apache.org/xml-soap/deployment"
           id="urn:HelloWorld">

   <dd:provider type="java"
               scope="Application"
               methods="sayHelloWorld">

    <dd:java class="HelloWorld" static="false" />

  </dd:provider>

<dd:faultListener>org.apache.soap.server.DOMFaultListener</dd:faultListener>

  <dd:mappings />

</dd:service>
```

3. `HelloWorld` クラスをコンパイルし、これを Web サーバーのクラスパスに移動します。

4. コマンド ライン ツールを使用して、Web サービスを配置します。

C#

C# では、.NET Framework クラスと Visual Studio IDE を使用して、Web サービスをより簡単に作成できます。

.NET Framework と Visual Studio を使用して、C# で Web サービスを作成するには

1. Visual Studio で Web サービス アプリケーションを作成します。詳細については、「Java 開発者のための C# アプリケーションの種類の説明」を参照してください。作成されたコードを次に示します。

C#

```
using System;
```

```
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {

    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }

}
```

2. 行 `[WebService(Namespace = "http://tempuri.org/")]` を見つけ、`"http://tempuri.org/"` を `"http://tempuri.org/"` に変更します。

C# Web サービスを実行するには

1. サービスをコンパイルして実行します。Web ブラウザのアドレス バーに「http://localhost/website1/service.asmx」と入力します。**localhost** は IIS Web サーバーの名前、**Service** はサービスの名前 (この場合は `Service`) です。
2. 出力は次のとおりです。

```
The following operations are supported. For a formal definition, please review the Ser
vice Description.
HelloWorld
```

3. HelloWorld リンクをクリックして、Service1 の `HelloWorld` メソッドを呼び出します。出力は次のとおりです。

```
Click here for a complete list of operations.
HelloWorld
Test
To test the operation using the HTTP POST protocol, click the 'Invoke' button.

SOAP 1.1
...
SOAP 1.2
...
HTTP POST
...
```

4. [Invoke] をクリックして、Service1 の `HelloWorld` メソッドを呼び出します。出力は次のとおりです。

```
<?xml version="1.0" encoding="utf-8" ?>
  <string xmlns="http://HowToDevelopWebServicesTest/">Hello World</string>
```

Web サービスの詳細については、以下のトピックを参照してください。

- XML Web サービス クライアントの構築
- XML Web サービスの説明

- 方法 : XML Web サービス メソッドを作成する
- チュートリアル : Visual Basic または Visual C# を使った XML Web サービスの作成
- チュートリアル : Visual Web Developer での ASP.NET Web サービスの作成と使用

Java から C# への自動変換の詳細については、「Java Language Conversion Assistant 3.0 の新機能」を参照してください。

参照

概念

C# プログラミング ガイド

その他の技術情報

Java 経験者が C# で開発する場合

モバイル デバイスおよびデータ (C# と Java の比較)

C# および .NET Compact Framework を使用すると、デスクトップ データベース プログラミングで使用するのと同一の概念および同様の API を使ってモバイル デバイス上のデータベース データにアクセスしたり管理したりできます。モバイル デバイスでは、ADO.NET が、Pocket PC や Smartphone を含む Windows CE デバイスを対象としたデスクトップ API のサブセットを提供します。詳細については、「データベース アクセス (C# と Java の比較)」を参照してください。

Java

Java では、J2ME および JDBC を使用して、モバイル デバイスからデータベースにアクセスできます。詳細については、「データベース アクセス (C# と Java の比較)」を参照してください。J2ME は、すべてのデバイスで使用できる単一の API ではなく、単一の開発環境を提供しません。さらに J2ME は、構成に応じて KVM または JVM のいずれかの仮想マシン内で実行する必要があります。

C#

C# でデータベースの読み取り操作を実行する場合は、デスクトップでもモバイル デバイスでも、接続、コマンド、データ テーブルなどの、なじみのある概念を使用できます。System.Data.SqlServerCe 名前空間およびクラスを使用するだけで操作を実行できます。たとえば、以下を使用できます。

- データベース接続のために SqlCeConnection を使用できます。
- SQL コマンド オブジェクトとして SqlCeCommand を使用できます。
- データ テーブル オブジェクトとして、DataTable などの結果セット オブジェクトを使用できます。

.NET Framework では、DataAdapter を使用して上記のクラスを簡単に併用できます。**SqlCeConnection** オブジェクトは、SqlCeDataAdapter オブジェクトの接続プロパティを使用して設定できます。

実行するクエリは、DataAdapter の SelectCommand プロパティを使用して指定するか、または単に接続オブジェクトと共に、DataAdapter のコンストラクタに渡します。

C#

```
da = new SqlCeDataAdapter("SELECT * FROM Users", cn);
```

DataTable オブジェクトは、DataAdapter オブジェクトの Fill メソッドを使用して作成します。**DataTable** オブジェクトには、クエリによって返された結果データ セットが含まれます。**DataTable** オブジェクトを反復処理し、Rows コレクションを使用してデータ行にアクセスできます。

モバイル デバイス上の SQL Server CE (SQLCE) データベースのテーブル行にアクセスする方法を次のコードに示します。

C#

```
namespace DataAccessCE
{
    using System.Data;
    using System.Data.SqlServerCe;

    class DataAccessCE
    {
        public static string connectionString = "";
        public static SqlCeConnection cn = null;
        public static SqlCeDataAdapter da = null;
        public static DataTable dt = new DataTable();

        static void Main()
        {
            connectionString = "Data Source=\\My Documents\\Database.sdf" ;
            cn = new SqlCeConnection(connectionString);

            da = new SqlCeDataAdapter("SELECT * FROM Users", cn);
            da.Fill(dt);

            foreach (DataRow dr in dt.Rows)
            {
                System.Console.WriteLine(dr[0]);
```

```
                }
            }
        }
    }
```

詳細については、次のトピックを参照してください。

- ADO.NET の概要
- データへのアクセス (Visual Studio)
- ADO.NET の応用例
- レプリケーション
- サブスクリプションの同期

Java から C# への自動変換の詳細については、「Java Language Conversion Assistant 3.0 の新機能」を参照してください。

コードのコンパイル

SQLCE データベースをアプリケーションから操作する前に、**System.Data.SqlServerCe** への参照をプロジェクトに追加する必要があります。参照を追加するには、開発環境の [プロジェクト] メニューの [参照の追加] をクリックします。次に、[参照の追加] ダイアログ ボックスで **System.Data.SqlServerCe** コンポーネントを選択します。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

プログラミングの注意点

コードをコンパイルして実行するには、以下が必要です。以下がない場合、`da.Fill(dt);` は失敗し、例外がスローされます。

- デバイスにインストールされている SQL Server CE。
- SQLCE データベース (Database.sdf) に対するテスト用のデータを含むデータベース テーブル。このテーブルは、SQL CE ツールを使用してデバイス上に作成できます。また、SQL Server デスクトップからレプリケートして .sdf ファイルを生成することもできます。.sdf ファイルは、プロジェクトに追加することも、接続文字列で指定されたディレクトリに手動でコピーすることもできます。

参照

関連項目

スマート デバイス
SqlConnection
SqlCommand

概念

C# プログラミング ガイド
ADO.NET DataSet

その他の技術情報

Java 経験者が C# で開発する場合
スマート デバイス開発
ADO.NET での DataSet の使用
ADO.NET for the Java Programmer

Java 開発者のための C# アプリケーションの種類の説明

C# アプリケーションには、Windows コンソール アプリケーション、Windows フォーム アプリケーション、ASP.NET Web アプリケーション、ASP.NET Web サービス アプリケーション、スマート デバイス アプリケーション、ActiveX アプリケーション、セットアップおよび配置アプリケーションなどの種類があります。

コンソール アプリケーション

コンソール アプリケーションは、フォームではなく、標準のコマンド ライン入出力を使用して入出力を実現します。コンソール アプリケーションでは、入出力を処理するために System.IO クラスを使用します。`System.IO.Console.WriteLine()` のようにメソッドの前にクラス名を使用したり、プログラムの先頭に **using** ステートメントを配置したりできます。コンソール アプリケーションは、Visual Studio や、Microsoft® メモ帳などのテキスト エディタを含む他の開発環境を使用して簡単に作成できます。詳細については、「Visual Studio の紹介」、「コンソール アプリケーションの作成 (Visual C#)」、「Hello World -- 最初のプログラム (C# プログラミング ガイド)」、または「Main() とコマンド ライン引数 (C# プログラミング ガイド)」を参照してください。

フォーム アプリケーション

フォーム アプリケーションには、入力用のボタンやリスト ボックスのような、一般的な Windows グラフィカル ユーザー インターフェイスが用意されています。フォーム アプリケーションは、System.Windows.Forms 名前空間内のクラスを使用します。フォーム アプリケーションは、Visual Studio や、Microsoft® メモ帳などのテキスト エディタを含む他の開発環境を使用して簡単に作成できます。Windows フォーム アプリケーションの作成の詳細については、「方法 : Windows アプリケーション プロジェクトを作成する」、「ASP.NET Web アプリケーションの作成 (Visual C#)」、または「ASP.NET Web アプリケーションの作成 (Visual C#)」を参照してください。

ASP.NET Web アプリケーション

ASP.NET アプリケーションは、コンソールやフォーム アプリケーションではなく、Web ブラウザに表示される Web アプリケーションです。ASP.NET アプリケーションでは、ブラウザからの入出力を処理するために、System.Web 名前空間や System.Web.UI などのクラスを使用します。`using System.Web.UI.HtmlControls;` のようにメソッドの前にクラス名を使用したり、プログラムの先頭に **using** ステートメントを配置したりできます。ASP.NET アプリケーションは、Visual Studio や、Microsoft® メモ帳などのテキスト エディタを含む他の開発環境を使用して簡単に作成できます。ASP.NET アプリケーションを作成する方法の詳細については、「Visual Web Developer」を参照してください。Visual Studio .NET を使用して ASP.NET アプリケーションを作成する方法の詳細については、「アプリケーション ダイアグラムでの ASP.NET アプリケーションの概要」を参照してください。ASP.NET の詳細については、「.NET Framework の ASP.NET Web アプリケーション」を参照してください。また、ASP.NET アプリケーションをデバッグする方法の詳細については、「ASP.NET Web アプリケーションのデバッグ」および「デバッグの準備 : ASP.NET Web アプリケーション」を参照してください。

ASP.NET Web サービス アプリケーション

ASP.NET Web サービスには、URL、HTTP、および XML を使用してアクセスできるので、プラットフォームや言語とは無関係にどのプログラムでも ASP.NET Web サービスにアクセスできます。ASP.NET Web サービス アプリケーションは、フォームのコンソール、Web ブラウザ、またはスマート デバイスに表示できます。ASP.NET Web サービス アプリケーションは、**System.Web** および System.Web.Services の名前空間とクラスを使用します。ASP.NET Web サービス アプリケーションは、Visual Studio や、Microsoft® メモ帳などのテキスト エディタを含む他の開発環境を使用して簡単に作成できます。Web サービス アプリケーションの作成の詳細については、「データのアクセスと表示 (Visual C#)」および「方法 : ASP.NET Web サービス プロジェクトを作成する」を参照してください。ASP.NET Web サービスを既存のプロジェクトに追加する方法の詳細については、「方法 : マネージ コードを使用して既存の Web プロジェクトに XML Web サービスを追加する」を参照してください。ASP.NET Web サービスの詳細については、「チュートリアル : Visual Web Developer での ASP.NET Web サービスの作成と使用」および「チュートリアル : Visual Basic または Visual C# を使った XML Web サービスの作成」を参照してください。また、ASP.NET Web サービス アプリケーションをデバッグする方法の詳細については、「デバッグの準備 : XML Web サービス プロジェクト」を参照してください。

この他に ASP.NET Web サービスに関連する以下のようなトピックがあります。

- XML Web サービス クライアントの構築
- XML Web サービスの説明
- 方法 : XML Web サービス メソッドを作成する
- チュートリアル : Visual Basic または Visual C# を使った XML Web サービスの作成
- チュートリアル : Visual Web Developer での ASP.NET Web サービスの作成と使用
- How to Use Visual Studio .NET 2003 to build and test an XML Web service

スマート デバイス アプリケーション

スマート デバイス アプリケーションは、PDA や Smartphone などのモバイル デバイスで動作します。スマート デバイス アプリケーションは、コン

ソール アプリケーション、Windows フォーム アプリケーション、ASP.NET クライアント、または Web クライアントとして作成でき、コンソール、フォーム、または Web ブラウザに表示されます。スマート デバイス アプリケーションは、デスクトップ アプリケーションと同じ名前空間とクラスを使用します。ただし、このアプリケーションは、.NET Framework ではなく Compact Framework を使用します。Windows モバイル デバイス アプリケーションの開発とデスクトップ アプリケーションの開発の違いの詳細については、
「デバイス アプリケーション開発とデスクトップ アプリケーション開発の比較」を参照してください。開発環境のバージョンには、モバイル デバイスでの C# アプリケーションの一部またはすべての種類の開発をサポートできるものがあります。ASP.NET アプリケーションの作成の詳細については、「新しいデバイス プロジェクトの開始」および「スマート デバイス アプリケーション ウィザード」を参照してください。

この他に ASP.NET Web サービスに関連する以下のようなトピックがあります。

- Use Visual Studio .NET 2003 to build and test a mobile Web application
- スマート デバイス アプリケーションの開発
- Smart Device Programmability Features of Visual Studio .NET
- スマート デバイス ハードウェアの考慮事項
- ASP.NET モバイル コントロール クイック スタート
- ASP.NET モバイル コントロールのサンプル
- スマート デバイスのチュートリアル
- チュートリアル : デバイス対応の Windows フォーム アプリケーションの作成
- チュートリアル : デバイス プロジェクトでの Windows フォームのデバッグ

ActiveX コントロール

Java Beans と同様に、ActiveX コントロールは、"OLE オブジェクト" およびコンポーネント オブジェクト モデル (COM) オブジェクトに相当するコンポーネントです。最も単純な形式の ActiveX コントロールは、IUnknown インターフェイスをサポートする COM オブジェクトです。ActiveX コントロールは、Internet Explorer からソフトウェア開発ツール、エンド ユーザー生産性向上ツールに至るまでのさまざまなコンテナで再利用するためのプログラミング可能なソフトウェア コンポーネントの主要な開発アーキテクチャです。ActiveX コントロールの詳細については、以下のトピックを参照してください。

- Introduction to ActiveX
- Packaging ActiveX Controls
- Using ActiveX Controls to Automate Your Web
- Designing Secure ActiveX Controls
- Using ActiveX Controls with Windows Forms in Visual Studio .NET
- ActiveX コントロールのデバッグ

セットアップおよび配置アプリケーション

Visual Studio には、デスクトップ、Web、およびスマート デバイスのセットアップおよび配置プロジェクト用のテンプレートが用意されています。デスクトップ、Web、およびモバイル デバイスでの C# アプリケーションの一部またはすべての種類のセットアップおよび配置は、さまざまなバージョンの開発環境でサポートされています。詳細については、次のトピックを参照してください。

- .NET Framework アプリケーションの配置.
- デバイス アプリケーションの配布
- デバイスへのアプリケーション インストール
- チュートリアル : デバイス プロジェクトのカスタム CAB ファイルの生成
- Cabinet Packaging : Internet Explorer Code Download and the Java Package Manager

関連トピック

- .NET Framework のネットワーク操作の基礎
- Windows フォーム アプリケーションの基礎
- Java Language Conversion Assistant 3.0 の新機能

- .NET Framework のサンプル

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C# への移行
Java 開発者のための C# コード例
Java 開発者のための C# プログラミング言語
Visual C# について
Visual C# IDE の使用

Java アプリケーションの Visual C# への変換

Microsoft Visual Studio 2005 には、Visual J++ ® Version 6.0 で作成されたプロジェクトまたは Java 言語で記述されたプロジェクトを Visual C#® に変換する機能があるため、.NET Framework を利用できます。Java Language Conversion Assistant を使用して、既存の Visual J++ 6.0 プロジェクトまたは Java 言語のファイルから新しい Visual C# プロジェクトを生成します。Java Language Conversion Assistant ウィザードを使用すると、既存のファイルを簡単に変換できます。

このセクションの内容

Java Language Conversion Assistant 3.0 の新機能

このバージョンの Java Language Conversion Assistant には、次の新機能が含まれています。

Visual J++ または Java 言語から Visual C# へのプロジェクトの変換

Java Language Conversion Assistant の使用方法について説明します。

Web アプリケーションの変換

Java Language Conversion Assistant を使用して Web アプリケーションを変換するときに Web アプリケーションを適合させる方法について説明します。

各種 Java 言語アプリケーションの変換

Java Language Conversion Assistant を使用して各種のアプリケーションを変換する際に、実行する必要がある処理と発生する可能性がある問題について説明します。

JLCA 診断メッセージ (パッケージ別)

Java Language Conversion Assistant で生成されるエラー メッセージの一覧を示します。

関連するセクション

Java Language Conversion Assistant ウィザード

ウィザードの使用方法について説明します。

JSP カスタム タグ ライブラリの変換

カスタム タグの実行時の動作をエミュレートするためにサポート クラスが作成される Java 言語クラスの一覧を示します。

変換後のサーブレット クラスを開く前のコンパイル

変換後のサーブレット クラスの Web フォームを開く方法について説明します。

トラブルシューティング : コード ページを一致させるには

コード ページが存在しない場合や一致しない場合の対処方法について説明します。

未変換コードの手動アップグレード

自動的に変換できなかったコードをアップグレードする方法について説明します。

C# リファレンス

Visual C# 言語の概要を説明します。

Java Language Conversion Assistant 3.0 の新機能

このバージョンの Java Language Conversion Assistant には、次の新機能が含まれています。

- EJB アプリケーションの変換のサポート
- CORBA アプリケーションの変換のサポート
- RMI アプリケーションの変換のサポート
- JMS アプリケーションの変換のサポート
- シリアル化アプリケーションの変換のサポート
- JNDI を使用するアプリケーションの変換のサポート
- JAAS アプリケーションの変換のサポート
- JCE アプリケーションの変換のサポート
- Java Swing アプリケーションの変換のサポート
- JAXP アプリケーションの変換のサポート
- TRAX アプリケーションの変換のサポート
- JavaMail を使用するアプリケーションの変換のサポート
- コマンド ライン プロパティ スイッチ

EJB

Enterprise JavaBeans は、すべての種類の Enterprise JavaBeans (Message-driven beans、Session beans、および Entity beans) をサポートするために System.EnterpriseServices 名前空間のクラスに変換されます。

CORBA

CORBA (Common Request Broker Architecture) のクラスは、分散コンピューティングをサポートするために System.Runtime.Remoting 名前空間のクラスに変換されます。

RMI

RMI (Remote Method Invocation) のクラスは、分散アプリケーションとリモート オブジェクトをサポートするために **System.Runtime.Remoting** 名前空間のクラスに変換されます。

JMS

JMS (Java Message Service) のクラスは、Windows メッセージ キューを使用する、System.Messaging 名前空間のクラスに変換されます。

シリアル化アプリケーション

java.io.Serializable インターフェイスを実装するクラスは、System.Runtime.Serialization.ISerializable インターフェイスを実装するように変換されます。

JNDI を使用するアプリケーション

JNDI (Java Naming and Directory Interface) のクラスは、名前付けサービスとディレクトリ サービスを提供するために System.DirectoryServices 名前空間のクラスに変換されます。一部のメソッドは、RMI と CORBA のリモート操作をサポートするために **System.Runtime.Remoting** のメソッドに変換されます。

JAAS

JAAS (Java Authentication and Authorization Service) のクラスは、すべてのセキュリティ サービスと認証サービスを提供するために System.Security 名前空間のクラスに変換されます。

JCE

JCE (Java Cryptography Extension) のクラスは、メッセージの暗号化と復号化を提供してセキュリティを強化するために System.Security.Cryptography 名前空間のクラスに変換されます。

Java Swing

javax.swing パッケージのクラスは、ユーザー インターフェイス コンポーネントおよびコントロールを提供するために System.Windows.Forms 名前空間のクラスに変換されます。

JAXP

Java API for XML 処理のクラスは、System.Xml 名前空間のクラスに変換されます。SAX モデルと DOM モデルの両方がサポートされます。

TRAX

Transformation API for XML のクラスは、System.Xml.Xsl 名前空間のクラスに変換されます。

JavaMail を使用するアプリケーション

JavaMail API のクラスは、SMTP (Simple Mail Transfer Protocol) を使用したメッセージの作成および送信をサポートするために System.Web.Mail 名前空間のクラスに変換されます。

プロパティ スイッチ

get/set/is メソッドをメソッドのままにしておくか、プロパティに変換するかを選択できるようにするために、コマンド ライン スイッチ (**/ProcessGetSetOff**) が提供されます。既定では、JLCA はこれらのメソッドをプロパティに変換します。コマンド ライン スイッチを切り替えて、これらのメソッドの大部分をメソッドのままにすることもできます。他の変換との互換性を保つために、次のパターンは常にプロパティに変換されます。

- ActiveX のメソッドは、.NET Framework の特定のプロパティに変換されます。
- CORBA 属性。
- .NET Framework で必要なアクセサ、ミューテータ、および状態チェックの各メソッド。
- 自動的に生成された .NET Framework のプロパティ。

bean のプロパティの設定をエミュレートするために使用される SupportClass コードでは、**get/set** メソッドではなくプロパティが想定されるため、ワイルドカード文字 (*) に設定されたプロパティと共に **setProperty** メソッドを使用する JSP ページは正しく変換されません。

Taglib ハンドラ クラスのプロパティは **get/set** メソッドに変換されます。また、カスタム コントロールはプロパティを使用してカスタム HTML タグ内の ASPX 属性への対応付けを行うため、プロパティ値を設定するための生成済み ASPX ページで使用されるコードは破壊されます。

ビジュアルなコンポーネントのアクセサ メソッドが、コンポーネントを所有するスレッド以外のスレッドで呼び出される場合、そのメソッドは Invoke メソッドに変換されます。このメソッドは、**ProcessGetSetOff** フラグがオンの場合に **object** 型を返します。戻り値は、元のメソッドの戻り値と同じ型にキャストする必要があります。

サポートされない Java 言語パッケージ

CORBA と .NET Framework リモート処理のアーキテクチャに違いがあるため、CORBA の次のパッケージとクラスはサポートされません。

- **org.omg.CosNaming**
- **org.omg.CosNamingContextExtPackage**
- **org.omg.Dynamic**
- **org.omg.IOP**
- **org.omg.IOP.CodecFactoryPackage**
- **org.omg.IOP.CodecPackage**
- **org.omg.Messaging**
- **org.omg.PortableInterceptor**
- **org.omg.PortableInterceptor.ORBInitInfoPackage**

Swing と Windows フォームのアーキテクチャに違いがあるため、次の Swing パッケージはサポートされません。

- **javax.swing.plaf**
- **javax.swing.plaf.basic**
- **javax.swing.plaf.metal**

- **javax.swing.plaf.multi**

サードパーティ ドライバを実装するためのインターフェイスが含まれるため、次のパッケージはサポートされません。

- **javax.sql**

参照

関連項目

コマンド ライン スイッチ

その他の技術情報

Java アプリケーションの Visual C# への変換

各種 Java 言語アプリケーションの変換

Visual J++ または Java 言語から Visual C# へのプロジェクトの変換

Java Language Conversion Assistant を使用すると、Visual J++ 6.0 プロジェクトや Java 言語のファイルを変換し、Visual C# で開発を続行できます。Java Language Conversion Assistant は、既存のプロジェクトを変更するのではなく、元のプロジェクトに基づいて新しい Visual C# プロジェクトを作成します。変換時に生成されるエラー、警告、問題は、新しいプロジェクトが生成された後で、変換レポートに表示されます。自動的に変換できないプロジェクトの部分を手動で変換できるように、これらのエラー、警告、および問題も変換後のコードのコメントに出力されます。

メモ：

Visual C# に慣れていない場合は、この処理を実行する前に、多少の時間をかけて習熟しておく必要があります。詳細については、「C# 言語と .NET Framework の概要」を参照してください。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual J++ プロジェクト

Visual J++ プロジェクトを変換するには

1. Visual Studio を起動します。
2. [ファイル] メニューの [開く] をポイントし、[変換] をクリックします。
3. [Java Language Conversion Assistant] をクリックし、[OK] をクリックします。
4. [ソース ファイル] ページの [Visual J++ 6.0 プロジェクト] をクリックします。
5. [プロジェクト ファイルの選択] ページの [参照] をクリックします。
6. 該当する .vjp ファイルを見つけ、選択します。

   メモ：

   .vjp ファイルの CLASSPATH ディレクティブが .jar ファイルまたは .class ファイルを指定している場合、これらファイルは無視されます。

7. [新しいプロジェクトを作成するディレクトリを指定してください。] ページで、作成する新しいプロジェクトの名前およびディレクトリを指定します。
8. [変換の開始] ページの [次へ] をクリックします。

Java 言語のプロジェクト

Java 言語のプロジェクトを変換するには

1. Visual Studio を起動します。
2. [ファイル] メニューの [開く] をポイントし、[変換] をクリックします。
3. [Java Language Conversion Assistant] をクリックし、[OK] をクリックします。
4. [ソース ファイル] ページの [プロジェクト ファイルを含むディレクトリ] をクリックします。
5. [ソース ディレクトリの選択] ページの [参照] をクリックします。
6. 該当するプロジェクトを見つけ、選択します。

   メモ：

選択したディレクトリ内のファイルは表示されませんが、そのディレクトリ内のすべての .jav ファイルおよび .java ファイルが変換されます。Java Language Conversion Assistant では、最も無関係なファイルは無視されますが、変換を始める前にソース フォルダをクリーンアップする必要があります。

7. プロジェクトに必要な他のファイルを 2 番目のテキスト ボックスに追加します。
8. [新しいプロジェクトの構成] ページで次の項目を指定します。
   - 作成するプロジェクトの名前。
   - プロジェクトの出力の種類。
9. [新しいプロジェクトを作成するディレクトリを指定してください。] ページで、作成する新しいプロジェクトの名前およびディレクトリを指定します。
10. [変換の開始] ページの [次へ] をクリックします。
11. 自動的に変換できなかったコードを修正します。詳細については、「未変換コードの手動アップグレード」を参照してください。

参照

処理手順

未変換コードの手動アップグレード
Java Language Conversion Assistant ウィザード
トラブルシューティング : コード ページを一致させるには

関連項目

[変換] ダイアログ ボックス
コマンド ライン スイッチ

概念

Java Language Conversion Assistant

Java Language Conversion Assistant ウィザード

Java Language Conversion Assistant ウィザードを使用して、Visual J++ および Java 言語のプロジェクトを Visual C# に変換できます。このウィザードを使用すると、Visual J++ や Java 言語から Visual C# に変換するために、新しいプロジェクトを作成し、元のプロジェクトのファイルをコピーできます。変換処理で発生したエラー、警告、問題の詳細を示すレポートが生成されます。エラー、警告、および問題は、新しいプロジェクトのコードのコメントとして注記され、タスク一覧で表示できます。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Java Language Conversion Assistant ウィザードを表示するには

1. [ファイル] メニューの [開く] をポイントし、[変換] をクリックします。
2. [Java Language Conversion Assistant] をクリックし、[OK] をクリックします。

   セキュリティに関するメモ Java から変換したすべての C# コードは、セキュリティ問題に対応した Java Language Conversion Assistant ツールを使用して見直す必要があります。保護されていない Java コードは、保護されていない C# コードに変換されます。また、このツールは、認証などのセキュリティに関連するいくつかのクラスを含む、特定の Java クラスのコードの移行を行いません。この場合、アップグレード レポートには、移行されなかったコードが注記されます。このレポートを確認し、セキュリティ問題の発生を軽減することが重要です。

参照

処理手順

Visual J++ または Java 言語から Visual C# へのプロジェクトの変換
未変換コードの手動アップグレード

概念

Java Language Conversion Assistant

Java Language Conversion Assistant

Java Language Conversion Assistant は、Visual J++ 6.0 プロジェクトおよび Java 言語のファイルを Visual C# に変換するツールです。これらのファイルを Visual C# に変換すると、.NET Framework の利点を活かしながら、既存のコード ベースを活用できます。

新しい Visual C# プロジェクトには、既存の Visual J++ または Java 言語のコードから自動的に生成できる新しい Visual C# コードがすべて含まれます。詳細については、「Visual J++ または Java 言語から Visual C# へのプロジェクトの変換」を参照してください。

Java Language Conversion Assistant を使用すると、Visual J++ または Java 言語のアプリケーション、およびアプレット プロジェクトを変換できます。これらのプロジェクトは次のように変換されます。

変換前	変換後
アプリケーション	Windows フォーム アプリケーション
アプレット	Web ユーザー コントロール
JSP ページまたはサーブレット	Web アプリケーション

変換後の Web ユーザー コントロールは、アプレットの場合と同様にブラウザでホストできます。HTML ページでは、ホスト対象のコントロールを `APPLET` タグではなく `OBJECT` タグで宣言します。コントロールを指定するには `classid` 属性を使用します。この属性では、コントロールのパスとコントロールの完全修飾名を指定し、両者をシャープ記号 (#) で区切ります。次に例を示します。

```
<OBJECT id="myControl" classid="http:ControlLibrary1.dll#ControlLibrary1.myControl" VIEWAST
EXT></OBJECT>
```

コントロールを正しく表示するには、コントロールを格納した .dll ファイルを、コントロールを表示する Web ページと同じ仮想ディレクトリに置くか、グローバル アセンブリ キャッシュにインストールする必要があります。

サポート クラス

Visual C# では使用できない元のプロジェクトの機能を変換するために、Java Language Conversion Assistant は、元の機能を複製する "サポート クラス" を作成します。"サポート クラス" は "マネージャ" とも呼ばれます。サポート クラスは、エミュレートする対象のクラスとはアーキテクチャが大きく異なる場合があります。変換後のプロジェクトではアプリケーションの元のアーキテクチャができる限り保持されますが、サポート クラスの最終的な目標は元の機能を複製することです。

変換レポート

プロジェクト内の一部のコードは、自動的に変換できない場合があります。Java Language Conversion Assistant ウィザードを実行した後、変換処理の間に検出されたすべてのエラー、警告、および問題の詳細を示す変換レポートを表示できます。新しいプロジェクトのコードのうち、未変換のコードには UPGRADE_TODO というコメントが付きます。変換コメントはタスク一覧に表示できます。各変換コメントには、コードを手動で変換する方法を示すヘルプ トピックへのリンクが含まれています。詳細については、「未変換コードの手動アップグレード」を参照してください。

セキュリティに関するメモ：
Java から変換したすべての C# コードは、セキュリティ問題に対応した Java Language Conversion Assistant ツールを使用して見直す必要があります。保護されていない Java コードは、保護されていない C# コードに変換されます。また、このツールは、認証などのセキュリティに関連するいくつかのクラスを含む、特定の Java クラスのコードの移行を行いません。この場合、アップグレード レポートには、移行されなかったコードが注記されます。このレポートを確認し、セキュリティ問題を防ぐことが重要です。

参照

処理手順

Visual J++ または Java 言語から Visual C# へのプロジェクトの変換
未変換コードの手動アップグレード

その他の技術情報

Java アプリケーションの Visual C# への変換

未変換コードの手動アップグレード

Java Language Conversion Assistant で Visual J++ プロジェクトまたは Java 言語のファイルを変換すると、自動的に変換できなかったコードが新しい Visual C# プロジェクトに含まれている場合があります。新しいプロジェクトのコードには、そのコードを Visual C# に手動で変換できるようにコメントが挿入されます。

コメント タイプ	説明
UPGRADE_TODO	自動的に変換できなかったコード
UPGRADE_WARNING	問題のあるコード
UPGRADE ISSUE	問題のあるコード
UPGRADE_NOTE	元のコードとは異なる動作をする可能性のあるコード

アプリケーションをコンパイルする前に修正する必要のある、コンパイラ エラーが存在する場合もあります。各コメントには、問題に関する短い説明と、コードの変換方法を説明するヘルプ トピックへのリンクが含まれます。

メモ：

使用している設定またはエディションによっては、ヘルプの記載と異なるダイアログ ボックスやメニュー コマンドが表示される場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

未変換コードを手動でアップグレードするには

- Java Language Conversion Assistant ウィザードを実行した後で、プロジェクトを Visual Studio で開きます。

参照

処理手順

Visual J++ または Java 言語から Visual C# へのプロジェクトの変換

概念

Java Language Conversion Assistant

トラブルシューティング：コード ページを一致させるには

変換後のコードに正しくない文字が含まれている場合は、文字エンコーディングのコード ページが存在しないか、一致していないことが主な原因です。Java Language Conversion Assistant では、次のような方法でエンコーディングの一致が行われます。

- 現在のシステム ANSI コード ページでエンコードされたファイルは、変換後もこのエンコーディングが維持されます。
- その他のエンコーディングはすべて UTF-8 エンコーディングに変換されます。

Unicode バイト オーダー マークで始まる Unicode ファイルは、自動的に Unicode として識別されます。ファイルの先頭にバイト オーダー マークがなく、エンコーディング スイッチが指定されていない場合、そのファイルはシステムの現在の ANSI コード ページであると見なされます。

現在使用しているシステム ANSI コード ページを変更するには

1. [コントロール パネル] を開き、[地域のオプション] (Windows 2000) または [地域と言語のオプション] (Windows XP) をダブルクリックします。
2. [詳細設定] をクリックし、適切なコード ページをクリックします。

ソース ファイルが非 ANSI 文字エンコーディングである場合や、Unicode ソース ファイルの先頭にバイト オーダー マークがない場合は、エンコーディング スイッチを使用する必要があります。

エンコーディング スイッチでは、次の文字エンコーディングを指定できます。

エンコーディング	スイッチ
ラテン語アルファベット No. 2 (中央ヨーロッパ)	ISO-8859-2
ラテン語アルファベット No. 3	ISO-8859-3
ラテン語アルファベット No. 4 (バルト諸国)	ISO-8859-4
ラテン語/キリル語アルファベット	ISO-8859-5
ラテン語/アラビア語アルファベット	ISO-8859-6
ラテン語/ギリシャ語アルファベット	ISO-8859-7
ラテン語/ヘブライ語アルファベット	ISO-8859-8
ラテン語アルファベット No. 9	ISO-8859-15
日本語	EUC-JP
韓国語	EUC-KR
簡体字中国語	EUC-CN
Chinese National Standard	GB18030
バイト オーダー マークがない UTF-8	UTF-8

特定のエンコーディングを使用するには、そのエンコーディングのサポートがシステムにインストールされている必要があります。たとえば、Windows 2000 Server コンピュータで GB18030 エンコーディングを使用するには、GB18030 サポート パッケージをダウンロードしてインストールする必要があります。詳細については、システムのマニュアルを参照してください。エンコーディング スイッチを使用する場合、プロジェクト内のすべてのファイルは同じエンコーディングである必要があります。

JLCA でサポートされていないエンコーディング システムを使用している場合は、変換できない文字が失われる可能性があります。状況によっては、ファイルが失われる可能性があります。

参照

概念

応答エンコーディングの変更

コマンド ライン スイッチ

次の表は、Java Language Conversion Assistant 3.0 で使用できるコマンド ライン スイッチです。

スイッチ	機能
/?	メッセージを印刷します。
/Out	ターゲット ディレクトリを指定します。既定値は \OutDir です。
/Verbose	すべて出力するように指定します。
/NoLogo	著作権画面を表示しないように指定します。
/NoLog	ログを書き込まないように指定します。
/LogFile	ログ ファイル名を指定します。
/Encoding	入力ファイルのエンコーディングを指定します。
/ProjectName	ディレクトリ変換のプロジェクト名を指定します。
/ProjectType	プロジェクトの種類を指定します。オプションには、EXE、WinExe、Library、および ASP.NET があります。既定値は WinExe です。
/VRoot	Internet Information Server の仮想ルートを指定します。
/ContextPath	仮想ルートで置き換えるコンテキスト パスを指定します。
/Domain	URL 用に置き換えるドメインを指定します。
/NoExtensibility	機能拡張マップを無効にします。
/JDK	変換に使用する JDK を指定します。オプションには VJ と J2EE があります。
/ProcessGetSetOff	プロパティ変換を無効にします。

プロパティ スイッチ

get/set/is メソッドをメソッドのままにしておくか、プロパティに変換するかを選択できるように、コマンド ライン スイッチ (**/ProcessGetSetOff**) が用意されています。既定では、JLCA はこれらのメソッドをプロパティに変換します。コマンド ライン スイッチを切り替えて、これらのほとんどをメソッドのままにすることもできます。他の変換との互換性のために、次のパターンは常にプロパティに変換されます。

- ActiveX のメソッドは、.NET Framework 内の特定のプロパティに変換されます。
- CORBA 属性。
- アクセサ、ミューテータ、および .NET Framework で必要となる状態チェック メソッド。
- .NET Framework プロパティを自動的に生成します。

プロパティがワイルドカード文字 (*) に設定された **setProperty** メソッドを使用する JSP ページは、正しく変換されません。bean プロパティの設定をエミュレートする SupportClass コードが **get/set** メソッドではなくプロパティを想定しているためです。

Taglib ハンドラ クラスのプロパティは、**get/set** メソッドに変換されます。また、プロパティ値を設定するために生成された ASPX ページで使用されるコードは、カスタム コントロールがプロパティを使用してカスタム HTML タグ内の ASPX 属性をマップするため、分割されます。

ビジュアル コンポーネントのアクセサ メソッドがコンポーネントを所有するスレッド以外のスレッドから呼び出された場合は、System.Windows.Forms.Control.Invoke メソッドに変換されます。このメソッドは、**ProcessGetSetOff** フラグをオンにすると、**object** 型

を返します。戻り値は、元のメソッドの戻り値と同じ型にキャストする必要があります。

参照

その他の技術情報

Java アプリケーションの Visual C# への変換

Web アプリケーションの変換

Web アプリケーションを JSP から ASP.NET に変換するときは、追加情報が必要です。Java Language Conversion Assistant によって、アプリケーションのルート ディレクトリが設定され、サーブレット クラスが別のディレクトリに移動され、リンクが置換され、Web.xml ファイルで定義されているタグ ライブラリ記述子が解決されます。URL 変換を強化するために Web アプリケーション用のコンテキスト パスとアプリケーション ドメインを設定できます。

このセクションの内容

Web アプリケーションの変換の種類の選択

JSP ページやサーブレット クラスを含む Web アプリケーションを変換するには、変換のセットアップ時に、変換と出力の種類を適切に選択する必要があります。[Java 言語ファイルのディレクトリ] を選択し、出力の種類を Web アプリケーションに設定します。

参照

その他の技術情報

Web アプリケーションの変換

アプリケーション ドメインとコンテキスト パスの設定

Web アプリケーションを変換するときは、まず出力の種類として Web アプリケーションを選択する必要があります。

Web アプリケーションの仮想ルートを入力した後は、アプリケーション内のリンクの変換方法を拡張するために、アプリケーション ドメインを指定するオプションが表示されます。アプリケーション ドメインを指定しない場合は、すべてのリンク (外部リンクを含む) が ASP.NET リンクに変換されます。

アプリケーションのコンテキスト パスを指定することもできます。

参照

概念

Web アプリケーションの変換の種類の選択

その他の技術情報

Web アプリケーションの変換

要求パラメータのエンコーディング

Java 言語では、サーブレット コンテナは、ISO 8859-1 エンコーディングを使用して要求パラメータをエンコードします。Web アプリケーションが別のエンコーディングを扱う場合は、通常、以下のいずれかの例のコードを含みます。

例

```
String text = request.getParameter("text");
text = new String( text.getBytes("ISO-8559-1"), charset )
```

もう 1 つの例を次に示します。

```
String text = request.getParameter("text");
BufferedReader reader = new BufferedReader(new InputStreamReader(new StringBufferInputStrea
m(text), charset));
text = reader.readLine();
```

どちらの場合も、*charset* パラメータは、デコードする要求のエンコーディングを表す文字列です。

.NET Framework では、ASP.NET と、Web.xml ファイルで設定されるパラメータによってデコードが自動的に行われるため、次の例に示すように、このコードは不要です。

例

```
<globalization requestEncoding="euc-jp" responseEncoding="euc-jp"/>
```

Web アプリケーションを変換した場合は、この機能を行う変換後の Java 言語コードはもはや不要であるため、該当のコードを削除できます。

参照

その他の技術情報

Web アプリケーションの変換

Web アプリケーションのディレクトリ構造の違い

Web アプリケーションを変換するときに、Java Language Conversion Assistant によって、新しいアプリケーション用に別のディレクトリ構造が設定されます。

すべてのサーブレット クラスが、Web アプリケーションのルート ディレクトリに移動されます。変換のプロセスで作成されたディレクトリ構造は、元の Java 言語のプロジェクトでサーブレットが配置されているディレクトリに必ずしも対応してはいません。

例

次の例では、Web アプリケーションのルート ディレクトリは \website\AppName です。サーブレット クラスは、変換後のディレクトリ構造では別のディレクトリに移動されます。

元のディレクトリ構造

```
mt\src
mt\src\com\ais\servlets
mt\src\com\ais\servlets\Servlet1.java
mt\src\com\ais\servlets\Servlet2.java
mt\src\com\ais\beans
mt\src\com\ais\beans\CatalogBean.java
mt\src\com\ais\customtags
mt\src\com\ais\customtags\TableTag.java
mt\misc\pack1\myclasses
mt\misc\pack1\myclasses\DbHelper.java
mt\website
mt\website\AppName
mt\website\AppName\jsp\
mt\website\AppName\jsp\Search.jsp
mt\website\AppName\jsp\content\Catalog.jsp
mt\website\AppName\html\Index.html
mt\website\AppName\images\Logo.gif
mt\website\AppName\WEB-INF
mt\website\AppName\WEB-INF\Web.xml
mt\website\AppName\WEB-INF\MyTags.tld
```

相当する変換後のディレクトリ

```
outdir\src
outdir\src\com\ais\beans
outdir\src\com\ais\beans\CatalogBean.cs
outdir\src\com\ais\customtags
outdir\src\com\ais\customtags\TableTag.cs
outdir\misc\pack1\myclasses
outdir\misc\pakc1\myclasses\Dbhelper.cs
outdir\website
outdir\website\AppName
outdir\website\AppName\com\ais\servlets\Servlet1.aspx
outdir\website\AppName\com\ais\servlets\Servlet2.aspx
outdir\website\AppName\jsp\
outdir\website\AppName\jsp\Search.aspx
outdir\website\AppName\jsp\content\Catalog.aspx
outdir\website\AppName\html\Index.html
outdir\website\AppName\images\Logo.gif
outdir\website\AppName\WEB-INF
outdir\website\AppName\WEB-INF\Web.xml
outdir\website\AppName\WEB-INF\MyTags.tld
```

参照

その他の技術情報

Web アプリケーションの変換

リンクの置換

Java Language Conversion Assistant は、変換後のディレクトリ構造で動作するようにすべてのリンクを置換します。ここでは、Web アプリケーションのディレクトリ構造の違いで定義されているディレクトリ構造を仮定します。

URL とその変更方法

URL には基本的に 3 つの種類があります。

- ドメイン情報を含む絶対 URL

```
http://www.example.com/servlet/pack1/Servlet1
http://www.example.com/page.jsp
```

- ドメイン情報を含まない絶対 URL

```
/servlet/pack1/Servlet1
/directory/page2.jsp
```

- 相対 URL

```
pack1.Servlet1
page3.jsp
```

クライアントが Web アプリケーションにアクセスするために使用する URL は、次のような形式です。

```
http://hoststring/ContextPath/resource
```

この例では、要素は次のように定義されます。

hoststring

仮想ホストまたは *hostname*:*portNumber* にマップされるホスト名。

ContextPath

アプリケーションにアクセスするディレクトリ構造。

resource

Web.xml 構成ファイルで定義されるファイル、JSP ページ、またはサーブレットへの参照。

Java Language Conversion Assistant は次のような変更を行います。

- コンテキスト パスが存在する場合は、そのコンテキスト パスを、変換で定義される対応する仮想ルートの名前と Web アプリケーション ルートに変更します。
- コンテキスト パスが存在しない場合は、リンクに VROOT が挿入されます。
- ファイル拡張子が .jsp である場合は、.aspx に変更されます。
- Web.xml 構成ファイルを使用してサーブレット参照を検索し、変換します。

例

元の URL

```
http://hoststring/ContextPath/dir/JSP1.jsp?param1=1
http://HostNameProvided/dir/JSP1.jsp?param1=1
```

相当する変換後の URL

```
http://hoststring/VRoot/dir/JSP1.aspx?param1=1
```

```
http://HostNameProvided/VRoot/dir/JSP1.aspx?param1=1
```

リンクにサーブレット トークンとサーブレットの完全な名前が含まれている場合、サーブレット トークンは削除され、完全な名前に含まれるドット (.) はスラッシュ (/) に変更されます。サーブレットへのリンクが相対的である場合、参照は VROOT と変換後のサーブレットのディレクトリ構造を含めるように変更されます。

例

元の URL

```
http://hoststring/ContextPath/servlet/com.ais.servlets.Servlet1
com.ais.servlets.Servlet1
```

相当する変換後の URL

```
http://hoststring/VRoot/com/ais/servlets/Servlet1.aspx
/VRoot/com/ais/servlets/Servlet1.aspx
```

サーブレット トークンの後に Web.xml ファイルにあるサーブレット名が続く場合は、サーブレットの完全な名前がリンクに挿入されます。

例

元の XML

```
<servlet>
    <servlet-name>test</servlet-name>
    <servlet-class>com.ais.servlets.Servlet1</servlet-class>
</servlet>
```

元の URL

```
http://hoststring/ContextPath/servlets/test
```

相当する変換後の URL

```
http://hoststring/VRoot/com/ais/servlets/Servlet1.aspx
```

Web アプリケーション名の後に Web.xml ファイルのサーブレット マップで定義されているパターンが続く場合は、サーブレットの名前が決定され、リンクに挿入されます。パターンにはワイルドカード文字 (*) を含めることができます。

例

元の XML

```
<servlet>
    <servlet-name>test</servlet-name>
    <servlet-class>com.ais.servlets.Servlet1</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>test</servlet-name>
    <url-pattern>/*.asp</url-pattern>
</servlet-mapping>
```

元の URL

```
http://hoststring/ContextPath/abc.asp
```

相当する変換後の URL

```
http://hoststring/VRoot/com/ais/servlets/Servlet1.aspx
```

実行時に計算されるリンクは、Java Language Conversion Assistant では変換されません。そのため、コードを手動で変更し、適切な ASP.NET リンクを作成する必要があります。これは、次のような場合に当てはまります。

- URL パラメータ値が実行時に計算される、`<%=%>`、`jsp:forward`、`jsp:include`、または `jsp:execute` の各式、および `<a href>` と `<form action>` の HTML タグを持つリンク。

- そのパラメータが Web リソースへのパスを表す任意の **javax.servlet** メソッド。たとえば、URL パラメータ値が実行時に計算される **Response.redirect** メソッドがあります。

例

元の Java 言語のコード

```
<jsp:include page="<%=value%>">
```

相当する Visual C# のコード

```
<%-- //UPGRADE_TODO: Expected format of parameters of action jsp:include are different in t
he equivalent in .NET Framework --%>
<% Server.Execute(value)%>
```

参照

概念

Web アプリケーションのディレクトリ構造の違い

その他の技術情報

Web アプリケーションの変換

タグ ライブラリ記述子の解決

taglib ディレクティブの **uri** 属性が Web.xml ファイルで定義されている **taglib-uri** を参照する場合、タグ ライブラリ記述子 (TLD) の場所にはそのファイルの情報が使用されます。

例

元の XML

```
<taglib>
    <taglib-uri>/MYTEST</taglib-uri>
    <taglib-location>WEB-INF/myTagLib.tld</taglib-location>
</taglib>
```

元の JSP

```
<%taglib uri="/MYTEST" prefix="myTag"%>
```

解析される相当する TLD

```
/WEB-INF/myTagLib.tld
```

参照

その他の技術情報

Web アプリケーションの変換

応答エンコーディングの変更

変換対象の Java 言語の Web アプリケーションに、出力エンコーディングが UTF-8 以外の JSP ページや HTML ページが含まれている場合は、Web.config ファイルを変更すると、生成された ASP.NET Web アプリケーションの既定の応答エンコーディングを変更できます。**Globalization** 要素を検索し、**responseEncoding** 属性を目的のエンコーディングに変更します。

参照

処理手順

トラブルシューティング : コード ページを一致させるには

JSP カスタム タグ ライブラリの変換

JSP カスタム タグと .NET Framework Web ユーザー コントロールとの違いによって、JSP カスタム タグの処理をエミュレートするサポート クラスのセットが変換時に作成されます。これらのクラスは、変換後の JSP カスタム タグ クラス ハンドラのベースになります。サポート クラスは、**javax.servlet.jsp.tagext** パッケージ内の次のクラスについて作成されます。

元の Java 言語のクラス	生成されるサポート クラス
BodyContent	**WCBodyContent**
Tag	**WCBase**
IterationTag	**WCIterationBase**
BodyTag	**WCBodyBase**
BodyTagSupport	**WCBodyImpl**
TagSupport	**WCIterationImpl**

参照

その他の技術情報

Web アプリケーションの変換

変換後のサーブレット クラスを開く前のコンパイル

サーブレット クラスを変換するときには、アプリケーションの Web フォーム ページを開く前に、変換のすべての問題を解決し、コードをコンパイルする必要があります。コードをコンパイルするまで、Web フォーム ページをデザイナ ビューで開くことはできません。コンパイル前に Web フォーム ページを開こうとすると、エラーが発生します。

参照

その他の技術情報

Web アプリケーションの変換

Web タグの例

System.Web.UI.HtmlTextWriter クラスのメソッド、および単一のコントロールで複数の HTML 要素を作成する方法を次のコード例に示します。

例

```
using System;
using System.Web;
using System.Web.UI;
using System.Collections.Specialized;
namespace CustomControls
{
    public class Rendered2 : Control, IPostBackDataHandler, IPostBackEventHandler
    {
        private String text1;
        private String text2;
        private String text = "Press button to see if you won.";
        private int number = 100;
        private int Sum
        {
            get
            {
                return Int32.Parse(text1) + Int32.Parse(text2);
            }
        }
        public int Number
        {
            get
            {
                return number;
            }
            set
            {
                number = value;
            }
        }
        public String Text {
            get
            {
                return text;
            }
            set
            {
                text = value;
            }
        }
        public event CheckEventHandler Check;
        protected virtual void OnCheck(CheckEventArgs ce)
        {
            if (Check != null)
            {
                Check(this,ce);
            }
        }
        public virtual bool LoadPostData(string postDataKey, NameValueCollection values)
        {
            text1 = values[UniqueID + "t1"];
            text2 = values[UniqueID+ "t2"];
            Page.RegisterRequiresRaiseEvent(this);
            return false;
        }
        public virtual void RaisePostDataChangedEvent() {
        }
        public void RaisePostBackEvent(string eventArgument)
        {
```

```
            OnCheck(new CheckEventArgs(Sum - Number));
        }
        protected override void Render(HtmlTextWriter writer)
        {
            writer.RenderBeginTag(HtmlTextWriterTag.H3);
            writer.Write("Enter a number:");
            writer.RenderEndTag();
            writer.AddAttribute(HtmlTextWriterAttribute.Type,"Text");
            writer.AddAttribute(HtmlTextWriterAttribute.Name,this.UniqueID + "t1");
            writer.AddAttribute(HtmlTextWriterAttribute.Value,"0");
            writer.RenderBeginTag(HtmlTextWriterTag.Input);
            writer.RenderEndTag();
            writer.RenderBeginTag(HtmlTextWriterTag.H3);
            writer.Write("Enter another number:");
            writer.RenderEndTag();
            writer.AddAttribute(HtmlTextWriterAttribute.Type,"Text");
            writer.AddAttribute(HtmlTextWriterAttribute.Name,this.UniqueID + "t2");
            writer.AddAttribute(HtmlTextWriterAttribute.Value,"0");
            writer.RenderBeginTag(HtmlTextWriterTag.Input);
            writer.RenderEndTag();
            writer.RenderBeginTag(HtmlTextWriterTag.Br);
            writer.RenderEndTag();
            writer.AddAttribute(HtmlTextWriterAttribute.Type,"Submit");
            writer.AddAttribute( HtmlTextWriterAttribute.Name,this.UniqueID);
            writer.AddAttribute(HtmlTextWriterAttribute.Value,"Submit");
            writer.AddStyleAttribute(HtmlTextWriterStyle.Height,"25 px");
            writer.AddStyleAttribute(HtmlTextWriterStyle.Width,"100 px");
            writer.RenderBeginTag(HtmlTextWriterTag.Input);
            writer.RenderEndTag();
            writer.RenderBeginTag(HtmlTextWriterTag.Br);
            writer.RenderEndTag();
            writer.RenderBeginTag(HtmlTextWriterTag.Span);
            writer.Write(this.Text);
            writer.RenderEndTag();
        }
    }
}
//CheckEvent.cs.
//Contains the code for the custom event data class CheckEventArgs.
//Also defines the event handler for the Check event.
using System;
namespace CustomControls
{
    public class CheckEventArgs : EventArgs
    {
        private bool match = false;
        public CheckEventArgs (int difference)
        {
            if (difference == 0)
            {
                match = true;
            }
        }
        public bool Match
        {
            get
            {
                return match;
            }
        }
    }
    public delegate void CheckEventHandler(object sender, CheckEventArgs ce);
}
```

System.Web.UI.WebControls.WebControl クラスのコンストラクタを使用して HTML `TextArea`要素を作成し、それを Web フォーム ページに表示する方法を次のコード例に示します。

```
<font face="Courier New" size="2" color="#000080">
<%@ Page Language="C#" %>
<html>
<head>
<script runat="server">
void Button1_Click(Object sender, EventArgs e)
{
    WebControl wc = new WebControl(HtmlTextWriterTag.Textarea);
    PlaceHolder1.Controls.Add(wc);
}
</script>
</head>
<body>
<form runat="server">
<h3>WebControl Constructor Example</h3>
<p>
<asp:PlaceHolder id="PlaceHolder1" runat="Server" />
<br>
<asp:Button id="Button1" Text="Click to create a new TextArea" OnClick="Button1_Click" runa
t="Server" />
<p>
</form>
</body>
</html>
```

WriteBeginTag メソッドを使用して、指定した HTML 要素のタブ間隔および開始タグを **HtmlTextWriter** 出力ストリームに書き込みます。

このメソッドは HTML タグの終了文字 (>) を書き込まないため、要素に HTML 属性を追加できます。タグを閉じるには、TagRightChar 定数を使用します。**WriteBeginTag** を SelfClosingTagEnd 定数と使用して、自己終了型の HTML 要素を書き込みます。

このメソッドは、タグまたは属性のマップを許可せず、HTML 要素を各要求に対して同じように描画するカスタム サーバー コントロールによって使用されます。

```
// Create a manually rendered tag.
writer.WriteBeginTag("img");
writer.WriteAttribute("alt", "AtlValue");
writer.WriteAttribute("myattribute", "No "encoding " required", false);
writer.Write(HtmlTextWriter.TagRightChar);
writer.WriteEndTag("img");
writer.WriteLine();
writer.Indent--;
writer.RenderEndTag();
```

次の例に示すように、インスタンスを取得できます。

```
HtmlTextWriterTag myTag;
myTag = HtmlTextWriterTag.Area;
```

参照

関連項目

System.Web
HtmlTextWriter
WebControl

各種 Java 言語アプリケーションの変換

ここでは、Java Language Conversion Assistant を使用して各種のアプリケーションを変換するときに、実行する処理と発生する可能性がある問題について説明します。

このセクションの内容

JavaBeans アプリケーションの変換

JavaBeans の変換について説明します。必要となる手動の処理についても示します。

EJB アプリケーションの変換

Enterprise JavaBeans テクノロジを使用するアプリケーションの変換処理に関する主な問題について説明します。

CORBA アプリケーションの変換

CORBA アプリケーションの変換処理について説明します。CORBA と .NET Framework リモート処理のアーキテクチャの違いについても示します。

RMI アプリケーションの変換

RMI (Remote Method Invocation) アプリケーションの変換について説明します。

JMS アプリケーションの変換

JMS (Java Message Service) アプリケーションを変換するときに発生する問題について説明します。

シリアル化されたアプリケーションの変換

シリアル化が必要なアプリケーションの変換処理について説明します。

JNDI を使用しているアプリケーションの変換

JNDI (Java Naming and Directory Interface) を使用するアプリケーションの変換の問題について説明します。

データ パッケージの変換

データベース アプリケーションの変換について説明します。

JAAS アプリケーションの変換

JAAS (Java Authentication and Authorization Service) アプリケーションに関する変換の問題について説明します。

JCE アプリケーションの変換

暗号化アプリケーションの変換の問題について説明します。

Java Swing アプリケーションの変換

Java Swing アプリケーションに関する変換の問題について説明します。

ActiveX コントロールの変換

Microsoft ActiveX コントロールと ActiveX オートメーション オブジェクトに関する変換の問題について説明します。

Javax.swing のサンプル

Java Swing アプリケーションの変換用のコード サンプルを示します。このコード サンプルでは、Windows フォーム コントロールの **IExtenderProvider** インターフェイス、ユーザー定義イベント、および **DataSource** プロパティが使用されています。

方法 : リモート処理のセキュリティ レベルを設定する

既定のセキュリティ レベルを調整して、オブジェクトをシリアル化する方法について説明します。

関連するセクション

[変換] ダイアログ ボックス

プロジェクトで使用する変換ツールを選択できる [変換] ダイアログ ボックスについて説明します。

Web タグの例

Web アプリケーションの変換用のコード サンプルを示します。このコード サンプルでは、**System.Web.UI.HtmlTextWriter** クラスと **System.Web.UI.WebControl** クラスが使用されています。

コマンド ライン スイッチ

Java Language Conversion Assistant で使用できるコマンド ライン スイッチ、特にプロパティ スイッチについて説明します。

JavaBeans アプリケーションの変換

JavaBeans アプリケーションを変換する前に、アプリケーションにマニフェスト ファイルがあることを確認します。ない場合は、他のファイルと同じように処理されます。マニフェスト ファイルは、MANIFEST.MF というファイル名で、META-INF という名前のフォルダにあります。既存のファイルの名前を必要に応じて変更できます。

マニフェスト ファイル内で標準規則によって JavaBeans と識別されるアプリケーションだけが、JavaBeans アプリケーションと見なされて変換されます。次に例を示します。

```
Name: x\y class
Java-Bean: True | False
```

この例では、x はパッケージを表し、y は bean クラス コンポーネントを表しています。

bean を正確に識別することが、その bean がどのクラスに変換されるか、および関連付けられた **BeanInfo** クラスに含まれる情報が変換処理で適用されるかの両方に影響します。たとえば、**JPanel** オブジェクトは、通常、System.Windows.Forms.Panel オブジェクトに変換されます。ただし、そのオブジェクトがマニフェスト ファイルで JavaBeans として識別されている場合は、System.Windows.Forms.UserControl オブジェクトに変換されます。

java.awt.Panel クラスまたは **javax.swing.JPanel** クラスから継承する Visual JavaBeans は、**System.Windows.Forms.UserControl** クラスから継承するように変換されます。

関連付けられている **BeanInfo** の各クラスの情報は変換処理中に適用されますが、.NET Framework には直接これらのクラスに相当するものはありません。

参照

関連項目

UserControl

EJB アプリケーションの変換

エンタープライズ JavaBeans (EJB) アプリケーションを変換すると、変換後のプロジェクトには EJB 関連クラスとインターフェイス ファイルのみが含まれます。アプリケーションが他のユーティリティ クラスに依存している場合は、そのクラスを手動でプロジェクトに追加する必要があります。すべてのユーティリティ クラスは既定でクライアント プロジェクトに含まれ、必要なら手動で除外できます。

.NET Framework では名前付け検索の方法が異なるため、名前付けコンテキストに関連してコンパイル エラーが発生します。関連するコードは Visual C# ファイルでコメント アウトできます。

変換後にコンポーネントのセキュリティを手動で設定する必要があります。

メッセージ ドリブン bean はコンポーネントに変換されるため、出力をコンソールに書き込むすべてのコードを確認します。

コンポーネントを配置するときは、DLL に厳密な名前で署名する必要があります。キー ファイルを生成して、アセンブリに手動で統合します。

配置後に、変換したコンポーネントのトランザクション設定値を確認します。

コンポーネントを Windows 2000 Server に発行する場合、すべてのサービス コンポーネントの **PrivateComponent** タグを削除する必要があります。

クライアント プロジェクトをインストールした後、サービス コンポーネントへの参照を手動で追加します。

トランザクション設定

サービス コンポーネントを 1 つのトランザクション属性と関連付けることができます。Java 言語では、トランザクション属性はメソッド レベルで適用できます。相当するトランザクション属性を次の表に示します。

EJB	**.NET Framework**
NotSupported	NotSupported
Supports	Supported
Required	Required
RequiresNew	RequiresNew
必須	No .NET Framework equivalent
Never	No .NET Framework equivalent

変換時には、EJB 配置記述子内で指定されたトランザクション属性、および各属性が EJB メソッドと関連付けられる回数に基づいて、Java Language Conversion Assistant がコンポーネントの最も代表的な .NET Framework トランザクション属性を決定しようとします。メソッドに指定されている EJB トランザクション属性がコンポーネントに対して選択された .NET Framework トランザクション属性と異なる場合、Visual C# コードで警告が生成されます。これらの警告メッセージを探してください。

セキュリティ設定

プログラムまたは Component Services Explorer のいずれかで、次の .NET Framework セキュリティ設定を行います。

- 承認
- セキュリティ レベル
- 認証レベル
- 偽装レベル

セキュリティ チェックを実行するには承認が有効になっている必要があります。System.EnterpriseServices.ApplicationAccessControlAttribute クラスのプロパティを設定して、セキュリティ チェックを有効にします。これらのプロパティは残りの設定 (セキュリティ レベル、認証レベル、偽装レベル) を行うために使用されます。

セキュリティは、プロセス レベルで、またはプロセス レベルとコンポーネント レベルの両方で設定できます。System.EnterpriseServices.AccessChecksLevelOption 列挙型では次のオプションが定義されます。

- Application
- ApplicationComponent

セキュリティ チェックがプロセス レベルでのみ実行される場合、インターフェイス レベル、コンポーネント レベル、およびメソッド レベルでのロール ベースのセキュリティ設定は無視され、無効になります。変換後のアプリケーションを **ApplicationComponent** セキュリティ レベルに設定してください。

認証レベルは、ユーザー ID が .NET Framework アプリケーションに関して認証される方法を制御します。System.EnterpriseServices.AuthenticationOption 列挙型の次の値のいずれかを使用します。

- System.EnterpriseServices.AuthenticationOption.None
- Connect
- Call
- Packet
- Integrity
- Privacy

既定の認証レベルは **Packet** です。このレベルでは、ユーザーから送信されるすべてのパケットが認証されます。変換後のアプリケーションを **Packet** 認証レベルに設定してください。

偽装とは、元のアプリケーションから別のアプリケーションまたはセキュリティで保護されたリソースにアクセスするときに、クライアントのセキュリティ ID が渡される方法を指します。このため、サーバーはある程度までクライアントを偽装できます。System.EnterpriseServices.ImpersonationLevelOption 列挙型には次のオプションがあります。

- Anonymous
- Default
- Delegate
- Identify
- Impersonate

.NET Framework アプリケーションの既定の偽装レベルは **Impersonate** です。このレベルでは、サーバーはクライアントとして動作します。ただし、他のコンピュータ上のオブジェクトやリソースにサーバーがクライアントの代わりにアクセスすることはできません。変換後のアプリケーションを **Impersonate** レベルに設定します。

発行

JLCA は EJB プロジェクトを COM アプリケーションに変換します。EJB ソース ファイル フォルダに JAR ファイルと配置記述子ファイル Ejb-jar.xml が両方ともある場合は、JAR ファイルに含まれる配置記述子も解析されるため、変換後のプロジェクトではアプリケーションが 2 つ生成されます。

EJB が変換されるときに、.NET トランザクション属性のサービス コンポーネントへの割り当てが試みられます。1 つの EJB に対してローカル インターフェイスとリモート インターフェイスに異なるトランザクション属性がグローバルに割り当てられている場合、.NET トランザクション属性は割り当てられません。同様に、1 つのインターフェイスにトランザクション属性がグローバルに割り当てられ、他のインターフェイスにはメソッドごとのトランザクション属性が割り当てられている場合、.NET トランザクション属性は割り当てられません。メソッドごとのトランザクション属性の割り当ては、グローバルなトランザクション属性の割り当てがない場合にのみ、.NET トランザクション属性を決定するために使用されます。.NET サービス コンポーネントと Java 言語ではトランザクション属性の処理方法が異なるため、設定を再確認して必要な調整を行ってください。

変換中に既定のセキュリティ チェックが追加されますが、プロジェクトによってはこれが不要なことがあります。Java 言語のソース コードでメソッドごとのセキュリティ ロールを設定している場合や一定のメソッドを除外している場合は、変換後に設定を再確認してください。

参照

関連項目

System.EnterpriseServices

CORBA アプリケーションの変換

CORBA アプリケーションを変換する前に、OMG IDL ファイルがあることを確認してください。それ以外の場合、Java ファイルは他のクラスとして扱われます。

変換

必要なコードがソース ファイル内のスケルトンやスタブのコードに含まれていないことを確認してください。含まれている場合、変換によって失われます。

変換先として指定するディレクトリが、CLASSPATH に含まれている必要があります。

CORBA アプリケーションは、HTTP 経由の .NET Framework リモート処理に既定で変換されます。つまり、変換されたアプリケーションは CORBA クライアントと CORBA サーバーを使用しません。

.NET Framework リモート処理は、ネーム サービスや中間層が介在しないクライアント/サーバー接続を使用します。したがって、ORB (Object Request Broker) はありません。

CORBA ネーム サービスの作成に必要なコードは、.NET Framework リモート処理では不要となります。この中には、通常プロパティのハッシュテーブルや配列にパックされるクラスとコレクションが含まれます。

次のパッケージ内のクラスに対するすべての参照をコメント アウトします。

- **org.omg.CosNaming**
- **org.omg.CosNamingContextPackage**

CORBA アプリケーションは、最初に名前付けコンテキストを取得する必要があります。Java Language Conversion Assistant は、このコンテキストをリモート検索として扱うため、.NET Framework リモート処理では不要になります。次のコード行はコメント アウトできます。

```
NamingContext ncRef = (org.omg.CosNaming.NamingContext) Activator.GetObject(typeof(org.omg.
CosNaming.NamingContext), "http://<Host>:<Port>/<Name>");
```

セキュリティの問題

.NET Framework では、分散型通信 (リモート処理) の既定のセキュリティ レベルは **low** です。これは、ユーザー定義のオブジェクト型をリモート メソッドに渡す際に影響します。詳細については、「方法 : リモート処理のセキュリティ レベルを設定する」を参照してください。

参照

関連項目

System.Runtime.Remoting

概念

JNDI を使用しているアプリケーションの変換

RMI アプリケーションの変換

Java 言語では、RMI (Remote Method Invocation) は、サーバー、クライアント、およびオブジェクト レジストリの 3 つの部分で構成されます。

サーバー側アプリケーションのコンパイル時に 2 つのファイルが生成されます。"スケルトン" ファイルがサーバー側に生成され、"スタブ" ファイルがクライアント側に生成されます。どちらのファイルも、メソッドの呼び出しとデータの転送を処理するために内部で使用されます。サーバー側ではスケルトン ファイルが使用され、クライアント側ではスタブ ファイルが使用されます。

RMI クライアントとサーバーを .NET Framework に変換しても、Visual C# の RMI クライアントとサーバーにはなりません。既定で、.NET Framework リモート処理が HTTP 経由で使用されます。メソッドの呼び出しとデータの転送は内部で処理されるので、スケルトン ファイルやスタブ ファイルのようなファイルは生成されません。

RMI クライアントは、文字列 URL を送信してリモート オブジェクトを要求します。この文字列の形式は、rmi://*host*:*port*/*name* です。*port* の部分は、リモート オブジェクトを登録するためにサーバーで使用されるポートです。既定値は 1099 です。

.NET Framework では、文字列 URL は使用されるチャネル プロトコルを表し、その形式は *protocol*://*host*:*port*/*name* です。*port* の部分は、クライアントからの要求を待機するためにサーバーで登録されたポートです。

Java 言語では、**java.rmi.server.UnicastRemoteObject** クラスを拡張し、そのすべてのメソッドが **RemoteException** エラーを発生させるオブジェクトは、リモート オブジェクトであると見なされます。**RemoteException** を継承するほとんどすべての例外は、System.Runtime.Remoting.RemotingException エラーに変換されます。アプリケーションで特定の **RemotingException** サブクラスを別の方法で扱う場合は、コードを手動で調整する必要があります。

.NET Framework では、System.MarshalByRefObject クラスを継承するオブジェクトは、リモート オブジェクトであると見なされます。

各 RMI インターフェイスは、それぞれ別の Visual Studio プロジェクトに変換されます。プロジェクト参照を更新して、リモート インターフェイス アセンブリを含める必要があります。クライアントとサーバーのエントリ ポイントが変換後のソース ツリーに含まれる場合は、出力されたコードを独立した Visual Studio プロジェクトとして分離させる必要があります。クライアントとサーバーのプロジェクトは、どちらも共有 DLL アセンブリを参照する必要があります。

セキュリティの問題

.NET Framework では、分散通信 (リモート処理) の既定のセキュリティ レベルは **low** です。これは、ユーザー定義オブジェクト型をリモート メソッドに渡す場合に影響します。詳細については、「方法 : リモート処理のセキュリティ レベルを設定する」を参照してください。

参照

関連項目

RemotingException
MarshalByRefObject

概念

JNDI を使用しているアプリケーションの変換

JMS アプリケーションの変換

Java Message Services (JMS) は、Windows メッセージ キューを使用するように変換されます。したがって、接続、コネクション ファクトリ、およびセッションを作成するコードは、変換時にコメント アウトされます。関連のある変数がコードの他の場所にある場合は、それらを削除する必要があります。Visual C# コード内では、プロパティおよび JNDI に関連するコードは無視してください。

パブリッシャ サブスクライバ モードは、ポイント ツー ポイント モードとまったく同じ方法で変換されます。

JMS の次の要素はサポートされません。

- トピック
- 持続性のあるサブスクライバ
- メッセージ セレクタ
- 例外リスナ

複数の送信先があるメッセージングを .NET Framework で使用するには、次のしくみのいずれかを選択する必要があります。

- 配布リスト
- 複数要素形式名
- マルチキャスト アドレス

これらのしくみでは、送信先キューは文字列値で表され、System.Messaging.MessageQueue.Path プロパティに設定されます。

Windows メッセージ キューを使用するには、キューまたはトピックを表す文字列を手動で変更する必要があります。**TemporaryQueue** オブジェクトは、物理キューに変換されます。このキューは明示的に削除する必要があります。

.NET Framework では、トランザクション キューの扱い方が異なります。System.Messaging.MessageQueueTransaction オブジェクトを使用して、トランザクション キューを設定し、トランザクション管理を書き直す必要があります。

メッセージの受信確認は、.NET Framework では自動的に管理されます。メッセージの受信確認がコードに手動で書かれている場合は、その部分を削除する必要があります。

接続を使用できない場合、Windows メッセージ キューは接続を再び試みません。

XA オブジェクトは変換されません。分散トランザクションは、.NET Framework では書き直す必要があります。

変換されたアプリケーションは、JMS 対応 MOM と対話できません。

参照

関連項目

System.Messaging
MessageQueueTransaction

シリアル化されたアプリケーションの変換

Java 言語では、**java.io.Serializable** インターフェイスを直接実装するか、継承をとおして実装するオブジェクトが、シリアル化できるオブジェクトです。.NET Framework は、System.Runtime.Serialization.ISerializable インターフェイスを似たような方法で使用しますが、SerializableAttribute 属性も使用できます。したがって、オブジェクトを実行時にプログラムでシリアル化できるかどうかを確認することはできません。Java 言語コードで **Serializable** インターフェイスを引数として受け取るメソッドや戻り値として返すメソッドには注意してください。これは .NET Framework で使用できる同等のメソッドとのやりとりはできません。相互に読み取りや書き込みはできません。

Java 言語では、シリアル化できるクラスは通常 **readObject** メソッドと **writeObject** メソッドを実装します。**writeObject** メソッドはオーバーロードされたコンストラクタに変換され、**readObject** メソッドは `GetObjectData` メソッドに変換されます。ほとんどの場合、変換されたコードは、それ以上変更しなくてもコンパイルして実行できます。ただし、マーシャリングとマーシャリング解除のカスタム コードを定義したクラスは大幅に書き直す必要があります。

参照

関連項目

ISerializable

JNDI を使用しているアプリケーションの変換

変換後のアプリケーションで機能が保持されるには、両方の環境で同じサービス プロバイダを利用できることが必要です。Java 言語と .NET Framework の両方でサポートされるのは LDAP/ActiveDirectory のみです。

一般的に名前付けおよびディレクトリ サービスをサポートしている JNDI メソッドは、System.DirectoryServices 名前空間のメソッドに変換されます。一般的に RMI および CORBA をサポートしているメソッドは System.Runtime.Remoting 名前空間に変換されます。いくつかのメソッドが 2 つの目的をサポートしているため、**System.Runtime.Remoting** メソッドの中には、変換後のコードで **System.DirectoryServices** メソッドに変更する必要があるものもあります。

Java 言語では、環境を定義するハッシュ テーブルを受け入れるコンストラクタを使用して、コンテキストをインスタンス化できます。System.DirectoryServices.DirectoryEntry クラスにはそのようなコンストラクタはありません。

プロバイダを識別するディレクトリ エントリ パスは大文字と小文字が区別され、たとえば `ldap` を `LDAP` のように変更する必要があります。

Java 言語では、検索パラメータがコンテキストとは別に定義されて、次にコンテキスト検索メソッドに渡されます。.NET Framework では、検索パラメータはディレクトリ エントリと組み合わせて定義され、System.DirectoryServices.DirectorySearcher オブジェクトを作成することによって検索されます。この違いに対応するために、コードを変更する必要があります。

DirContext.bind メソッドと **DirContext.rebind** メソッドの代わりに、**System.DirectoryServices.DirectoryEntry** クラスおよびその関連クラスを使用して、ディレクトリのオブジェクトを追加したり置き換えたりします。

Context.createSubcontext メソッドの代わりに System.DirectoryServices.DirectoryEntries.Add(System.String,System.String) メソッドを使用し、新しいエントリのプロパティを設定します。

Context メソッドで **javax.rmi.naming** パッケージ内に相当するものがある場合、そのすべてのメソッドは System.Runtime.Remoting.RemotingServices メソッドに変換されます。これには **Context.bind**、**Context.lookup**、**Context.rebind**、および **Context.unbind** が含まれます。

javax.naming.ldap パッケージはサポートされません。

参照

概念

CORBA アプリケーションの変換
RMI アプリケーションの変換

データ パッケージの変換

java.sql パッケージおよび **javax.sql** パッケージを使用するアプリケーションは、System.Data.OleDb 名前空間に変換されます。別のデータベース システムを使用する場合は、これらの参照を System.Data.SQLClient や System.Data.ODBC などのシステムのクライアントに変換できます。

各 .NET Framework データ プロバイダは接続を自動的にプールします。接続プールは、ODBC Data Provider、OleDB Data Provider、および SQL Server Data Provider でそれぞれ構成できます。

すべての接続文字列を .NET Framework 形式に変換する必要があります。次の例は、さまざまな種類のデータベース接続を変換する方法を示しています。

例

次の例は、JDBC-ODBC ブリッジを使用する方法を示しています。

元の Visual J++ コード

```
String dbUrl = "jdbc:odbc:BiblioODBC";
Connection c = DriverManager.getConnection(dbUrl);
```

相当する Visual C# のコード

```
String dbUrl = "Provider = MSDASQL; DSN = BiblioODBC";
ODBCConnection c = "DriverManager.getConnection(dbUrl);
```

次の例は、Microsoft SQL Server から URL を使用する方法を示しています。

元の Visual J++ コード

```
String dbUrl = "jdbc:microsoft:sqlserver://MySQLServer:1433; DatabaseName=pubs";
Connection c = "DriverManager.getConnection(dbUrl, username, pwd);
```

相当する Visual C# のコード

```
String dbUrl = "Provider = SQLOLEDB; DataSource = MySQLServer; Initial Catalog = pubs";
SQLConnection c = DriverManager.getConnection (dbUrl + "; User ID = " + username + "; PWD +
" + pwd);
```

元の Oracle 接続文字列

次の例は、Oracle データベース接続文字列を変更する方法を示しています。

```
String jdbcUrl = "jdbc:oracle:thin:@localhost:1521:ORCL";
conn = DriverManager.getConnection(jdbcUrl, user, password);
```

相当する Visual C# のコード

```
String jdbcUrl = "Provider = MSDAORA; Data Source = localhost:1521;";
OracleConnection conn = DriverManager.getConnection(jdbcUrl + "User ID = " user + "; Passwo
rd = " + password);
```

更新可能で、スクロール可能な結果セットは確実に変換されるとは限りません。サポート クラスが用意されていますが、手動で何らかの調整を行う必要があります。このような結果セットにアプリケーションが大きく依存している場合、結果セットが自動的に変換される System.Data.OleDb.OleDbDataReader クラスではなく、System.Data.DataView クラスまたは System.Data.DataTable クラスを使用します。

Clob データ型と **Blob** データ型は、それぞれ文字配列およびバイト配列に変換されます。文字配列とバイト配列では動作がそれぞれ異なり、メモリの消費量が増加する場合もあります。変換後のコードはデータを順番にフェッチして配列を作成します。データ ストリームをディスク ファイルに書き込むこともできます。また、Oracle サーバー上で **Large Object Binary** データ型を表すために使用される System.Data.OracleClient.OracleLob クラスも使用できます。**STRUCT** 型、**ARRAY** 型、ユーザー定義型など、新しいデータ型によっては .NET Framework でサポートされない場合もあります。System.Data.OracleClient.OracleBFile マネージ クラスは Oracle **BFile** データ型を処理します。

データ アクセスの種類はデータベースの種類に固有です。OleDb ドライバはほとんどのデータベースで使用できるため、すべての JDBC コードが OleDb 型に変換されます。場合によっては、この型を変換後に変更して、パフォーマンスやサポートを強化することもできます。.NET Framework では他にも、ODBC、SQL、Oracle などのデータ アダプタを使用できます。

DataSources は割り当てられません。ただし、場合によっては、OleDbConnection、または System.Data 名前空間に属する他の任意のクライアント接続クラスを使用して機能の一部を表すことができます。**javax.sql.DataSource** インターフェイスのログ機能は、System.Diagnostics.EventLog クラスを使用して実装できます。

javax.sql パッケージは確実に変換されるとは限りません。このパッケージのインターフェイスは、サードパーティ ドライバによって実装する必要があります。

Driver インターフェイスおよび **DriverManager** クラスは推奨されないため、現在では必要ありません。これらに関連するいくつかのメソッドも変換できますが、これらは不要なメソッドです。

参照

関連項目

System.Data.OleDb
System.Data.SqlClient
System.Data.Odbc
DataAdapter
DataTable
OleDbDataReader

概念

接続プールについて
接続プールの使用

JAAS アプリケーションの変換

.NET Framework は、プリンシパル クラス、ID クラス、およびアクセス許可クラスと共にロール ベースのセキュリティを使用してセキュリティ対策を行います。独自のセキュリティ モジュールを作成しなくても、.NET Framework にある組み込みのモジュールから選択できます。JAAS (Java Authentication and Authorization Service) アプリケーションを .NET Framework に変換する場合、この 2 つのセキュリティ対策の違いを考慮する必要があります。

すべての JAAS 構成ファイルは、Java Language Conversion Assistant で処理できるよう、名前を JAAS.config に変更する必要があります。これらのファイルは App.config ファイルに変換され、サポートクラス メソッドでこれを使用することで認証モジュールを取得して認証マネージャに登録できます。

LoginContext クラスは、動作の異なる静的クラス **System.Security.AuthenticationModule** に変換されます。

LoginModule クラスは、IAuthenticationModule インターフェイスに変換されます。Java 言語では、**LoginContext** オブジェクトは、コールバック ハンドラを使用してユーザーからの入力を要求し、ログイン モジュールを使用してユーザーを認証する **LoginModule** オブジェクトを登録します。.NET Framework では、認証モジュールが認証マネージャに登録され、認証マネージャは登録された認証モジュールをループして認証情報を返します。

汎用例外 System.Security.SecurityException は、別の JAAS 例外の代わりに使用されます。

Subject クラスは、動作の異なる System.Security.Principal.GenericPrincipal クラスに変換されます。Windows 認証で使用する System.Security.Principal.WindowsPrincipal クラスもあります。

参照

関連項目

System.Security
System.Security.Principal
GenericPrincipal
WindowsPrincipal

JCE アプリケーションの変換

Java 言語では、ジェネリック クラスまたはジェネリック インターフェイスがすべての暗号化クラスの基本クラスとして使用されます。.NET Framework では、System.Security.Cryptography.AsymmetricAlgorithm クラスがすべての公開キー アルゴリズムの基本クラスです。変換時に、System.Security.Cryptography.SymmetricAlgorithm クラスのラッパーとして **CryptoSupport** サポート クラスが生成されます。非対称アルゴリズムを手動で再実装する必要があります。

Java 言語では、すべての暗号化アルゴリズムの種類は、暗号化の種類を示すリテラル文字列を使用して **Cipher.getInstance** メソッドによって生成されます。暗号化の種類ごとに、.NET Framework の個別のクラスが使用されます。比較的限定された暗号化クラスが用意されています。

参照

関連項目

System.Security.Cryptography
SymmetricAlgorithm
AsymmetricAlgorithm

Java Swing アプリケーションの変換

Swing アプリケーションは System.Windows.Forms 名前空間に変換されます。したがって、次の Swing 要素では、Visual C# に直接これに相当するものがなく、変換できません。

- **javax.swing.plaf** パッケージはサポートされません。.NET Framework には Swing をプラグ可能な外観またはそれを参照するメソッドにカプセル化するクラスに相当するものはありません。
- 表示と編集機能はサポートされていません。
- Java AWT レイアウトはサポートされていません。これらは Swing 固有のものではありませんが、多くの Swing アプリケーションに影響します。

次の要素は、Java Language Conversion Assistant で変換後、サポートが制限されているか、または手動によるかなりの調整が必要かを示しています。

- モデルは一部だけサポートされます。
- **JTree** および **JTable** などのクラスはモデルに依存しているため、.NET Framework で相当するものに直接変換されません。ヘルプに実装されているサポート クラスはこれらのクラスと一致しますが、すべての項目には対応しません。
- **ContentPane** クラスとその他の中間レベルのコンテナは直接変換されません。たとえば、**JFrame** オブジェクトにはその内容ペインにコントロールが含まれますが、.NET Framework System.Windows.Forms.Form クラスには、中間オブジェクトなしでそれ自体のコントロールが含まれます。これにより継承の問題が発生し、変換には主要なユーザーの入力が必要です。
- 定数値によって動作が指定されるクラスは自動的には完全に変換されません。たとえば、**FileDialog** クラスには、ファイルを開くか、または保存するかを指定する定数があります。これは、Visual C# 内の 2 つの別々のクラスによって処理されます。
- イベント処理は .NET Framework ポート内の別のモデルに組み込まれているため、すべての Swing イベントは変換後に手動による調整が必要です。変換を処理するイベントのサンプルは、「Javax.swing のサンプル」トピックに含まれます。

参照

関連項目

Javax.swing のエラー メッセージ
System.Windows.Forms

概念

Javax.swing のサンプル

ActiveX コントロールの変換

com.ms.wfc.ax パッケージには、Microsoft ActiveX コントロールおよび ActiveX オートメーション オブジェクトのクラスとインターフェイスが含まれています。

.NET Framework では、Windows フォームがホストできるのは、Windows フォーム コントロールだけです。ActiveX コントロールは、System.Windows.Forms.AxHost クラスから派生するラッパー クラス内の Windows フォーム コントロールのように動作させるために、ラップする必要があります。

ActiveX コンポーネントは、次のいずれかの方法で変換できます。

- Visual Studio を使用します。これはタイプ ライブラリ内の COM 型をアセンブリ内のメタデータに自動的に変換します。
- インポート タイプ ライブラリを使用します。これはコマンド ライン スイッチを提供して、メタデータを調整し、相互運用アセンブリと名前空間を生成します。
- カスタム ラッパーを使用します。これは最も手間のかかるオプションですが、アプリケーション ニーズに合わせてカスタマイズできます。

参照

関連項目

AxHost

Javax.swing のサンプル

次の例は、エクステンダを使用して、System.ComponentModel.IExtenderProvider インターフェイスによりプロパティをコントロールに提供する方法を示しています。

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Globalization;
namespace Extenders
{
    [ProvideProperty("MyString", typeof(Control))]
    public class UserControl1 : System.Windows.Forms.UserControl, IExtenderProvider
    {
        private System.ComponentModel.Container components = null;
        public UserControl1()
        {
            InitializeComponent();
        }
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if(components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }
        #region Component Designer generated code
            private void InitializeComponent()
            {
                this.Name = "UserControl1";
                this.Size = new System.Drawing.Size(170, 160);
            }
        #endregion
        [Browsable(true)]
        public String GetMyString(Control control)
        {
            return "This is my string.";
        }
        public void SetMyString(Control container, String value)
        {
        }
        public bool CanExtend(Object control)
        {
            return true;
        }
    }
}
```

次の例は、ユーザー定義イベントの処理と発生を実装する方法を示しています。

```
namespace EventSample
{
    using System;
    using System.ComponentModel;
    //Class that contains the data for
    //the alarm event. Derives from System.EventArgs.
    public class AlarmEventArgs : EventArgs
```

```
    {
        private readonly bool snoozePressed;
        private readonly int nrings;
        public AlarmEventArgs(bool snoozePressed, int nrings)
        {
            this.snoozePressed = snoozePressed;
            this.nrings = nrings;
        }
        //The NumRings property returns the number of rings
        //that the alarm clock has sounded when the alarm event
        //is generated.
        public int NumRings
        {
            get { return nrings;
        }
    }
    //The SnoozePressed property indicates whether the snooze
    //button is pressed on the alarm when the alarm event is generated.
    public bool SnoozePressed
    {
        get {return snoozePressed;}
    }
    //The AlarmText property that contains the wake-up message.
    public string AlarmText
    {
        get
        {
            if (snoozePressed)
            {
                return ("Wake up! Snooze time is over.");
            }
            else
            {
                return ("Wake Up!");
            }
        }
    }
}
    //Delegate declaration.
    public delegate void AlarmEventHandler(object sender, AlarmEventArgs e);
    //The Alarm class that raises the alarm event.
    public class AlarmClock
    {
        private bool snoozePressed = false;
        private int nrings = 0;
        private bool stop = false;
        //The Stop property indicates whether the
        //alarm should be turned off.
        public bool Stop
        {
            get {return stop;}
            set {stop = value;}
        }
        //The SnoozePressed property indicates whether the snooze
        //button is pressed on the alarm when the alarm event is
        //generated.
        public bool SnoozePressed
        {
            get {return snoozePressed;}
            set {snoozePressed = value;}
        }
        //The event member that is of type AlarmEventHandler.
        public event AlarmEventHandler Alarm;
        //The protected OnAlarm method raises the event by invoking
        //the delegates. The sender is always this, the current instance
        //of the class.
        protected virtual void OnAlarm(AlarmEventArgs e)
        {
```

```
            if (Alarm != null)
            {
                //Invokes the delegates.
                Alarm(this, e);
            }
        }
        //This alarm clock does not have
        //a user interface.
        //To simulate the alarm mechanism, it has a loop
        //that raises the alarm event at every iteration
        //with a time delay of 300 milliseconds,
        //if snooze is not pressed. If snooze is pressed,
        //the time delay is 1000 milliseconds.
        public void Start()
        {
            for (;;)
            {
                nrings++;
                if (stop)
                {
                    break;
                }
                else if (snoozePressed)
                {
                    System.Threading.Thread.Sleep(1000);
                    {
                        AlarmEventArgs e = new AlarmEventArgs(snoozePressed,
                        nrings);
                        OnAlarm(e);
                    }
                }
                else
                {
                    System.Threading.Thread.Sleep(300);
                    AlarmEventArgs e = new AlarmEventArgs(snoozePressed, nrings);
                    OnAlarm(e);
                }
            }
        }
    }
    //The WakeMeUp class that has a method AlarmRang that handles the
    //alarm event.
    public class WakeMeUp
    {
        public void AlarmRang(object sender, AlarmEventArgs e)
        {
            Console.WriteLine(e.AlarmText +"\n");
            if (!(e.SnoozePressed))
            {
                if (e.NumRings % 10 == 0)
                {
                    Console.WriteLine("Let alarm ring? Enter Y");
                    Console.WriteLine(" Press Snooze? Enter N");
                    Console.WriteLine(" Stop Alarm? Enter Q");
                    String input = Console.ReadLine();
                    if (input.Equals("Y") ||input.Equals("y")) return;
                    else if (input.Equals("N") || input.Equals("n"))
                    {
                        ((AlarmClock)sender).SnoozePressed = true;
                         return;
                    }
                    else
                    {
                        ((AlarmClock)sender).Stop = true;
                        return;
                    }
                }
            }
```

```
            else
            {
                Console.WriteLine(" Let alarm ring? Enter Y");
                Console.WriteLine(" Stop alarm? Enter Q");
                String input = Console.ReadLine();
                if (input.Equals("Y") || input.Equals("y")) return;
                else
                {
                    ((AlarmClock)sender).Stop = true;
                    return;
                }
            }
        }
    }
    //The driver class that hooks up the event handling method of
    //WakeMeUp to the alarm event of an Alarm object using a delegate.
    //In a forms-based application, the driver class is the
    //form.
    public class AlarmDriver
    {
        public static void Main (string[] args)
        {
            //Instantiates the event receiver.
            WakeMeUp w= new WakeMeUp();
           //Instantiates the event source.
           AlarmClock clock = new AlarmClock();
           //Connects the AlarmRang method to the Alarm event.
           clock.Alarm += new AlarmEventHandler(w.AlarmRang);
           clock.Start();
        }
    }
}
```

次の例は、コントロールの `DataSource` プロパティを使用して、モデルの特性をエミュレートする方法を示しています。

```
public class DataBinding : System.Collections.ArrayList
{
    public DataBinding()
    {
    }
    public void AddNew(int index, object obj)
    {
        this.Add(obj);
    }
}
DataBinding s_model = new DataBinding();
DataBinding i_model = new DataBinding();
System.Windows.Forms.ListBox listBox1 = new System.Windows.Forms.ListBox();
public void SetModel(int whichModel)
{
    if (whichModel==0)
    {
        i_model.Add("i_model Value1");
        i_model.Add("i_model Value2");
        listBox1.DataSource = i_model;
    }
    else
    {
        s_model.Add("s_model Value1");
        s_model.Add("s_model Value2");
        listBox1.DataSource = s_model;
    }
}
```

参照

概念

com.ms.wfc.data.Connection の例

com.ms.wfc.data.Connection メソッドの使用方法の例を次に示します。

1. Microsoft ActiveX データ オブジェクト (ADO) 2.7 ライブラリ (Adodb.dll) への参照を追加します。
2. Connection オブジェクト (**ConnectionClass** クラス) でデータベースへの接続を作成します。

- **ConnectionClass.Open** メソッドを呼び出します。

  Provider パラメータは Jet 4 OLE DB プロバイダを示し、*Data Source* パラメータはデータベースの物理的な位置を指します。

参照

その他の技術情報

ADO API Reference

方法：リモート処理のセキュリティ レベルを設定する

.NET Framework では、分散型通信 (リモート処理) の既定のセキュリティ レベルは **low** です。この設定は、ユーザー定義のオブジェクト型をリモート メソッドに渡す際に影響します。場合によっては、オブジェクトのシリアル化を許可するために、既定のセキュリティ レベルを **full** に調整する必要があります。

セキュリティ レベルを調整するには

- 変換時に生成された構成ファイルを変更します。

  または

- 変換後のコードを編集します。

アプリケーションを実行し、次の説明と共に System.Runtime.Serialization.SerializationException が発生するまでは、この調整を行う必要があるかどうかは不明です。

セキュリティ制限のため、System.Runtime.Remoting.ObjRef 型にアクセスできません。

参照

概念

CORBA アプリケーションの変換
RMI アプリケーションの変換

JLCA 診断メッセージ (パッケージ別)

ここには、Java Language Conversion Assistant のすべての診断メッセージへのリンクが用意されています。リンクは、パッケージごとに分類されています。

Java 言語のエラー、警告、および問題

(1002) コンパイル エラー : 無効なステートメント

(1003) 注意 : 宣言が final から変数に変更された

(1005) 注意 : 初期化式がメソッドまたはコンストラクタに移動された

(1011) 注意 : 新しく生成されたコード

(1012) 注意 : ラベル付きの Break ステートメントはサポートされていない

(1014) 注意 : ラベル付きステートメントが Java Language Conversion Assistant によって移動された

(1015) メモ : ラベル付きの Continue ステートメントはサポートされていない

(1019) 注意 : 内部クラスはそれを囲んでいるインスタンスとともには提供されない

(1021) 注意 : 内部クラスのメンバ フィールド this$0 はサポートされていない

(1022) 注意 : ローカル クラスの宣言はサポートされていない

(1023) メモ : final 変数が内部クラスにメンバとしてコピーされた

(1024) グローバル注意 : 匿名クラスの宣言はサポートされていない

(1025) 注意 : C# ではインターフェイスにインターフェイスを入れ子にできない

(1027) メモ : .NET Framework では同期化されたメソッドはサポートされていない

(1042) グローバル警告: Visual C# のデータ型は異なっている可能性があります。

(1043) 修正が必要 : .NET Framework では異なる値を返すメソッドまたはプロパティ

(1045) 注意 : .NET Framework ではインターフェイスにフィールドを含めることはできない

(1062) 修正が必要 : データベース列の NULL 値は OleDbDataReader.isDBNull を使用して確認できる

(1063) 修正が必要 : 接続文字列を .NET Framework と互換性のある形式に変更する必要がある

(1064) 修正が必要 : 接続文字列を .NET Framework と互換性のある形式に変更し、それに java.util.Properties の情報を追加する必要がある

(1066) 修正が必要 : OleDbConnection コンストラクタの接続文字列にはパラメータとして setLoginTimeout を含める必要がある

(1075) メモ : Font コンストラクタは等価でないインスタンスを生成する可能性がある

(1077) 修正が必要 : サポートされていないクラス継承

(1078) 修正が必要 : クラスで使用されるリソースが有効なリソース ファイルであることを確認する必要がある

(1079) コンパイル エラー : 抽象クラスから継承されたメソッドはすべて実装される必要がある

(1083) 注意 : ビット配列が同じサイズでないと行えない処理

(1086) コンパイル エラー : 整数変数には代入できないフィールド

(1088) 修正が必要 : .NET の構造体には NULL 値は定義されていない

(1089) 修正が必要 : 構成ファイルのアクセス方法と形式が異なっている

(1091) 実行時の警告: 変換後のコンストラクタはパラメータ数が異なる

(1092) 修正が必要 : パラメータの値または型が期待されるものと異なっている

(1093) 修正が必要 : .NET クラス メンバの値にアクセスするにはコードを変更する必要がある

(1095) To Do: このリリースではマップされていないコード要素

(1096) 注意 : 列挙型に割り当てられるフィールドが整数値で初期化されている

(1099) メモ : メソッドによってスローされる例外が異なる可能性がある

(1100) 注意 : 同じ例外の try ステートメント内に複数の catch 句がある

(1101) 実行時の警告 : 割り当てられたパラメータによって例外がスローされる可能性がある

(1102) コンパイル エラー : クラスに変更を加えてスーパークラスが呼び出されるようにする必要がある

(1103) コンパイル エラー : 異なる型に変換されたフィールド

(1104) 修正が必要 : ConnectionString プロパティを使用してログオンおよびパスワードを設定する必要がある

(1105) 修正が必要 : 不必要なメソッド呼び出し

(1107) グローバル注意 : DataGrid クラスの setAllowAddNew、setAllowDelete、および setAllowUpdate の変換

(1108) 修正が必要 : 呼び出しの前にプロパティまたはメソッドの設定または呼び出しが行われる必要がある

(1109) コンパイル エラー : 変換されたパラメータにメソッドのパラメータ型との互換性がない

(1113) コンパイル エラー : メンバがパッケージの内部使用のために予約されている

(1114) 修正が必要 : OpenFileDialog および SaveFileDialog クラスの setFilter のパラメータを変更する必要がある

(1115) グローバル警告 : 安全でないコードをコンパイルするにはコンパイラ スイッチが必要である

(1116) 修正が必要 : COM カスタム マーシャラが無効になる可能性

(1118) 修正が必要: グループ化されたチェック ボックスでは同時に複数を選択できる

(1119) 注意: ユーザー コントロール イベントの属性が無効

(1123) 注意: クラスからインターフェイスへのマッピング

(1124) 注意: メソッド本体から削除されたコード

(1128) コンパイル エラー: fillay 句の break ステートメントは無効

(1130) 修正が必要: java.sql.DatabaseMetaData に相当するクラスは System.Data.DataTable でクエリ結果を処理

(1132) 修正が必要: 変換先のクラスは誤った形式の URL に対して例外をスローしない

(1133) グローバル警告 : equals メソッドが異なる値を返す可能性がある

(1135) グローバル エラー: メイン ウィンドウ開始には System.Windows.Forms.Application.Run メソッドの呼び出しが必要

(1137) 修正が必要: 変換後の get メソッドまたは set メソッドでプロパティ名が重複

(1139) グローバル警告: 変換されないディレクトリ

(1140) グローバル警告: スレッド名の設定は 1 回のみ

(1141) 注意: COM コクラスに対する Java の呼び出し可能ラッパー クラスがコメントにされた

(1142) 修正が必要: イベントのメソッドをオーバーロードすることが必要

(1143) 修正が必要: override でないメソッド

(1144) 実行時の警告: componentHidden へのロジックの追加

(1145) 実行時の警告: componentShown へのロジックの追加

(1146) 修正が必要: 複数行にわたるクラスのインスタンスの宣言と構築

(1147) 修正が必要: Sealed とマークされた <classname> クラス

(1148) 注意: 同期されたキーワードの削除

(1151) 修正が必要: アクセスできないコンストラクタ

(1152) 修正が必要: レジストリへのアクセスを手動で変換

(1153) 修正が必要: 変換されていないステートメント

(1154) メモ: com.ms.wfc.data.ui.DataNavigator クラスから変換されたクラスのコンストラクタのパラメータ

(1155) 修正が必要: Delegate または MulticastDelegate から派生するクラス

(1156) コンパイル エラー: 間違ったステートメント

(1157) コンパイル エラー: パラメータの型が同じメソッド

(1158) 修正が必要: ファイルにアクセスする方法の変更

(1159) グローバル警告: アーカイブのパスが作業ディレクトリに依存

(1160) 修正が必要: 他のインターフェイスの機能を含むインターフェイス

(1167) 注意: リテラル文字列なしでの getParameter の呼び出し

(1168) 修正が必要: getParameter でパラメータ名の変更が必要

(1169) メモ: プロパティの既定値の変更

(1170) グローバル警告: クラスが構造体に変換され、Null 値を格納できず返せない

(1171) 注意 : getSource メソッドはイベント処理ルーチン内に含まれる必要がある

(1172) グローバル エラー : 入力プロジェクト ファイルが壊れている

(1173) 修正が必要 : 16 進リテラルは異なる値を返す

(1174) 注意 : .NET Framework ではバイナリ ファイルの読み込み方法が異なる

(1175) グローバル警告: unsigned long

(1177) 実行時の警告 : インターフェイスの実装が単一クラスに変換された

(1178) 修正が必要 : メソッドの修正が必要

(1179) グローバル警告 : OleDbManager 接続文字列を要求

(1180) 注意 : サポートされるトランザクション属性は 1 つだけ

(1181) 実行時の警告 : C# コードで式が複数回使用されている

(1182) 修正が必要 : セッション パラメータの格納順序が異なる

(1183) 実行時の警告 : Init 機能が変換された

(1184) 修正が必要 : <Target Name> で受け取るインスタンスの型を変更することが必要

(1185) 修正が必要 : キュー アクセスの取得が異なる

(1186) コンパイル エラー: クラス階層が異なるため、コンパイル エラーが発生する可能性があります。

(1187) 実行時の警告 : ASP.NET ページを読み込むたびに <Member type> が実行される可能性がある

(1188) 注意 : 変換された RMI メソッドについてプロトコルを選択できる

(1189) 修正が必要 : 複数の送信先があるメッセージングでは送信先を特別な形式にする必要がある

(1190) 修正が必要 : リモート オブジェクトのアクティベーション

(1191) メモ : インスタンス フィールドが静的に変換された

(1192) 修正が必要: シリアル化コードが正しく変換されない可能性がある

(1193) コンパイル エラー : メソッドの変換でシリアル化コードが未使用

(1194) 実行時の警告 : シリアル化値のキーは 1 回のみ使用する必要があり、対応するキーに一致する必要がある

(1195) 修正が必要 : 非仮想メソッドのオーバーライド修飾子で変更

(1196) メモ: リモート インターフェイス ファイル変換

(1197) 修正が必要 : 関数名の String 値でインスタンス化されたデリゲート

(1198) 注意 : サーブレット クラスの継承により変数の不正操作の可能性がある

(1199) メモ : 各 javadoc コメントがマージされている

(1200) 注意 : ベース クラスの共有フィールドが new 修飾子で再宣言されている

(1201) 修正が必要 : 複数チャネルの登録

(1202) 修正が必要 : 参照の変換でユーザーによる修正が必要

(1203) メモ : クラス インターフェイスを使用してパブリック メンバを公開する

(1204) 注意 : アクセス修飾子が変更されている

(1205) 修正が必要 : プリミティブ型のパラメータを持つメソッドがラッパー クラス型のパラメータでオーバーロードされた

(1206) グローバル注意 : 変換されたプロジェクトは Visual Studio デザイナ ビューで開く前にコンパイルすることが必要

(1207) 修正が必要 : COM オブジェクトから拡張される型はすべてのメソッドをオーバーライドすることが必要

(1208) グローバル注意 : System.Data.OleDb 名前空間は別の名前空間と置き換えることができる

(1209) 注意 : メソッドがプロパティに変換されない

(1210) グローバル警告 : 逆シリアル化のレベルを変更する必要がある可能性がある

(1211) 実行時の警告 : チャネルの登録が予告なしに失敗することがある

(1212) 注意 : 配列の範囲が確認されない

(1213) グローバル注意 : 変換中に構成ファイルが追加された

(1214) コンパイル エラー : ユーザーはサーバー アドレスを入力する必要がある

(1215) コンパイル エラー: サーバー ポートを入力する必要があります。

(1216) コンパイル エラー : ユーザーはリモート オブジェクトの名前を入力する必要がある

(1217) グローバル警告 : Java 以外の言語はサポートされない

(1218) グローバル警告 : Java 言語の非ネイティブな定義が Visual C# のネイティブな構築に変換されている

(1219) グローバル注意 : System.MarshalByRefObject.InitializeLifetimeService オーバーライド

(1220) 修正が必要 : Reset メソッドを実装することが必要

(1221) 修正が必要 : メソッドでのアプレットからの親呼び出し

(1222) 実行時の警告 : 相当するコンストラクタにはより多くのパラメータがある

(1223) グローバル警告 : 双方向言語の出力

(1224) 修正が必要 : メソッドの戻り値の型が異なる

(1225) グローバル警告 : プリミティブ型キャストの動作が異なる

(1227) メモ : Visual レイアウト用にコピーされたコード

(1228) 注意 : コメント アウトされたコードは InitializeComponent メソッドに移動された

(1230) 実行時の警告 : 他のフォームの前に ContainerControl の追加

(1231) 修正が必要 : 複数の Java クラスのメンバが Visual C# 内の同じメンバに変換される

(1232) 修正が必要 : クラス ロジックを維持するには、メソッドまたはプロパティの実装が必要

(1233) 注意 : 変換後のサーブレット クラスで、"this" キーワードが異なる動作をする可能性がある

(1234) 注意 : 現在、内部クラスをシリアル化できる

(1235) コンパイル エラー : .NET Framework には対応する基本クラスのメソッドがない

(1236) 修正が必要 : サーブレット フィルタのリストを確認する

(1237) 注意 : EJB はサービス コンポーネントに変換された

(1238) 修正が必要 : サポートされていない EJB コールバック メソッド

(1239) グローバル警告 : クラスやインターフェイスが複数の EJB で使用されている

(1240) 修正が必要 : EJB bean のコンテナ管理による持続性とのデータ バインディングの再実装

(1241) 修正が必要 : EJBHome インターフェイス メソッドの移動と適用

(1242) 修正が必要 : EJB 2.0 CMP の finder メソッドの再実装

(1243) 修正が必要 : EJB 1.1 の CMP の検索メソッドの再実装

(1244) 修正が必要 : EJB 2.0 CMP リレーションシップの再実装

(1245) 修正が必要 : メソッド レベルでサポートされていない EJB トランザクション属性

(1246) 修正が必要 : サポートされていない EJB トランザクション属性

(1247) グローバル警告 : トランザクション コンテキスト内でスローされる例外が現在のトランザクションを常にロールバックする

(1248) 警告: EJB メソッドが除外されませんでした。

(1249) 警告 : チェックされていないメソッドのセキュリティ アクセス チェックは無効にできない

(1250) グローバル警告: セキュリティ レベルを変更する必要があります。

(1251) メモ : カスタム マーシャリングとマーシャリング解除が無視される

(1252) グローバル警告 : システム インクルードが無視されている

(1253) 修正が必要 : EJB リソース参照

(1254) 修正が必要 : EJB 参照を相当する .NET の参照に置換

(1255) グローバル注意 : 現在のコンポーネントに対する EJB 環境エントリが必要

(1256) グローバル注意 : EJB の再入はサポートされていない

(1257) グローバル警告 :JNDI セキュリティ

(1258) 修正が必要 : リモート コンテキストの初期化を手動で調整

(1259) 修正が必要 : リモート オブジェクト登録では既定のコンストラクタのみを使用

(1260) 修正が必要 : IDL の値型の既定のファクトリは、Visual C# では必要ない

(1261) 注意 : Servlet destroy メソッドはランタイムによって呼び出されない。

(1262) 修正が必要 : 使用できない型またはパッケージのインポート宣言によるコンパイル エラーの可能性がある

(1263) 修正が必要 : Web アプリケーションのフィルタ動作が異なる

(1264) グローバル警告: .NET Framework でのイベントの動作とは異なります。

(1265) 実行時の警告 : 相当するメソッドのパラメータが少ない

(1266) 実行時の警告 : System.MarshalByRefObject から派生したオブジェクトは独自のドメイン内でしかシリアル化できない

(1267) 注意 : 外観レイアウト用のコードの一部が InitializeComponent に移動された

(1268) 注意 : シリアル化できるクラス用にパラメータなしのコンストラクタが追加された

(1269) 修正が必要 : Visual C# に複数の相当するメンバがあるクラスのメンバは、動作しないことがある

(1270) 修正が必要 : コンストラクタの変更が必要

(1271) メモ : このメソッドは削除可能

(1272) 修正が必要 : サポートされていない IDL プリプロセッサ ディレクティブが変換されなかった

(1273) メモ : Context が変換されなかった

(1274) グローバル警告 : メンバ実行シーケンスが異なる可能性がある

(1275) 注意 : RMI サーバー アプリケーションが終了前にクライアント要求を待機している

(1277) 修正が必要 : 相当するクラスはシリアル化できない

(1278) メモ : メソッドが返る前に Out パラメータが割り当てられている必要がある

(1279) 実行時の警告 : デリゲートの作成に渡されたメソッド名が、有効でない可能性がある

(1280) 修正が必要 : サードパーティ パッケージが見つかったが、変換されているとは限らない

(1281) グローバル警告 : 不正な XML 文字によりリモート処理例外が発生する可能性

(1282) グローバル警告 : Visual C# があいまいな名前空間を正しく解決できない

(1283) グローバル警告 : 変換された ASP プロジェクトまたは JSP アプレットの移動

(1284) 注意 : getModifiers メソッド シミュレーションがいくつかのコントロールで動作しないことがある

(1285) グローバル警告: 変換されたプロパティ エディタの実装は動作が異なる可能性がある

(1286) 注意 : 値型実装が共有アセンブリに移動された

(1287) To Do: 変換文字列は、サポートされない可能性があります。

(1288) 注意 : キーを処理する暗号化クラスの動作が異なる

(1290) グローバル警告 : サウンドをサポートするには DirectX をインストールする必要がある

(1291) 修正が必要 : 構造体には Visual C# の NULL に相当する値はない

(1292) 注意 : UI スクロール フィールドはスクロール バーの変換に影響を与えない

(1293) グローバル警告 : すべての JNDI 例外が System.Exception にマッピングされている

(1294) コンパイル エラー : com.ms.wfc.html パッケージはサポートされない

(1295) 修正が必要 : Visual C# にはオブジェクトをパラメータとして持つコンストラクタがない

(1296) 注意 : アクセス修飾子の変更

(1264) グローバル警告: データにアクセスする前に、列挙子を開始しなければなりません。

(1298) 注意 : BeanInfo 機能が相当する Visual C# コンポーネントに転送される

(1299) 修正が必要 : 返された式を手動で変換する必要がある

(1300) 修正が必要: TypeConverter に適切な System.ComponentModel.Design.Serialization.InstanceDescriptor が必要です。

(1301) グローバル警告 : JavaBeans ソースはプロジェクト ディレクトリにある必要がある

(1302) グローバル エラー : ValueBase の変換によって型の不一致が発生している可能性がある

(1303) 注意 : ref キーワードが構造体パラメータに追加された

(1304) メモ: JavaBeans を変換すると、デプロイメントに問題が発生する可能性があります。

(1305) グローバル警告: AwtUI コンポーネントのイベント ハンドラ リンクの損失

(1306) 注意 : メソッドまたはプロパティの実装が追加された

(1307) 修正が必要 : ステートメントを InitializeComponent に移動できない

(1308) 注意 : リスナのメソッドが変換されたが、使用されない

(1309) 注意 : デリゲートに異なる戻り値がある可能性がある

(1310) 修正が必要 : EJB クラスに多重継承の問題がある

(1311) 修正が必要 : 考えられる EJB に配置記述子がない

(1312) 実行時の警告 :Mandatory トランザクション属性に相当する属性がない

(1313) 注意 : ステートメントを参照しない場合、ステートメントは不要

(1314) グローバル警告 : 変換後の EJB で、既定のセキュリティ機能が有効になっている

(1315) 修正が必要 : パラメータを参照渡しできない

(1316) グローバル警告 : EJB トランザクション属性が適用されなかった

(1317) コンパイル エラー : ValueType 配列の割り当て、比較、またはパラメータとしての引き渡しを実行できない

(1318) グローバル警告 : JAAS 構成ファイルの名前を変更する必要がある

(2001) 実行時の警告: 一意でなければならないサーバー側のタグがファイルのインクルードによってページ内で繰り返される可能性

(2002) 実行時の警告: aspx ではファイルを動的にインクルードできない

(2003) 注意: ASP.NET での buffer 属性の有効な値は 2 つのみ

(2004) 注意: 要求スコープからセッション スコープへの変更

(2005) グローバル エラー: Web.Config customErrors モードのプロパティをオンに設定する必要

(2006) 修正が必要 : ファイルが変換されない

(2007) 注意 : カスタム タグ内部のブロックをメソッドに移動するときにスクリプト変数が範囲外になる可能性がある

(2008) 注意 : カスタム タグ内部のブロックを移動するときに制御構造が正常に動作しない

(4010) グローバル エラー : ActiveX 参照が異なっている

(4011) グローバル エラー : ファイル内に無効な文字が見つかった

(4012) グローバル警告 : システム上でコード ページを使用できない

(4013) 注意 : 指定された TLD ファイルが見つからないか解析できない

(4014) グローバル警告 : Typelib が登録されているものと異なる

(4015) 注意 : ユーザー コントロール ID が変更されている

(4020) グローバル エラー : アーカイブ ファイルが圧縮解除できない

(4021) グローバル エラー : アーカイブ ファイルを読み込むことができない

(5000) 修正が必要: Virtual に設定されていないメソッド

参照

その他の技術情報

JLCA 診断メッセージ (パッケージ別)

Java アプリケーションの Visual C# への変換

Com.ms.activex のエラー メッセージ

com.ms.activex は変換されませんでした。

com.ms.activex.ActiveXControl は変換されませんでした。

com.ms.activex.ActiveXControlListener は変換されませんでした。

com.ms.activex.ActiveXControlServices は変換されませんでした。

com.ms.activex.ActiveXInputStream は変換されませんでした。

com.ms.activex.ActiveXOutputStream は変換されませんでした。

com.ms.activex.ActiveXToolkit は変換されませんでした。

com.ms.activex.PropertyDialogThread は変換されませんでした。

Com.ms.awt のエラーメッセージ

com.ms.awt.AccessibleWrapper は変換されませんでした。

com.ms.awt.AWTFinalizeable は変換されませんでした。

com.ms.awt.AWTFinalizer は変換されませんでした。

com.ms.awt.AWTPermission.check は変換されませんでした。

com.ms.awt.AWTPermission.pid は変換されませんでした。

com.ms.awt.CaretX は変換されませんでした。

com.ms.awt.CharsetString は変換されませんでした。

com.ms.awt.CharToByteSymbol は変換されませんでした。

com.ms.awt.ColorX.ColorX(float, float, float) は変換されませんでした。

com.ms.awt.ColorX.ColorX(int) は変換されませんでした。

com.ms.awt.ColorX.ColorX(int, int, int) は変換されませんでした。

com.ms.awt.ColorX.getHilight は変換されませんでした。

com.ms.awt.ColorX.getShadow は変換されませんでした。

com.ms.awt.Device.Device は変換されませんでした。

com.ms.awt.Device.getBasics は変換されませんでした。

com.ms.awt.Device.getDisplayContext は変換されませんでした。

com.ms.awt.DrawingSurface は変換されませんでした。

com.ms.awt.DrawingSurfaceInfo は変換されませんでした。

com.ms.awt.EventFilterListener は変換されませんでした。

com.ms.awt.FocusEvent.getOtherComponent は変換されませんでした。

com.ms.awt.FocusingTextField は変換されませんでした。

com.ms.awt.FontDescriptor は変換されませんでした。

com.ms.awt.FontMetricsX.bytesWidth は変換されませんでした。

com.ms.awt.FontMetricsX.CHAR_KERNING は変換されませんでした。

com.ms.awt.FontMetricsX.charsWidth は変換されませんでした。

com.ms.awt.FontMetricsX.FontMetricsX は変換されませんでした。

com.ms.awt.FontMetricsX.getAveCharWidth は変換されませんでした。

com.ms.awt.FontMetricsX.getFace は変換されませんでした。

com.ms.awt.FontMetricsX.getFontMetrics は変換されませんでした。

com.ms.awt.FontMetricsX.getLeading は変換されませんでした。

com.ms.awt.FontMetricsX.getMaxAdvance は変換されませんでした。

com.ms.awt.FontMetricsX.getWidths は変換されませんでした。

com.ms.awt.FontMetricsX.stringWidth は変換されませんでした。

com.ms.awt.FontX.chooseFont は変換されませんでした。

com.ms.awt.FontX.EMBEDDED は変換されませんでした。

com.ms.awt.FontX.getAttributeList は変換されませんでした。

com.ms.awt.FontX.getFlags は変換されませんでした。

com.ms.awt.FontX.getFlagsVal は変換されませんでした。

com.ms.awt.FontX.getFont は変換されませんでした。

com.ms.awt.FontX.getFontList は変換されませんでした。

com.ms.awt.FontX.getFontNativeData は変換されませんでした。

com.ms.awt.FontX.getNativeData は変換されませんでした。

com.ms.awt.FontX.getStyleVal は変換されませんでした。

com.ms.awt.FontX.isTypeable は変換されませんでした。

com.ms.awt.FontX.matchFace は変換されませんでした。

com.ms.awt.FontX.OUTLINE は変換されませんでした。

com.ms.awt.FontX.USEDFONT は変換されませんでした。

com.ms.awt.GenericEvent は変換されませんでした。

com.ms.awt.GraphicsX.bitBlt は変換されませんでした。

com.ms.awt.GraphicsX.cng は変換されませんでした。

com.ms.awt.GraphicsX.comp は変換されませんでした。

com.ms.awt.GraphicsX.copyArea は変換されませんでした。

com.ms.awt.GraphicsX.drawBezier は変換されませんでした。

com.ms.awt.GraphicsX.drawChars は変換されませんでした。

com.ms.awt.GraphicsX.drawCharsWithoutFxFont は変換されませんでした。

com.ms.awt.GraphicsX.drawOutlineChar は変換されませんでした。

com.ms.awt.GraphicsX.drawOutlinePolygon は変換されませんでした。

com.ms.awt.GraphicsX.drawPixels は変換されませんでした。

com.ms.awt.GraphicsX.drawRoundRect は変換されませんでした。

com.ms.awt.GraphicsX.drawScanLines は変換されませんでした。

com.ms.awt.GraphicsX.drawT2Curve は変換されませんでした。

com.ms.awt.GraphicsX.fill3DRect は変換されませんでした。

com.ms.awt.GraphicsX.fillRoundRect は変換されませんでした。

com.ms.awt.GraphicsX.gdc は変換されませんでした。

com.ms.awt.GraphicsX.getColorType は変換されませんでした。

com.ms.awt.GraphicsX.getGlyphOutline は変換されませんでした。

com.ms.awt.GraphicsX.go は変換されませんでした。

com.ms.awt.GraphicsX.GraphicsX は変換されませんでした。

com.ms.awt.GraphicsX.image は変換されませんでした。

com.ms.awt.GraphicsX.originX は変換されませんでした。

com.ms.awt.GraphicsX.originY は変換されませんでした。

com.ms.awt.GraphicsX.resetSurfaceParams は変換されませんでした。

com.ms.awt.GraphicsX.setClip は変換されませんでした。

com.ms.awt.GraphicsX.setPaintMode は変換されませんでした。

com.ms.awt.GraphicsX.setSurfaceOffset は変換されませんでした。

com.ms.awt.GraphicsX.setSurfaceOwner は変換されませんでした。

com.ms.awt.GraphicsX.setSurfaceVisRgn は変換されませんでした。

com.ms.awt.GraphicsX.setXORMode は変換されませんでした。

com.ms.awt.GraphicsXConstants.BDR_VALID は変換されませんでした。

com.ms.awt.HeavyComponent は変換されませんでした。

com.ms.awt.HorizBagLayout は変換されませんでした。

com.ms.awt.ImageX は変換されませんでした。

com.ms.awt.ListLayout は変換されませんでした。

com.ms.awt.MenuBarX.getItemID は変換されませんでした。

com.ms.awt.MenuBarX.MenuBarX(int) は変換されませんでした。

com.ms.awt.MenuBarX.MenuBarX(int, Applet, String) は変換されませんでした。

com.ms.awt.MenuBarX.MenuBarX(int, String) は変換されませんでした。

com.ms.awt.MenuItemX.addNotify は変換されませんでした。

com.ms.awt.MenuItemX.getID は変換されませんでした。

com.ms.awt.MenuX.CheckMenuItem は変換されませんでした。

com.ms.awt.MenuX.getItemID は変換されませんでした。

com.ms.awt.MenuXConstants は変換されませんでした。

com.ms.awt.OrientableFlowLayout は変換されませんでした。

com.ms.awt.PhysicalDrawingSurface は変換されませんでした。

com.ms.awt.PlatformFont は変換されませんでした。

com.ms.awt.VariableGridLayout は変換されませんでした。

com.ms.awt.VerticalBagLayout は変換されませんでした。

com.ms.awt.WClipboard.lostClipboard は変換されませんでした。

com.ms.awt.WClipboard.lostSelectionOwnership は変換されませんでした。

com.ms.awt.WClipboard.setToolkit は変換されませんでした。

com.ms.awt.WClipboard.WClipboard は変換されませんでした。

com.ms.awt.WComponentPeer は変換されませんでした。

com.ms.awt.WDragSession は変換されませんでした。

com.ms.awt.WEventQueue は変換されませんでした。

com.ms.awt.WFileDialogPeer は変換されませんでした。

com.ms.awt.WGuiCallback は変換されませんでした。

com.ms.awt.WHeavyPeer は変換されませんでした。

com.ms.awt.Win32SystemResourceDecoder は変換されませんでした。

com.ms.awt.WinEvent.notify は変換されませんでした。

com.ms.awt.WPrintGraphics は変換されませんでした。

com.ms.awt.WPrintJob は変換されませんでした。

com.ms.awt.WToolkit は変換されませんでした。

Com.ms.com のエラー メッセージ

com.ms.com.AnsiStringMarshaller は変換されませんでした。

com.ms.com.AnsiStringRef は変換されませんでした。

com.ms.com.COAUTHIDENTITY.cbByValSize は変換されませんでした。

com.ms.com.COAUTHIDENTITY.domain は変換されませんでした。

com.ms.com.COAUTHIDENTITY.flags は変換されませんでした。

com.ms.com.COAUTHIDENTITY.fromPtr は変換されませんでした。

com.ms.com.COAUTHIDENTITY.password は変換されませんでした。

com.ms.com.COAUTHIDENTITY.toExternal は変換されませんでした。

com.ms.com.COAUTHIDENTITY.toJava は変換されませんでした。

com.ms.com.COAUTHIDENTITY.toPtr は変換されませんでした。

com.ms.com.COAUTHIDENTITY.user は変換されませんでした。

com.ms.com.COAUTHINFO.pAuthIdentityData は変換されませんでした。

com.ms.com.COAUTHINFO.pwszServerPrincName は変換されませんでした。

com.ms.com.ComContext は変換されませんでした。

com.ms.com.ComException.ComException は変換されませんでした。

com.ms.com.ComException.getHelpContext は変換されませんでした。

com.ms.com.ComException.getHResult は変換されませんでした。

com.ms.com.ComException.hr は変換されませんでした。

com.ms.com.ComException.m_helpContext は変換されませんでした。

com.ms.com.ComException.m_helpFile は変換されませんでした。

com.ms.com.ComException.m_source は変換されませんでした。

com.ms.com.ComFailException.ComFailException は変換されませんでした。

com.ms.com.ComLib.ComLib は変換されませんでした。

com.ms.com.ComLib.declareMessagePumpThread は変換されませんでした。

com.ms.com.ComLib.executeOnContext は変換されませんでした。

com.ms.com.ComLib.freeUnusedLibraries は変換されませんでした。

com.ms.com.ComLib.IENVNextMarshalerC2J は変換されませんでした。

com.ms.com.ComLib.IENVNextMarshalerJ2C は変換されませんでした。

com.ms.com.ComLib.IID_IDispatch は変換されませんでした。

com.ms.com.ComLib.IID_IUnknown は変換されませんでした。

com.ms.com.ComLib.isEqualUnknown は変換されませんでした。

com.ms.com.ComLib.jcdwClassOffsetOf は変換されませんでした。

com.ms.com.ComLib.jcdwOffsetOf は変換されませんでした。

com.ms.com.ComLib.makeProxyRef は変換されませんでした。

com.ms.com.ComLib.ownsCleanup は変換されませんでした。

com.ms.com.ComLib.ptrToUnknown は変換されませんでした。

com.ms.com.ComLib.setDataWrapperSize は変換されませんでした。

com.ms.com.ComLib.startMTAThread は変換されませんでした。

com.ms.com.ComLib.supportsInterface は変換されませんでした。

com.ms.com.ComLib.threadStartMTA は変換されませんでした。

com.ms.com.ComLib.unknownToPtr は変換されませんでした。

com.ms.com.ComSuccessException.ComSuccessException は変換されませんでした。

com.ms.com.CONNECTDATA.pUnk は変換されませんでした。

com.ms.com.COSERVERINFO.pAuthInfo は変換されませんでした。

com.ms.com.COSERVERINFO.pwszName は変換されませんでした。

com.ms.com.CUnknown は変換されませんでした。

com.ms.com.CustomLib は変換されませんでした。

com.ms.com.Dispatch は変換されませんでした。

com.ms.com.DispatchProxy は変換されませんでした。

com.ms.com.Generic は変換されませんでした。

com.ms.com.Guid.Guid は変換されませんでした。

com.ms.com.Guid.setByStr は変換されませんでした。

com.ms.com.IAccessible.iid は変換されませんでした。

com.ms.com.IAccessibleDefault.iid は変換されませんでした。

com.ms.com.IBindCtx.iid は変換されませんでした。

com.ms.com.IBindCtx.RegisterObjectBound は変換されませんでした。

com.ms.com.IBindCtx.RevokeObjectBound は変換されませんでした。

com.ms.com.IBindCtx.RevokeObjectParam は変換されませんでした。

com.ms.com.IClassFactory は変換されませんでした。

com.ms.com.IClassFactory2 は変換されませんでした。

com.ms.com.IConnectionPoint.Advise は変換されませんでした。

com.ms.com.IConnectionPoint.iid は変換されませんでした。

com.ms.com.IConnectionPointContainer.iid は変換されませんでした。

com.ms.com.IEnumConnectionPoints.iid は変換されませんでした。

com.ms.com.IEnumConnectionPoints.Next は変換されませんでした。

com.ms.com.IEnumConnections.iid は変換されませんでした。

com.ms.com.IEnumConnections.Next は変換されませんでした。

com.ms.com.IEnumConnections.Skip は変換されませんでした。

com.ms.com.IEnumMoniker.iid は変換されませんでした。

com.ms.com.IEnumMoniker.Next は変換されませんでした。

com.ms.com.IEnumSTATSTG は変換されませんでした。

com.ms.com.IEnumString.iid は変換されませんでした。

com.ms.com.IEnumString.Next は変換されませんでした。

com.ms.com.IEnumUnknown は変換されませんでした。

com.ms.com.IEnumVariant.Clone は変換されませんでした。

com.ms.com.IExternalConnectionSink は変換されませんでした。

com.ms.com.IIDIsMarshaler は変換されませんでした。

com.ms.com.ILicenseMgr は変換されませんでした。

com.ms.com.ILockBytes は変換されませんでした。

com.ms.com.IMarshal.GetMarshalSizeMax は変換されませんでした。

com.ms.com.IMarshal.GetUnmarshalClass は変換されませんでした。

com.ms.com.IMarshal.iid は変換されませんでした。

com.ms.com.IMoniker.BindToObject は変換されませんでした。

com.ms.com.IMoniker.iid は変換されませんでした。

com.ms.com.IMoniker.IsDirty は変換されませんでした。

com.ms.com.IMoniker.IsEqual は変換されませんでした。

com.ms.com.IMoniker.IsRunning は変換されませんでした。

com.ms.com.IParseDisplayName は変換されませんでした。

com.ms.com.IPersist.iid は変換されませんでした。

com.ms.com.IPersistFile.iid は変換されませんでした。

com.ms.com.IPersistFile.IsDirty は変換されませんでした。

com.ms.com.IPersistFile.Save は変換されませんでした。

com.ms.com.IPersistStorage は変換されませんでした。

com.ms.com.IPersistStream.GetClassID は変換されませんでした。

com.ms.com.IPersistStream.iid は変換されませんでした。

com.ms.com.IPersistStreamInit は変換されませんでした。

com.ms.com.IPropertyNotifySink は変換されませんでした。

com.ms.com.IROTData は変換されませんでした。

com.ms.com.IRunningObjectTable.GetObject は変換されませんでした。

com.ms.com.IRunningObjectTable.GetTimeOfLastChange は変換されませんでした。

com.ms.com.IRunningObjectTable.iid は変換されませんでした。

com.ms.com.IRunningObjectTable.IsRunning は変換されませんでした。

com.ms.com.IRunningObjectTable.NoteChangeTime は変換されませんでした。

com.ms.com.IRunningObjectTable.Register は変換されませんでした。

com.ms.com.ISequentialStream は変換されませんでした。

com.ms.com.ISequentialStream.iid は変換されませんでした。

com.ms.com.IServiceProvider は変換されませんでした。

com.ms.com.IStorage は変換されませんでした。

com.ms.com.IStream.CopyTo は変換されませんでした。

com.ms.com.IStream.iid は変換されませんでした。

com.ms.com.IStream.LOCK_EXCLUSIVE は変換されませんでした。

com.ms.com.IStream.LOCK_ONLYONCE は変換されませんでした。

com.ms.com.IStream.LOCK_WRITE は変換されませんでした。

com.ms.com.IStream.Read は変換されませんでした。

com.ms.com.IStream.Seek は変換されませんでした。

com.ms.com.IStream.STATFLAG_DEFAULT は変換されませんでした。

com.ms.com.IStream.STATFLAG_NONAME は変換されませんでした。

com.ms.com.IStream.STATFLAG_NOOPEN は変換されませんでした。

com.ms.com.IStream.STGC_DANGEROUSLYCOMMITMERELYTODISKCACHE は変換されませんでした。

com.ms.com.IStream.STGC_DEFAULT は変換されませんでした。

com.ms.com.IStream.STGC_ONLYIFCURRENT は変換されませんでした。

com.ms.com.IStream.STGC_OVERWRITE は変換されませんでした。

com.ms.com.IStream.STREAM_SEEK_CUR は変換されませんでした。

com.ms.com.IStream.STREAM_SEEK_END は変換されませんでした。

com.ms.com.IStream.STREAM_SEEK_SET は変換されませんでした。

com.ms.com.IStream.Write は変換されませんでした。

com.ms.com.LicenseMgr は変換されませんでした。

com.ms.com.LICINFO は変換されませんでした。

com.ms.com.MULTI_QI は変換されませんでした。

com.ms.com.NoAutoMarshaling は変換されませんでした。

com.ms.com.NoAutoScripting は変換されませんでした。

com.ms.com.SafeArray.destroy は変換されませんでした。

com.ms.com.SafeArray.getFeatures は変換されませんでした。

com.ms.com.SafeArray.getNumLocks は変換されませんでした。

com.ms.com.SafeArray.getPhysicalSafeArray は変換されませんでした。

com.ms.com.SafeArray.getvt は変換されませんでした。

com.ms.com.SafeArray.reinit は変換されませんでした。

com.ms.com.SafeArray.reinterpretType は変換されませんでした。

com.ms.com.SizeIsMarshaler は変換されませんでした。

com.ms.com.STATSTG.clsid_data1 は変換されませんでした。

com.ms.com.STATSTG.clsid_data2 は変換されませんでした。

com.ms.com.STATSTG.clsid_data3 は変換されませんでした。

com.ms.com.STATSTG.STGTY_LOCKBYTES は変換されませんでした。

com.ms.com.STATSTG.STGTY_PROPERTY は変換されませんでした。

com.ms.com.STATSTG.STGTY_STORAGE は変換されませんでした。

com.ms.com.STATSTG.STGTY_STREAM は変換されませんでした。

com.ms.com.StdCOMClassObject は変換されませんでした。

com.ms.com.UniStringMarshaller は変換されませんでした。

com.ms.com.UniStringRef は変換されませんでした。

com.ms.com.Variant.changeType は変換されませんでした。

com.ms.com.Variant.clone は変換されませんでした。

com.ms.com.Variant.cloneIndirect は変換されませんでした。

com.ms.com.Variant.getDate は変換されませんでした。

com.ms.com.Variant.getEmpty は変換されませんでした。

com.ms.com.Variant.getErrorRef は変換されませんでした。

com.ms.com.Variant.getNull は変換されませんでした。

com.ms.com.Variant.getVariantArray は変換されませんでした。

com.ms.com.Variant.getVariantArrayRef は変換されませんでした。

com.ms.com.Variant.getvt は変換されませんでした。

com.ms.com.Variant.noParam は変換されませんでした。

com.ms.com.Variant.putError は変換されませんでした。

com.ms.com.Variant.putSafeArrayRefHelper は変換されませんでした。

com.ms.com.Variant.toError は変換されませんでした。

com.ms.com.Variant.toScriptObject は変換されませんでした。

com.ms.com.Variant.toVariantArray は変換されませんでした。

com.ms.com.Variant.Variant は変換されませんでした。

com.ms.com.Variant.VariantByref は変換されませんでした。

com.ms.com.Variant.VariantClear は変換されませんでした。

com.ms.com.Variant.VariantCurrency は変換されませんでした。

com.ms.com.Variant.VariantEmpty は変換されませんでした。

com.ms.com.Variant.VariantError は変換されませんでした。

com.ms.com.Variant.VariantNull は変換されませんでした。

com.ms.com.Variant.VariantTypeMask は変換されませんでした。

com.ms.com.Variant.VariantVariant は変換されませんでした。

Com.ms.directx のエラー メッセージ

com.ms.directX.D3dFindDeviceResult.GetGuid は変換されませんでした。

com.ms.directX.D3dFindDeviceSearch.getGuid は変換されませんでした。

com.ms.directX.D3dFindDeviceSearch.setGuid は変換されませんでした。

com.ms.directX.Direct3dRMFace.getVertices は変換されませんでした。

com.ms.directX.Direct3dRMMesh.getVertices は変換されませんでした。

com.ms.directX.Direct3dRMMesh.setVertices は変換されませんでした。

com.ms.directX.Direct3dRMMeshBuilder.addFaces は変換できませんでした。

com.ms.directX.Direct3dRMMeshBuilder.getVertices は変換できませんでした。

com.ms.directX.DirectDraw.createPalette は変換されませんでした。

com.ms.directX.DirectDraw.setCooperativeLevel は変換されませんでした。

com.ms.directX.DirectDrawClipper.resetSurfaceParams は変換されませんでした。

com.ms.directX.DirectDrawClipper.setComponent は変換されませんでした。

com.ms.directX.DirectDrawClipper.setSurfaceOffset は変換されませんでした。

com.ms.directX.DirectDrawClipper.setSurfaceOwner は変換されませんでした。

com.ms.directX.DirectDrawClipper.setSurfaceVisRgn は変換されませんでした。

com.ms.directX.DirectDrawPalette.getColorEntries は変換されませんでした。

com.ms.directX.DirectDrawPalette.getPaletteEntries は変換されませんでした。

com.ms.directX.DirectDrawPalette.setEntries は変換されませんでした。

com.ms.directX.DirectSound.createSoundBuffer は変換されませんでした。

com.ms.directX.DirectSound.setCooperativeLevel は変換されませんでした。

com.ms.directX.DirectSoundBuffer.getFormat は変換されませんでした。

com.ms.directX.DirectSoundBuffer.initialize は変換されませんでした。

com.ms.directX.DirectSoundBuffer.setFormat は変換されませんでした。

com.ms.directX.DirectSoundResource.loadWaveFile は変換されませんでした。

com.ms.directX.DirectSoundResource.loadWaveResource は変換されませんでした。

com.ms.directX.DirectXConstants は変換されませんでした。

com.ms.directX.PaletteEntry は変換されませんでした。

com.ms.directX.WaveFormatEx は変換されませんでした。

Com.ms.dll のエラーメッセージ

com.ms.dll.Callback は変換されませんでした。

com.ms.dll.DllLib.addrOf は変換されませんでした。

com.ms.dll.DllLib.addrOfPinnedObject は変換されませんでした。

com.ms.dll.DllLib.copy は変換されませんでした。

com.ms.dll.DllLib.DllLib は変換されませんでした。

com.ms.dll.DllLib.freePinnedHandle は変換されませんでした。

com.ms.dll.DllLib.getPinnedHandle は変換されませんでした。

com.ms.dll.DllLib.getPinnedObject は変換されませんでした。

com.ms.dll.DllLib.isStruct は変換されませんでした。

com.ms.dll.DllLib.propagateStructFields は変換されませんでした。

com.ms.dll.DllLib.resize は変換されませんでした。

com.ms.dll.ParameterCountMismatchError は変換されませんでした。

com.ms.dll.StringMarshaler は変換されませんでした。

Com.ms.dxmedia のエラー メッセージ

com.ms.dxmedia.AppTriggeredEvent は変換されませんでした。

com.ms.dxmedia.ArrayBvr.ArrayBvr は変換されませんでした。

com.ms.dxmedia.ArrayBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.ArrayBvr.newUninitBvr は変換されませんでした。

com.ms.dxmedia.ArrayBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Bbox2Bvr.Bbox2Bvr は変換されませんでした。

com.ms.dxmedia.Bbox2Bvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.Bbox2Bvr.getMax は変換されませんでした。

com.ms.dxmedia.Bbox2Bvr.getMin は変換されませんでした。

com.ms.dxmedia.Bbox2Bvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.Bbox2Bvr.newUninitBvr は変換されませんでした。

com.ms.dxmedia.Bbox2Bvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Bbox3Bvr.Bbox3Bvr は変換されませんでした。

com.ms.dxmedia.Bbox3Bvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.Bbox3Bvr.getMax は変換されませんでした。

com.ms.dxmedia.Bbox3Bvr.getMin は変換されませんでした。

com.ms.dxmedia.Bbox3Bvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.Bbox3Bvr.newUninitBvr は変換されませんでした。

com.ms.dxmedia.Bbox3Bvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Behavior.Behavior は変換されませんでした。

com.ms.dxmedia.Behavior.debug は変換されませんでした。

com.ms.dxmedia.Behavior.extract は変換されませんでした。

com.ms.dxmedia.Behavior.getCOMBvr は変換されませんでした。

com.ms.dxmedia.Behavior.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.BooleanBvr.BooleanBvr は変換されませんでした。

com.ms.dxmedia.BooleanBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.BooleanBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.BooleanBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.CallbackNotifier は変換されませんでした。

com.ms.dxmedia.CameraBvr.CameraBvr は変換されませんでした。

com.ms.dxmedia.CameraBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.CameraBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.CameraBvr.newUninitBvr は変換されませんでした。

com.ms.dxmedia.CameraBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.ColorBvr.ColorBvr は変換されませんでした。

com.ms.dxmedia.ColorBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.ColorBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.ColorBvr.NewUninitBvr は変換されませんでした。

com.ms.dxmedia.ColorBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Cycler は変換されませんでした。

com.ms.dxmedia.DashStyleBvr.DashStyleBvr は変換されませんでした。

com.ms.dxmedia.DashStyleBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.DashStyleBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.DashStyleBvr.newUninitBvr は変換されませんでした。

com.ms.dxmedia.DashStyleBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.DefaultErrReceiver は変換されませんでした。

com.ms.dxmedia.DXMApplet は変換されませんでした。

com.ms.dxmedia.DXMCanvas は変換されませんでした。

com.ms.dxmedia.DXMCanvasBase は変換されませんでした。

com.ms.dxmedia.DXMDebugCallback は変換されませんでした。

com.ms.dxmedia.DXMEvent.DXMEvent は変換されませんでした。

com.ms.dxmedia.DXMEvent.getCOMPtr は変換されませんでした。

com.ms.dxmedia.DXMEvent.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.DXMEvent.registerCallback は変換されませんでした。

com.ms.dxmedia.DXMEvent.setCOMBvr は変換されませんでした。

com.ms.dxmedia.DXMException.DXMException は変換されませんでした。

com.ms.dxmedia.EndStyleBvr.EndStyleBvr は変換されませんでした。

com.ms.dxmedia.EndStyleBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.EndStyleBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.EndStyleBvr.newUninitBvr は変換されませんでした。

com.ms.dxmedia.EndStyleBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.ErrorAndWarningReceiver は変換されませんでした。

com.ms.dxmedia.EventCallbackObject は変換されませんでした。

com.ms.dxmedia.FontStyleBvr.FontStyleBvr は変換されませんでした。

com.ms.dxmedia.FontStyleBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.FontStyleBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.FontStyleBvr.NewUninitBvr は変換されませんでした。

com.ms.dxmedia.FontStyleBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.GeometryBvr.GeometryBvr は変換されませんでした。

com.ms.dxmedia.GeometryBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.GeometryBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.GeometryBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.ImageBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.ImageBvr.ImageBvr は変換されませんでした。

com.ms.dxmedia.ImageBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.ImageBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.JoinStyleBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.JoinStyleBvr.JoinStyleBvr は変換されませんでした。

com.ms.dxmedia.JoinStyleBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.JoinStyleBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.LineStyleBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.LineStyleBvr.LineStyleBvr は変換されませんでした。

com.ms.dxmedia.LineStyleBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.LineStyleBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.MatteBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.MatteBvr.MatteBvr は変換されませんでした。

com.ms.dxmedia.MatteBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.MatteBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.MicrophoneBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.MicrophoneBvr.MicrophoneBvr は変換されませんでした。

com.ms.dxmedia.MicrophoneBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.MicrophoneBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Model.cleanup は変換されませんでした。

com.ms.dxmedia.Model.createModel は変換されませんでした。

com.ms.dxmedia.Model.getImportBase は変換されませんでした。

com.ms.dxmedia.Model.modifyPreferences は変換されませんでした。

com.ms.dxmedia.Model.receiveInputImages は変換されませんでした。

com.ms.dxmedia.Model.setImportBase は変換されませんでした。

com.ms.dxmedia.ModelMakerApplet は変換されませんでした。

com.ms.dxmedia.ModifiableBehavior は変換されませんでした。

com.ms.dxmedia.MontageBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.MontageBvr.MontageBvr は変換されませんでした。

com.ms.dxmedia.MontageBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.MontageBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.NumberBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.NumberBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.NumberBvr.NumberBvr は変換されませんでした。

com.ms.dxmedia.NumberBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.PairObject.PairObject は変換されませんでした。

com.ms.dxmedia.Path2Bvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.Path2Bvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.Path2Bvr.Path2Bvr は変換されませんでした。

com.ms.dxmedia.Path2Bvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.PickableGeometry.PickableGeometry は変換されませんでした。

com.ms.dxmedia.PickableImage.PickableImage は変換されませんでした。

com.ms.dxmedia.Point2Bvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.Point2Bvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.Point2Bvr.Point2Bvr は変換されませんでした。

com.ms.dxmedia.Point2Bvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Point3Bvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.Point3Bvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.Point3Bvr.Point3Bvr は変換されませんでした。

com.ms.dxmedia.Point3Bvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Preferences.COLOR_KEY_BLUE は変換されませんでした。

com.ms.dxmedia.Preferences.COLOR_KEY_GREEN は変換されませんでした。

com.ms.dxmedia.Preferences.COLOR_KEY_RED は変換されませんでした。

com.ms.dxmedia.Preferences.DITHERING は変換されませんでした。

com.ms.dxmedia.Preferences.ENGINE_OPTIMIZATIONS は変換されませんでした。

com.ms.dxmedia.Preferences.FILL_MODE は変換されませんでした。

com.ms.dxmedia.Preferences.FILL_MODE_POINT は変換されませんでした。

com.ms.dxmedia.Preferences.FILL_MODE_SOLID は変換されませんでした。

com.ms.dxmedia.Preferences.FILL_MODE_WIREFRAME は変換されませんでした。

com.ms.dxmedia.Preferences.MAX_FRAMES_PER_SEC は変換されませんでした。

com.ms.dxmedia.Preferences.OVERRIDE_APPLICATION_PREFERENCES は変換されませんでした。

com.ms.dxmedia.Preferences.PERSPECTIVE_CORRECT は変換されませんでした。

com.ms.dxmedia.Preferences.RGB_LIGHTING_MODE は変換されませんでした。

com.ms.dxmedia.Preferences.SHADE_MODE は変換されませんでした。

com.ms.dxmedia.Preferences.SHADE_MODE_FLAT は変換されませんでした。

com.ms.dxmedia.Preferences.SHADE_MODE_GOURAUD は変換されませんでした。

com.ms.dxmedia.Preferences.SHADE_MODE_PHONG は変換されませんでした。

com.ms.dxmedia.Preferences.TEXTURE_QUALITY は変換されませんでした。

com.ms.dxmedia.Preferences.TEXTURE_QUALITY_LINEAR は変換されませんでした。

com.ms.dxmedia.Preferences.TEXTURE_QUALITY_NEAREST は変換されませんでした。

com.ms.dxmedia.Preferences.USE_3D_HW は変換されませんでした。

com.ms.dxmedia.Preferences.USE_VIDEOMEM は変換されませんでした。

com.ms.dxmedia.PropertyDispatcher は変換されませんでした。

com.ms.dxmedia.SoundBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.SoundBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.SoundBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.SoundBvr.SoundBvr は変換されませんでした。

com.ms.dxmedia.Statics.makeBvrFromInterface は変換されませんでした。

com.ms.dxmedia.StaticsBase._site は変換されませんでした。

com.ms.dxmedia.StaticsBase.BvrHook は変換されませんでした。

com.ms.dxmedia.StaticsBase.checkRead は変換されませんでした。

com.ms.dxmedia.StaticsBase.cm は変換されませんでした。

com.ms.dxmedia.StaticsBase.foot は変換されませんでした。

com.ms.dxmedia.StaticsBase.getCOMPtr は変換されませんでした。

com.ms.dxmedia.StaticsBase.handleError は変換されませんでした。

com.ms.dxmedia.StaticsBase.importGeometry は変換されませんでした。

com.ms.dxmedia.StaticsBase.importImage は変換されませんでした。

com.ms.dxmedia.StaticsBase.importMovie (非同期) は変換されませんでした。

com.ms.dxmedia.StaticsBase.importMovie (同期) は変換されませんでした。

com.ms.dxmedia.StaticsBase.importSound (非同期) は変換されませんでした。

com.ms.dxmedia.StaticsBase.importSound (同期) は変換されませんでした。

com.ms.dxmedia.StaticsBase.inch は変換されませんでした。

com.ms.dxmedia.StaticsBase.meter は変換されませんでした。

com.ms.dxmedia.StaticsBase.mm は変換されませんでした。

com.ms.dxmedia.StaticsBase.registerErrorAndWarningReceiver は変換されませんでした。

com.ms.dxmedia.StaticsBase.unregisterCallback は変換されませんでした。

com.ms.dxmedia.StringBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.StringBvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.StringBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.StringBvr.StringBvr は変換されませんでした。

com.ms.dxmedia.Transform2Bvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.Transform2Bvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.Transform2Bvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Transform2Bvr.Transform2Bvr は変換されませんでした。

com.ms.dxmedia.Transform3Bvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.Transform3Bvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.Transform3Bvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Transform3Bvr.Transform3Bvr は変換されませんでした。

com.ms.dxmedia.TupleBvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.TupleBvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.TupleBvr.TupleBvr は変換されませんでした。

com.ms.dxmedia.UntilNotifierCB は変換されませんでした。

com.ms.dxmedia.Vector2Bvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.Vector2Bvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.Vector2Bvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Vector2Bvr.Vector2Bvr は変換されませんでした。

com.ms.dxmedia.Vector3Bvr.getCOMPtr は変換されませんでした。

com.ms.dxmedia.Vector3Bvr.newUninitBehavior は変換されませんでした。

com.ms.dxmedia.Vector3Bvr.setCOMBvr は変換されませんでした。

com.ms.dxmedia.Vector3Bvr.Vector3Bvr は変換されませんでした。

com.ms.dxmedia.Viewer.getCurrentTickTime は変換されませんでした。

com.ms.dxmedia.Viewer.registerErrorAndWarningReceiver は変換されませんでした。

com.ms.dxmedia.Viewer.startModel は変換されませんでした。

com.ms.dxmedia.Viewer.tick は変換されませんでした。

Com.ms.fx のエラーメッセージ

com.ms.fx.BaseColor.BaseColor は変換されませんでした。

com.ms.fx.fullTxtRun は変換されませんでした。

com.ms.fx.FxBrushPen.drawBytesCallback は変換されませんでした。

com.ms.fx.FxBrushPen.drawCharsCallback は変換されませんでした。

com.ms.fx.FxBrushPen.drawRoundRectCallback は変換されませんでした。

com.ms.fx.FxBrushPen.drawScanLinesCallback は変換されませんでした。

com.ms.fx.FxBrushPen.drawStringCallback は変換されませんでした。

com.ms.fx.FxBrushPen.fill3DRectCallback は変換されませんでした。

com.ms.fx.FxBrushPen.fillRoundRectCallback は変換されませんでした。

com.ms.fx.FxCaret は変換されませんでした。

com.ms.fx.FxColor.brightenColor は変換されませんでした。

com.ms.fx.FxColor.darkenColor は変換されませんでした。

com.ms.fx.FxColor.FxColor は変換されませんでした。

com.ms.fx.FxColor.getHilight は変換されませんでした。

com.ms.fx.FxColor.getShadow は変換されませんでした。

com.ms.fx.FxComponentImage は変換されませんでした。

com.ms.fx.FxComponentTexture.FxComponentTexture は変換されませんでした。

com.ms.fx.FxCurve は変換されませんでした。

com.ms.fx.FxEllipse は変換されませんでした。

com.ms.fx.FxFont.ANTIALIAS は変換されませんでした。

com.ms.fx.FxFont.drawEffects は変換されませんでした。

com.ms.fx.FxFont.FONTXFONT は変換されませんでした。

com.ms.fx.FxFont.FxFont は変換されませんでした。

com.ms.fx.FxFont.getAttributeList は変換されませんでした。

com.ms.fx.FxFont.getEmboldenedFont は変換されませんでした。

com.ms.fx.FxFont.getFlags は変換されませんでした。

com.ms.fx.FxFont.getFlagsVal は変換されませんでした。

com.ms.fx.FxFont.getFont は変換されませんでした。

com.ms.fx.FxFont.getFontList は変換されませんでした。

com.ms.fx.FxFont.getStyleVal は変換されませんでした。

com.ms.fx.FxFont.matchFace は変換されませんでした。

com.ms.fx.FxFont.STRIKEOUT は変換されませんでした。

com.ms.fx.FxFont.UNDERLINE は変換されませんでした。

com.ms.fx.FxFont.USEDFONT は変換されませんでした。

com.ms.fx.FxFontMetrics は変換されませんでした。

com.ms.fx.FxFontMetricsOther は変換されませんでした。

com.ms.fx.FxFormattedText は変換されませんでした。

com.ms.fx.FxGraphicMetaFile.addObjectToTable は変換されませんでした。

com.ms.fx.FxGraphicMetaFile.debugging は変換されませんでした。

com.ms.fx.FxGraphicMetaFile.enumerate は変換されませんでした。

com.ms.fx.FxGraphicMetaFile.FxGraphicMetaFile は変換されませんでした。

com.ms.fx.FxGraphicMetaFile.hdc は変換されませんでした。

com.ms.fx.FxGraphicMetaFile.hdcStates は変換されませんでした。

com.ms.fx.FxGraphicMetaFile.play は変換されませんでした。

com.ms.fx.FxGraphicMetaFile.records は変換されませんでした。

com.ms.fx.FxGraphicMetaFile.removeObjectFromTable は変換されませんでした。

com.ms.fx.FxGraphics.drawBezier は変換されませんでした。

com.ms.fx.FxGraphics.drawBorder は変換されませんでした。

com.ms.fx.FxGraphics.drawChars は変換されませんでした。

com.ms.fx.FxGraphics.drawCharsWithoutFxFont は変換されませんでした。

com.ms.fx.FxGraphics.drawOutlineChar は変換されませんでした。

com.ms.fx.FxGraphics.drawOutlinePolygon は変換されませんでした。

com.ms.fx.FxGraphics.drawPixels は変換されませんでした。

com.ms.fx.FxGraphics.drawScanLines は変換されませんでした。

com.ms.fx.FxGraphics.drawString は変換されませんでした。

com.ms.fx.FxGraphics.drawString(int, int) は変換できませんでした。

com.ms.fx.FxGraphics.drawStringFormatted は変換されませんでした。

com.ms.fx.FxGraphics.drawStringWithoutFxFont は変換されませんでした。

com.ms.fx.FxGraphics.drawT2Curve は変換されませんでした。

com.ms.fx.FxGraphics.fill3DRect は変換されませんでした。

com.ms.fx.FxGraphics.getClip は変換されませんでした。

com.ms.fx.FxGraphics.getExtendedGraphics は変換されませんでした。

com.ms.fx.FxGraphics.getGlyphOutline は変換されませんでした。

com.ms.fx.FxGraphics.intelliFont は変換されませんでした。

com.ms.fx.FxGraphics.setClip は変換されませんでした。

com.ms.fx.FxGraphicsOVM.baseGraphics は変換されませんでした。

com.ms.fx.FxGraphicsOVM.clearRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM.clipRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM.copyArea は変換されませんでした。

com.ms.fx.FxGraphicsOVM.create は変換されませんでした。

com.ms.fx.FxGraphicsOVM.dispose は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawArc は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawBezier は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawBorder は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawBytes は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawChars は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawChars(char[], int, int, int, int) は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawChars(char[], int, int, int, int, Rectangle, int, int[], int[]) は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawCharsWithoutFxFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawLine は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawOutlinePolygon は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawOval は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawPixels は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawPolygon は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawPolyline は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawRoundRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawScanLines は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawString は変換されませんでした。

com.ms.fx.FxGraphicsOVM.drawStringWithoutFxFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM.excludeClip は変換されませんでした。

com.ms.fx.FxGraphicsOVM.fillArc は変換されませんでした。

com.ms.fx.FxGraphicsOVM.fillOval は変換されませんでした。

com.ms.fx.FxGraphicsOVM.fillPolygon は変換されませんでした。

com.ms.fx.FxGraphicsOVM.fillRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM.fillRoundRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM.FxGraphicsOVM は変換されませんでした。

com.ms.fx.FxGraphicsOVM.getBaseGraphics は変換されませんでした。

com.ms.fx.FxGraphicsOVM.getClipBounds は変換されませんでした。

com.ms.fx.FxGraphicsOVM.getClipRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM.getClipRegion は変換されませんでした。

com.ms.fx.FxGraphicsOVM.getColor は変換されませんでした。

com.ms.fx.FxGraphicsOVM.getFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM.getFontMetrics は変換されませんでした。

com.ms.fx.FxGraphicsOVM.getGlyphOutline は変換されませんでした。

com.ms.fx.FxGraphicsOVM.getTranslation は変換されませんでした。

com.ms.fx.FxGraphicsOVM.hardClipRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM.intersectClip は変換されませんでした。

com.ms.fx.FxGraphicsOVM.nativeSetFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM.originX は変換されませんでした。

com.ms.fx.FxGraphicsOVM.originY は変換されませんでした。

com.ms.fx.FxGraphicsOVM.runningAFC は変換されませんでした。

com.ms.fx.FxGraphicsOVM.setClip は変換されませんでした。

com.ms.fx.FxGraphicsOVM.setClip(int, int, int) は変換できませんでした。

com.ms.fx.FxGraphicsOVM.setClip(Region) は変換されませんでした。

com.ms.fx.FxGraphicsOVM.setClip(Shape) は変換できませんでした。

com.ms.fx.FxGraphicsOVM.setColor は変換されませんでした。

com.ms.fx.FxGraphicsOVM.setFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM.setPaintMode は変換されませんでした。

com.ms.fx.FxGraphicsOVM.setXORMode は変換されませんでした。

com.ms.fx.FxGraphicsOVM.systemInterface は変換されませんでした。

com.ms.fx.FxGraphicsOVM11 は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.clearRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.clipRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.copyArea は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.dispose は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawArc は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawBezier は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawBorder は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawBytes は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawChars は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawCharsWithoutFxFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawLine は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawOutlinePolygon は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawOval は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawPixels は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawPolygon は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawPolyline は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawRoundRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawScanLines は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawString は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.drawStringWithoutFxFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.excludeClip は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.fillArc は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.fillOval は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.fillPolygon は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.fillRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.fillRoundRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.getBaseGraphics は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.getClipBounds は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.getClipRect は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.getClipRegion は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.getColor は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.getFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.getFontMetrics は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.getGlyphOutline は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.getTranslation は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.intersectClip は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.nativeSetFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.originX は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.originY は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.runningAFC は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.setClip は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.setColor は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.setFont は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.setPaintMode は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.setXORMode は変換されませんでした。

com.ms.fx.FxGraphicsOVM11.systemInterface は変換されませんでした。

com.ms.fx.FxOutlineFont は変換されませんでした。

com.ms.fx.FxPen.calcLineValues は変換されませんでした。

com.ms.fx.FxPen.drawRoundRectCallback は変換されませんでした。

com.ms.fx.FxPen.drawScanLinesCallback は変換されませんでした。

com.ms.fx.FxPen.fillRectCallback は変換されませんでした。

com.ms.fx.FxPen.FxPen は変換されませんでした。

com.ms.fx.FxPen.myDrawOval は変換されませんでした。

com.ms.fx.FxRubberPen は変換されませんでした。

com.ms.fx.FxStateConfigurableImage は変換されませんでした。

com.ms.fx.FxStateConfigurableUIImage.FxStateConfigurableUIImage は変換されませんでした。

com.ms.fx.FxStateConfigurableUIImage.getImageState は変換されませんでした。

com.ms.fx.FxStyledPen は変換されませんでした。

com.ms.fx.FxStyledPen.FxStyledPen は変換されませんでした。

com.ms.fx.FxSystemFont は変換されませんでした。

com.ms.fx.FxSystemIcon は変換されませんでした。

com.ms.fx.FxText.buffer は変換されませんでした。

com.ms.fx.FxText.getChar は変換されませんでした。

com.ms.fx.FxText.getWordBreak は変換されませんでした。

com.ms.fx.FxText.isDelimiter は変換されませんでした。

com.ms.fx.FxText.isWhite は変換されませんでした。

com.ms.fx.FxText.nChars は変換されませんでした。

com.ms.fx.FxText.setText は変換されませんでした。

com.ms.fx.FxTexture.DRAW_BL は変換されませんでした。

com.ms.fx.FxTexture.DRAW_BOTTOM は変換されませんでした。

com.ms.fx.FxTexture.DRAW_BR は変換されませんでした。

com.ms.fx.FxTexture.DRAW_CENTER は変換されませんでした。

com.ms.fx.FxTexture.DRAW_LEFT は変換されませんでした。

com.ms.fx.FxTexture.DRAW_RIGHT は変換されませんでした。

com.ms.fx.FxTexture.DRAW_TL は変換されませんでした。

com.ms.fx.FxTexture.DRAW_TOP は変換されませんでした。

com.ms.fx.FxTexture.DRAW_TR は変換されませんでした。

com.ms.fx.FxTexture.drawScanLinesCallback は変換されませんでした。

com.ms.fx.FxTexture.FxTexture は変換されませんでした。

com.ms.fx.FxTexture.getBottomAxis は変換されませんでした。

com.ms.fx.FxTexture.getInner は変換されませんでした。

com.ms.fx.FxTexture.getLeftAxis は変換されませんでした。

com.ms.fx.FxTexture.getPinOrigin は変換されませんでした。

com.ms.fx.FxTexture.getRightAxis は変換されませんでした。

com.ms.fx.FxTexture.getSnapDraw は変換されませんでした。

com.ms.fx.FxTexture.getStretch は変換されませんでした。

com.ms.fx.FxTexture.getTopAxis は変換されませんでした。

com.ms.fx.FxTexture.getUpdatedAreasMask は変換されませんでした。

com.ms.fx.FxTexture.imageUpdate は変換されませんでした。

com.ms.fx.FxTexture.REPEAT_PIN は変換されませんでした。

com.ms.fx.FxTexture.setAxis は変換されませんでした。

com.ms.fx.FxTexture.setPinOrigin は変換されませんでした。

com.ms.fx.FxTexture.setSnapDraw は変換されませんでした。

com.ms.fx.FxTexture.setStretch は変換されませんでした。

com.ms.fx.FxTexture.setUpdateCallback は変換されませんでした。

com.ms.fx.FxTexture.setUpdatedAreasMask は変換されませんでした。

com.ms.fx.FxTexture.size は変換されませんでした。

com.ms.fx.FxTexture.SNAP_EDGES は変換されませんでした。

com.ms.fx.FxTexture.STRETCH_MIDDLE は変換されませんでした。

com.ms.fx.FxTexture.STRETCH_OUTER は変換されませんでした。

com.ms.fx.FxToolkit は変換されませんでした。

com.ms.fx.GlyphMetrics は変換されませんでした。

com.ms.fx.GlyphOutline は変換されませんでした。

com.ms.fx.IFxShape は変換されませんでした。

com.ms.fx.IFxSystemInterface は変換されませんでした。

com.ms.fx.IFxTextCallback は変換されませんでした。

com.ms.fx.IFxTextConstants は変換されませんでした。

com.ms.fx.IFxTextConstants.DIRLAYOUT は変換されませんでした。

com.ms.fx.IFxTextConstants.htaCenter は変換されませんでした。

com.ms.fx.IFxTextConstants.htaJustified は変換されませんでした。

com.ms.fx.IFxTextConstants.htaLeft は変換されませんでした。

com.ms.fx.IFxTextConstants.htaRight は変換されませんでした。

com.ms.fx.IFxTextConstants.htaScriptDefault は変換されませんでした。

com.ms.fx.IFxTextConstants.htaStretch は変換されませんでした。

com.ms.fx.IFxTextConstants.MOVE_DOWN は変換されませんでした。

com.ms.fx.IFxTextConstants.MOVE_LEFT は変換されませんでした。

com.ms.fx.IFxTextConstants.MOVE_RIGHT は変換されませんでした。

com.ms.fx.IFxTextConstants.MOVE_UP は変換されませんでした。

com.ms.fx.IFxTextConstants.NEXT_DOWN は変換されませんでした。

com.ms.fx.IFxTextConstants.NEXT_LEFT は変換されませんでした。

com.ms.fx.IFxTextConstants.NEXT_RIGHT は変換されませんでした。

com.ms.fx.IFxTextConstants.NEXT_UP は変換されませんでした。

com.ms.fx.IFxTextConstants.OPAQUE_BODY は変換されませんでした。

com.ms.fx.IFxTextConstants.OPAQUE_POST は変換されませんでした。

com.ms.fx.IFxTextConstants.OPAQUE_PRIOR は変換されませんでした。

com.ms.fx.IFxTextConstants.SCRIPT_DEF は変換されませんでした。

com.ms.fx.IFxTextConstants.tdBT_LR は変換されませんでした。

com.ms.fx.IFxTextConstants.tdBT_RL は変換されませんでした。

com.ms.fx.IFxTextConstants.tdHebrewNormal は変換されませんでした。

com.ms.fx.IFxTextConstants.tdJapanTradNormal は変換されませんでした。

com.ms.fx.IFxTextConstants.tdLatinNormal は変換されませんでした。

com.ms.fx.IFxTextConstants.tdLeftToRightReading は変換されませんでした。

com.ms.fx.IFxTextConstants.tdLR_BT は変換されませんでした。

com.ms.fx.IFxTextConstants.tdLR_TB は変換されませんでした。

com.ms.fx.IFxTextConstants.tdMongolianNormal は変換されませんでした。

com.ms.fx.IFxTextConstants.tdRightToLeftReading は変換されませんでした。

com.ms.fx.IFxTextConstants.tdRL_BT は変換されませんでした。

com.ms.fx.IFxTextConstants.tdRL_TB は変換されませんでした。

com.ms.fx.IFxTextConstants.tdScriptDefault は変換されませんでした。

com.ms.fx.IFxTextConstants.tdTB_LR は変換されませんでした。

com.ms.fx.IFxTextConstants.tdTB_RL は変換されませんでした。

com.ms.fx.IFxTextConstants.tdVisualLayout は変換されませんでした。

com.ms.fx.IFxTextConstants.VISUAL_DEF は変換されませんでした。

com.ms.fx.IFxTextConstants.vtaBaseline は変換されませんでした。

com.ms.fx.IFxTextConstants.vtaBottom は変換されませんでした。

com.ms.fx.IFxTextConstants.vtaCenter は変換されませんでした。

com.ms.fx.IFxTextConstants.vtaScriptDefault は変換されませんでした。

com.ms.fx.IFxTextConstants.vtaStretch は変換されませんでした。

com.ms.fx.IFxTextConstants.vtaTop は変換されませんでした。

com.ms.fx.IFxTextConstants.wwCleanEdges は変換されませんでした。

com.ms.fx.IFxTextConstants.wwKeepWordIntact は変換されませんでした。

com.ms.fx.IFxTextConstants.wwMask は変換されませんでした。

com.ms.fx.IFxTextConstants.wwNone は変換されませんでした。

com.ms.fx.IFxTextConstants.wwTypeMask は変換されませんでした。

com.ms.fx.IFxTextConstants.wwVirtualRectEnd は変換されませんでした。

com.ms.fx.IFxTextConstants.wwVirtualRectSide は変換されませんでした。

com.ms.fx.IFxTextConstants.wwWrap は変換されませんでした。

com.ms.fx.IFxTextureUpdate は変換されませんでした。

com.ms.fx.OutlineCurve は変換されませんでした。

com.ms.fx.OutlinePolygon は変換されませんでした。

com.ms.fx.PeerConstants は変換されませんでした。

com.ms.fx.Region.clone は変換されませんでした。

com.ms.fx.Region.COMPLEX は変換されませんでした。

com.ms.fx.Region.complexity は変換されませんでした。

com.ms.fx.Region.copy は変換されませんでした。

com.ms.fx.Region.EMPTY~ は変換されませんでした。

com.ms.fx.Region.equals は変換されませんでした。

com.ms.fx.Region.fillBounds は変換されませんでした。

com.ms.fx.Region.getBounds は変換されませんでした。

com.ms.fx.Region.getGeometry は変換されませんでした。

com.ms.fx.Region.invert は変換されませんでした。

com.ms.fx.Region.isEmpty は変換されませんでした。

com.ms.fx.Region.Region は変換されませんでした。

com.ms.fx.Region.set は変換されませんでした。

com.ms.fx.Region.SIMPLE は変換されませんでした。

com.ms.fx.RegionConverter は変換されませんでした。

com.ms.fx.txtRun は変換されませんでした。

com.ms.fx.Version は変換されませんでした。

Com.ms.io のエラー メッセージ

com.ms.io.console.Console は変換されませんでした。

com.ms.io.console.DefaultConsole は変換されませんでした。

com.ms.io.ObjectInputStreamWithLoader は変換されませんでした。

com.ms.io.OffsetInputStreamFilter.mark は変換されませんでした。

com.ms.io.OffsetInputStreamFilter.reset は変換されませんでした。

com.ms.io.Path.hasExecExtensionType は変換されませんでした。

com.ms.io.Path.isRoot は変換されませんでした。

com.ms.io.Path.validateFilename は変換されませんでした。

com.ms.io.SystemInputStream は変換されませんでした。

com.ms.io.SystemOutputStream は変換されませんでした。

com.ms.io.UserFileDialog は変換されませんでした。

Com.ms.jdbc.odbc のエラー メッセージ

com.ms.jdbc.odbc.JdbcOdbcConnection.getAutoCommit は変換されませんでした。

com.ms.jdbc.odbc.JdbcOdbcConnection.getMetaData は変換されませんでした。

Com.ms.lang のエラー メッセージ

com.ms.lang.MulticastDelegate.invokeHelperMulticast は変換されませんでした。

com.ms.lang.RegKey.enumKey は変換されませんでした。

com.ms.lang.RegKey.finalize は変換されませんでした。

com.ms.lang.RegKey.getBinaryValue は変換されませんでした。

com.ms.lang.RegKey.KEYOPEN_ALL は変換されませんでした。

com.ms.lang.RegKey.KEYOPEN_CREATE は変換されませんでした。

com.ms.lang.RegKey.KEYOPEN_READ は変換されませんでした。

com.ms.lang.RegKey.KEYOPEN_WRITE は変換されませんでした。

com.ms.lang.RegKey.loadKey は変換されませんでした。

com.ms.lang.RegKey.queryInfo は変換されませんでした。

com.ms.lang.RegKey.replace は変換されませんでした。

com.ms.lang.RegKey.restore は変換されませんでした。

com.ms.lang.RegKey.unload は変換されませんでした。

com.ms.lang.RegQueryInfo は変換されませんでした。

com.ms.lang.SystemThread は変換されませんでした。

com.ms.lang.SystemX.arrayCompare は変換されませんでした。

com.ms.lang.SystemX.blockcopy は変換されませんでした。

com.ms.lang.SystemX.exitProcessAfterMainThreadReturns は変換されませんでした。

com.ms.lang.SystemX.getDeclaredMethodFromSignature は変換されませんでした。

com.ms.lang.SystemX.getDefaultInputManager は変換されませんでした。

com.ms.lang.SystemX.getInputManager は変換されませんでした。

com.ms.lang.SystemX.getKeyboardLanguageName は変換されませんでした。

com.ms.lang.SystemX.getKeyboardLanguages は変換されませんでした。

com.ms.lang.SystemX.getMethodFromSignature は変換されませんでした。

com.ms.lang.SystemX.getNativeServices は変換されませんでした。

com.ms.lang.SystemX.isLocalCharDBCSLeadByte は変換されませんでした。

com.ms.lang.SystemX.JavaStringToLocalString は変換されませんでした。

com.ms.lang.SystemX.LocalStringToJavaString は変換されませんでした。

com.ms.lang.SystemX.setInputManager は変換されませんでした。

com.ms.lang.SystemX.setKeyboardLanguage は変換されませんでした。

com.ms.lang.VerifyErrorEx は変換されませんでした。

Com.ms.mtx のエラーメッセージ

com.ms.mtx.AppServer は変換されませんでした。

com.ms.mtx.Context.createObject は変換されませんでした。

com.ms.mtx.Context.disableCommit は変換されませんでした。

com.ms.mtx.Context.enableCommit は変換されませんでした。

com.ms.mtx.Context.getContextId は変換されませんでした。

com.ms.mtx.Context.getDeactivateOnReturn は変換されませんでした。

com.ms.mtx.Context.getDirectCallerName は変換されませんでした。

com.ms.mtx.Context.getDirectCreatorName は変換されませんでした。

com.ms.mtx.Context.getMyTransactionVote は変換されませんでした。

com.ms.mtx.Context.getObjectContext は変換されませんでした。

com.ms.mtx.Context.getOriginalCallerName は変換されませんでした。

com.ms.mtx.Context.getOriginalCreatorName は変換されませんでした。

com.ms.mtx.Context.getProperty は変換されませんでした。

com.ms.mtx.Context.getPropertyNames は変換されませんでした。

com.ms.mtx.Context.getSafeRef は変換されませんでした。

com.ms.mtx.Context.getTransaction は変換されませんでした。

com.ms.mtx.Context.getTransactionId は変換されませんでした。

com.ms.mtx.Context.isCallerInRole は変換されませんでした。

com.ms.mtx.Context.isInTransaction は変換されませんでした。

com.ms.mtx.Context.isSecurityEnabled は変換されませんでした。

com.ms.mtx.Context.setAbort は変換されませんでした。

com.ms.mtx.Context.setComplete は変換されませんでした。

com.ms.mtx.Context.setDeactivateOnReturn は変換されませんでした。

com.ms.mtx.Context.setMyTransactionVote は変換されませんでした。

com.ms.mtx.Context.TxAbort は変換されませんでした。

com.ms.mtx.Context.TxCommit は変換されませんでした。

com.ms.mtx.IContextState.GetDeactivateOnReturn は変換されませんでした。

com.ms.mtx.IContextState.GetMyTransactionVote は変換されませんでした。

com.ms.mtx.IContextState.iid は変換されませんでした。

com.ms.mtx.IContextState.SetDeactivateOnReturn は変換されませんでした。

com.ms.mtx.IContextState.SetMyTransactionVote は変換されませんでした。

com.ms.mtx.IEnumNames は変換されませんでした。

com.ms.mtx.IGetContextProperties は変換されませんでした。

com.ms.mtx.IMTxAS は変換されませんでした。

com.ms.mtx.IMTxAS.GetObjectContext は変換されませんでした。

com.ms.mtx.IMTxAS.iid は変換されませんでした。

com.ms.mtx.IObjectContext.CreateInstance は変換されませんでした。

com.ms.mtx.IObjectContext.iid は変換されませんでした。

com.ms.mtx.IObjectContextInfo.iid は変換されませんでした。

com.ms.mtx.IObjectControl.iid は変換されませんでした。

com.ms.mtx.IObjectControl は変換されませんでした。

com.ms.mtx.ISecurityCallContext.getItem は変換されませんでした。

com.ms.mtx.ISecurityCallContext.iid は変換されませんでした。

com.ms.mtx.ISecurityCallersColl.iid は変換されませんでした。

com.ms.mtx.ISecurityIdentityColl.getItem は変換されませんでした。

com.ms.mtx.ISecurityIdentityColl.iid は変換されませんでした。

com.ms.mtx.SecurityProperty.GetDirectCallerName は変換されませんでした。

com.ms.mtx.SecurityProperty.GetOriginalCallerName は変換されませんでした。

com.ms.mtx.ISharedProperty.iid は変換されませんでした。

com.ms.mtx.ISharedPropertyGroup.iid は変換されませんでした。

com.ms.mtx.ISharedPropertyGroupManager.iid は変換されませんでした。

com.ms.mtx.ITransactionContextEx は変換されませんでした。

com.ms.mtx.MTx は変換されませんでした。

com.ms.mtx.ObjectContext.CreateInstance は変換されませんでした。

com.ms.mtx.ObjectContext.DisableCommit は変換されませんでした。

com.ms.mtx.ObjectContext.EnableCommit は変換されませんでした。

com.ms.mtx.ObjectContext.get_NewEnum は変換されませんでした。

com.ms.mtx.ObjectContext.getCount は変換されませんでした。

com.ms.mtx.ObjectContext.getItem は変換されませんでした。

com.ms.mtx.ObjectContext.getSecurity は変換されませんでした。

com.ms.mtx.ObjectContext.iid は変換されませんでした。

com.ms.mtx.ObjectContext.IsCallerInRole は変換されませんでした。

com.ms.mtx.ObjectContext.IsInTransaction は変換されませんでした。

com.ms.mtx.ObjectContext.IsSecurityEnabled は変換されませんでした。

com.ms.mtx.ObjectContext.SetAbort は変換されませんでした。

com.ms.mtx.ObjectContext.SetComplete は変換されませんでした。

com.ms.mtx.SecurityCallContext.getCallers は変換されませんでした。

com.ms.mtx.SecurityCallContext.getDirectCaller は変換されませんでした。

com.ms.mtx.SecurityCallContext.getNumCallers は変換されませんでした。

com.ms.mtx.SecurityCallContext.getProperty は変換されませんでした。

com.ms.mtx.SecurityCallContext.getPropertyNames は変換されませんでした。

com.ms.mtx.SecurityCallContext.isUserInRole は変換されませんでした。

com.ms.mtx.SecurityCaller は変換されませんでした。

com.ms.mtx.SecurityProperty.GetDirectCreatorName は変換されませんでした。

com.ms.mtx.SecurityProperty.GetOriginalCreatorName は変換されませんでした。

com.ms.mtx.SecurityProperty.iid は変換されませんでした。

com.ms.mtx.SharedPropertyGroupManager.clsid は変換されませんでした。

com.ms.mtx.SharedPropertyGroupManager.get_NewEnum は変換されませんでした。

com.ms.mtx.TransactionContextEx は変換されませんでした。

Com.ms.object のエラー メッセージ

com.ms.object.Category は変換されませんでした。

com.ms.object.dragdrop.DragHandler は変換されませんでした。

com.ms.object.dragdrop.DragHelper は変換されませんでした。

com.ms.object.dragdrop.DragProxy は変換されませんでした。

com.ms.object.dragdrop.DragSession.getDragModifiers は変換されませんでした。

com.ms.object.dragdrop.DragSource.DEFAULT_ACTION は変換されませんでした。

com.ms.object.dragdrop.DragSource.queryDragCursor は変換されませんでした。

com.ms.object.dragdrop.DragSource.queryDragStatus は変換されませんでした。

com.ms.object.IServiceObjectProvider は変換されませんでした。

com.ms.object.ISite は変換されませんでした。

com.ms.object.ISiteable は変換されませんでした。

com.ms.object.MetaObject は変換されませんでした。

com.ms.object.ObjectBag は変換されませんでした。

com.ms.object.SimpleTransferSession は変換されませんでした。

com.ms.object.TransferSession は変換されませんでした。

Com.ms.ui のエラーメッセージ

com.ms.ui.<ClassName>.add*Listener は変換されませんでした。

com.ms.ui.<ClassName>.addNotify は変換されませんでした。

com.ms.ui.<ClassName>.getPeer は変換されませんでした。

com.ms.ui.<ClassName>.handleEvent は変換されませんでした。

com.ms.ui.<ClassName>.isNotified は変換されませんでした。

com.ms.ui.<ClassName>.process*Event は変換されませんでした。

com.ms.ui.<ClassName>.remove*Listener は変換されませんでした。

com.ms.ui.<ClassName>.removeNotify は変換されませんでした。

com.ms.ui.<ClassName>.setListenerTracker を変換できませんでした。

com.ms.ui.<ClassName>BeanInfo は変換されませんでした。

com.ms.ui.AwtUIApplet.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIApplet.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIApplet.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIApplet.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIApplet.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.AwtUIApplet.AwtUIApplet は変換されませんでした。

com.ms.ui.AwtUIApplet.AwtUIApplet は変換されませんでした。

com.ms.ui.AwtUIApplet.destroy は変換されませんでした。

com.ms.ui.AwtUIApplet.getComponent は変換されませんでした。

com.ms.ui.AwtUIApplet.getHeader は変換されませんでした。

com.ms.ui.AwtUIApplet.getRoot は変換されませんでした。

com.ms.ui.AwtUIApplet.getTaskManager は変換されませんでした。

com.ms.ui.AwtUIApplet.getUIComponentCount は変換されませんでした。

com.ms.ui.AwtUIApplet.setHeader は変換されませんでした。

com.ms.ui.AwtUIApplet.setLayout は変換されませんでした。

com.ms.ui.AwtUIBand は変換されませんでした。

com.ms.ui.AwtUIBandBeanInfo.AwtUIBandBeanInfo は変換されませんでした。

com.ms.ui.AwtUIBandBeanInfo.getIcon は変換されませんでした。

com.ms.ui.AwtUIBandBox は変換されませんでした。

com.ms.ui.AwtUIBandBoxBeanInfo.AwtUIBandBoxBeanInfo は変換されませんでした。

com.ms.ui.AwtUIBandBoxBeanInfo.getIcon は変換されませんでした。

com.ms.ui.AwtUIButton._btn は変換されませんでした。

com.ms.ui.AwtUIButton.getStyle は変換されませんでした。

com.ms.ui.AwtUIButton.keyDown は変換されませんでした。

com.ms.ui.AwtUIButton.keyUp は変換されませんでした。

com.ms.ui.AwtUIButton.mouseUp を変換できませんでした。

com.ms.ui.AwtUIButton.setStyle は変換されませんでした。

com.ms.ui.AwtUICheckButton.getBase は変換されませんでした。

com.ms.ui.AwtUICheckButton.getSelectedObjects は変換されませんでした。

com.ms.ui.AwtUIChoice.add は変換されませんでした。

com.ms.ui.AwtUIChoice.addSelectedIndex は変換されませんでした。

com.ms.ui.AwtUIChoice.addSelectedIndices は変換されませんでした。

com.ms.ui.AwtUIChoice.addSelectedItem を変換できませんでした。

com.ms.ui.AwtUIChoice.addSelectedItems は変換されませんでした。

com.ms.ui.AwtUIChoice.getAnchorItem は変換されませんでした。

com.ms.ui.AwtUIChoice.getBase は変換されませんでした。

com.ms.ui.AwtUIChoice.getExtensionItem を変換できませんでした。

com.ms.ui.AwtUIChoice.getSelectedItem は変換されませんでした。

com.ms.ui.AwtUIChoice.getSelectedItems は変換されませんでした。

com.ms.ui.AwtUIChoice.getSelectedObjects は変換されませんでした。

com.ms.ui.AwtUIChoice.getSelectionMode は変換されませんでした。

com.ms.ui.AwtUIChoice.getStyle は変換されませんでした。

com.ms.ui.AwtUIChoice.removeSelectedIndex は変換されませんでした。

com.ms.ui.AwtUIChoice.removeSelectedIndices は変換されませんでした。

com.ms.ui.AwtUIChoice.removeSelectedItem は変換されませんでした。

com.ms.ui.AwtUIChoice.removeSelectedItems は変換されませんでした。

com.ms.ui.AwtUIChoice.setAnchorItem は変換されませんでした。

com.ms.ui.AwtUIChoice.setExtensionItem を変換できませんでした。

com.ms.ui.AwtUIChoice.setSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIChoice.setSelectedIndex を変換できませんでした。(Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIChoice.setSelectedIndices は変換されませんでした。

com.ms.ui.AwtUIChoice.setSelectedItem は変換されませんでした。

com.ms.ui.AwtUIChoice.setSelectedItems は変換されませんでした。

com.ms.ui.AwtUIChoice.setSelectionMode は変換されませんでした。

com.ms.ui.AwtUIChoice.setStyle は変換されませんでした。

com.ms.ui.AwtUIColumnViewer は変換されませんでした。

com.ms.ui.AwtUIControl.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIControl.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIControl.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.AwtUIControl.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.AwtUIControl.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIControl.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIControl.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIControl.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIControl.AwtUIControl は変換されませんでした。

com.ms.ui.AwtUIControl.getID を変換できませんでした。

com.ms.ui.AwtUIControl.isSelected は変換されませんでした。

com.ms.ui.<ClassName>.postEvent は変換されませんでした。

com.ms.ui.AwtUIControl.setChecked を変換できませんでした。

com.ms.ui.AwtUIControl.setHot は変換されませんでした。

com.ms.ui.AwtUIControl.setID は変換されませんでした。

com.ms.ui.AwtUIControl.setIndeterminate は変換されませんでした。

com.ms.ui.AwtUIControl.setLayout を変換できませんでした。

com.ms.ui.AwtUIControl.setPressed は変換されませんでした。

com.ms.ui.AwtUIControl.setReparent は変換されませんでした。

com.ms.ui.AwtUIControl.setSelected は変換されませんでした。

com.ms.ui.AwtUIDialog.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.AwtUIDialog.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.AwtUIDialog.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIDialog.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIDialog.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIDialog.getComponent は変換されませんでした。

com.ms.ui.AwtUIDialog.position は変換されませんでした。

com.ms.ui.AwtUIDialog.setLayout は変換されませんでした。

com.ms.ui.AwtUIDrawText.getCharFromScreen は変換されませんでした。

com.ms.ui.AwtUIDrawText.getCharLocation は変換されませんでした。

com.ms.ui.AwtUIDrawText.getOutline は変換されませんでした。

com.ms.ui.AwtUIDrawText.isAutoResizable は変換されませんでした。

com.ms.ui.AwtUIDrawText.setAutoResizable は変換されませんでした。

com.ms.ui.AwtUIDrawText.setHorizAlign は変換されませんでした。

com.ms.ui.AwtUIDrawText.setOutline は変換されませんでした。

com.ms.ui.AwtUIDrawText.setRefresh は変換されませんでした。

com.ms.ui.AwtUIDrawText.setVertAlign は変換されませんでした。

com.ms.ui.AwtUIDrawText.setWordWrap は変換されませんでした。

com.ms.ui.AwtUIEdit.getCharFromScreen を変換できませんでした。

com.ms.ui.AwtUIEdit.getCharLocation は変換されませんでした。

com.ms.ui.AwtUIEdit.getOutline は変換されませんでした。

com.ms.ui.AwtUIEdit.getWordWrap は変換されませんでした。

com.ms.ui.AwtUIEdit.isAutoResizable を変換できませんでした。

com.ms.ui.AwtUIEdit.setAutoResizable は変換されませんでした。

com.ms.ui.AwtUIEdit.setHorizAlign は変換されませんでした。

com.ms.ui.AwtUIEdit.setOutline は変換されませんでした。

com.ms.ui.AwtUIEdit.setRefresh を変換できませんでした。

com.ms.ui.AwtUIEdit.setVertAlign は変換されませんでした。

com.ms.ui.AwtUIEdit.setWordWrap は変換されませんでした。

com.ms.ui.AwtUIEdit.showCaret は変換されませんでした。

com.ms.ui.AwtUIFrame.add は変換されませんでした (Java Language Conversion Assistant) (4)

com.ms.ui.AwtUIFrame.add は変換されませんでした (Java Language Conversion Assistant) (4)

com.ms.ui.AwtUIFrame.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIFrame.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIFrame.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.AwtUIFrame.requestFocus は変換されませんでした。

com.ms.ui.AwtUIFrame.setLayout は変換されませんでした。

com.ms.ui.AwtUIGraphic.AwtUIGraphic は変換されませんでした。

com.ms.ui.AwtUIGraphic.getContentBounds は変換されませんでした。

com.ms.ui.AwtUIGraphic.imageUpdate は変換されませんでした。

com.ms.ui.AwtUIHost は変換されませんでした。

com.ms.ui.AwtUIHost.add は変換されませんでした (Java Language Conversion Assistant) (4)

com.ms.ui.AwtUIHost.add は変換されませんでした (Java Language Conversion Assistant) (4)

com.ms.ui.AwtUIHost.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIHost.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIHost.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.AwtUIHost.disableHostEvents は変換されませんでした。

com.ms.ui.AwtUIHost.enableHostEvents は変換されませんでした。

com.ms.ui.AwtUIHost.getComponent は変換されませんでした。

com.ms.ui.AwtUIHost.getHeader は変換されませんでした。

com.ms.ui.AwtUIHost.getPreferredSize は変換されませんでした。

com.ms.ui.AwtUIHost.getRoot は変換されませんでした。

com.ms.ui.AwtUIHost.getUIComponent は変換されませんでした。

com.ms.ui.AwtUIHost.invalidate は変換されませんでした。

com.ms.ui.AwtUIHost.layout は変換されませんでした。

com.ms.ui.AwtUIHost.listenerTracker は変換されませんでした。

com.ms.ui.AwtUIHost.obtainListenerTracker は変換されませんでした。

com.ms.ui.AwtUIHost.paint は変換されませんでした。

com.ms.ui.AwtUIHost.paintAll は変換されませんでした。

com.ms.ui.AwtUIHost.preferredSize は変換されませんでした。

com.ms.ui.AwtUIHost.preProcessHostEvent は変換されませんでした。

com.ms.ui.AwtUIHost.root は変換されませんでした。

com.ms.ui.AwtUIHost.setHeader は変換されませんでした。

com.ms.ui.AwtUIHost.setLayout は変換されませんでした。

com.ms.ui.AwtUIHost.setListenerHost は変換されませんでした。

com.ms.ui.AwtUIHost.show は変換されませんでした。

com.ms.ui.AwtUIHost.usingNewEvents を変換できませんでした。

com.ms.ui.AwtUIHost.validate は変換されませんでした。

com.ms.ui.AwtUIHost.validateTree は変換されませんでした。

com.ms.ui.AwtUIList.addSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIList.addSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIList.addSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIList.addSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIList.addSelectedItem は変換されませんでした。

com.ms.ui.AwtUIList.addSelectedItems は変換されませんでした。

com.ms.ui.AwtUIList.AwtUIList は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIList.AwtUIList を変換できませんでした。(Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIList.find は変換されませんでした。

com.ms.ui.AwtUIList.getSelectedIndex は変換されませんでした。

com.ms.ui.AwtUIList.getSelectedItem は変換されませんでした。

com.ms.ui.AwtUIList.remove は変換されませんでした。

com.ms.ui.AwtUIList.removeSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIList.removeSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIList.removeSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIList.removeSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIList.removeSelectedItem は変換されませんでした。

com.ms.ui.AwtUIList.removeSelectedItems は変換されませんでした。

com.ms.ui.AwtUIList.setSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIList.setSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIList.setSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIList.setSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIList.setSelectedItem は変換されませんでした。

com.ms.ui.AwtUIList.setSelectedItems は変換されませんでした。

com.ms.ui.AwtUIList.setSelectionMode は変換されませんでした。

com.ms.ui.AwtUIMarquee は変換されませんでした。

com.ms.ui.AwtUIMenuList.AwtUIMenuList は変換されませんでした。

com.ms.ui.AwtUIMenuList.getSelectedObjects は変換されませんでした。

com.ms.ui.AwtUIMessageBox は変換されませんでした。

com.ms.ui.AwtUIMessageBox.action は変換されませんでした。

com.ms.ui.AwtUIMessageBox.AwtUIMessageBox は変換されませんでした。

com.ms.ui.AwtUIMessageBox.doModal は変換されませんでした。

com.ms.ui.AwtUIMessageBox.doModalIndex は変換されませんでした。

com.ms.ui.AwtUIMessageBox.getButtonAlignment は変換されませんでした。

com.ms.ui.AwtUIMessageBox.getButtons は変換されませんでした。

com.ms.ui.AwtUIMessageBox.getDefaultButton は変換されませんでした。

com.ms.ui.AwtUIMessageBox.getFrame は変換されませんでした。

com.ms.ui.AwtUIMessageBox.getImage は変換されませんでした。

com.ms.ui.AwtUIMessageBox.getText は変換されませんでした。

com.ms.ui.AwtUIMessageBox.getTimeout は変換されませんでした。

com.ms.ui.AwtUIMessageBox.insets は変換されませんでした。

com.ms.ui.AwtUIMessageBox.keyDown は変換されませんでした。

com.ms.ui.AwtUIMessageBox.keyUp は変換されませんでした。

com.ms.ui.AwtUIMessageBox.preferredSize は変換されませんでした。

com.ms.ui.AwtUIMessageBox.setButtonAlignment は変換されませんでした。

com.ms.ui.AwtUIMessageBox.setButtons は変換されませんでした。

com.ms.ui.AwtUIMessageBox.setDefaultButton は変換されませんでした。

com.ms.ui.AwtUIMessageBox.setImage は変換されませんでした。

com.ms.ui.AwtUIMessageBox.setText は変換されませんでした。

com.ms.ui.AwtUIMessageBox.setTimeout は変換されませんでした。

com.ms.ui.AwtUIMessageBox.timeTriggered を変換できませんでした。

com.ms.ui.AwtUIPanel.setLayout は変換されませんでした。

com.ms.ui.AwtUIProgress.AwtUIProgress は変換されませんでした。

com.ms.ui.AwtUIProgress.getBase は変換されませんでした。

com.ms.ui.AwtUIPushButton.getBase は変換されませんでした。

com.ms.ui.AwtUIRadioButton.AwtUIRadioButton は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIRadioButton.AwtUIRadioButton は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIRadioButton.getBase は変換されませんでした。

com.ms.ui.AwtUIRadioButton.getSelectedObjects は変換されませんでした。

com.ms.ui.AwtUIRepeatButton.AwtUIRepeatButton は変換されませんでした。

com.ms.ui.AwtUIScrollBar は変換されませんでした。

com.ms.ui.AwtUIScrollBar.AwtUIScrollBar は変換されませんでした。

com.ms.ui.AwtUIScrollBar.getBase は変換されませんでした。

com.ms.ui.AwtUIScrollBar.getStyle は変換されませんでした。

com.ms.ui.AwtUIScrollBar.scrollLineDown は変換されませんでした。

com.ms.ui.AwtUIScrollBar.scrollLineUp は変換されませんでした。

com.ms.ui.AwtUIScrollBar.scrollPageDown は変換されませんでした。

com.ms.ui.AwtUIScrollBar.scrollPageUp は変換されませんでした。

com.ms.ui.AwtUIScrollBar.setScrollInfo は変換されませんでした。

com.ms.ui.AwtUIScrollBar.setScrollLine は変換されませんでした。

com.ms.ui.AwtUIScrollBar.setStyle は変換されませんでした。

com.ms.ui.AwtUIScrollBar.setUnitIncrement は変換されませんでした。

com.ms.ui.AwtUIScrollViewer は変換されませんでした。

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIScrollViewer.AwtUIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIScrollViewer.getContent は変換されませんでした。

com.ms.ui.AwtUIScrollViewer.getHLine は変換されませんでした。

com.ms.ui.AwtUIScrollViewer.getLine は変換されませんでした。

com.ms.ui.AwtUIScrollViewer.getVLine は変換されませんでした。

com.ms.ui.AwtUIScrollViewer.setContent は変換されませんでした。

com.ms.ui.AwtUIScrollViewer.setHLine を変換できませんでした。

com.ms.ui.AwtUIScrollViewer.setLine を変換できませんでした。

com.ms.ui.AwtUIScrollViewer.setVLine は変換されませんでした。

com.ms.ui.AwtUISplitViewer は変換されませんでした。

com.ms.ui.AwtUISplitViewer.add は変換されませんでした。

com.ms.ui.AwtUISplitViewer.AwtUISplitViewer は変換されませんでした。

com.ms.ui.AwtUISplitViewer.getComponent は変換されませんでした。

com.ms.ui.AwtUISplitViewer.getPos を変換できませんでした。

com.ms.ui.AwtUISplitViewer.getStyle を変換できませんでした。

com.ms.ui.AwtUISplitViewer.remove は変換されませんでした。

com.ms.ui.AwtUISplitViewer.setPos は変換されませんでした。

com.ms.ui.AwtUIStatus.getBase は変換されませんでした。

com.ms.ui.AwtUITabList は変換されませんでした。

com.ms.ui.AwtUITabList.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITabList.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITabList.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITabViewer.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITabViewer.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITabViewer.addTab は変換されませんでした。

com.ms.ui.AwtUITree は変換されませんでした。

com.ms.ui.AwtUITree.add は変換されませんでした (Java Language Conversion Assistant) (4)

com.ms.ui.AwtUITree.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.add は変換されませんでした (Java Language Conversion Assistant) (4)

com.ms.ui.AwtUITree.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.AwtUITree.addSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.addSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.addSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.addSelectedIndices を変換できませんでした。(Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.addSelectedItem は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.addSelectedItem は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.addSelectedItems は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.addSelectedItems は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.getExpander は変換されませんでした。

com.ms.ui.AwtUITree.getSelectedIndex は変換されませんでした。

com.ms.ui.AwtUITree.getSelectedIndices は変換されませんでした。

com.ms.ui.AwtUITree.getSelectedItems を変換できませんでした。

com.ms.ui.AwtUITree.getSelectedObjects は変換されませんでした。

com.ms.ui.AwtUITree.getSelectionMode は変換されませんでした。

com.ms.ui.AwtUITree.remove は変換されませんでした。

com.ms.ui.AwtUITree.removeSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.removeSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.removeSelectedIndices を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.removeSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.removeSelectedItem は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.removeSelectedItem は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.removeSelectedItems は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.removeSelectedItems は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.setSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.setSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.setSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.setSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.setSelectedItem は変換されませんでした。

com.ms.ui.AwtUITree.setSelectedItems は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUITree.setSelectedItems は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUITree.setSelectionMode は変換されませんでした。

com.ms.ui.AwtUIWindow.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.AwtUIWindow.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIWindow.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIWindow.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.AwtUIWindow.AwtUIWindow は変換されませんでした。

com.ms.ui.AwtUIWindow.requestFocus は変換されませんでした。

com.ms.ui.AwtUIWindow.setLayout は変換されませんでした。

com.ms.ui.ButtonFlowLayout は変換されませんでした。

com.ms.ui.ButtonFlowLayout.ButtonFlowLayout を変換できませんでした。

com.ms.ui.ButtonFlowLayout.computeUnitDimension は変換されませんでした。

com.ms.ui.ButtonFlowLayout.getHeightPad は変換されませんでした。

com.ms.ui.ButtonFlowLayout.getMinHeight は変換されませんでした。

com.ms.ui.ButtonFlowLayout.getMinWidth は変換されませんでした。

com.ms.ui.ButtonFlowLayout.getWidthPad は変換されませんでした。

com.ms.ui.ButtonFlowLayout.setHeightPad は変換されませんでした。

com.ms.ui.ButtonFlowLayout.setMinHeight は変換されませんでした。

com.ms.ui.ButtonFlowLayout.setMinWidth は変換されませんでした。

com.ms.ui.ButtonFlowLayout.setWidthPad は変換されませんでした。

com.ms.ui.ButtonPanel.ButtonPanel は変換されませんでした。

com.ms.ui.ButtonPanel.getDefaultButton は変換されませんでした。

com.ms.ui.event.UIActionEvent.<EventType> は変換されませんでした。

com.ms.ui.event.UIActionEvent.getActionCommand は変換されませんでした。

com.ms.ui.event.UIAdjustmentEvent.<EventType> は変換されませんでした。

com.ms.ui.event.UIBaseEvent.getID は変換されませんでした。

com.ms.ui.event.UIContainerEvent.<EventType> は変換されませんでした。

com.ms.ui.event.UIEvent は変換されませんでした。

com.ms.ui.event.UIFocusEvent.<EventType> は変換されませんでした。

com.ms.ui.event.UIFocusEvent.getArg は変換されませんでした。

com.ms.ui.event.UIFocusEvent.isTemporary は変換されませんでした。

com.ms.ui.event.UIInputEvent は変換されませんでした。

com.ms.ui.event.UIItemEvent.<EventType> を変換できませんでした。

com.ms.ui.event.UIItemEvent.getItem は変換されませんでした。

com.ms.ui.event.UIItemEvent.getStateChange は変換されませんでした。

com.ms.ui.event.UIKeyEvent.CHAR_UNDEFINED は変換されませんでした。

com.ms.ui.event.UIKeyEvent.getKeyChar は変換されませんでした。

com.ms.ui.event.UIKeyEvent.getKeyCode は変換されませんでした。

com.ms.ui.event.UIKeyEvent.getOldEventKey は変換されませんでした。

com.ms.ui.event.UIKeyEvent.KEY_EVENT_BASE は変換されませんでした。

com.ms.ui.event.UIKeyEvent.KEY_PRESSED は変換されませんでした。

com.ms.ui.event.UIKeyEvent.KEY_RELEASED は変換されませんでした。

com.ms.ui.event.UIKeyEvent.KEY_TYPED は変換されませんでした。

com.ms.ui.event.UIKeyEvent.VK_BACK_QUOTE は変換されませんでした。

com.ms.ui.event.UIKeyEvent.VK_EQUALS は変換されませんでした。

com.ms.ui.event.UIKeyEvent.VK_META は変換されませんでした。

com.ms.ui.event.UIKeyEvent.VK_UNDEFINED は変換されませんでした。

com.ms.ui.event.UIMouseEvent.<EventType> は変換されませんでした。

com.ms.ui.event.UIMouseEvent.getClickCount は変換されませんでした。

com.ms.ui.event.UIMouseEvent.getPoint は変換されませんでした。

com.ms.ui.event.UIMouseEvent.getX は変換されませんでした。

com.ms.ui.event.UIMouseEvent.getY は変換されませんでした。

com.ms.ui.event.UIMouseEvent.isPopupTrigger は変換されませんでした。

com.ms.ui.event.UINotifyEvent は変換されませんでした。

com.ms.ui.event.UITextEvent.<EventType> は変換されませんでした。

com.ms.ui.event.UIWindowEvent.<EventType> は変換されませんでした。

com.ms.ui.IAwtUIAdjustable は変換されませんでした。

com.ms.ui.IAwtUIItemSelectable は変換されませんでした。

com.ms.ui.IUIAccessible は変換されませんでした。

com.ms.ui.IUIAccessible.<ErrorCode> は変換されませんでした。

com.ms.ui.IUIAccessible.getBounds は変換されませんでした。

com.ms.ui.IUIAccessible.navigate は変換されませんでした。

com.ms.ui.IUIBand は変換されませんでした。

com.ms.ui.IUIComponent.action は変換されませんでした。

com.ms.ui.IUIComponent.adjustLayoutSize は変換されませんでした。

com.ms.ui.IUIComponent.deliverEvent は変換されませんでした。

com.ms.ui.IUIComponent.ensureVisible を変換できませんでした。

com.ms.ui.IUIComponent.getBounds は変換されませんでした。

com.ms.ui.IUIComponent.getCachedPreferredSize は変換されませんでした。

com.ms.ui.IUIComponent.getID は変換されませんでした。

com.ms.ui.IUIComponent.getLocation は変換されませんでした。

com.ms.ui.IUIComponent.getMaximumSize は変換されませんでした。

com.ms.ui.IUIComponent.getMinimumSize は変換されませんでした。

com.ms.ui.IUIComponent.getPreferredSize は変換されませんでした。

com.ms.ui.IUIComponent.getToolkit は変換されませんでした。

com.ms.ui.IUIComponent.isChecked は変換されませんでした。

com.ms.ui.IUIComponent.isHeightRelative は変換されませんでした。

com.ms.ui.IUIComponent.isHot は変換されませんでした。

com.ms.ui.IUIComponent.isIndeterminate は変換されませんでした。

com.ms.ui.IUIComponent.isInvalidating は変換されませんでした。

com.ms.ui.IUIComponent.isKeyable は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.IUIComponent.isKeyable を変換できませんでした。(Java Language Conversion Assistant) (2)

com.ms.ui.IUIComponent.isPressed は変換されませんでした。

com.ms.ui.IUIComponent.isRedrawing は変換されませんでした。

com.ms.ui.IUIComponent.isSelected は変換されませんでした。

com.ms.ui.IUIComponent.isValid は変換されませんでした。

com.ms.ui.IUIComponent.isWidthRelative は変換されませんでした。

com.ms.ui.IUIComponent.keyDown は変換されませんでした。

com.ms.ui.IUIComponent.keyUp は変換されませんでした。

com.ms.ui.IUIComponent.lostFocus は変換されませんでした。

com.ms.ui.IUIComponent.mouseClicked は変換されませんでした。

com.ms.ui.IUIComponent.mouseDown は変換されませんでした。

com.ms.ui.IUIComponent.mouseEnter を変換できませんでした。

com.ms.ui.IUIComponent.mouseExit は変換されませんでした。

com.ms.ui.IUIComponent.mouseMove は変換されませんでした。

com.ms.ui.IUIComponent.mouseUp は変換されませんでした。

com.ms.ui.IUIComponent.paint は変換されませんでした。

com.ms.ui.IUIComponent.paintAll は変換されませんでした。

com.ms.ui.IUIComponent.prepareImage を変換できませんでした。

com.ms.ui.IUIComponent.print を変換できませんでした。

com.ms.ui.IUIComponent.printAll は変換されませんでした。

com.ms.ui.IUIComponent.recalcPreferredSize は変換されませんでした。

com.ms.ui.IUIComponent.setBounds は変換されませんでした。

com.ms.ui.IUIComponent.setChecked は変換されませんでした。

com.ms.ui.IUIComponent.setFlags は変換されませんでした。

com.ms.ui.IUIComponent.setHot は変換されませんでした。

com.ms.ui.IUIComponent.setID は変換されませんでした。

com.ms.ui.IUIComponent.setIndeterminate は変換されませんでした。

com.ms.ui.IUIComponent.setInvalidating は変換されませんでした。

com.ms.ui.IUIComponent.setLocation は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.IUIComponent.setLocation は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.IUIComponent.setPressed は変換されませんでした。

com.ms.ui.IUIComponent.setRedrawing は変換されませんでした。

com.ms.ui.IUIComponent.setSelected は変換されませんでした。

com.ms.ui.IUIComponent.setValid は変換されませんでした。

com.ms.ui.IUIComponent.setVisible は変換されませんでした。

com.ms.ui.IUIComponent.validate は変換されませんでした。

com.ms.ui.IUIContainer.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.IUIContainer.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.IUIContainer.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.IUIContainer.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.IUIContainer.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.IUIContainer.adjustLayoutSize は変換されませんでした。

com.ms.ui.IUIContainer.continueInvalidate は変換されませんでした。

com.ms.ui.IUIContainer.ensureVisible は変換されませんでした。

com.ms.ui.IUIContainer.getChildBounds は変換されませんでした。

com.ms.ui.IUIContainer.getChildLocation は変換されませんでした。

com.ms.ui.IUIContainer.getChildSize は変換されませんでした。

com.ms.ui.IUIContainer.getComponentFromID は変換されませんでした。

com.ms.ui.IUIContainer.getEdge は変換されませんでした。

com.ms.ui.IUIContainer.getLayout は変換されませんでした。

com.ms.ui.IUIContainer.isOverlapping は変換されませんでした。

com.ms.ui.IUIContainer.navigate は変換されませんでした。

com.ms.ui.IUIContainer.paintComponents は変換されませんでした。

com.ms.ui.IUIContainer.passFocus は変換されませんでした。

com.ms.ui.IUIContainer.replace は変換されませんでした。

com.ms.ui.IUIContainer.setChildBounds は変換されませんでした。

com.ms.ui.IUIContainer.setChildLocation は変換されませんでした。

com.ms.ui.IUIContainer.setChildSize は変換されませんでした。

com.ms.ui.IUIContainer.setEdge は変換されませんでした。

com.ms.ui.IUIContainer.setHeader は変換されませんでした。

com.ms.ui.IUIContainer.setLayout は変換されませんでした。

com.ms.ui.IUILayoutManager は変換されませんでした。

com.ms.ui.IUIMenuLauncher は変換されませんでした。

com.ms.ui.IUIPosition は変換されませんでした。

com.ms.ui.IUIPropertyPage は変換されませんでした。

com.ms.ui.IUIRootContainer.componentMoved は変換されませんでした。

com.ms.ui.IUIRootContainer.endMenu は変換されませんでした。

com.ms.ui.IUIRootContainer.endTooltip は変換されませんでした。

com.ms.ui.IUIRootContainer.getFocus は変換されませんでした。

com.ms.ui.IUIRootContainer.getLaunchedMenu は変換されませんでした。

com.ms.ui.IUIRootContainer.launchMenu は変換されませんでした。

com.ms.ui.IUIRootContainer.launchTooltip は変換されませんでした。

com.ms.ui.IUIRootContainer.needsValidating は変換されませんでした。

com.ms.ui.IUIRootContainer.setFocus は変換されませんでした。

com.ms.ui.IUIScroll は変換されませんでした。

com.ms.ui.IUISelector は変換されませんでした。

com.ms.ui.IUISpinnerBuddy を変換できませんでした。

com.ms.ui.IUITree を変換できませんでした。

com.ms.ui.IUIWizardStep は変換されませんでした。

com.ms.ui.IWinEvent は変換されませんでした。

com.ms.ui.resource.DataBoundInputStream は変換されませんでした。

com.ms.ui.resource.ResourceDecoder は変換されませんでした。

com.ms.ui.resource.ResourceFormattingException は変換されませんでした。

com.ms.ui.resource.ResourceTypeListener は変換されませんでした。

com.ms.ui.resource.UIDialogLayout は変換されませんでした。

com.ms.ui.resource.Win32ResourceDecoder は変換されませんでした。

com.ms.ui.UIApplet.destroy は変換されませんでした。

com.ms.ui.UIApplet.getAppletContext は変換されませんでした。

com.ms.ui.UIApplet.getAudioClip は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIApplet.getAudioClip は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIApplet.getDocumentBase は変換されませんでした。

com.ms.ui.UIApplet.isActive は変換されませんでした。

com.ms.ui.UIApplet.play は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIApplet.play は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIApplet.setStub は変換されませんでした。

com.ms.ui.UIApplet.showStatus は変換されませんでした。

com.ms.ui.UIAwtHost.forwardEvents は変換されませんでした。

com.ms.ui.UIAwtHost.getCachedPreferredSize は変換されませんでした。

com.ms.ui.UIAwtHost.getMinimumSize は変換されませんでした。

com.ms.ui.UIAwtHost.getPreferredSize は変換されませんでした。

com.ms.ui.UIAwtHost.notifyEvent は変換されませんでした。

com.ms.ui.UIAwtHost.setFocused は変換されませんでした。

com.ms.ui.UIAwtHost.setValid は変換されませんでした。

com.ms.ui.UIBand は変換されませんでした。

com.ms.ui.UIBandBox は変換されませんでした。

com.ms.ui.UIBandThumb は変換されませんでした。

com.ms.ui.UIBarLayout は変換されませんでした。

com.ms.ui.UIBorderLayout は変換されませんでした。

com.ms.ui.UIButton.doDefaultAction は変換されませんでした。

com.ms.ui.UIButton.getStyle は変換されませんでした。

com.ms.ui.UIButton.keyDown は変換されませんでした。

com.ms.ui.UIButton.keyUp は変換されませんでした。

com.ms.ui.UIButton.mouseClicked は変換されませんでした。

com.ms.ui.UIButton.setHot は変換されませんでした。

com.ms.ui.UIButton.setStyle は変換されませんでした。

com.ms.ui.UIButtonBar は変換されませんでした。

com.ms.ui.UIButtonBar.<ButtonBarType> は変換されませんでした。

com.ms.ui.UIButtonBar.add は変換されませんでした。

com.ms.ui.UIButtonBar.addTo は変換されませんでした。

com.ms.ui.UIButtonBar.UIButtonBar は変換されませんでした。

com.ms.ui.UICanvas.getID は変換されませんでした。

com.ms.ui.UICanvas.setID は変換されませんでした。

com.ms.ui.UICardLayout は変換されませんでした。

com.ms.ui.UICheckButton.getCheckImageSize は変換されませんでした。

com.ms.ui.UICheckButton.getConvertedEvent は変換されませんでした。

com.ms.ui.UICheckButton.getMinimumSize を変換できませんでした。

com.ms.ui.UICheckButton.getPreferredSize は変換されませんでした。

com.ms.ui.UICheckButton.paint は変換されませんでした。

com.ms.ui.UICheckButton.setCheckImageSize は変換されませんでした。

com.ms.ui.UICheckButton.setHot は変換されませんでした。

com.ms.ui.UICheckButton.setID は変換されませんでした。

com.ms.ui.UICheckButton.setPressed は変換されませんでした。

com.ms.ui.UICheckButton.setSelected は変換されませんでした。

com.ms.ui.UICheckGroup.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UICheckGroup.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UICheckGroup.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UICheckGroup.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UICheckGroup.setHeader は変換されませんでした。

com.ms.ui.UICheckGroup.UICheckGroup は変換されませんでした。

com.ms.ui.UIChoice.add は変換されませんでした。

com.ms.ui.UIChoice.addSelectedIndex を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.UIChoice.addSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIChoice.addSelectedIndices は変換されませんでした。

com.ms.ui.UIChoice.addSelectedItem を変換できませんでした。

com.ms.ui.UIChoice.addSelectedItems は変換されませんでした。

com.ms.ui.UIChoice.adjustPopupListSize は変換されませんでした。

com.ms.ui.UIChoice.getAnchorItem は変換されませんでした。

com.ms.ui.UIChoice.getConvertedEvent は変換されませんでした。

com.ms.ui.UIChoice.getExtensionItem は変換されませんでした。

com.ms.ui.UIChoice.getMenu は変換されませんでした。

com.ms.ui.UIChoice.getPopupListMaxStringCount を変換できませんでした。

com.ms.ui.UIChoice.getPreferredSize は変換されませんでした。

com.ms.ui.UIChoice.getSelectedIndices は変換されませんでした。

com.ms.ui.UIChoice.getSelectedItem は変換されませんでした。

com.ms.ui.UIChoice.getSelectedItems を変換できませんでした。

com.ms.ui.UIChoice.getSelectionMode は変換されませんでした。

com.ms.ui.UIChoice.getStyle は変換されませんでした。

com.ms.ui.UIChoice.keyDown は変換されませんでした。

com.ms.ui.UIChoice.mouseDown は変換されませんでした。

com.ms.ui.UIChoice.remove は変換されませんでした。

com.ms.ui.UIChoice.removeSelectedIndex は変換されませんでした。

com.ms.ui.UIChoice.removeSelectedIndices は変換されませんでした。

com.ms.ui.UIChoice.removeSelectedItem を変換できませんでした。

com.ms.ui.UIChoice.removeSelectedItems は変換されませんでした。

com.ms.ui.UIChoice.setAnchorItem は変換されませんでした。

com.ms.ui.UIChoice.setExtensionItem は変換されませんでした。

com.ms.ui.UIChoice.setPopupListMaxStringCount は変換されませんでした。

com.ms.ui.UIChoice.setSelectedIndex を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.UIChoice.setSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIChoice.setSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIChoice.setSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIChoice.setSelectedItem を変換できませんでした。

com.ms.ui.UIChoice.setSelectionMode は変換されませんでした。

com.ms.ui.UIChoice.setStyle は変換されませんでした。

com.ms.ui.UIChoice.setUsingWindowsLook は変換されませんでした。

com.ms.ui.UIChoice.THICK は変換されませんでした。

com.ms.ui.UIColorDialog.UIColorDialog は変換されませんでした。

com.ms.ui.UIColumnHeader.isMoving は変換されませんでした。

com.ms.ui.UIColumnHeader.isSizing は変換されませんでした。

com.ms.ui.UIColumnHeader.isSizingLeft は変換されませんでした。

com.ms.ui.UIColumnHeader.isSizingRight は変換されませんでした。

com.ms.ui.UIColumnHeader.mouseDown は変換されませんでした。

com.ms.ui.UIColumnViewer は変換されませんでした。

com.ms.ui.UIComponent.action は変換されませんでした。

com.ms.ui.UIComponent.adjustLayoutSize は変換されませんでした。

com.ms.ui.UIComponent.clone を変換できませんでした。

com.ms.ui.UIComponent.contains は変換されませんでした。

com.ms.ui.UIComponent.deliverEvent は変換されませんでした。

com.ms.ui.UIComponent.doDefaultAction は変換されませんでした。

com.ms.ui.UIComponent.doLayout は変換されませんでした。

com.ms.ui.UIComponent.ensureVisible は変換されませんでした。

com.ms.ui.UIComponent.getBounds は変換されませんでした。

com.ms.ui.UIComponent.getCachedPreferredSize は変換されませんでした。

com.ms.ui.UIComponent.getComponentAt は変換されませんでした。

com.ms.ui.UIComponent.getDefaultAction は変換されませんでした。

com.ms.ui.UIComponent.getDescription は変換されませんでした。

com.ms.ui.UIComponent.getGraphics は変換されませんでした。

com.ms.ui.UIComponent.getHelp は変換されませんでした。

com.ms.ui.UIComponent.getKeyboardShortcut は変換されませんでした。

com.ms.ui.UIComponent.getLocation は変換されませんでした。

com.ms.ui.UIComponent.getLocationOnScreen は変換されませんでした。

com.ms.ui.UIComponent.getMaximumSize は変換されませんでした。

com.ms.ui.UIComponent.getMinimumSize は変換されませんでした。

com.ms.ui.UIComponent.getPreferredSize は変換されませんでした。

com.ms.ui.UIComponent.getRoleCode は変換されませんでした。

com.ms.ui.UIComponent.getRoot は変換されませんでした。

com.ms.ui.UIComponent.getSize は変換されませんでした。

com.ms.ui.UIComponent.getStateCode は変換されませんでした。

com.ms.ui.UIComponent.getToolkit は変換されませんでした。

com.ms.ui.UIComponent.getTreeLock は変換されませんでした。

com.ms.ui.UIComponent.getValueText は変換されませんでした。

com.ms.ui.UIComponent.imageUpdate は変換されませんでした。

com.ms.ui.UIComponent.invalidate は変換されませんでした。

com.ms.ui.UIComponent.isChecked は変換されませんでした。

com.ms.ui.UIComponent.isEnabled は変換されませんでした。

com.ms.ui.UIComponent.isHeightRelative は変換されませんでした。

com.ms.ui.UIComponent.isHot は変換されませんでした。

com.ms.ui.UIComponent.isIndeterminate は変換されませんでした。

com.ms.ui.UIComponent.isInvalidating を変換できませんでした。

com.ms.ui.UIComponent.isKeyable は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIComponent.isKeyable は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIComponent.isPressed は変換されませんでした。

com.ms.ui.UIComponent.isRedrawing は変換されませんでした。

com.ms.ui.UIComponent.isSelectable は変換されませんでした。

com.ms.ui.UIComponent.isShowing は変換されませんでした。

com.ms.ui.UIComponent.isValid を変換できませんでした。

com.ms.ui.UIComponent.isVisible は変換されませんでした。

com.ms.ui.UIComponent.isWidthRelative は変換されませんでした。

com.ms.ui.UIComponent.keyDown は変換されませんでした。

com.ms.ui.UIComponent.keyUp は変換されませんでした。

com.ms.ui.UIComponent.lostFocus は変換されませんでした。

com.ms.ui.UIComponent.mouseClicked は変換されませんでした。

com.ms.ui.UIComponent.mouseDown は変換されませんでした。

com.ms.ui.UIComponent.mouseDrag は変換されませんでした。

com.ms.ui.UIComponent.mouseEnter は変換されませんでした。

com.ms.ui.UIComponent.mouseExit は変換されませんでした。

com.ms.ui.UIComponent.mouseMove は変換されませんでした。

com.ms.ui.UIComponent.mouseUp は変換されませんでした。

com.ms.ui.UIComponent.navigate は変換されませんでした。

com.ms.ui.UIComponent.notifyEvent は変換されませんでした。

com.ms.ui.UIComponent.paint は変換されませんでした。

com.ms.ui.UIComponent.paintAll は変換されませんでした。

com.ms.ui.UIComponent.prepareImage を変換できませんでした。(Java Language Conversion Assistant) (2)

com.ms.ui.UIComponent.prepareImage は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIComponent.print は変換されませんでした。

com.ms.ui.UIComponent.printAll は変換されませんでした。

com.ms.ui.UIComponent.recalcPreferredSize は変換されませんでした。

com.ms.ui.UIComponent.requestFocus を変換できませんでした。

com.ms.ui.UIComponent.repaint は変換されませんでした。

com.ms.ui.UIComponent.requestFocus を変換できませんでした。

com.ms.ui.UIComponent.setBounds は変換されませんでした。

com.ms.ui.UIComponent.setChecked は変換されませんでした。

com.ms.ui.UIComponent.setFlags は変換されませんでした。

com.ms.ui.UIComponent.setHot は変換されませんでした。

com.ms.ui.UIComponent.setID は変換されませんでした。

com.ms.ui.UIComponent.setIndeterminate は変換されませんでした。

com.ms.ui.UIComponent.setInvalidating は変換されませんでした。

com.ms.ui.UIComponent.setLocation は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIComponent.setLocation は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIComponent.setPressed は変換されませんでした。

com.ms.ui.UIComponent.setRedrawing は変換されませんでした。

com.ms.ui.UIComponent.setSelected は変換されませんでした。

com.ms.ui.UIComponent.setSize は変換されませんでした。

com.ms.ui.UIComponent.setUsingWindowsLook は変換されませんでした。

com.ms.ui.UIComponent.setValid は変換されませんでした。

com.ms.ui.UIComponent.setValueText は変換されませんでした。

com.ms.ui.UIComponent.setVisible は変換されませんでした。

com.ms.ui.UIComponent.validate は変換されませんでした。

com.ms.ui.UIContainer.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIContainer.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIContainer.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIContainer.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIContainer.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIContainer.adjustLayoutSize は変換されませんでした。

com.ms.ui.UIContainer.continueInvalidate は変換されませんでした。

com.ms.ui.UIContainer.ensureVisible は変換されませんでした。

com.ms.ui.UIContainer.getChildBounds を変換できませんでした。

com.ms.ui.UIContainer.getChildIndex は変換されませんでした。

com.ms.ui.UIContainer.getChildLocation は変換されませんでした。

com.ms.ui.UIContainer.getChildSize は変換されませんでした。

com.ms.ui.UIContainer.getClientRect は変換されませんでした。

com.ms.ui.UIContainer.getComponent は変換されませんでした。

com.ms.ui.UIContainer.getComponentFromID は変換されませんでした。

com.ms.ui.UIContainer.getComponentIndex は変換されませんでした。

com.ms.ui.UIContainer.getComponents は変換されませんでした。

com.ms.ui.UIContainer.getEdge は変換されませんでした。

com.ms.ui.UIContainer.getFocusComponent は変換されませんでした。

com.ms.ui.UIContainer.getID は変換されませんでした。

com.ms.ui.UIContainer.getInsets は変換されませんでした。

com.ms.ui.UIContainer.getLayout は変換されませんでした。

com.ms.ui.UIContainer.getMinimumSize は変換されませんでした。

com.ms.ui.UIContainer.getName は変換されませんでした。

com.ms.ui.UIContainer.getPreferredSize は変換されませんでした。

com.ms.ui.UIContainer.gotFocus は変換されませんでした。

com.ms.ui.UIContainer.invalidateAll は変換されませんでした。

com.ms.ui.UIContainer.isHeightRelative は変換されませんでした。

com.ms.ui.UIContainer.isOverlapping は変換されませんでした。

com.ms.ui.UIContainer.isWidthRelative は変換されませんでした。

com.ms.ui.UIContainer.keyDown は変換されませんでした。

com.ms.ui.UIContainer.lostFocus は変換されませんでした。

com.ms.ui.UIContainer.mouseEnter は変換されませんでした。

com.ms.ui.UIContainer.mouseExit は変換されませんでした。

com.ms.ui.UIContainer.move は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIContainer.move は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIContainer.navigate は変換されませんでした。

com.ms.ui.UIContainer.notifyEvent は変換されませんでした。

com.ms.ui.UIContainer.paint は変換されませんでした。

com.ms.ui.UIContainer.paintAll は変換されませんでした。

com.ms.ui.UIContainer.paintComponents は変換されませんでした。

com.ms.ui.UIContainer.passFocus は変換されませんでした。

com.ms.ui.UIContainer.printAll は変換されませんでした。

com.ms.ui.UIContainer.remove は変換されませんでした。

com.ms.ui.UIContainer.removeAll は変換されませんでした。

com.ms.ui.UIContainer.removeAllChildren は変換されませんでした。

com.ms.ui.UIContainer.replace は変換されませんでした。

com.ms.ui.UIContainer.setChildBounds は変換されませんでした。

com.ms.ui.UIContainer.setChildLocation は変換されませんでした。

com.ms.ui.UIContainer.setChildSize は変換されませんでした。

com.ms.ui.UIContainer.setComponent は変換されませんでした。

com.ms.ui.UIContainer.setEdge は変換されませんでした。

com.ms.ui.UIContainer.setID は変換されませんでした。

com.ms.ui.UIContainer.setLayout は変換されませんでした。

com.ms.ui.UIContainer.setLocation は変換されませんでした。

com.ms.ui.UIContainer.setSize は変換されませんでした。

com.ms.ui.UIContextMenu.action は変換されませんでした。

com.ms.ui.UIContextMenu.ended は変換されませんでした。

com.ms.ui.UIContextMenu.getPlacement は変換されませんでした。

com.ms.ui.UIContextMenu.getRoot は変換されませんでした。

com.ms.ui.UIContextMenu.getUserItem は変換されませんでした。

com.ms.ui.UIContextMenu.UIContextMenu は変換されませんでした。

com.ms.ui.UIDialog.adjustLayoutSize は変換されませんでした。

com.ms.ui.UIDialog.isAutoPack は変換されませんでした。

com.ms.ui.UIDialog.keyDown は変換されませんでした。

com.ms.ui.UIDialog.keyUp は変換されませんでした。

com.ms.ui.UIDialog.position を変換できませんでした。

com.ms.ui.UIDialog.setAutoPack は変換されませんでした。

com.ms.ui.UIDialog.setModal は変換されませんでした。

com.ms.ui.UIDialog.UIDialog は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIDialog.UIDialog は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIDialogMapping は変換されませんでした。

com.ms.ui.UIDragDrop は変換されませんでした。

com.ms.ui.UIDrawText.ensureNotDirty は変換されませんでした。

com.ms.ui.UIDrawText.ensureVisible を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.UIDrawText.ensureVisible は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIDrawText.eoln は変換されませんでした。

com.ms.ui.UIDrawText.getCachedGraphics を変換できませんでした。

com.ms.ui.UIDrawText.getCaretWidth は変換されませんでした。

com.ms.ui.UIDrawText.getCharFromScreen は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIDrawText.getCharFromScreen は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIDrawText.getCharLocation は変換されませんでした。

com.ms.ui.UIDrawText.getMinimumSize を変換できませんでした。

com.ms.ui.UIDrawText.getOutline は変換されませんでした。

com.ms.ui.UIDrawText.getPreferredSize は変換されませんでした。

com.ms.ui.UIDrawText.getStartPoint は変換されませんでした。

com.ms.ui.UIDrawText.getVertAlign は変換されませんでした。

com.ms.ui.UIDrawText.getWordEdge を変換できませんでした。

com.ms.ui.UIDrawText.gotFocus を変換できませんでした。

com.ms.ui.UIDrawText.hideCaret は変換されませんでした。

com.ms.ui.UIDrawText.isAutoResizable は変換されませんでした。

com.ms.ui.UIDrawText.keyDown は変換されませんでした。

com.ms.ui.UIDrawText.lostFocus を変換できませんでした。

com.ms.ui.UIDrawText.mouseDown は変換されませんでした。

com.ms.ui.UIDrawText.mouseDrag は変換されませんでした。

com.ms.ui.UIDrawText.paint は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIDrawText.paint は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIDrawText.setAutoResizable は変換されませんでした。

com.ms.ui.UIDrawText.setCaretWidth は変換されませんでした。

com.ms.ui.UIDrawText.setCurrIndex は変換されませんでした。

com.ms.ui.UIDrawText.setHorizAlign は変換されませんでした。

com.ms.ui.UIDrawText.setInputMethod は変換されませんでした。

com.ms.ui.UIDrawText.setOutline は変換されませんでした。

com.ms.ui.UIDrawText.setRefresh は変換されませんでした。

com.ms.ui.UIDrawText.setTabs を変換できませんでした。

com.ms.ui.UIDrawText.setTextCallback は変換されませんでした。

com.ms.ui.UIDrawText.setUnderlined は変換されませんでした。

com.ms.ui.UIDrawText.setVertAlign を変換できませんでした。

com.ms.ui.UIDrawText.setWordWrap は変換されませんでした。

com.ms.ui.UIDrawText.showCaret は変換されませんでした。

com.ms.ui.UIEdit.allowUndo は変換されませんでした。

com.ms.ui.UIEdit.append は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIEdit.append は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIEdit.append は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UIEdit.clear は変換されませんでした。

com.ms.ui.UIEdit.EXCLUDE を変換できませんでした。

com.ms.ui.UIEdit.getConvertedEvent は変換されませんでした。

com.ms.ui.UIEdit.getMaskChars は変換されませんでした。

com.ms.ui.UIEdit.getMaskMode は変換されませんでした。

com.ms.ui.UIEdit.INCLUDE は変換されませんでした。

com.ms.ui.UIEdit.insert は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIEdit.insert は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIEdit.insert は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UIEdit.isRedoable は変換されませんでした。

com.ms.ui.UIEdit.isUndoable は変換されませんでした。

com.ms.ui.UIEdit.isUndoAllowed は変換されませんでした。

com.ms.ui.UIEdit.keyDown は変換されませんでした。

com.ms.ui.UIEdit.READONLY は変換されませんでした。

com.ms.ui.UIEdit.redo は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIEdit.redo は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIEdit.remove は変換されませんでした。

com.ms.ui.UIEdit.setMaskChars は変換されませんでした。

com.ms.ui.UIEdit.setMaskMode は変換されませんでした。

com.ms.ui.UIEditChoice.action は変換されませんでした。

com.ms.ui.UIEditChoice.getEditComponent は変換されませんでした。

com.ms.ui.UIEditChoice.keyDown は変換されませんでした。

com.ms.ui.UIEditChoice.lostFocus は変換されませんでした。

com.ms.ui.UIExpandButton は変換されませんでした。

com.ms.ui.UIFindReplaceDialog を変換できませんでした。

com.ms.ui.UIFixedFlowLayout は変換されませんでした。

com.ms.ui.UIFixedFlowLayout.computeUnitDimension は変換されませんでした。

com.ms.ui.UIFixedFlowLayout.getAlignment は変換されませんでした。

com.ms.ui.UIFixedFlowLayout.getMinimumSize は変換されませんでした。

com.ms.ui.UIFixedFlowLayout.getPreferredSize は変換されませんでした。

com.ms.ui.UIFixedFlowLayout.isOverlapping は変換されませんでした。

com.ms.ui.UIFixedFlowLayout.isWidthRelative は変換されませんでした。

com.ms.ui.UIFixedFlowLayout.layout は変換されませんでした。

com.ms.ui.UIFixedFlowLayout.navigate は変換されませんでした。

com.ms.ui.UIFixedFlowLayout.setAlignment を変換できませんでした。

com.ms.ui.UIFixedFlowLayout.UIFixedFlowLayout は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIFixedFlowLayout.UIFixedFlowLayout は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIFixedFlowLayout.UIFixedFlowLayout は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UIFlowLayout は変換されませんでした。

com.ms.ui.UIFlowLayout.getAlignment は変換されませんでした。

com.ms.ui.UIFlowLayout.getMinimumSize は変換されませんでした。

com.ms.ui.UIFlowLayout.getPreferredSize は変換されませんでした。

com.ms.ui.UIFlowLayout.isOverlapping は変換されませんでした。

com.ms.ui.UIFlowLayout.isWidthRelative は変換されませんでした。

com.ms.ui.UIFlowLayout.layout を変換できませんでした。

com.ms.ui.UIFlowLayout.navigate は変換されませんでした。

com.ms.ui.UIFlowLayout.setAlignment は変換されませんでした。

com.ms.ui.UIFlowLayout.setWrap は変換されませんでした。

com.ms.ui.UIFlowLayout.UIFlowLayout は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIFlowLayout.UIFlowLayout は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIFlowLayout.UIFlowLayout は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UIFontDialog.UIFontDialog は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIFontDialog.UIFontDialog は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIFrame.setMenuBar は変換されませんでした。

com.ms.ui.UIGraphic.getPreferredSize は変換されませんでした。

com.ms.ui.UIGraphic.imageUpdate を変換できませんでした。

com.ms.ui.UIGraphic.paint は変換されませんでした。

com.ms.ui.UIGraphic.UIGraphic は変換されませんでした。

com.ms.ui.UIGridBagConstraints は変換されませんでした。

com.ms.ui.UIGridBagLayout は変換されませんでした。

com.ms.ui.UIGridLayout は変換されませんでした。

com.ms.ui.UIGridLayout.continueInvalidate は変換されませんでした。

com.ms.ui.UIGridLayout.getMinimumSize は変換されませんでした。

com.ms.ui.UIGridLayout.getPreferredSize は変換されませんでした。

com.ms.ui.UIGridLayout.isOverlapping は変換されませんでした。

com.ms.ui.UIGridLayout.layout は変換されませんでした。

com.ms.ui.UIGridLayout.navigate は変換されませんでした。

com.ms.ui.UIGridLayout.UIGridLayout を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.UIGridLayout.UIGridLayout は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIGroup.adjustLayoutSize は変換されませんでした。

com.ms.ui.UIGroup.paint は変換されませんでした。

com.ms.ui.UIGroup.UIGroup は変換されませんでした。

com.ms.ui.UIHeaderRow.mouseDown は変換されませんでした。

com.ms.ui.UIHeaderRow.mouseDrag は変換されませんでした。

com.ms.ui.UIHeaderRow.mouseUp は変換されませんでした。

com.ms.ui.UIHeaderRow.navigate は変換されませんでした。

com.ms.ui.UIItem.<Position> は変換されませんでした。

com.ms.ui.UIItem.getGap は変換されませんでした。

com.ms.ui.UIItem.getImagePos は変換されませんでした。

com.ms.ui.UIItem.getPreferredSize は変換されませんでした。

com.ms.ui.UIItem.imageUpdate は変換されませんでした。

com.ms.ui.UIItem.paint は変換されませんでした。

com.ms.ui.UIItem.setFocused は変換されませんでした。

com.ms.ui.UIItem.setGap は変換されませんでした。

com.ms.ui.UIItem.setHot は変換されませんでした。

com.ms.ui.UIItem.setImagePos は変換されませんでした。

com.ms.ui.UIItem.setSelected は変換されませんでした。

com.ms.ui.UIItem.UIItem は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIItem.UIItem を変換できませんでした。(Java Language Conversion Assistant) (2)

com.ms.ui.UILayoutManager は変換されませんでした。

com.ms.ui.UILine.getMinimumSize は変換されませんでした。

com.ms.ui.UILine.getPreferredSize は変換されませんでした。

com.ms.ui.UILine.paint は変換されませんでした。

com.ms.ui.UIList.UIList は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIList.UIList は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIMarquee は変換されませんでした。

com.ms.ui.UIMenuButton.action は変換されませんでした。

com.ms.ui.UIMenuButton.ended は変換されませんでした。

com.ms.ui.UIMenuButton.getPlacement は変換されませんでした。

com.ms.ui.UIMenuButton.keyDown は変換されませんでした。

com.ms.ui.UIMenuButton.launch は変換されませんでした。

com.ms.ui.UIMenuButton.mouseDown は変換されませんでした。

com.ms.ui.UIMenuButton.requestFocus は変換されませんでした。

com.ms.ui.UIMenuButton.UIMenuButton は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIMenuButton.UIMenuButton は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIMenuButton.UIMenuButton は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIMenuItem.UIMenuItem は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIMenuItem.action は変換されませんでした。

com.ms.ui.UIMenuItem.getInsets は変換されませんでした。

com.ms.ui.UIMenuItem.keyDown は変換されませんでした。

com.ms.ui.UIMenuItem.paint は変換されませんでした。

com.ms.ui.UIMenuItem.UIMenuItem は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIMenuItem.UIMenuItem は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIMenuLauncher.cancel は変換されませんでした。

com.ms.ui.UIMenuLauncher.ended は変換されませんでした。

com.ms.ui.UIMenuLauncher.fitToScreen は変換されませんでした。

com.ms.ui.UIMenuLauncher.getDisplayer を変換できませんでした。

com.ms.ui.UIMenuLauncher.getPlacement は変換されませんでした。

com.ms.ui.UIMenuLauncher.isLaunched は変換されませんでした。

com.ms.ui.UIMenuLauncher.launch は変換されませんでした。

com.ms.ui.UIMenuLauncher.raiseEvent は変換されませんでした。

com.ms.ui.UIMenuLauncher.setFocused は変換されませんでした。

com.ms.ui.UIMenuLauncher.setHot は変換されませんでした。

com.ms.ui.UIMenuLauncher.setSelected は変換されませんでした。

com.ms.ui.UIMenuLauncher.UIMenuLauncher は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIMenuLauncher.UIMenuLauncher は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIMenuLauncher.UIMenuLauncher は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UIMenuList は変換されませんでした。

com.ms.ui.UIMenuList.action は変換されませんでした。

com.ms.ui.UIMenuList.getMenuLauncher は変換されませんでした。

com.ms.ui.UIMenuList.gotFocus は変換されませんでした。

com.ms.ui.UIMenuList.keyDown は変換されませんでした。

com.ms.ui.UIMenuList.lostFocus を変換できませんでした。

com.ms.ui.UIMenuList.mouseClicked は変換されませんでした。

com.ms.ui.UIMenuList.mouseEnter は変換されませんでした。

com.ms.ui.UIMenuList.requestFocus は変換されませんでした。

com.ms.ui.UIMenuList.setMenuLauncher は変換されませんでした。

com.ms.ui.UIMenuList.UIMenuList は変換されませんでした。

com.ms.ui.UIMessageBox は変換されませんでした。

com.ms.ui.UIMessageBox.action は変換されませんでした。

com.ms.ui.UIMessageBox.doModal は変換されませんでした。

com.ms.ui.UIMessageBox.doModalIndex は変換されませんでした。

com.ms.ui.UIMessageBox.getButtonAlignment を変換できませんでした。

com.ms.ui.UIMessageBox.getButtons は変換されませんでした。

com.ms.ui.UIMessageBox.getDefaultButton は変換されませんでした。

com.ms.ui.UIMessageBox.getFrame は変換されませんでした。

com.ms.ui.UIMessageBox.getImage は変換されませんでした。

com.ms.ui.UIMessageBox.getInsets は変換されませんでした。

com.ms.ui.UIMessageBox.getPreferredSize は変換されませんでした。

com.ms.ui.UIMessageBox.getText は変換されませんでした。

com.ms.ui.UIMessageBox.getTimeout は変換されませんでした。

com.ms.ui.UIMessageBox.insets は変換されませんでした。

com.ms.ui.UIMessageBox.keyDown は変換されませんでした。

com.ms.ui.UIMessageBox.keyUp は変換されませんでした。

com.ms.ui.UIMessageBox.setButtonAlignment は変換されませんでした。

com.ms.ui.UIMessageBox.setButtons は変換されませんでした。

com.ms.ui.UIMessageBox.setDefaultButton は変換されませんでした。

com.ms.ui.UIMessageBox.setImage は変換されませんでした。

com.ms.ui.UIMessageBox.setText は変換されませんでした。

com.ms.ui.UIMessageBox.setTimeout は変換されませんでした。

com.ms.ui.UIMessageBox.timeTriggered は変換されませんでした。

com.ms.ui.UIMessageBox.UIMessageBox は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIMessageBox.UIMessageBox は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIMSVMPopup は変換されませんでした。

com.ms.ui.UIOldEvent を変換できませんでした。

com.ms.ui.UIPanel.add は変換されませんでした。

com.ms.ui.UIPanel.getChild は変換されませんでした。

com.ms.ui.UIPanel.getChildCount は変換されませんでした。

com.ms.ui.UIPanel.getHeader は変換されませんでした。

com.ms.ui.UIPanel.getID は変換されませんでした。

com.ms.ui.UIPanel.getLayout は変換されませんでした。

com.ms.ui.UIPanel.paintAll は変換されませんでした。

com.ms.ui.UIPanel.setID は変換されませんでした。

com.ms.ui.UIPanel.setLayout は変換されませんでした。

com.ms.ui.UIPanel.setRedrawing は変換されませんでした。

com.ms.ui.UIPanel.UIPanel は変換されませんでした。

com.ms.ui.UIProgress.paint は変換されませんでした。

com.ms.ui.UIProgress.UIProgress は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIProgress.UIProgress は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIProgress.UIProgress は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UIProgress.update は変換されませんでした。

com.ms.ui.UIPropertyDialog は変換されませんでした。

com.ms.ui.UIPropertyPage は変換されませんでした。

com.ms.ui.UIPushButton.getConvertedEvent は変換されませんでした。

com.ms.ui.UIPushButton.paint は変換されませんでした。

com.ms.ui.UIPushButton.setChecked は変換されませんでした。

com.ms.ui.UIPushButton.setFocused は変換されませんでした。

com.ms.ui.UIPushButton.setPressed は変換されませんでした。

com.ms.ui.UIRadioButton.UIRadioButton は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIRadioButton.UIRadioButton は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIRadioGroup.add は変換されませんでした。

com.ms.ui.UIRadioGroup.UIRadioGroup は変換されませんでした。

com.ms.ui.UIRepeatButton.mouseClicked は変換されませんでした。

com.ms.ui.UIRepeatButton.setPressed は変換されませんでした。

com.ms.ui.UIRepeatButton.timeTriggered は変換されませんでした。

com.ms.ui.UIRepeatButton.UIRepeatButton は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIRepeatButton.UIRepeatButton は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIRepeatButton.UIRepeatButton は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIRepeatButton.UIRepeatButton は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIRepeatButton.UIRepeatButton は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIRow.getName は変換されませんでした。

com.ms.ui.UIRow.requestFocus は変換されませんでした。

com.ms.ui.UIRow.setSelected を変換できませんでした。

com.ms.ui.UIRowLayout は変換されませんでした。

com.ms.ui.UIScroll は変換されませんでした。

com.ms.ui.UIScroll.scrollLineDown は変換されませんでした。

com.ms.ui.UIScroll.scrollLineUp を変換できませんでした。

com.ms.ui.UIScroll.scrollPageDown は変換されませんでした。

com.ms.ui.UIScroll.scrollPageUp は変換されませんでした。

com.ms.ui.UIScroll.setScrollInfo を変換できませんでした。

com.ms.ui.UIScroll.setScrollLine は変換されませんでした。

com.ms.ui.UIScrollBar を変換できませんでした。

com.ms.ui.UIScrollBar.<Direction> は変換されませんでした。

com.ms.ui.UIScrollBar.add は変換されませんでした。

com.ms.ui.UIScrollBar.getLayoutComponent は変換されませんでした。

com.ms.ui.UIScrollBar.setStyle は変換されませんでした。

com.ms.ui.UIScrollBar.setUsingWindowsLook は変換されませんでした。

com.ms.ui.UIScrollBar.UIScrollBar を変換できませんでした。(Java Language Conversion Assistant) (1)

com.ms.ui.UIScrollBar.UIScrollBar は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIScrollBar.UIScrollBar は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UIScrollThumb は変換されませんでした。

com.ms.ui.UIScrollViewer は変換されませんでした。

com.ms.ui.UIScrollViewer.<Position> は変換されませんでした。

com.ms.ui.UIScrollViewer.add を変換できませんでした。

com.ms.ui.UIScrollViewer.CONT は変換されませんでした。

com.ms.ui.UIScrollViewer.CONTENT は変換されませんでした。

com.ms.ui.UIScrollViewer.ensureVisible は変換されませんでした。

com.ms.ui.UIScrollViewer.getContent は変換されませんでした。

com.ms.ui.UIScrollViewer.getHLine は変換されませんでした。

com.ms.ui.UIScrollViewer.getLayoutComponent は変換されませんでした。

com.ms.ui.UIScrollViewer.getLine は変換されませんでした。

com.ms.ui.UIScrollViewer.getMinimumSize は変換されませんでした。

com.ms.ui.UIScrollViewer.getPosition は変換されませんでした。

com.ms.ui.UIScrollViewer.getPreferredSize は変換されませんでした。

com.ms.ui.UIScrollViewer.getRoleCode は変換されませんでした。

com.ms.ui.UIScrollViewer.getVLine は変換されませんでした。

com.ms.ui.UIScrollViewer.getXPosition を変換できませんでした。

com.ms.ui.UIScrollViewer.getYPosition は変換されませんでした。

com.ms.ui.UIScrollViewer.isKeyable は変換されませんでした。

com.ms.ui.UIScrollViewer.keyDown は変換されませんでした。

com.ms.ui.UIScrollViewer.remove は変換されませんでした。

com.ms.ui.UIScrollViewer.replace は変換されませんでした。

com.ms.ui.UIScrollViewer.setContent は変換されませんでした。

com.ms.ui.UIScrollViewer.setHLine は変換されませんでした。

com.ms.ui.UIScrollViewer.setLine は変換されませんでした。

com.ms.ui.UIScrollViewer.setScrollStyle は変換されませんでした。

com.ms.ui.UIScrollViewer.setUsingWindowsLook は変換されませんでした。

com.ms.ui.UIScrollViewer.setVLine は変換されませんでした。

com.ms.ui.UIScrollViewer.setXPosition は変換されませんでした。

com.ms.ui.UIScrollViewer.setYPosition を変換できませんでした。

com.ms.ui.UIScrollViewer.UIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UIScrollViewer.UIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIScrollViewer.UIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIScrollViewer.UIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIScrollViewer.UIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIScrollViewer.UIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UIScrollViewer.UIScrollViewer は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UISelector は変換されませんでした。

com.ms.ui.UISelector.addSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UISelector.addSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UISelector.addSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UISelector.addSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UISelector.find は変換されませんでした。

com.ms.ui.UISelector.getSelectedIndex は変換されませんでした。

com.ms.ui.UISelector.getSelectedItem は変換されませんでした。

com.ms.ui.UISelector.remove は変換されませんでした。

com.ms.ui.UISelector.removeSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UISelector.removeSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UISelector.removeSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UISelector.removeSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UISelector.setSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UISelector.setSelectedIndex は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UISelector.setSelectedIndices は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UISelector.setSelectedIndices could not be converted (Java Language Conversion Assistant) (2)

com.ms.ui.UISelector.setSelectedItem は変換されませんでした。

com.ms.ui.UISelector.setSelectionMode は変換されませんでした。

com.ms.ui.UISingleContainer.add は変換されませんでした。

com.ms.ui.UISingleContainer.getChildBounds は変換されませんでした。

com.ms.ui.UISingleContainer.getChildLocation は変換されませんでした。

com.ms.ui.UISingleContainer.getChildSize は変換されませんでした。

com.ms.ui.UISingleContainer.getComponent は変換されませんでした。

com.ms.ui.UISingleContainer.getHeader は変換されませんでした。

com.ms.ui.UISingleContainer.layout は変換されませんでした。

com.ms.ui.UISingleContainer.mouseExit は変換されませんでした。

com.ms.ui.UISingleContainer.relayout は変換されませんでした。

com.ms.ui.UISingleContainer.remove は変換されませんでした。

com.ms.ui.UISingleContainer.setHeader を変換できませんでした。

com.ms.ui.UISingleContainer.setID は変換されませんでした。

com.ms.ui.UISingleContainer.setName は変換されませんでした。

com.ms.ui.UISingleContainer.UISingleContainer は変換されませんでした。

com.ms.ui.UISlider.add は変換されませんでした。

com.ms.ui.UISlider.clearSelection は変換されませんでした。

com.ms.ui.UISlider.getLayoutComponent は変換されませんでした。

com.ms.ui.UISlider.getSelectionEnd は変換されませんでした。

com.ms.ui.UISlider.getSelectionStart は変換されませんでした。

com.ms.ui.UISlider.setSelection は変換されませんでした。

com.ms.ui.UISlider.setSelectionEnd は変換されませんでした。

com.ms.ui.UISlider.setSelectionStart は変換されませんでした。

com.ms.ui.UISpinner は変換されませんでした。

com.ms.ui.UISpinner.<Direction> は変換されませんでした。

com.ms.ui.UISpinner.<Type> を変換できませんでした。

com.ms.ui.UISpinner.add を変換できませんでした。

com.ms.ui.UISpinner.getLayoutComponent は変換されませんでした。

com.ms.ui.UISpinner.getMinimumSize は変換されませんでした。

com.ms.ui.UISpinner.getPreferredSize は変換されませんでした。

com.ms.ui.UISpinner.getRoleCode は変換されませんでした。

com.ms.ui.UISpinner.getScrollPos は変換されませんでした。

com.ms.ui.UISpinner.keyDown は変換されませんでした。

com.ms.ui.UISpinner.layout は変換されませんでした。

com.ms.ui.UISpinner.RAISED は変換されませんでした。

com.ms.ui.UISpinner.requestFocus は変換されませんでした。

com.ms.ui.UISpinner.scrollLineDown は変換されませんでした。

com.ms.ui.UISpinner.scrollLineUp は変換されませんでした。

com.ms.ui.UISpinner.scrollPageDown は変換されませんでした。

com.ms.ui.UISpinner.scrollPageUp は変換されませんでした。

com.ms.ui.UISpinner.setScrollInfo を変換できませんでした。

com.ms.ui.UISpinner.setStyle は変換されませんでした。

com.ms.ui.UISpinner.UISpinner は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UISpinner.UISpinner は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UISpinnerEdit.BORDER は変換されませんでした。

com.ms.ui.UISpinnerEdit.CENTER は変換されませんでした。

com.ms.ui.UISpinnerEdit.getEditStyle を変換できませんでした。

com.ms.ui.UISpinnerEdit.setEditStyle は変換されませんでした。

com.ms.ui.UISpinnerEdit.UISpinnerEdit は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UISpinnerEdit.UISpinnerEdit は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UISplitLayout は変換されませんでした。

com.ms.ui.UISplitViewer を変換できませんでした。

com.ms.ui.UISplitViewer.<Type> は変換されませんでした。

com.ms.ui.UISplitViewer.getComponent は変換されませんでした。

com.ms.ui.UISplitViewer.getFloater は変換されませんでした。

com.ms.ui.UISplitViewer.getPos を変換できませんでした。

com.ms.ui.UISplitViewer.getStyle は変換されませんでした。

com.ms.ui.UISplitViewer.mouseDown は変換されませんでした。

com.ms.ui.UISplitViewer.mouseDrag は変換されませんでした。

com.ms.ui.UISplitViewer.mouseUp は変換されませんでした。

com.ms.ui.UISplitViewer.setPos は変換されませんでした。

com.ms.ui.UISplitViewer.UISplitViewer は変換されませんでした。

com.ms.ui.UIStateComponent.adjustLayoutSize は変換されませんでした。

com.ms.ui.UIStateComponent.disableEvents を変換できませんでした。

com.ms.ui.UIStateComponent.enableEvents は変換されませんでした。

com.ms.ui.UIStateComponent.getCachedPreferredSize を変換できませんでした。

com.ms.ui.UIStateComponent.getConvertedEvent は変換されませんでした。

com.ms.ui.UIStateComponent.getFlags は変換されませんでした。

com.ms.ui.UIStateComponent.HEADER は変換されませんでした。

com.ms.ui.UIStateComponent.isInvalidating は変換されませんでした。

com.ms.ui.UIStateComponent.isRedrawing は変換されませんでした。

com.ms.ui.UIStateComponent.isSelected は変換されませんでした。

com.ms.ui.UIStateComponent.isValid は変換されませんでした。

com.ms.ui.UIStateComponent.listeners は変換されませんでした。

com.ms.ui.UIStateComponent.obtainListenerTracker は変換されませんでした。

com.ms.ui.UIStateComponent.recalcPreferredSize は変換されませんでした。

com.ms.ui.UIStateComponent.setChecked は変換されませんでした。

com.ms.ui.UIStateComponent.setFlags は変換されませんでした。

com.ms.ui.UIStateComponent.setHot は変換されませんでした。

com.ms.ui.UIStateComponent.setIndeterminate を変換できませんでした。

com.ms.ui.UIStateComponent.setInvalidating は変換されませんでした。

com.ms.ui.UIStateComponent.setPressed は変換されませんでした。

com.ms.ui.UIStateComponent.setRedrawing は変換されませんでした。

com.ms.ui.UIStateComponent.setReparent を変換できませんでした。

com.ms.ui.UIStateComponent.setSelected は変換されませんでした。

com.ms.ui.UIStateComponent.setUsingWindowsLook は変換されませんでした。

com.ms.ui.UIStateComponent.setValid は変換されませんでした。

com.ms.ui.UIStateContainer.add は変換されませんでした。

com.ms.ui.UIStateContainer.adjustLayoutSize は変換されませんでした。

com.ms.ui.UIStateContainer.disableEvents は変換されませんでした。

com.ms.ui.UIStateContainer.enableEvents は変換されませんでした。

com.ms.ui.UIStateContainer.getBackground は変換されませんでした。

com.ms.ui.UIStateContainer.getCachedPreferredSize は変換されませんでした。

com.ms.ui.UIStateContainer.getConvertedEvent は変換されませんでした。

com.ms.ui.UIStateContainer.getCursor は変換されませんでした。

com.ms.ui.UIStateContainer.getEdge は変換されませんでした。

com.ms.ui.UIStateContainer.getFlags は変換されませんでした。

com.ms.ui.UIStateContainer.getFont は変換されませんでした。

com.ms.ui.UIStateContainer.getForeground は変換されませんでした。

com.ms.ui.UIStateContainer.getIndex は変換されませんでした。

com.ms.ui.UIStateContainer.getParent は変換されませんでした。

com.ms.ui.UIStateContainer.isChecked は変換されませんでした。

com.ms.ui.UIStateContainer.isEnabled は変換されませんでした。

com.ms.ui.UIStateContainer.isFocused は変換されませんでした。

com.ms.ui.UIStateContainer.isInvalidating は変換されませんでした。

com.ms.ui.UIStateContainer.isRedrawing は変換されませんでした。

com.ms.ui.UIStateContainer.isSelected は変換されませんでした。

com.ms.ui.UIStateContainer.isValid は変換されませんでした。

com.ms.ui.UIStateContainer.isVisible は変換されませんでした。

com.ms.ui.UIStateContainer.listeners は変換されませんでした。

com.ms.ui.UIStateContainer.obtainConvertedEvent は変換されませんでした。

com.ms.ui.UIStateContainer.obtainListenerTracker は変換されませんでした。

com.ms.ui.UIStateContainer.recalcPreferredSize は変換されませんでした。

com.ms.ui.UIStateContainer.setBackground は変換されませんでした。

com.ms.ui.UIStateContainer.setChecked は変換されませんでした。

com.ms.ui.UIStateContainer.setCursor は変換されませんでした。

com.ms.ui.UIStateContainer.setEdge は変換されませんでした。

com.ms.ui.UIStateContainer.setEnabled は変換されませんでした。

com.ms.ui.UIStateContainer.setFlags は変換されませんでした。

com.ms.ui.UIStateContainer.setFont は変換されませんでした。

com.ms.ui.UIStateContainer.setForeground は変換されませんでした。

com.ms.ui.UIStateContainer.setHot は変換されませんでした。

com.ms.ui.UIStateContainer.setIndeterminate は変換されませんでした。

com.ms.ui.UIStateContainer.setIndex は変換されませんでした。

com.ms.ui.UIStateContainer.setInvalidating は変換されませんでした。

com.ms.ui.UIStateContainer.setParent を変換できませんでした。

com.ms.ui.UIStateContainer.setPressed は変換されませんでした。

com.ms.ui.UIStateContainer.setRedrawing は変換されませんでした。

com.ms.ui.UIStateContainer.setReparent は変換されませんでした。

com.ms.ui.UIStateContainer.setSelected は変換されませんでした。

com.ms.ui.UIStateContainer.setUsingWindowsLook は変換されませんでした。

com.ms.ui.UIStateContainer.setValid は変換されませんでした。

com.ms.ui.UIStateContainer.setVisible は変換されませんでした。

com.ms.ui.UIStateContainer.UIStateContainer は変換されませんでした。

com.ms.ui.UIStatic は変換されませんでした。

com.ms.ui.UIStatic.HCENTER は変換されませんでした。

com.ms.ui.UIStatic.setFlags は変換されませんでした。

com.ms.ui.UIStatic.VCENTER は変換されませんでした。

com.ms.ui.UISystem は変換されませんでした。

com.ms.ui.UITab.paint は変換されませんでした。

com.ms.ui.UITab.setSelected を変換できませんでした。

com.ms.ui.UITab.UITab は変換されませんでした。

com.ms.ui.UITabLayout を変換できませんでした。

com.ms.ui.UITabList は変換されませんでした。

com.ms.ui.UITabList.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UITabList.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UITabList.add を変換できませんでした。(Java Language Conversion Assistant) (2)

com.ms.ui.UITabList.add を変換できませんでした。(Java Language Conversion Assistant) (2)

com.ms.ui.UITabList.layout を変換できませんでした。

com.ms.ui.UITabList.paint を変換できませんでした。

com.ms.ui.UITabList.passFocus は変換されませんでした。

com.ms.ui.UITabList.setSelectedItem は変換されませんでした。

com.ms.ui.UITabListLayout を変換できませんでした。

com.ms.ui.UITabViewer.add は変換されませんでした。

com.ms.ui.UITabViewer.addTab を変換できませんでした。

com.ms.ui.UITabViewer.getConvertedEvent は変換されませんでした。

com.ms.ui.UITabViewer.paint は変換されませんでした。

com.ms.ui.UITabViewer.removeTab は変換されませんでした。

com.ms.ui.UITabViewer.requestFocus は変換されませんでした。

com.ms.ui.UITabViewer.setSelectedItem は変換されませんでした。

com.ms.ui.UIText.getPreferredSize は変換されませんでした。

com.ms.ui.UIText.paint は変換されませんでした。

com.ms.ui.UIText.setFocused は変換されませんでした。

com.ms.ui.UIText.setHot は変換されませんでした。

com.ms.ui.UIText.setSelected を変換できませんでした。

com.ms.ui.UIThreePanelLayout は変換されませんでした。

com.ms.ui.UIThumb は変換されませんでした。

com.ms.ui.UITree は変換されませんでした。

com.ms.ui.UITree.action は変換されませんでした。

com.ms.ui.UITree.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UITree.add は変換されませんでした (Java Language Conversion Assistant) (2)

com.ms.ui.UITree.add は変換されませんでした (Java Language Conversion Assistant) (3)

com.ms.ui.UITree.add は変換されませんでした (Java Language Conversion Assistant) (4)

com.ms.ui.UITree.add は変換されませんでした (Java Language Conversion Assistant) (1)

com.ms.ui.UITree.getAttachRect は変換されませんでした。

com.ms.ui.UITree.getExpander は変換されませんでした。

com.ms.ui.UITree.keyDown を変換できませんでした。

com.ms.ui.UITree.remove は変換されませんでした。

com.ms.ui.UITree.setChecked は変換されませんでした。

com.ms.ui.UITree.setExpanded は変換されませんでした。

com.ms.ui.UITree.setExpander は変換されませんでした。

com.ms.ui.UITree.setLayout は変換されませんでした。

com.ms.ui.UITreeLayout は変換されませんでした。

com.ms.ui.UIVerticalFlowLayout は変換されませんでした。

com.ms.ui.UIViewer.ensureVisible は変換されませんでした。

com.ms.ui.UIViewer.isOverlapping は変換されませんでした。

com.ms.ui.UIViewer.mouseDown は変換されませんでした。

com.ms.ui.UIViewer.mouseDrag は変換されませんでした。

com.ms.ui.UIViewer.mouseUp を変換できませんでした。

com.ms.ui.UIViewer.requestFocus は変換されませんでした。

com.ms.ui.UIViewer.timeTriggered は変換されませんでした。

com.ms.ui.UIWindow.keyDown は変換されませんでした。

com.ms.ui.UIWindow.pack は変換されませんでした。

com.ms.ui.UIWindow.setSize は変換されませんでした。

com.ms.ui.UIWinEvent は変換されませんでした。

com.ms.ui.UIWizard は変換されませんでした。

com.ms.ui.UIWizardStep は変換されませんでした。

com.ms.ui.Version は変換されませんでした。

Com.ms.util のエラー メッセージ

com.ms.util.ArraySort.compare は変換されませんでした。

com.ms.util.ArraySort.sort は変換されませんでした。

com.ms.util.ArraySort.swap は変換されませんでした。

com.ms.util.cab.CabEnumerator は変換されませんでした。

com.ms.util.cab.CabException は変換されませんでした。

com.ms.util.cab.CabFileEntry は変換されませんでした。

com.ms.util.cab.CabFolderEntry は変換されませんでした。

com.ms.util.cab.CabProgressInterface は変換されませんでした。

com.ms.util.EventLog.reportEvent は変換されませんでした。

com.ms.util.HTMLTokenizer は変換されませんでした。

com.ms.util.IIntRangeComparator は変換されませんでした。

com.ms.util.IncludeExcludeIntRanges は変換されませんでした。

com.ms.util.IncludeExcludeWildcards は変換されませんでした。

com.ms.util.ini.IniFile は変換されませんでした。

com.ms.util.ini.IniSection は変換されませんでした。

com.ms.util.ini.IniSyntaxErrorException は変換されませんでした。

com.ms.util.IntRanges.compare は変換されませんでした。

com.ms.util.IntRanges.compareSet は変換されませんでした。

com.ms.util.IntRanges.ComparisonResulttoString は変換されませんでした。

com.ms.util.IntRanges.condense は変換されませんでした。

com.ms.util.IntRanges.contains は変換されませんでした。

com.ms.util.IntRanges.indexOf は変換されませんでした。

com.ms.util.IntRanges.intersect は変換されませんでした。

com.ms.util.IntRanges.IntRanges は変換されませんでした。

com.ms.util.IntRanges.invertComparisonResult は変換されませんでした。

com.ms.util.IntRanges.lock は変換されませんでした。

com.ms.util.IntRanges.parse は変換されませんでした。

com.ms.util.IntRanges.removeRange は変換されませんでした。

com.ms.util.IntRanges.removeRanges は変換されませんでした。

com.ms.util.IntRanges.removeRanges(int, int) は変換されませんでした。

com.ms.util.IntRanges.removeRanges(int, int, IntRangeComparator) は変換されませんでした。

com.ms.util.IntRanges.removeSingleton は変換されませんでした。

com.ms.util.IntRanges.size は変換されませんでした。

com.ms.util.IntRanges.sort は変換されませんでした。

com.ms.util.IntRanges.unlock は変換されませんでした。

com.ms.util.IWildcardExpressionComparator は変換されませんでした。

com.ms.util.OrdinalMap.insertInCognate は変換されませんでした。

com.ms.util.OrdinalMap.moveCognate は変換されませんでした。

com.ms.util.OrdinalMap.newCognate は変換されませんでした。

com.ms.util.OrdinalMap.removeFromCognate は変換されませんでした。

com.ms.util.ProvideSetComparisonInfo は変換されませんでした。

com.ms.util.Queue.capacity は変換されませんでした。

com.ms.util.Queue.elementFromHead は変換されませんでした。

com.ms.util.Queue.elementFromTail は変換されませんでした。

com.ms.util.Queue.hasMoreElements は変換されませんでした。

com.ms.util.Queue.nextElement は変換されませんでした。

com.ms.util.Queue.setCapacity は変換されませんでした。

com.ms.util.Queue.toString は変換されませんでした。

com.ms.util.SetComparer は変換されませんでした。

com.ms.util.SetComparison は変換されませんでした。

com.ms.util.SetTemplate.foundAt は変換されませんでした。

com.ms.util.SetTemplate.foundCognate は変換されませんでした。

com.ms.util.SetTemplate.foundIn は変換されませんでした。

com.ms.util.SetTemplate.generation は変換されませんでした。

com.ms.util.SetTemplate.insertInCognate は変換されませんでした。

com.ms.util.SetTemplate.keys は変換されませんでした。

com.ms.util.SetTemplate.locate は変換されませんでした。

com.ms.util.SetTemplate.moveCognate は変換されませんでした。

com.ms.util.SetTemplate.newCognate は変換されませんでした。

com.ms.util.SetTemplate.rehash は変換されませんでした。

com.ms.util.SetTemplate.removeFromCognate は変換されませんでした。

com.ms.util.SetTemplate.reportFinds は変換されませんでした。

com.ms.util.SetTemplate.reportRequests は変換されませんでした。

com.ms.util.SetTemplate.reportUnits は変換されませんでした。

com.ms.util.SetTemplate.SetTemplate は変換されませんでした。

com.ms.util.Sort.compare は変換されませんでした。

com.ms.util.Sort.doSort は変換されませんでした。

com.ms.util.Sort.swap は変換されませんでした。

com.ms.util.StringComparison.ascending は変換されませんでした。

com.ms.util.StringComparison.descending は変換されませんでした。

com.ms.util.SystemVersionManager は変換されませんでした。

com.ms.util.Task は変換されませんでした。

com.ms.util.TaskManager は変換されませんでした。

com.ms.util.ThreadLocalStorage は変換されませんでした。

com.ms.util.Timer.getRepeat は変換されませんでした。

com.ms.util.Timer.getUserID は変換されませんでした。

com.ms.util.Timer.Timer は変換されませんでした。

com.ms.util.TimerEvent.getSource は変換されませんでした。

com.ms.util.TimerEvent.getTime は変換されませんでした。

com.ms.util.TimerEvent.getUserID は変換されませんでした。

com.ms.util.TimerListener は変換されませんでした。

com.ms.util.UnsignedIntRanges.contains は変換されませんでした。

com.ms.util.UnsignedIntRanges.indexOf は変換されませんでした。

com.ms.util.UnsignedIntRanges.intersect は変換されませんでした。

com.ms.util.UnsignedIntRanges.removeRange は変換されませんでした。

com.ms.util.UnsignedIntRanges.UnsignedIntRanges は変換されませんでした。

com.ms.util.VectorSort.compare は変換されませんでした。

com.ms.util.VectorSort.swap は変換されませんでした。

com.ms.util.WildcardExpression は変換されませんでした。

com.ms.util.zip.ZipInputStreamEx は変換されませんでした。

com.ms.util.zip.ZipOutputStreamEx は変換されませんでした。

Com.ms.wfc のエラー メッセージ

com.ms.wfc.Rectangle.Editor を変換できませんでした。

Com.ms.wfc.app のエラー メッセージ

com.ms.wfc.app.Application.addOnSettingChange は変換されませんでした。

com.ms.wfc.app.Application.addOnSystemShutdown は変換されませんでした。

com.ms.wfc.app.Application.allocThreadStorage は変換されませんでした。

com.ms.wfc.app.Application.createThread は変換されませんでした。

com.ms.wfc.app.Application.doEvents は変換されませんでした。

com.ms.wfc.app.Application.freeThreadStorage は変換されませんでした。

com.ms.wfc.app.Application.getParkingForm は変換されませんでした。

com.ms.wfc.app.Application.getThreadStorage は変換されませんでした。

com.ms.wfc.app.Application.removeOnSettingChange は変換されませんでした。

com.ms.wfc.app.Application.removeOnSystemShutdown は変換されませんでした。

com.ms.wfc.app.Application.runDialog は変換されませんでした。

com.ms.wfc.app.Application.setThreadStorage は変換されませんでした。

com.ms.wfc.app.CharacterSet は変換されませんでした。

com.ms.wfc.app.Clipboard.Clipboard は変換されませんでした。

com.ms.wfc.app.DataFormats.CF_CSV を変換できませんでした。

com.ms.wfc.app.DataFormats.CF_WFCOBJECT を変換できませんでした。

com.ms.wfc.app.DataFormats.DataFormats は変換されませんでした。

com.ms.wfc.app.DataFormats.Format.Format は変換されませんでした。

com.ms.wfc.app.DataFormats.Format.win32Handle は変換されませんでした。

com.ms.wfc.app.DataObject.OleDAdvise は変換されませんでした。

com.ms.wfc.app.DataObject.OleDUnadvise は変換されませんでした。

com.ms.wfc.app.DataObject.OleEnumDAdvise は変換されませんでした。

com.ms.wfc.app.DataObject.OleEnumFormatEtc は変換されませんでした。

com.ms.wfc.app.DataObject.OleGetCanonicalFormatEtc は変換されませんでした。

com.ms.wfc.app.DataObject.OleGetData は変換されませんでした。

com.ms.wfc.app.DataObject.OleGetDataHere は変換されませんでした。

com.ms.wfc.app.DataObject.OleQueryGetData は変換されませんでした。

com.ms.wfc.app.DataObject.OleSetData は変換されませんでした。

com.ms.wfc.app.IMessageFilter.postFilterMessage は変換されませんでした。

com.ms.wfc.app.KeywordVk は変換されませんでした。

com.ms.wfc.app.Languages.Languages を変換できませんでした。

com.ms.wfc.app.Locale.CalendarType は変換されませんでした。

com.ms.wfc.app.Locale.compareStrings を変換できませんでした。

com.ms.wfc.app.Locale.DateFormatOrder は変換されませんでした。

com.ms.wfc.app.Locale.getCalendarType は変換されませんでした。

com.ms.wfc.app.Locale.getCenturyFormat は変換されませんでした。

com.ms.wfc.app.Locale.getCharacterSet は変換されませんでした。

com.ms.wfc.app.Locale.getCompareIgnoreCase は変換されませんでした。

com.ms.wfc.app.Locale.getCompareIgnoreKana は変換されませんでした。

com.ms.wfc.app.Locale.getCompareIgnoreKashida は変換されませんでした。

com.ms.wfc.app.Locale.getCompareIgnoreNonSpace は変換されませんでした。

com.ms.wfc.app.Locale.getCompareIgnoreSymbols は変換されませんでした。

com.ms.wfc.app.Locale.getCompareIgnoreWidth は変換されませんでした。

com.ms.wfc.app.Locale.getCountryCode は変換されませんでした。

com.ms.wfc.app.Locale.getDefaultCountry は変換されませんでした。

com.ms.wfc.app.Locale.getEnglishCurrencyName は変換されませんでした。

com.ms.wfc.app.Locale.getLeadingZero は変換されませんでした。

com.ms.wfc.app.Locale.getNativeDigits は変換されませんでした。

com.ms.wfc.app.Locale.getSortId は変換されませんでした。

com.ms.wfc.app.Locale.getSupportedLocales は変換されませんでした。

com.ms.wfc.app.Locale.Languages.Locale.Languages を変換できませんでした。

com.ms.wfc.app.Locale.Languages.RHAETO_ROMAN は変換されませんでした。

com.ms.wfc.app.Locale.LeadingZeros は変換されませんでした。

com.ms.wfc.app.Locale.Locale は変換されませんでした。

com.ms.wfc.app.Locale.MeasurementSystem は変換されませんでした。

com.ms.wfc.app.Locale.NegativeNumberMode.Locale.NegativeNumberMode を変換できませんでした。

com.ms.wfc.app.Locale.OptionalCalendarType は変換されませんでした。

com.ms.wfc.app.Locale.PositiveCurrencyMode.Locale.PositiveCurrencyMode を変換できませんでした。

com.ms.wfc.app.Locale.setCalendarType は変換されませんでした。

com.ms.wfc.app.Locale.setCompareIgnoreCase は変換されませんでした。

com.ms.wfc.app.Locale.setCompareIgnoreKana は変換されませんでした。

com.ms.wfc.app.Locale.setCompareIgnoreKashida は変換されませんでした。

com.ms.wfc.app.Locale.setCompareIgnoreNonSpace は変換されませんでした。

com.ms.wfc.app.Locale.setCompareIgnoreSymbols は変換されませんでした。

com.ms.wfc.app.Locale.setCompareIgnoreWidth は変換されませんでした。

com.ms.wfc.app.Locale.setFirstDayOfWeek は変換されませんでした。

com.ms.wfc.app.Locale.setFirstWeekOfYear は変換されませんでした。

com.ms.wfc.app.Locale.setLeadingZero は変換されませんでした。

com.ms.wfc.app.Locale.setListSeparator は変換されませんでした。

com.ms.wfc.app.Locale.setMeasurementSystem は変換されませんでした。

com.ms.wfc.app.Locale.Sort は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.CHINESE_SIMPLIFIED は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.CHINESE_TRADITIONAL は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.DUTCH_BELGIAN は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.ENGLISH_AUS は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.ENGLISH_CAN は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.ENGLISH_EIRE は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.ENGLISH_NZ は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.ENGLISH_UK は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.ENGLISH_US は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.FRENCH_BELGIAN は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.FRENCH_CANADIAN は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.FRENCH_SWISS は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.GERMAN_AUSTRIAN は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.GERMAN_SWISS は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.ITALIAN_SWISS は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.Locale.SubLanguages は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.NORWEGIAN_BOKMAL は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.NORWEGIAN_NYNORSK は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.PORTUGUESE_BRAZILIAN は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.SERBO_CROATIAN_CYRILLIC は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.SERBO_CROATIAN_LATIN は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.SPANISH_MEXICAN は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.SPANISH_MODERN は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.SubLanguages は変換されませんでした。

com.ms.wfc.app.Locale.SubLanguages.SYS_DEFAULT は変換されませんでした。

com.ms.wfc.app.Message.free は変換されませんでした。

com.ms.wfc.app.MethodInvoker.MethodInvoker は変換されませんでした。

com.ms.wfc.app.NegativeNumberMode.NegativeNumberMode を変換できませんでした。

com.ms.wfc.app.PositiveCurrencyMode.PositiveCurrencyMode を変換できませんでした。

com.ms.wfc.app.Registry.Registry は変換されませんでした。

com.ms.wfc.app.RegistryKey.getBaseKey は変換されませんでした。

com.ms.wfc.app.SendKeysHookProc は変換されませんでした。

com.ms.wfc.app.SKEvent は変換されませんでした。

com.ms.wfc.app.SubLanguages.SubLanguages を変換できませんでした。

com.ms.wfc.app.SystemInformation.getArrange は変換されませんでした。

com.ms.wfc.app.ThreadExceptionEventHandler.ThreadExceptionEventHandler を変換できませんでした。

com.ms.wfc.app.Time.getConstructorArgs は変換されませんでした。

com.ms.wfc.app.Time.getExtension は変換されませんでした。

com.ms.wfc.app.Time.save は変換されませんでした。

com.ms.wfc.app.Time.Time は変換されませんでした。

com.ms.wfc.app.Time.toSystemTime は変換されませんでした。

com.ms.wfc.app.Time.toVariant は変換されませんでした。

com.ms.wfc.app.Window.callback は変換されませんでした。

Com.ms.wfc.ax のエラー メッセージ

com.ms.wfc.ax は変換されませんでした。

com.ms.wfc.ax._POINTL は変換されませんでした。

com.ms.wfc.ax.ActiveX は変換されませんでした。

com.ms.wfc.ax.Ambients は変換されませんでした。

com.ms.wfc.ax.IAdviseSink は変換されませんでした。

com.ms.wfc.ax.ICategorizeProperties は変換されませんでした。

com.ms.wfc.ax.IClassFactory は変換されませんでした。

com.ms.wfc.ax.IClassFactory2 は変換されませんでした。

com.ms.wfc.ax.IDDispatch は変換されませんでした。

com.ms.wfc.ax.IEDispatch は変換されませんでした。

com.ms.wfc.ax.IEnumOLEVERB は変換されませんでした。

com.ms.wfc.ax.IEnumUnknown は変換されませんでした。

com.ms.wfc.ax.IExtender は変換されませんでした。

com.ms.wfc.ax.IFont は変換されませんでした。

com.ms.wfc.ax.IFontDisp は変換されませんでした。

com.ms.wfc.ax.IGetOleObject は変換されませんでした。

com.ms.wfc.ax.IGetVBAObject は変換されませんでした。

com.ms.wfc.ax.IMyDispatch は変換されませんでした。

com.ms.wfc.ax.IObjectIdentity は変換されませんでした。

com.ms.wfc.ax.IObjectWithSite は変換されませんでした。

com.ms.wfc.ax.IOleClientSite は変換されませんでした。

com.ms.wfc.ax.IOleContainer は変換されませんでした。

com.ms.wfc.ax.IOleControl は変換されませんでした。

com.ms.wfc.ax.IOleControlSite は変換されませんでした。

com.ms.wfc.ax.IOleInPlaceActiveObject は変換されませんでした。

com.ms.wfc.ax.IOleInPlaceFrame は変換されませんでした。

com.ms.wfc.ax.IOleInPlaceObject は変換されませんでした。

com.ms.wfc.ax.IOleInPlaceSite は変換されませんでした。

com.ms.wfc.ax.IOleInPlaceUIWindow は変換されませんでした。

com.ms.wfc.ax.IOleWindow は変換されませんでした。

com.ms.wfc.ax.IParseDisplayName は変換されませんでした。

com.ms.wfc.ax.IPerPropertyBrowsing は変換されませんでした。

com.ms.wfc.ax.IPersist は変換されませんでした。

com.ms.wfc.ax.IPersist.iid は変換されませんでした。

com.ms.wfc.ax.IPersistPropertyBag は変換されませんでした。

com.ms.wfc.ax.IPersistStorage は変換されませんでした。

com.ms.wfc.ax.IPersistStream は変換されませんでした。

com.ms.wfc.ax.IPersistStreamInit は変換されませんでした。

com.ms.wfc.ax.IPicture は変換されませんでした。

com.ms.wfc.ax.IPictureDisp は変換されませんでした。

com.ms.wfc.ax.IPropertyBag は変換されませんでした。

com.ms.wfc.ax.IQuickActivate は変換されませんでした。

com.ms.wfc.ax.ISimpleFrameSite は変換されませんでした。

com.ms.wfc.ax.ISpecifyPropertyPages は変換されませんでした。

com.ms.wfc.ax.IVBFormat は変換されませんでした。

com.ms.wfc.ax.IVBGetControl は変換されませんでした。

com.ms.wfc.ax.tagCADWORD は変換されませんでした。

com.ms.wfc.ax.tagCALPOLESTR は変換されませんでした。

com.ms.wfc.ax.tagCAUUID は変換されませんでした。

com.ms.wfc.ax.tagCONTROLINFO は変換されませんでした。

com.ms.wfc.ax.tagDISPPARAMS は変換されませんでした。

com.ms.wfc.ax.tagLICINFO は変換されませんでした。

com.ms.wfc.ax.tagLOGPALETTE は変換されませんでした。

com.ms.wfc.ax.tagMSG は変換されませんでした。

com.ms.wfc.ax.tagOIFI は変換されませんでした。

com.ms.wfc.ax.tagOleMenuGroupWidths は変換されませんでした。

com.ms.wfc.ax.tagOLEVERB は変換されませんでした。

com.ms.wfc.ax.tagPOINTF は変換されませんでした。

com.ms.wfc.ax.tagQACONTAINER は変換されませんでした。

com.ms.wfc.ax.tagQACONTROL は変換されませんでした。

com.ms.wfc.ax.tagRECT は変換されませんでした。

com.ms.wfc.ax.tagSIZE は変換されませんでした。

com.ms.wfc.ax.tagSIZEL は変換されませんでした。

Com.ms.wfc.core のエラー メッセージ

com.ms.wfc.core.ArrayDialog は変換されませんでした。

com.ms.wfc.core.ArrayEditor.createNewInstance は変換されませんでした。

com.ms.wfc.core.ArrayEditor.defaultPropInfo は変換されませんでした。

com.ms.wfc.core.ArrayEditor.editValue は変換されませんでした。

com.ms.wfc.core.ArrayEditor.getBaseName は変換されませんでした。

com.ms.wfc.core.ArrayEditor.getDisplayText は変換されませんでした。

com.ms.wfc.core.ArrayEditor.getTextFromValue は変換されませんでした。

com.ms.wfc.core.ArrayEditor.getTypeDescription は変換されませんでした。

com.ms.wfc.core.BooleanEditor.getStyle は変換されませんでした。

com.ms.wfc.core.BooleanEditor.getValueFromText は変換されませんでした。

com.ms.wfc.core.CancelEventHandler.CancelEventHandler は変換されませんでした。

com.ms.wfc.core.CategoryAttribute.Position は変換されませんでした。

com.ms.wfc.core.Component.appendEventHandlers は変換されませんでした。

com.ms.wfc.core.Component.componentChanged は変換されませんでした。

com.ms.wfc.core.Component.getDisposing は変換されませんでした。

com.ms.wfc.core.Component.getResource は変換されませんでした。

com.ms.wfc.core.Component.getService は変換されませんでした。

com.ms.wfc.core.ComponentInfo.getClassInfo は変換されませんでした。

com.ms.wfc.core.ComponentInfo.getDefaultEventInfo は変換されませんでした。

com.ms.wfc.core.ComponentInfo.getExtenders は変換されませんでした。

com.ms.wfc.core.ComponentManager.createClassInfo は変換されませんでした。

com.ms.wfc.core.ComponentManager.createValueEditor は変換されませんでした。

com.ms.wfc.core.ComponentManager.getComponentInfo は変換されませんでした。

com.ms.wfc.core.ComponentManager.getValueEditor は変換されませんでした。

com.ms.wfc.core.ComponentManager.registerClassInfo は変換されませんでした。

com.ms.wfc.core.ComponentManager.registerValueEditorClass は変換されませんでした。

com.ms.wfc.core.ConstructorArg は変換されませんでした。

com.ms.wfc.core.ConstructorArg.ConstructorArg は変換されませんでした。

com.ms.wfc.core.CustomizerVerb.addOnVerbExecute は変換されませんでした。

com.ms.wfc.core.CustomizerVerb.CustomizerVerb は変換されませんでした。

com.ms.wfc.core.CustomizerVerb.getBitmap は変換されませんでした。

com.ms.wfc.core.CustomizerVerb.getData は変換されませんでした。

com.ms.wfc.core.CustomizerVerb.onVerbExecute は変換されませんでした。

com.ms.wfc.core.CustomizerVerb.performVerbExecute は変換されませんでした。

com.ms.wfc.core.CustomizerVerb.removeOnVerbExecute は変換されませんでした。

com.ms.wfc.core.CustomizerVerb.setBitmap は変換されませんでした。

com.ms.wfc.core.CustomizerVerb.setData は変換されませんでした。

com.ms.wfc.core.DescriptionAttribute.getDescription は変換されませんでした。

com.ms.wfc.core.DesignForm.DesignForm は変換されませんでした。

com.ms.wfc.core.DesignForm.getPage は変換されませんでした。

com.ms.wfc.core.DesignForm.getPageCount は変換されませんでした。

com.ms.wfc.core.DesignForm.showForm は変換されませんでした。

com.ms.wfc.core.DesignPage.getPageMessage は変換されませんでした。

com.ms.wfc.core.DesignPage.NULLVALUE は変換されませんでした。

com.ms.wfc.core.DesignPage.objects は変換されませんでした。

com.ms.wfc.core.DesignPage.onReadProperty は変換されませんでした。

com.ms.wfc.core.DesignPage.onWriteProperty は変換されませんでした。

com.ms.wfc.core.DesignPage.properties は変換されませんでした。

com.ms.wfc.core.DesignPage.setObjects は変換されませんでした。

com.ms.wfc.core.DoubleEditor.getValueFromText は変換されませんでした。

com.ms.wfc.core.Enum.Enum は変換されませんでした。

com.ms.wfc.core.Event.Event は変換されませんでした。

com.ms.wfc.core.Event.extendedInfo は変換されませんでした。

com.ms.wfc.core.EventHandler.EventHandler は変換されませんでした。

com.ms.wfc.core.EventInfo.addEventHandler は変換されませんでした。

com.ms.wfc.core.EventInfo.getAddMethod は変換されませんでした。

com.ms.wfc.core.EventInfo.getMulticast は変換されませんでした。

com.ms.wfc.core.EventInfo.getRemoveMethod は変換されませんでした。

com.ms.wfc.core.EventInfo.getType は変換されませんでした。

com.ms.wfc.core.EventInfo.removeEventHandler は変換されませんでした。

com.ms.wfc.core.ExtenderInfo は変換されませんでした。

com.ms.wfc.core.FieldsEditor は変換されませんでした。

com.ms.wfc.core.FloatEditor.getValueFromText は変換されませんでした。

com.ms.wfc.core.IAttributes は変換されませんでした。

com.ms.wfc.core.IClassInfo は変換されませんでした。

com.ms.wfc.core.IComponent.dispose は変換されませんでした。

com.ms.wfc.core.IComponent.getChildOf は変換されませんでした。

com.ms.wfc.core.IComponentSite.componentChanged は変換されませんでした。

com.ms.wfc.core.IComponentSite.getDisposing は変換されませんでした。

com.ms.wfc.core.IConstructable は変換されませんでした。

com.ms.wfc.core.IContainer.dispose は変換されませんでした。

com.ms.wfc.core.IContainer.getComponent は変換されませんでした。

com.ms.wfc.core.IContainer.getComponents は変換されませんでした。

com.ms.wfc.core.ICustomizer.getDesignPages は変換されませんでした。

com.ms.wfc.core.ICustomizer.getHitTest は変換されませんでした。

com.ms.wfc.core.ICustomizer.getVerbs は変換されませんでした。

com.ms.wfc.core.IDesignPage.getSize は変換されませんでした。

com.ms.wfc.core.IDesignPage.setObjects は変換されませんでした。

com.ms.wfc.core.IDesignPage.setSize は変換されませんでした。

com.ms.wfc.core.IEditorHost.getHostHandle は変換されませんでした。

com.ms.wfc.core.IEditorSite.getType は変換されませんでした。

com.ms.wfc.core.IEditorSite.getValueOwner は変換されませんでした。

com.ms.wfc.core.IEditorSite.getValueOwners は変換されませんでした。

com.ms.wfc.core.IResourceLoader は変換されませんでした。

com.ms.wfc.core.IResourceManager.containsProperty は変換されませんでした。

com.ms.wfc.core.IResourceManager.getNames は変換されませんでした。

com.ms.wfc.core.IResourceManager.getProperties は変換されませんでした。

com.ms.wfc.core.IResourceManager.saveAllProperties は変換されませんでした。

com.ms.wfc.core.IResourceManager.setProperty は変換されませんでした。

com.ms.wfc.core.IResourceManager.setResourceSet は変換されませんでした。

com.ms.wfc.core.IValueEditor.getConstantName は変換されませんでした。

com.ms.wfc.core.IValueEditor.getValueFromSubPropertyValues は変換されませんでした。

com.ms.wfc.core.IValueEditor.STYLE_IMMEDIATE は変換されませんでした。

com.ms.wfc.core.IValueEditor.STYLE_NOARRAYEXPANSION は変換されませんでした。

com.ms.wfc.core.IValueEditor.STYLE_NOARRAYMULTISELECT は変換されませんでした。

com.ms.wfc.core.IValueEditor.STYLE_NOEDITABLETEXT は変換されませんでした。

com.ms.wfc.core.IValueEditor.STYLE_PAINTVALUE は変換されませんでした。

com.ms.wfc.core.IValueEditor.STYLE_PROPERTIES は変換されませんでした。

com.ms.wfc.core.IValueEditor.STYLE_SHOWFIELDS は変換されませんでした。

com.ms.wfc.core.IValueEditor.STYLE_VALUES は変換されませんでした。

com.ms.wfc.core.MemberAttribute.MemberAttribute は変換されませんでした。

com.ms.wfc.core.NonEditableReferenceEditor.getStyle は変換されませんでした。

com.ms.wfc.core.NonEditableReferenceEditor.NonEditableReferenceEditor は変換されませんでした。

com.ms.wfc.core.NonPersistableAttribute は変換されませんでした。

com.ms.wfc.core.PropertyInfo.canResetValue は変換されませんでした。

com.ms.wfc.core.PropertyInfo.getGetMethod は変換されませんでした。

com.ms.wfc.core.PropertyInfo.getIsReadOnly は変換されませんでした。

com.ms.wfc.core.PropertyInfo.getResetMethod は変換されませんでした。

com.ms.wfc.core.PropertyInfo.getSetMethod は変換されませんでした。

com.ms.wfc.core.PropertyInfo.getShouldPersistMethod は変換されませんでした。

com.ms.wfc.core.PropertyInfo.getType は変換されませんでした。

com.ms.wfc.core.PropertyInfo.getValue は変換されませんでした。

com.ms.wfc.core.PropertyInfo.getValueEditor は変換されませんでした。

com.ms.wfc.core.PropertyInfo.resetValue は変換されませんでした。

com.ms.wfc.core.PropertyInfo.setValue は変換されませんでした。

com.ms.wfc.core.PropertyInfo.shouldPersistValue は変換されませんでした。

com.ms.wfc.core.ReferenceEditor.container は変換されませんでした。

com.ms.wfc.core.ReferenceEditor.getStyle は変換されませんでした。

com.ms.wfc.core.ReferenceEditor.getTextFromValue は変換されませんでした。

com.ms.wfc.core.ReferenceEditor.getValueFromText は変換されませんでした。

com.ms.wfc.core.ReferenceEditor.getValues は変換されませんでした。

com.ms.wfc.core.ReferenceEditor.none は変換されませんでした。

com.ms.wfc.core.ReferenceEditor.ReferenceEditor は変換されませんでした。

com.ms.wfc.core.ReferenceEditor.type は変換されませんでした。

com.ms.wfc.core.ResourceManager.containsProperty は変換されませんでした。

com.ms.wfc.core.ResourceManager.EXTENSION は変換されませんでした。

com.ms.wfc.core.ResourceManager.getLocaleFileNameFromBase は変換されませんでした。

com.ms.wfc.core.ResourceManager.getNames は変換されませんでした。

com.ms.wfc.core.ResourceManager.getProperties は変換されませんでした。

com.ms.wfc.core.ResourceManager.getResource は変換されませんでした。

com.ms.wfc.core.ResourceManager.getResourceLoader は変換されませんでした。

com.ms.wfc.core.ResourceManager.ResourceManager は変換されませんでした。

com.ms.wfc.core.ResourceManager.save は変換されませんでした。

com.ms.wfc.core.ResourceManager.saveAllProperties は変換されませんでした。

com.ms.wfc.core.ResourceManager.setBaseName は変換されませんでした。

com.ms.wfc.core.ResourceManager.setProperty は変換されませんでした。

com.ms.wfc.core.ResourceManager.setResourceLoader は変換されませんでした。

com.ms.wfc.core.ResourceManager.setResourceSet は変換されませんでした。

com.ms.wfc.core.ResourceReader は変換されませんでした。

com.ms.wfc.core.ResourceWriter は変換されませんでした。

com.ms.wfc.core.ShowInToolboxAttribute.NO は変換されませんでした。

com.ms.wfc.core.ShowInToolboxAttribute.YES は変換されませんでした。

com.ms.wfc.core.StringListDialog は変換されませんでした。

com.ms.wfc.core.StringListEditor は変換されませんでした。

com.ms.wfc.core.Sys は変換されませんでした。

com.ms.wfc.core.ValueEditor.editValue は変換されませんでした。

com.ms.wfc.core.ValueEditor.getValueFromSubPropertyValues は変換されませんでした。

com.ms.wfc.core.ValueEditorAttribute.valueEditorType は変換されませんでした。

com.ms.wfc.core.VerbExecuteEvent は変換されませんでした。

com.ms.wfc.core.VerbExecuteEventHandler は変換されませんでした。

com.ms.wfc.core.WFCException.getBaseException は変換されませんでした。

com.ms.wfc.core.WFCException.printStackTrace は変換されませんでした。

Com.ms.wfc.data のエラー メッセージ

com.ms.wfc.data.AdoEnums は変換されませんでした。

com.ms.wfc.data.AdoEvent は変換されませんでした。

com.ms.wfc.data.AdoException.AdoException は変換されませんでした。

com.ms.wfc.data.AdoProperties.AdoProperties は変換されませんでした。

com.ms.wfc.data.BooleanDataFormat は変換されませんでした。

com.ms.wfc.data.Connection.addOnBeginTransComplete は変換されませんでした。

com.ms.wfc.data.Connection.addOnCommitTransComplete は変換されませんでした。

com.ms.wfc.data.Connection.addOnConnectComplete は変換されませんでした。

com.ms.wfc.data.Connection.addOnDisconnect は変換されませんでした。

com.ms.wfc.data.Connection.addOnExecuteComplete は変換されませんでした。

com.ms.wfc.data.Connection.addOnInfoMessage は変換されませんでした。

com.ms.wfc.data.Connection.addOnRollbackTransComplete は変換されませんでした。

com.ms.wfc.data.Connection.addOnWillConnect は変換されませんでした。

com.ms.wfc.data.Connection.addOnWillExecute は変換されませんでした。

com.ms.wfc.data.Connection.Connection は変換されませんでした。

com.ms.wfc.data.Connection.getPeer は変換されませんでした。

com.ms.wfc.data.Connection.openSchema は変換されませんでした。

com.ms.wfc.data.Connection.removeOnBeginTransComplete は変換されませんでした。

com.ms.wfc.data.Connection.removeOnCommitTransComplete は変換されませんでした。

com.ms.wfc.data.Connection.removeOnConnectComplete は変換されませんでした。

com.ms.wfc.data.Connection.removeOnDisconnect は変換されませんでした。

com.ms.wfc.data.Connection.removeOnExecuteComplete は変換されませんでした。

com.ms.wfc.data.Connection.removeOnInfoMessage は変換されませんでした。

com.ms.wfc.data.Connection.removeOnRollbackTransComplete は変換されませんでした。

com.ms.wfc.data.Connection.removeOnWillConnect は変換されませんでした。

com.ms.wfc.data.Connection.removeOnWillExecute は変換されませんでした。

com.ms.wfc.data.Connection.setConnectionString は変換されませんでした。

com.ms.wfc.data.ConnectionEvent は変換されませんでした。

com.ms.wfc.data.ConnectionEventHandler は変換されませんでした。

com.ms.wfc.data.DataFormat は変換されませんでした。

com.ms.wfc.data.DataFormat.FormatWrapper は変換されませんでした。

com.ms.wfc.data.DateDataFormat は変換されませんでした。

com.ms.wfc.data.DateFormat.valid は変換されませんでした。

com.ms.wfc.data.dsl.<ClassName>.clsid は変換されませんでした。

com.ms.wfc.data.dsl.<ClassName>.iid は変換されませんでした。

com.ms.wfc.data.dsl._COAUTHIDENTITY._COAUTHIDENTITY は変換されませんでした。

com.ms.wfc.data.dsl._COAUTHINFO._COAUTHINFO は変換されませんでした。

com.ms.wfc.data.dsl._COSERVERINFO._COSERVERINFO は変換されませんでした。

com.ms.wfc.data.dsl._RemotableHandle._RemotableHandle は変換されませんでした。

com.ms.wfc.data.dsl.tagMULTI_QI は変換されませんでした。

com.ms.wfc.data.EventHandlerList.clear は変換されませんでした。

com.ms.wfc.data.EventHandlerList.getList は変換されませんでした。

com.ms.wfc.data.EventsListener は変換されませんでした。

com.ms.wfc.data.Field.getDataTimestamp は変換されませんでした。

com.ms.wfc.data.Field.getDispatch は変換されませんでした。

com.ms.wfc.data.Field.getGuid は変換されませんでした。

com.ms.wfc.data.Field.getInt は変換されませんでした。

com.ms.wfc.data.Field.getObject は変換されませんでした。

com.ms.wfc.data.Field.getTime は変換されませんでした。

com.ms.wfc.data.Field.getTimestamp は変換されませんでした。

com.ms.wfc.data.Field.setDataDate は変換されませんでした。

com.ms.wfc.data.Field.setDispatch は変換されませんでした。

com.ms.wfc.data.Field.setGuid は変換されませんでした。

com.ms.wfc.data.Field.setTime は変換されませんでした。

com.ms.wfc.data.Field.setTimestamp は変換されませんでした。

com.ms.wfc.data.IDataFormat は変換されませんでした。

com.ms.wfc.data.IDataFormat.Editor は変換されませんでした。

com.ms.wfc.data.IDataSource.addDataSourceListener は変換されませんでした。

com.ms.wfc.data.IDataSource.getDataMember は変換されませんでした。

com.ms.wfc.data.IDataSource.getDataMemberCount は変換されませんでした。

com.ms.wfc.data.IDataSource.getDataMemberName は変換されませんでした。

com.ms.wfc.data.IDataSource.iid は変換されませんでした。

com.ms.wfc.data.IDataSource.removeDataSourceListener は変換されませんでした。

com.ms.wfc.data.IDataSourceListener は変換されませんでした。

com.ms.wfc.data.IMarshal.iid は変換されませんでした。

com.ms.wfc.data.IPersist.iid は変換されませんでした。

com.ms.wfc.data.IPersistStream.GetSizeMax は変換されませんでした。

com.ms.wfc.data.IPersistStream.iid は変換されませんでした。

com.ms.com.IPersistFile.iid は変換されませんでした。

com.ms.wfc.data.NumberDataFormat は変換されませんでした。

com.ms.wfc.data.ObjectProxy.call は変換されませんでした。

com.ms.wfc.data.rds.<ClassName>.clsid は変換されませんでした。

com.ms.wfc.data.rds.<ClassName>.iid は変換されませんでした。

com.ms.wfc.data.Recordset.addDataSourceListener は変換されませんでした。

com.ms.wfc.data.Recordset.addNew は変換されませんでした。

com.ms.wfc.data.Recordset.addOnEndOfRecordset は変換されませんでした。

com.ms.wfc.data.Recordset.addOnFetchComplete は変換されませんでした。

com.ms.wfc.data.Recordset.addOnFetchProgress は変換されませんでした。

com.ms.wfc.data.Recordset.addOnFieldChangeComplete は変換されませんでした。

com.ms.wfc.data.Recordset.addOnMoveComplete は変換されませんでした。

com.ms.wfc.data.Recordset.addOnRecordChangeComplete は変換されませんでした。

com.ms.wfc.data.Recordset.addOnRecordsetChangeComplete は変換されませんでした。

com.ms.wfc.data.Recordset.addOnWillChangeField は変換されませんでした。

com.ms.wfc.data.Recordset.addOnWillChangeRecord は変換されませんでした。

com.ms.wfc.data.Recordset.addOnWillChangeRecordset は変換されませんでした。

com.ms.wfc.data.Recordset.addOnWillMove は変換されませんでした。

com.ms.wfc.data.Recordset.DisconnectObject は変換されませんでした。

com.ms.wfc.data.Recordset.GetClassID は変換されませんでした。

com.ms.wfc.data.Recordset.getCommand は変換されませんでした。

com.ms.wfc.data.Recordset.getDataMember は変換されませんでした。

com.ms.wfc.data.Recordset.getDataMemberCount は変換されませんでした。

com.ms.wfc.data.Recordset.getDataMemberName は変換されませんでした。

com.ms.wfc.data.Recordset.GetMarshalSizeMax は変換されませんでした。

com.ms.wfc.data.Recordset.getRecordset は変換されませんでした。

com.ms.wfc.data.Recordset.getRows は変換されませんでした。

com.ms.wfc.data.Recordset.GetSizeMax は変換されませんでした。

com.ms.wfc.data.Recordset.GetUnmarshalClass は変換されませんでした。

com.ms.wfc.data.Recordset.IsDirty は変換されませんでした。

com.ms.wfc.data.Recordset.Load は変換されませんでした。

com.ms.wfc.data.Recordset.MarshalInterface は変換されませんでした。

com.ms.wfc.data.Recordset.move は変換されませんでした。

com.ms.wfc.data.Recordset.nextRecordset は変換されませんでした。

com.ms.wfc.data.Recordset.open は変換されませんでした。

com.ms.wfc.data.Recordset.open(Object) は変換されませんでした。

com.ms.wfc.data.Recordset.Recordset(IDataSource) は変換できませんでした。

com.ms.wfc.data.Recordset.Recordset(IDataSource, String) は変換できませんでした。

com.ms.wfc.data.Recordset.Recordset(Object) は変換できませんでした。

com.ms.wfc.data.Recordset.release は変換されませんでした。

com.ms.wfc.data.Recordset.ReleaseMarshalData は変換されませんでした。

com.ms.wfc.data.Recordset.removeDataSourceListener は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnEndOfRecordset は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnFetchComplete は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnFetchProgress は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnFieldChangeComplete は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnMoveComplete は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnRecordChangeComplete は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnRecordsetChangeComplete は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnWillChangeField は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnWillChangeRecord は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnWillChangeRecordset は変換されませんでした。

com.ms.wfc.data.Recordset.removeOnWillMove は変換されませんでした。

com.ms.wfc.data.Recordset.setCommand は変換されませんでした。

com.ms.wfc.data.Recordset.UnmarshalInterface は変換されませんでした。

com.ms.wfc.data.RecordsetEvent は変換されませんでした。

com.ms.wfc.data.RecordsetEventHandler は変換されませんでした。

com.ms.wfc.data.StringManager.getString は変換されませんでした。

Com.ms.wfc.data.adodb のエラー メッセージ

com.ms.wfc.data.adodb.<ClassName>.clsid は変換されませんでした。

com.ms.wfc.data.adodb.<ClassName>.iid は変換されませんでした。

com.ms.wfc.data.adodb._Command.setActiveConnection は変換されませんでした。

com.ms.wfc.data.adodb._Connection.setConnectionString を変換できませんでした。

com.ms.wfc.data.adodb._Recordset.getCollect は変換されませんでした。

com.ms.wfc.data.adodb._Recordset.setCollect は変換されませんでした。

com.ms.wfc.data.adodb.Command.setActiveConnection は変換されませんでした。

com.ms.wfc.data.adodb.Connection.setConnectionString を変換できませんでした。

com.ms.wfc.data.adodb.Recordset.getCollect は変換されませんでした。

com.ms.wfc.data.adodb.Recordset.setCollect は変換されませんでした。

Com.ms.wfc.data.ui のエラー メッセージ

com.ms.wfc.data.ui.Column.getAllowSizing は変換されませんでした。

com.ms.wfc.data.ui.Column.getBackColor は変換されませんでした。

com.ms.wfc.data.ui.Column.getDataFormat は変換されませんでした。

com.ms.wfc.data.ui.Column.getDataType は変換されませんでした。

com.ms.wfc.data.ui.Column.getFont は変換されませんでした。

com.ms.wfc.data.ui.Column.getForeColor は変換されませんでした。

com.ms.wfc.data.ui.Column.getSelectedBackColor は変換されませんでした。

com.ms.wfc.data.ui.Column.getSelectedForeColor は変換されませんでした。

com.ms.wfc.data.ui.Column.getVisible は変換されませんでした。

com.ms.wfc.data.ui.Column.getWriteAllowed は変換されませんでした。

com.ms.wfc.data.ui.Column.setAlignment は変換されませんでした。

com.ms.wfc.data.ui.Column.setAllowSizing は変換されませんでした。

com.ms.wfc.data.ui.Column.setBackColor は変換されませんでした。

com.ms.wfc.data.ui.Column.setDataFormat は変換されませんでした。

com.ms.wfc.data.ui.Column.setFont は変換されませんでした。

com.ms.wfc.data.ui.Column.setForeColor は変換されませんでした。

com.ms.wfc.data.ui.Column.setIndex は変換されませんでした。

com.ms.wfc.data.ui.Column.setSelectedBackColor は変換されませんでした。

com.ms.wfc.data.ui.Column.setSelectedForeColor は変換されませんでした。

com.ms.wfc.data.ui.Column.setValue は変換されませんでした。

com.ms.wfc.data.ui.Column.setVisible は変換されませんでした。

com.ms.wfc.data.ui.Column.shouldPersistBackColor は変換されませんでした。

com.ms.wfc.data.ui.Column.shouldPersistFont は変換されませんでした。

com.ms.wfc.data.ui.Column.shouldPersistForeColor は変換されませんでした。

com.ms.wfc.data.ui.Column.shouldPersistSelectedBackColor は変換されませんでした。

com.ms.wfc.data.ui.Column.shouldPersistSelectedForeColor は変換されませんでした。

com.ms.wfc.data.ui.ColumnEditingEvent は変換されませんでした。

com.ms.wfc.data.ui.ColumnEditingEventHandler は変換されませんでした。

com.ms.wfc.data.ui.ColumnEvent は変換されませんでした。

com.ms.wfc.data.ui.ColumnEventHandler は変換されませんでした。

com.ms.wfc.data.ui.ColumnResizeEvent は変換されませんでした。

com.ms.wfc.data.ui.ColumnResizeEventHandler は変換されませんでした。

com.ms.wfc.data.ui.ColumnsEditor は変換されませんでした。

com.ms.wfc.data.ui.ColumnsEditorDialog は変換されませんでした。

com.ms.wfc.data.ui.ColumnUpdatingEvent は変換されませんでした。

com.ms.wfc.data.ui.ColumnUpdatingEventHandler は変換されませんでした。

com.ms.wfc.data.ui.ConnectionStringEditor は変換されませんでした。

com.ms.wfc.data.ui.DataBinder は変換されませんでした。

com.ms.wfc.data.ui.DataBinder.Customizer は変換されませんでした。

com.ms.wfc.data.ui.DataBinder.GeneralPage は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.bindTarget は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.CallbackType は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.commitChange は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.DataBinding は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.FalseArgument は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.getConstructorArgs は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.getDataFormat は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.getFieldName は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.getPropertyValue は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.getTarget は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.m_propertyChange は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.onChanged は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.onChanging は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.onTargetDispose は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.propertyChange は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.refreshPropertyValue は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.setDataFormat は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.setFieldName は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.setPropertyValue は変換されませんでした。

com.ms.wfc.data.ui.DataBinding.setTarget は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addColumn は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnColumnEdited は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnColumnEditing は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnColumnResize は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnColumnUpdated は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnColumnUpdating は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnDeleted は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnDeleting は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnError は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnHeaderClick は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnInserted は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnInserting は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnPositionChange は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnRowResize は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnScroll は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnSelChange は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnUpdated は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.addOnUpdating は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.Customizer は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.DataGrid は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.DataGridErrorDialog は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.fireErrorEvent は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getAllowAddNew は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getAllowArrows は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getAllowDelete は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getAllowRowSizing は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getAllowUpdate は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getCurrentRow は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getCurrentRowModified は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getDisplayIndex は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getDynamicColumns は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getEnterAction は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getFirstRow は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getHeaderLineCount は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getRowHeight は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getScrollbars は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getSelectedColumns は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getSelectedRows は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getShowPhantom は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getTabAction は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.getWrapCellPointer は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.makeCurrentCellVisible は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.MIN_COLUMN_WIDTH は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.MIN_ROW_HEIGHT は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onColumnEdited は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onColumnEditing は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onColumnResize は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onColumnUpdated は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onColumnUpdating は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onDeleted は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onDeleting は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onError は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onHeaderClick は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onInserted は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onInserting は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onPositionChange は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onRowResize は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onScroll は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onSelChange は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onUpdated は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.onUpdating は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.rebind は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnColumnEdited は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnColumnEditing は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnColumnResize は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnColumnUpdated は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnColumnUpdating は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnDeleted は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnDeleting は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnError は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnHeaderClick は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnInserted は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnInserting は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnPositionChange は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnRowResize は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnScroll は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnSelChange は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnUpdated は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.removeOnUpdating は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.rowBookmark は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.rowTop は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.scroll は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setAllowAddNew は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setAllowArrows は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setAllowDelete は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setAllowRowSizing は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setAllowUpdate は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setCurrentRow は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setDynamicColumns は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setEnterAction は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setFirstRow は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setHeaderLineCount は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setLeftColumn は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setRowHeight は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setScrollbars は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setSelectedColumns は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setSelectedRows は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setTabAction は変換されませんでした。

com.ms.wfc.data.ui.DataGrid.setWrapCellPointer は変換されませんでした。

com.ms.wfc.data.ui.DataMemberEditor は変換されませんでした。

com.ms.wfc.data.ui.DataNavigator.Customizer は変換されませんでした。

com.ms.wfc.data.ui.DataNavigator.getDataMember は変換されませんでした。

com.ms.wfc.data.ui.DataNavigator.moveFirst は変換されませんでした。

com.ms.wfc.data.ui.DataNavigator.moveLast は変換されませんでした。

com.ms.wfc.data.ui.DataNavigator.moveNext は変換されませんでした。

com.ms.wfc.data.ui.DataNavigator.movePrevious は変換されませんでした。

com.ms.wfc.data.ui.DataNavigator.propertyChanged は変換されませんでした。

com.ms.wfc.data.ui.DataNavigator.setDataMember は変換されませんでした。

com.ms.wfc.data.ui.DataNavigator.setDataSource は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addDataSourceListener は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnBeginTransComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnCommitTransComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnConnectComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnDisconnect は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnEndOfRecordset は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnExecuteComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnFetchComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnFetchProgress は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnFieldChangeComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnInfoMessage は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnMoveComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnRecordChangeComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnRecordsetChangeComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnRollbackTransComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnWillChangeField は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnWillChangeRecord は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnWillChangeRecordset は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnWillConnect は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnWillExecute は変換されませんでした。

com.ms.wfc.data.ui.DataSource.addOnWillMove は変換されませんでした。

com.ms.wfc.data.ui.DataSource.dataMemberAdded は変換されませんでした。

com.ms.wfc.data.ui.DataSource.dataMemberChanged は変換されませんでした。

com.ms.wfc.data.ui.DataSource.dataMemberRemoved は変換されませんでした。

com.ms.wfc.data.ui.DataSource.DataSource は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getAsyncConnect は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getAsyncExecute は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getAsyncFetch は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getCacheSize は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getCursorLocation は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getCursorType は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getDataMember は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getDataMemberCount は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getDataMemberName は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getDesignTimeData は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getIsolationLevel は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getLockType は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getMaxRecords は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getMode は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getParentDataSource は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getParentFieldName は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getPassword は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getPrepared は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getRecordset は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getSort は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getStayInSync は変換されませんでした。

com.ms.wfc.data.ui.DataSource.getUserId は変換されませんでした。

com.ms.wfc.data.ui.DataSource.isChildDataSource は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeDataSourceListener は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnBeginTransComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnCommitTransComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnConnectComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnDisconnect は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnEndOfRecordset は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnExecuteComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnFetchComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnFetchProgress は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnFieldChangeComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnInfoMessage は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnMoveComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnRecordChangeComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnRecordsetChangeComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnRollbackTransComplete は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnWillChangeField は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnWillChangeRecord は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnWillChangeRecordset は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnWillConnect は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnWillExecute は変換されませんでした。

com.ms.wfc.data.ui.DataSource.removeOnWillMove は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setAsyncConnect は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setAsyncExecute は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setAsyncFetch は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setCacheSize は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setConnectionString は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setConnectionTimeout は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setCursorLocation は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setCursorType は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setDesignTimeData は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setFilter は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setIsolationLevel は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setLockType は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setMaxRecords は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setMode は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setParentDataSource は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setParentFieldName は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setPassword は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setPrepared は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setSort は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setStayInSync は変換されませんでした。

com.ms.wfc.data.ui.DataSource.setUserId は変換されませんでした。

com.ms.wfc.data.ui.DataSource.unrealize は変換されませんでした。

com.ms.wfc.data.ui.EnterAction は変換されませんでした。

com.ms.wfc.data.ui.ErrorEvent は変換されませんでした。

com.ms.wfc.data.ui.ErrorEventHandler は変換されませんでした。

com.ms.wfc.data.ui.FieldNameEditor は変換されませんでした。

com.ms.wfc.data.ui.GridLineStyle.GridLineStyle は変換されませんでした。

com.ms.wfc.data.ui.GridLineStyle.RAISED3D は変換されませんでした。

com.ms.wfc.data.ui.GridLineStyle.SUNKEN3D は変換されませんでした。

com.ms.wfc.data.ui.PositionChangeEvent.lastColumn は変換されませんでした。

com.ms.wfc.data.ui.PositionChangeEvent.lastRow は変換されませんでした。

com.ms.wfc.data.ui.PositionChangeEvent.PositionChangeEvent は変換されませんでした。

com.ms.wfc.data.ui.PositionChangeEventHandler.PositionChangeEventHandler は変換されませんでした。

com.ms.wfc.data.ui.PropertyNameEditor は変換されませんでした。

com.ms.wfc.data.ui.RecordsetProvider は変換されませんでした。

com.ms.wfc.data.ui.TabAction は変換されませんでした。

Com.ms.wfc.io のエラー メッセージ

com.ms.wfc.io.BufferedStream.getStream は変換されませんでした。

com.ms.wfc.io.BufferedStream.readCore は変換されませんでした。

com.ms.wfc.io.BufferedStream.readStringCharsAnsi は変換されませんでした。

com.ms.wfc.io.BufferedStream.readStringNull は変換されませんでした。

com.ms.wfc.io.BufferedStream.readStringNullAnsi は変換されませんでした。

com.ms.wfc.io.BufferedStream.writeCore は変換されませんでした。

com.ms.wfc.io.CodePage.ANSI は変換されませんでした。

com.ms.wfc.io.CodePage.CodePage は変換されませんでした。

com.ms.wfc.io.CodePage.MAC は変換されませんでした。

com.ms.wfc.io.CodePage.OEM は変換されませんでした。

com.ms.wfc.io.DataStream.comStream は変換されませんでした。

com.ms.wfc.io.DataStream.DataStream は変換されませんでした。

com.ms.wfc.io.DataStream.fromComStream は変換されませんでした。

com.ms.wfc.io.DataStream.getComStream は変換されませんでした。

com.ms.wfc.io.DataStream.readCore は変換されませんでした。

com.ms.wfc.io.DataStream.readStringNull は変換されませんでした。

com.ms.wfc.io.DataStream.readStringNullAnsi は変換されませんでした。

com.ms.wfc.io.DataStream.readUTF は変換されませんでした。

com.ms.wfc.io.DataStream.toComStream は変換されませんでした。

com.ms.wfc.io.DataStream.writeCore は変換されませんでした。

com.ms.wfc.io.DataStream.writeStringChars は変換されませんでした。

com.ms.wfc.io.DataStream.writeStringCharsAnsi は変換されませんでした。

com.ms.wfc.io.DataStream.writeStringNull は変換されませんでした。

com.ms.wfc.io.DataStream.writeStringNullAnsi は変換されませんでした。

com.ms.wfc.io.DataStream.writeUTF は変換されませんでした。

com.ms.wfc.io.DataStreamFromComStream は変換されませんでした。

com.ms.wfc.io.File.createDirectory は変換されませんでした。

com.ms.wfc.io.File.File は変換されませんでした。

com.ms.wfc.io.File.getDirectory は変換されませんでした。

com.ms.wfc.io.File.getFiles は変換されませんでした。

com.ms.wfc.io.File.getName は変換されませんでした。

com.ms.wfc.io.File.handle は変換されませんでした。

com.ms.wfc.io.File.openStandardError は変換されませんでした。

com.ms.wfc.io.File.openStandardInput は変換されませんでした。

com.ms.wfc.io.File.openStandardOutput は変換されませんでした。

com.ms.wfc.io.File.readCore は変換されませんでした。

com.ms.wfc.io.File.writeCore は変換されませんでした。

com.ms.wfc.io.FileEnumerator.close は変換されませんでした。

com.ms.wfc.io.FileEnumerator.FileEnumerator は変換されませんでした。

com.ms.wfc.io.FileEnumerator.finalize は変換されませんでした。

com.ms.wfc.io.FileEnumerator.getAttributes は変換されませんでした。

com.ms.wfc.io.IDataStream.getComStream は変換されませんでした。

com.ms.wfc.io.FileEnumerator.getSize は変換されませんでした。

com.ms.wfc.io.IDataStream.readStringNull は変換されませんでした。

com.ms.wfc.io.IDataStream.readStringNullAnsi は変換されませんでした。

com.ms.wfc.io.IDataStream.readUTF は変換されませんでした。

com.ms.wfc.io.IDataStream.writeStringChars は変換されませんでした。

com.ms.wfc.io.IDataStream.writeStringCharsAnsi は変換されませんでした。

com.ms.wfc.io.IDataStream.writeStringNull は変換されませんでした。

com.ms.wfc.io.IDataStream.writeStringNullAnsi は変換されませんでした。

com.ms.wfc.io.IDataStream.writeUTF は変換されませんでした。

com.ms.wfc.io.IDataStreamProvider は変換されませんでした。

com.ms.wfc.io.MemoryStream.MemoryStream は変換されませんでした。

com.ms.wfc.io.MemoryStream.readCore は変換されませんでした。

com.ms.wfc.io.MemoryStream.readStringCharsAnsi は変換されませんでした。

com.ms.wfc.io.MemoryStream.readStringNull は変換されませんでした。

com.ms.wfc.io.MemoryStream.readStringNullAnsi は変換されませんでした。

com.ms.wfc.io.MemoryStream.writeCore は変換されませんでした。

com.ms.wfc.io.MemoryStream.writeTo は変換されませんでした。

com.ms.wfc.io.Reader.Reader は変換されませんでした。

com.ms.wfc.io.TextReader.TextReader(ByteStream) は変換されませんでした。

com.ms.wfc.io.TextReader.TextReader(ByteStream, int) は変換されませんでした。

com.ms.wfc.io.TextReader.TextReader(ByteStream, int, int) は変換されませんでした。

com.ms.wfc.io.WinIOException.getErrorCode は変換されませんでした。

com.ms.wfc.io.WinIOException.WinIOException は変換されませんでした。

com.ms.wfc.io.Writer.Writer は変換されませんでした。

Com.ms.wfc.ole32 のエラー メッセージ

com.ms.wfc.ole32.DBBINDING.pTypeInfo は変換されませんでした。

com.ms.wfc.ole32.DBCOLUMNINFO.pTypeInfo は変換されませんでした。

com.ms.wfc.ole32.IEnumFORMATETC.iid は変換されませんでした。

com.ms.wfc.ole32.IEnumSTATDATA.iid は変換されませんでした。

com.ms.wfc.ole32.ILockBytes.iid は変換されませんでした。

com.ms.wfc.ole32.IMalloc.iid は変換されませんでした。

com.ms.wfc.ole32.IOleDataFormat.iid は変換されませんでした。

com.ms.wfc.ole32.IOleDataObject.iid は変換されませんでした。

com.ms.wfc.ole32.IOleDataObjectWithIStream.iid は変換されませんでした。

com.ms.wfc.ole32.IOleDropSource.iid は変換されませんでした。

com.ms.wfc.ole32.IOleDropTarget.iid は変換されませんでした。

com.ms.wfc.ole32.IPersist.iid は変換されませんでした。

com.ms.wfc.ole32.IPersistStream.iid は変換されませんでした。

com.ms.wfc.ole32.IPicture.iid は変換されませんでした。

com.ms.wfc.ole32.IStorage.iid は変換されませんでした。

Com.ms.wfc.ui のエラー メッセージ

com.ms.wfc.ui.AnchorEditor.editValue は変換されませんでした。

com.ms.wfc.ui.AnchorEditor.getConstantName は変換されませんでした。

com.ms.wfc.ui.AnchorEditor.getTextFromValue は変換されませんでした。

com.ms.wfc.ui.Animation を変換できませんでした。

com.ms.wfc.ui.AnimationFileNameEditor は変換されませんでした。

com.ms.wfc.ui.AutoSizeEvent は変換されませんでした。

com.ms.wfc.ui.AutoSizeEventHandler は変換されませんでした。

com.ms.wfc.ui.AxHost.AboutBoxDelegate は変換されませんでした。

com.ms.wfc.ui.AxHost.AxPropertyInfo は変換されませんでした。

com.ms.wfc.ui.AxHost.begin を変換できませんでした。

com.ms.wfc.ui.AxHost.DispidAttribute は変換されませんでした。

com.ms.wfc.ui.AxHost.fireOnClick は変換されませんでした。

com.ms.wfc.ui.AxHost.fireOnDblClick は変換されませんでした。

com.ms.wfc.ui.AxHost.fireOnKeyDown は変換されませんでした。

com.ms.wfc.ui.AxHost.fireOnKeyPress は変換されませんでした。

com.ms.wfc.ui.AxHost.fireOnKeyUp は変換されませんでした。

com.ms.wfc.ui.AxHost.fireOnMouseDown は変換されませんでした。

com.ms.wfc.ui.AxHost.fireOnMouseMove は変換されませんでした。

com.ms.wfc.ui.AxHost.fireOnMouseUp は変換されませんでした。

com.ms.wfc.ui.AxHost.Flags は変換されませんでした。

com.ms.wfc.ui.AxHost.getAmbientProperty は変換されませんでした。

com.ms.wfc.ui.AxHost.getClientAttributes は変換されませんでした。

com.ms.wfc.ui.AxHost.getColorFromOleColor は変換されませんでした。

com.ms.wfc.ui.AxHost.getFontFromIFont は変換されませんでした。

com.ms.wfc.ui.AxHost.getFontFromIFontDisp は変換されませんでした。

com.ms.wfc.ui.AxHost.getIFontDispFromFont は変換されませんでした。

com.ms.wfc.ui.AxHost.getIFontFromFont は変換されませんでした。

com.ms.wfc.ui.AxHost.getIPictureDispFromPicture は変換されませんでした。

com.ms.wfc.ui.AxHost.getIPictureFromPicture は変換されませんでした。

com.ms.wfc.ui.AxHost.getOleColorFromColor は変換されませんでした。

com.ms.wfc.ui.AxHost.getPictureFromIPicture は変換されませんでした。

com.ms.wfc.ui.AxHost.getPictureFromIPictureDisp は変換されませんでした。

com.ms.wfc.ui.AxHost.propertyChanged は変換されませんでした。

com.ms.wfc.ui.AxHost.setAboutBoxDelegate は変換されませんでした。

com.ms.wfc.ui.AxHost.setTopLevel は変換されませんでした。

com.ms.wfc.ui.AxHost.shouldPersistContainingForm は変換されませんでした。

com.ms.wfc.ui.AxHost.State.save は変換されませんでした。

com.ms.wfc.ui.AxHost.State.State は変換されませんでした。

com.ms.wfc.ui.AxPersist は変換されませんでした。

com.ms.wfc.ui.Bitmap.Bitmap は変換されませんでした。

com.ms.wfc.ui.Bitmap.Bitmap(Bitmap, Bitmap) は変換できませんでした。

com.ms.wfc.ui.Bitmap.Bitmap(int) は変換できませんでした。

com.ms.wfc.ui.Bitmap.Bitmap(int, int, int, int, short[]) は変換できませんでした。

com.ms.wfc.ui.Bitmap.Bitmap(IPicture) は変換できませんでした。

com.ms.wfc.ui.Bitmap.Bitmap(Palette) は変換できませんでした。

com.ms.wfc.ui.Bitmap.copyHandle は変換されませんでした。

com.ms.wfc.ui.Bitmap.destroyHandle は変換されませんでした。

com.ms.wfc.ui.Bitmap.drawStretchTo は変換されませんでした。

com.ms.wfc.ui.Bitmap.drawTo は変換されませんでした。

com.ms.wfc.ui.Bitmap.getColorMask は変換されませんでした。

com.ms.wfc.ui.Bitmap.getGraphics は変換されませんでした。

com.ms.wfc.ui.Bitmap.getHandle は変換されませんでした。

com.ms.wfc.ui.Bitmap.getMonochromeMask は変換されませんでした。

com.ms.wfc.ui.Bitmap.getPICTDESC は変換されませんでした。

com.ms.wfc.ui.Bitmap.getTransparent は変換されませんでした。

com.ms.wfc.ui.Bitmap.getTransparentColor は変換されませんでした。

com.ms.wfc.ui.Bitmap.initialize は変換されませんでした。

com.ms.wfc.ui.Brush.Brush(Bitmap) は変換できませんでした。

com.ms.wfc.ui.Brush.Brush(int) は変換できませんでした。

com.ms.wfc.ui.Brush.copyHandle は変換されませんでした。

com.ms.wfc.ui.Brush.destroyHandle は変換されませんでした。

com.ms.wfc.ui.Brush.getHandle は変換されませんでした。

com.ms.wfc.ui.Brush.HALFTONE は変換されませんでした。

com.ms.wfc.ui.BrushStyle.HOLLOW は変換されませんでした。

com.ms.wfc.ui.BrushStyle.PATTERN は変換されませんでした。

com.ms.wfc.ui.Button.propertyChanged は変換されませんでした。

com.ms.wfc.ui.CheckBox.getGroupValue は変換されませんでした。

com.ms.wfc.ui.CheckBox.setGroupValue は変換されませんでした。

com.ms.wfc.ui.CheckBox.setTextAlign は変換されませんでした。

com.ms.wfc.ui.CheckedListBox.propertyChanged は変換されませんでした。

com.ms.wfc.ui.Color.Color を変換できませんでした。

com.ms.wfc.ui.Color.Editor.ColorPicker を変換できませんでした。

com.ms.wfc.ui.Color.Editor.ColorPicker.Palette を変換できませんでした。

com.ms.wfc.ui.Color.Editor.ColorPicker.Palette.CustomColorDialog を変換できませんでした。

com.ms.wfc.ui.Color.fromCMYK は変換されませんでした。

com.ms.wfc.ui.Color.getConstructorArgs は変換されませんでした。

com.ms.wfc.ui.Color.toString は変換されませんでした。

com.ms.wfc.ui.ColumnClickEventHandler.ColumnClickEventHandler を変換できませんでした。

com.ms.wfc.ui.ComboBox.propertyChanged は変換されませんでした。

com.ms.wfc.ui.ContentAlignment.toTextFormat は変換されませんでした。

com.ms.wfc.ui.Control.getTopLevel を変換できませんでした。

com.ms.wfc.ui.Control.getVisible は変換されませんでした。

com.ms.wfc.ui.Control.invokeAsync は変換されませんでした。

com.ms.wfc.ui.Control.PROP_BACKCOLOR は変換されませんでした。

com.ms.wfc.ui.Control.PROP_CURSOR は変換されませんでした。

com.ms.wfc.ui.Control.PROP_ENABLED は変換されませんでした。

com.ms.wfc.ui.Control.PROP_FONT は変換されませんでした。

com.ms.wfc.ui.Control.PROP_FORECOLOR は変換されませんでした。

com.ms.wfc.ui.Control.PROP_LOCATION は変換されませんでした。

com.ms.wfc.ui.Control.PROP_PARENTBACKCOLOR は変換されませんでした。

com.ms.wfc.ui.Control.PROP_PARENTFONT は変換されませんでした。

com.ms.wfc.ui.Control.PROP_PARENTFORECOLOR は変換されませんでした。

com.ms.wfc.ui.Control.PROP_SIZE は変換されませんでした。

com.ms.wfc.ui.Control.PROP_TEXT は変換されませんでした。

com.ms.wfc.ui.Control.PROP_VISIBLE は変換されませんでした。

com.ms.wfc.ui.Control.propertyChanged は変換されませんでした。

com.ms.wfc.ui.Control.sendMessage は変換されませんでした。

com.ms.wfc.ui.Control.setTabIndex を変換できませんでした。

com.ms.wfc.ui.Control.setTopLevel を変換できませんでした。

com.ms.wfc.ui.Control.STYLE_ACCEPTSCHILDREN は変換されませんでした。

com.ms.wfc.ui.CoordinateSystem は変換されませんでした。

com.ms.wfc.ui.Cursor.Cursor は変換されませんでした。

com.ms.wfc.ui.Cursor.destroyHandle は変換されませんでした。

com.ms.wfc.ui.Cursor.getPICTDESC は変換されませんでした。

com.ms.wfc.ui.Cursor.initialize は変換されませんでした。

com.ms.wfc.ui.DateBoldEventHandler.DateBoldEventHandler を変換できませんでした。

com.ms.wfc.ui.DateRangeEventHandler.DateRangeEventHandler を変換できませんでした。

com.ms.wfc.ui.DateTimeFormatEvent は変換されませんでした。

com.ms.wfc.ui.DateTimeFormatEventHandler は変換されませんでした。

com.ms.wfc.ui.DateTimeFormatQueryEvent は変換されませんでした。

com.ms.wfc.ui.DateTimeFormatQueryEventHandler は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.addOnFormat は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.addOnFormatQuery は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.addOnUserString は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.getAllowUserString は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.getMaxDate を変換できませんでした。

com.ms.wfc.ui.DateTimePicker.getMinDate を変換できませんでした。

com.ms.wfc.ui.DateTimePicker.onFormat は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.onFormatQuery は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.onUserString は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.removeOnFormat は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.removeOnFormatQuery は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.removeOnUserString は変換されませんでした。

com.ms.wfc.ui.DateTimePicker.setAllowUserString は変換されませんでした。

com.ms.wfc.ui.DateTimeUserStringEvent は変換されませんでした。

com.ms.wfc.ui.DateTimeUserStringEventHandler は変換されませんでした。

com.ms.wfc.ui.DateTimeWmKeyDownEvent.DateTimeWmKeyDownEvent は変換されませんでした。

com.ms.wfc.ui.DateTimeWmKeyDownEvent.format は変換されませんでした。

com.ms.wfc.ui.DateTimeWmKeyDownEvent.time は変換されませんでした。

com.ms.wfc.ui.DateTimeWmKeyDownEventHandler.DateTimeWmKeyDownEventHandler を変換できませんでした。

com.ms.wfc.ui.Dimensions.Editor を変換できませんでした。

com.ms.wfc.ui.Dimensions.getConstructorArgs は変換されませんでした。

com.ms.wfc.ui.Dimensions.save は変換されませんでした。

com.ms.wfc.ui.DockEditor.editValue は変換されませんでした。

com.ms.wfc.ui.DockEditor.getConstantName は変換されませんでした。

com.ms.wfc.ui.DockEditor.getTextFromValue は変換されませんでした。

com.ms.wfc.ui.DocumentReadyEvent は変換されませんでした。

com.ms.wfc.ui.DocumentReadyEventHandler は変換されませんでした。

com.ms.wfc.ui.DocumentReadyEventHandler.DocumentReadyEventHandler を変換できませんでした。

com.ms.wfc.ui.DragEventHandler.DragEventHandler を変換できませんでした。

com.ms.wfc.ui.DrawItemEventHandler.DrawItemEventHandler を変換できませんでした。

com.ms.wfc.ui.Edit.propertyChanged は変換されませんでした。

com.ms.wfc.ui.Edit.setTextAlign を変換できませんでした。

com.ms.wfc.ui.Edit.undo は変換されませんでした。

com.ms.wfc.ui.Editor.createExtensionsString を変換できませんでした。

com.ms.wfc.ui.Editor.createFilterEntry を変換できませんでした。

com.ms.wfc.ui.Editor.getExtensions を変換できませんでした。

com.ms.wfc.ui.Editor.getFileDialogDescription を変換できませんでした。

com.ms.wfc.ui.Editor.loadFromStream を変換できませんでした。

com.ms.wfc.ui.EraseBackgroundEvent は変換されませんでした。

com.ms.wfc.ui.FileDialog.promptFileCreate は変換されませんでした。

com.ms.wfc.ui.FileDialog.promptFileNotFound は変換されませんでした。

com.ms.wfc.ui.FileDialog.promptFileOverwrite は変換されませんでした。

com.ms.wfc.ui.FloodFillType は変換されませんでした。

com.ms.wfc.ui.Font.ANSI_VAR は変換されませんでした。

com.ms.wfc.ui.Font.destroyHandle は変換されませんでした。

com.ms.wfc.ui.Font.DEVICE_DEFAULT は変換されませんでした。

com.ms.wfc.ui.Font.equalsBase は変換されませんでした。

com.ms.wfc.ui.Font.Font は変換されませんでした。

com.ms.wfc.ui.Font.getCharacterSet は変換されませんでした。

com.ms.wfc.ui.Font.getConstructorArgs は変換されませんでした。

com.ms.wfc.ui.Font.getFamily は変換されませんでした。

com.ms.wfc.ui.Font.getFontMetrics は変換されませんでした。

com.ms.wfc.ui.Font.getOrientation は変換されませんでした。

com.ms.wfc.ui.Font.getPitch は変換されませんでした。

com.ms.wfc.ui.Font.getStock は変換されませんでした。

com.ms.wfc.ui.Font.OEM_FIXED は変換されませんでした。

com.ms.wfc.ui.Font.save は変換されませんでした。

com.ms.wfc.ui.Font.SYSTEM は変換されませんでした。

com.ms.wfc.ui.Font.SYSTEM_FIXED は変換されませんでした。

com.ms.wfc.ui.FontDescriptor は変換されませんでした。

com.ms.wfc.ui.FontDevice は変換されませんでした。

com.ms.wfc.ui.FontDialog.getFontDevice は変換されませんでした。

com.ms.wfc.ui.FontDialog.getPrinterDC は変換されませんでした。

com.ms.wfc.ui.FontDialog.getScalableOnly は変換されませんでした。

com.ms.wfc.ui.FontDialog.getTrueTypeOnly は変換されませんでした。

com.ms.wfc.ui.FontDialog.getWysiwyg は変換されませんでした。

com.ms.wfc.ui.FontDialog.hookProc は変換されませんでした。

com.ms.wfc.ui.FontDialog.setFontDevice は変換されませんでした。

com.ms.wfc.ui.FontDialog.setPrinterDC は変換されませんでした。

com.ms.wfc.ui.FontDialog.setScalableOnly は変換されませんでした。

com.ms.wfc.ui.FontDialog.setTrueTypeOnly は変換されませんでした。

com.ms.wfc.ui.FontDialog.setWysiwyg は変換されませんでした。

com.ms.wfc.ui.FontFamily は変換されませんでした。

com.ms.wfc.ui.FontMetrics は変換されませんでした。

com.ms.wfc.ui.FontMetrics.ascent を変換できませんでした。

com.ms.wfc.ui.FontMetrics.charSet を変換できませんでした。

com.ms.wfc.ui.FontMetrics.descent を変換できませんでした。

com.ms.wfc.ui.FontMetrics.FontMetrics を変換できませんでした。

com.ms.wfc.ui.FontMetrics.height を変換できませんでした。

com.ms.wfc.ui.FontPitch は変換されませんでした。

com.ms.wfc.ui.FontSize.CELLHEIGHT は変換されませんでした。

com.ms.wfc.ui.FontSize.CENTIMETERS は変換されませんでした。

com.ms.wfc.ui.FontSize.CHARACTERHEIGHT は変換されませんでした。

com.ms.wfc.ui.FontSize.EM は変換されませんでした。

com.ms.wfc.ui.FontSize.EX は変換されませんでした。

com.ms.wfc.ui.FontType は変換されませんでした。

com.ms.wfc.ui.FontWeight.EXTRALIGHT は変換されませんでした。

com.ms.wfc.ui.FontWeight.LIGHT は変換されませんでした。

com.ms.wfc.ui.FontWeight.THIN は変換されませんでした。

com.ms.wfc.ui.Form.checkCloseDialog を変換できませんでした。

com.ms.wfc.ui.Form.FORMSTATE_AUTOSCALING は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_AUTOSCROLLING は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_BORDERSTYLE は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_CONTROLBOX は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_HELPBUTTON は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_HSCROLLVISIBLE は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_KEYPREVIEW は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_MAXIMIZEBOX は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_MINIMIZEBOX は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_PALETTEMODE は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_SETCLIENTSIZE は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_SHOWWINDOWONCREATE は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_STARTPOS は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_TASKBAR は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_TOPMOST は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_USERHASSCROLLED は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_VSCROLLVISIBLE は変換されませんでした。

com.ms.wfc.ui.Form.FORMSTATE_WINDOWSTATE は変換されませんでした。

com.ms.wfc.ui.Form.getFormState は変換されませんでした。

com.ms.wfc.ui.Form.getPalette は変換されませんでした。

com.ms.wfc.ui.Form.getPaletteMode は変換されませんでした。

com.ms.wfc.ui.Form.getPaletteSource は変換されませんでした。

com.ms.wfc.ui.Form.hasFormState は変換されませんでした。

com.ms.wfc.ui.Form.notifyPaletteChange は変換されませんでした。

com.ms.wfc.ui.Form.onMDIChildActivate は変換されませんでした。

com.ms.wfc.ui.Form.onNewPalette は変換されませんでした。

com.ms.wfc.ui.Form.propertyChanged は変換されませんでした。

com.ms.wfc.ui.Form.setAutoScaleBaseSize を変換できませんでした。

com.ms.wfc.ui.Form.setBorderStyle は変換されませんでした。

com.ms.wfc.ui.Form.setControlBox を変換できませんでした。

com.ms.wfc.ui.Form.setFormState は変換されませんでした。

com.ms.wfc.ui.Form.setMaximizeBox を変換できませんでした。

com.ms.wfc.ui.Form.setMinimizeBox を変換できませんでした。

com.ms.wfc.ui.Form.setNewControls を変換できませんでした。

com.ms.wfc.ui.Form.setPaletteMode は変換されませんでした。

com.ms.wfc.ui.Form.setPaletteSource は変換されませんでした。

com.ms.wfc.ui.Form.setShowInTaskbar を変換できませんでした。

com.ms.wfc.ui.Form.setStartPosition を変換できませんでした。

com.ms.wfc.ui.FormPaletteMode は変換されませんでした。

com.ms.wfc.ui.GiveFeedbackEventHandler.GiveFeedbackEventHandler を変換できませんでした。

com.ms.wfc.ui.Graphics.drawArc は変換されませんでした。

com.ms.wfc.ui.Graphics.drawChord は変換されませんでした。

com.ms.wfc.ui.Graphics.drawPie は変換されませんでした。

com.ms.wfc.ui.Graphics.drawString を変換できませんでした。

com.ms.wfc.ui.Graphics.floodFill は変換されませんでした。

com.ms.wfc.ui.Graphics.getCoordinateSystem は変換されませんでした。

com.ms.wfc.ui.Graphics.getDeviceOrigin は変換されませんでした。

com.ms.wfc.ui.Graphics.getDevicePoint は変換されませんでした。

com.ms.wfc.ui.Graphics.getDeviceScale は変換されませんでした。

com.ms.wfc.ui.Graphics.getFontDescriptors は変換されませんでした。

com.ms.wfc.ui.Graphics.getHandle は変換されませんでした。

com.ms.wfc.ui.Graphics.getLogicalPoint は変換されませんでした。

com.ms.wfc.ui.Graphics.getLogicalSizeX は変換されませんでした。

com.ms.wfc.ui.Graphics.getLogicalSizeY は変換されませんでした。

com.ms.wfc.ui.Graphics.getOpaque は変換されませんでした。

com.ms.wfc.ui.Graphics.getPageOrigin は変換されませんでした。

com.ms.wfc.ui.Graphics.getPageScale は変換されませんでした。

com.ms.wfc.ui.Graphics.getPhysicalSizeX は変換されませんでした。

com.ms.wfc.ui.Graphics.getPhysicalSizeY は変換されませんでした。

com.ms.wfc.ui.Graphics.getPixel は変換されませんでした。

com.ms.wfc.ui.Graphics.getTextSize は変換されませんでした。

com.ms.wfc.ui.Graphics.getTextSpace は変換されませんでした。

com.ms.wfc.ui.Graphics.Graphics は変換されませんでした。

com.ms.wfc.ui.Graphics.invert は変換されませんでした。

com.ms.wfc.ui.Graphics.renderPalette は変換されませんでした。

com.ms.wfc.ui.Graphics.scroll は変換されませんでした。

com.ms.wfc.ui.Graphics.setCoordinateOrigin は変換されませんでした。

com.ms.wfc.ui.Graphics.setCoordinateScale は変換されませんでした。

com.ms.wfc.ui.Graphics.setCoordinateSystem は変換されませんでした。

com.ms.wfc.ui.Graphics.setHandle は変換されませんでした。

com.ms.wfc.ui.Graphics.setOpaque は変換されませんでした。

com.ms.wfc.ui.Graphics.setPixel は変換されませんでした。

com.ms.wfc.ui.Graphics.setTextSpace は変換されませんでした。

com.ms.wfc.ui.Help.Help は変換されませんでした。

com.ms.wfc.ui.HelpEvent.component は変換されませんでした。

com.ms.wfc.ui.HelpEvent.contextId は変換されませんでした。

com.ms.wfc.ui.HelpEvent.contextType は変換されませんでした。

com.ms.wfc.ui.HelpEvent.controlId は変換されませんでした。

com.ms.wfc.ui.HelpEvent.HelpEvent(int, int, IComonent, int, Point) を変換できませんでした。

com.ms.wfc.ui.HelpEvent.HelpEvent(Object, int, int, IComonent, Int, Point) を変換できませんでした。

com.ms.wfc.ui.HelpEventHandler.HelpEventHandler を変換できませんでした。

com.ms.wfc.ui.HelpFileFileNameEditor は変換されませんでした。

com.ms.wfc.ui.HelpProvider.shouldPersistShowHelp は変換されませんでした。

com.ms.wfc.ui.HTMLControl を変換できませんでした。

com.ms.wfc.ui.HTMLControl.add を変換できませんでした。

com.ms.wfc.ui.HTMLControl.addOnDocumentReady を変換できませんでした。

com.ms.wfc.ui.HTMLControl.getAmbientProperty を変換できませんでした。

com.ms.wfc.ui.HTMLControl.HTMLControl を変換できませんでした。

com.ms.wfc.ui.HTMLControl.onDocumentReady を変換できませんでした。

com.ms.wfc.ui.HTMLControl.propertyChanged を変換できませんでした。

com.ms.wfc.ui.HTMLControl.removeOnDocumentReady を変換できませんでした。

com.ms.wfc.ui.HTMLControl.setBoundElements を変換できませんでした。

com.ms.wfc.ui.HTMLControl.setNewHTMLElements を変換できませんでした。

com.ms.wfc.ui.HTMLControl.setURL を変換できませんでした。

com.ms.wfc.ui.IActiveXCustomPropertyDialog は変換されませんでした。

com.ms.wfc.ui.Icon.getPICTDESC は変換されませんでした。

com.ms.wfc.ui.Icon.Icon は変換されませんでした。

com.ms.wfc.ui.Icon.initialize は変換されませんでした。

com.ms.wfc.ui.Icon.loadPicture は変換されませんでした。

com.ms.wfc.ui.IHandleHook は変換されませんでした。

com.ms.wfc.ui.Image.copyHandle は変換されませんでした。

com.ms.wfc.ui.Image.dirty は変換されませんでした。

com.ms.wfc.ui.Image.drawStretchTo は変換されませんでした。

com.ms.wfc.ui.Image.drawTo は変換されませんでした。

com.ms.wfc.ui.Image.getExtension は変換されませんでした。

com.ms.wfc.ui.Image.getHandle は変換されませんでした。

com.ms.wfc.ui.Image.getPICTDESC は変換されませんでした。

com.ms.wfc.ui.Image.getPicture は変換されませんでした。

com.ms.wfc.ui.Image.Image は変換されませんでした。

com.ms.wfc.ui.Image.initialize は変換されませんでした。

com.ms.wfc.ui.Image.loadImage は変換されませんでした。

com.ms.wfc.ui.Image.loadPicture は変換されませんでした。

com.ms.wfc.ui.Image.PictureList は変換されませんでした。

com.ms.wfc.ui.ImageIndexEditor は変換されませんでした。

com.ms.wfc.ui.ImageList.createHandle は変換されませんでした。

com.ms.wfc.ui.ImageList.destroyHandle は変換されませんでした。

com.ms.wfc.ui.ImageList.getBackColor は変換されませんでした。

com.ms.wfc.ui.ImageList.getIcon は変換されませんでした。

com.ms.wfc.ui.ImageList.getMaskColor は変換されませんでした。

com.ms.wfc.ui.ImageList.getUseMask は変換されませんでした。

com.ms.wfc.ui.ImageList.ImageList は変換されませんでした。

com.ms.wfc.ui.ImageList.recreateHandle は変換されませんでした。

com.ms.wfc.ui.ImageList.SetBackColor は変換できませんでした。

com.ms.wfc.ui.ImageList.setImages は変換されませんでした。

com.ms.wfc.ui.ImageList.setMaskColor は変換されませんでした。

com.ms.wfc.ui.ImageList.setUseMask は変換されませんでした。

com.ms.wfc.ui.ImageListStreamer.getExtension は変換されませんでした。

com.ms.wfc.ui.ImageListStreamer.ImageListStreamer は変換されませんでした。

com.ms.wfc.ui.ImageListStreamer.save は変換されませんでした。

com.ms.wfc.ui.InputLangChangeEventHandler.InputLangChangeEventHandler を変換できませんでした。

com.ms.wfc.ui.InputLangChangeRequestEvent.InputLangChangeRequestEvent を変換できませんでした。

com.ms.wfc.ui.InputLangChangeRequestEventHandler.InputLangChangeRequestEventHandler を変換できませんでした。

com.ms.wfc.ui.ISelectionService.setSelectionStyle は変換されませんでした。

com.ms.wfc.ui.ItemCheckEventHandler.ItemCheckEventHandler を変換できませんでした。

com.ms.wfc.ui.ItemDragEventHandler.ItemDragEventHandler を変換できませんでした。

com.ms.wfc.ui.KeyEvent.KeyEvent を変換できませんでした。

com.ms.wfc.ui.KeyEventHandler.KeyEventHandler を変換できませんでした。

com.ms.wfc.ui.KeyPressEvent.KeyPressEvent を変換できませんでした。

com.ms.wfc.ui.KeyPressEventHandler.KeyPressEventHandler を変換できませんでした。

com.ms.wfc.ui.Label.propertyChanged は変換されませんでした。

com.ms.wfc.ui.Label.setTextAlign は変換されませんでした。

com.ms.wfc.ui.LabelEditEventHandler.LabelEditEventHandler を変換できませんでした。

com.ms.wfc.ui.LayoutEventHandler.LayoutEventHandler を変換できませんでした。

com.ms.wfc.ui.ListItem.getConstructorArgs は変換されませんでした。

com.ms.wfc.ui.ListItem.ListItem は変換されませんでした。

com.ms.wfc.ui.ListItem.ListItem(Stream) を変換できませんでした。

com.ms.wfc.ui.ListItem.ListItem(String, int, String[]) を変換できませんでした。

com.ms.wfc.ui.ListItem.ListItem(String, String[]) を変換できませんでした。

com.ms.wfc.ui.ListItem.save は変換されませんでした。

com.ms.wfc.ui.ListItem.setSubItem を変換できませんでした。

com.ms.wfc.ui.MDIWindowDialog は変換されませんでした。

com.ms.wfc.ui.MeasureItemEventHandler.MeasureItemEventHandler を変換できませんでした。

com.ms.wfc.ui.Menu.Menu は変換されませんでした。

com.ms.wfc.ui.Menu.mergeMenu を変換できませんでした。

com.ms.wfc.ui.MenuItem.setChecked は変換されませんでした。

com.ms.wfc.ui.Metafile.copyHandle は変換されませんでした。

com.ms.wfc.ui.Metafile.destroyHandle は変換されませんでした。

com.ms.wfc.ui.Metafile.drawStretchTo は変換されませんでした。

com.ms.wfc.ui.Metafile.drawTo は変換されませんでした。

com.ms.wfc.ui.Metafile.getHandle は変換されませんでした。

com.ms.wfc.ui.Metafile.getPICTDESC は変換されませんでした。

com.ms.wfc.ui.Metafile.getRenderedSize は変換されませんでした。

com.ms.wfc.ui.Metafile.initialize は変換されませんでした。

com.ms.wfc.ui.Metafile.Metafile は変換されませんでした。

com.ms.wfc.ui.MonthCalendar.propertyChanged は変換されませんでした。

com.ms.wfc.ui.MouseEvent.MouseEvent は変換されませんでした。

com.ms.wfc.ui.MouseEventHandler.MouseEventHandler を変換できませんでした。

com.ms.wfc.ui.NodeLabelEditEventHandler.NodeLabelEditEventHandler を変換できませんでした。

com.ms.wfc.ui.PaintEvent.graphics は変換されませんでした。

com.ms.wfc.ui.PaintEventHandler.PaintEventHandler を変換できませんでした。

com.ms.wfc.ui.Palette は変換されませんでした。

com.ms.wfc.ui.Palette.clone は変換されませんでした。

com.ms.wfc.ui.Palette.copyHandle は変換されませんでした。

com.ms.wfc.ui.Palette.destroyHandle は変換されませんでした。

com.ms.wfc.ui.Palette.dispose は変換されませんでした。

com.ms.wfc.ui.Palette.equals は変換されませんでした。

com.ms.wfc.ui.Palette.finalize は変換されませんでした。

com.ms.wfc.ui.Palette.getHalftonePalette は変換されませんでした。

com.ms.wfc.ui.Palette.getHandle は変換されませんでした。

com.ms.wfc.ui.Palette.getPaletteSupported は変換されませんでした。

com.ms.wfc.ui.Palette.Palette は変換されませんでした。

com.ms.wfc.ui.Pen.copyHandle は変換されませんでした。

com.ms.wfc.ui.Pen.destroyHandle は変換されませんでした。

com.ms.wfc.ui.Pen.getHandle は変換されませんでした。

com.ms.wfc.ui.Pen.Pen は変換されませんでした。

com.ms.wfc.ui.PenEntry は変換されませんでした。

com.ms.wfc.ui.PenStyle.INSIDEFRAME は変換されませんでした。

com.ms.wfc.ui.PenStyle.NULL は変換されませんでした。

com.ms.wfc.ui.PictureBox.setImage は変換されませんでした。

com.ms.wfc.ui.Point.Editor を変換できませんでした。

com.ms.wfc.ui.Point.getConstructorArgs は変換されませんでした。

com.ms.wfc.ui.Point.Point は変換されませんでした。

com.ms.wfc.ui.Point.save は変換されませんでした。

com.ms.wfc.ui.PopulatedMenusEditor は変換されませんでした。

com.ms.wfc.ui.ProgressBar.addOnValueChanged は変換されませんでした。

com.ms.wfc.ui.ProgressBar.removeOnValueChanged は変換されませんでした。

com.ms.wfc.ui.QueryContinueDragEventHandler.QueryContinueDragEventHandler を変換できませんでした。

com.ms.wfc.ui.RadioButton.setTextAlign は変換されませんでした。

com.ms.wfc.ui.Radix は変換されませんでした。

com.ms.wfc.ui.RasterOp は変換されませんでした。

com.ms.wfc.ui.ReadyStateEvent は変換されませんでした。

com.ms.wfc.ui.ReadyStateEventHandler は変換されませんでした。

com.ms.wfc.ui.Rebar.addBand は変換されませんでした。

com.ms.wfc.ui.Rebar.addOnAutoSize は変換されませんでした。

com.ms.wfc.ui.Rebar.addOnHeightChange は変換されませんでした。

com.ms.wfc.ui.Rebar.addOnLayoutChange は変換されませんでした。

com.ms.wfc.ui.Rebar.applyAutoSize は変換されませんでした。

com.ms.wfc.ui.Rebar.getBandBorders は変換されませんでした。

com.ms.wfc.ui.Rebar.getBands は変換されませんでした。

com.ms.wfc.ui.Rebar.getDoubleClickToggle は変換されませんでした。

com.ms.wfc.ui.Rebar.getFixedOrder は変換されませんでした。

com.ms.wfc.ui.Rebar.getOrientation は変換されませんでした。

com.ms.wfc.ui.Rebar.onAutoSize は変換されませんでした。

com.ms.wfc.ui.Rebar.onHeightChange は変換されませんでした。

com.ms.wfc.ui.Rebar.onLayoutChange は変換されませんでした。

com.ms.wfc.ui.Rebar.removeAllBands は変換されませんでした。

com.ms.wfc.ui.Rebar.removeBand は変換されませんでした。

com.ms.wfc.ui.Rebar.removeOnAutoSize は変換されませんでした。

com.ms.wfc.ui.Rebar.removeOnHeightChange は変換されませんでした。

com.ms.wfc.ui.Rebar.removeOnLayoutChange は変換されませんでした。

com.ms.wfc.ui.Rebar.setBandBorders は変換されませんでした。

com.ms.wfc.ui.Rebar.setBands は変換されませんでした。

com.ms.wfc.ui.Rebar.setComponentSite は変換されませんでした。

com.ms.wfc.ui.Rebar.setDoubleClickToggle は変換されませんでした。

com.ms.wfc.ui.Rebar.setFixedOrder は変換されませんでした。

com.ms.wfc.ui.Rebar.setNewControls は変換されませんでした。

com.ms.wfc.ui.Rebar.setOrientation は変換されませんでした。

com.ms.wfc.ui.RebarBand.getAllowVariableHeight は変換されませんでした。

com.ms.wfc.ui.RebarBand.getAllowVertical は変換されませんでした。

com.ms.wfc.ui.RebarBand.getBandBreak は変換されませんでした。

com.ms.wfc.ui.RebarBand.getChildControl は変換されませんでした。

com.ms.wfc.ui.RebarBand.getChildEdge は変換されませんでした。

com.ms.wfc.ui.RebarBand.getFixedBitmap は変換されませんでした。

com.ms.wfc.ui.RebarBand.getGrowBy は変換されませんでした。

com.ms.wfc.ui.RebarBand.getHeaderWidth は変換されませんでした。

com.ms.wfc.ui.RebarBand.getIdealWidth は変換されませんでした。

com.ms.wfc.ui.RebarBand.getImageIndex は変換されませんでした。

com.ms.wfc.ui.RebarBand.getIndex は変換されませんでした。

com.ms.wfc.ui.RebarBand.getMaxInitialHeight は変換されませんでした。

com.ms.wfc.ui.RebarBand.getMinChildHeight は変換されませんでした。

com.ms.wfc.ui.RebarBand.getMinChildWidth は変換されませんでした。

com.ms.wfc.ui.RebarBand.getVisibleGripper は変換されませんでした。

com.ms.wfc.ui.RebarBand.maximize は変換されませんでした。

com.ms.wfc.ui.RebarBand.minimize は変換されませんでした。

com.ms.wfc.ui.RebarBand.setAllowVariableHeight は変換されませんでした。

com.ms.wfc.ui.RebarBand.setAllowVertical は変換されませんでした。

com.ms.wfc.ui.RebarBand.setBandBreak は変換されませんでした。

com.ms.wfc.ui.RebarBand.setChildControl は変換されませんでした。

com.ms.wfc.ui.RebarBand.setChildEdge は変換されませんでした。

com.ms.wfc.ui.RebarBand.setFixedBitmap は変換されませんでした。

com.ms.wfc.ui.RebarBand.setGrowBy は変換されませんでした。

com.ms.wfc.ui.RebarBand.setHeaderWidth は変換されませんでした。

com.ms.wfc.ui.RebarBand.setIdealWidth は変換されませんでした。

com.ms.wfc.ui.RebarBand.setImageIndex は変換されませんでした。

com.ms.wfc.ui.RebarBand.setIndex は変換されませんでした。

com.ms.wfc.ui.RebarBand.setMaxInitialHeight は変換されませんでした。

com.ms.wfc.ui.RebarBand.setMinChildHeight は変換されませんでした。

com.ms.wfc.ui.RebarBand.setMinChildWidth は変換されませんでした。

com.ms.wfc.ui.RebarBand.setVisibleGripper は変換されませんでした。

com.ms.wfc.ui.RebarBand.updateStyle は変換されませんでした。

com.ms.wfc.ui.Rectangle.Editor.Editor は変換されませんでした。

com.ms.wfc.ui.Rectangle.getConstructorArgs は変換されませんでした。

com.ms.wfc.ui.Rectangle.Rectangle は変換されませんでした。

com.ms.wfc.ui.Rectangle.save は変換されませんでした。

com.ms.wfc.ui.Rectangle.setBounds は変換されませんでした。

com.ms.wfc.ui.Rectangle.toRECT は変換されませんでした。

com.ms.wfc.ui.Region.copyHandle は変換されませんでした。

com.ms.wfc.ui.Region.createPolygonal は変換されませんでした。

com.ms.wfc.ui.Region.destroyHandle は変換されませんでした。

com.ms.wfc.ui.Region.getHandle を変換できませんでした。

com.ms.wfc.ui.RequestResizeEventHandler.RequestResizeEventHandler を変換できませんでした。

com.ms.wfc.ui.RichEdit.DLL_RICHEDIT は変換されませんでした。

com.ms.wfc.ui.RichEdit.getDelimiter は変換されませんでした。

com.ms.wfc.ui.RichEdit.getFollowPunctuation は変換されませんでした。

com.ms.wfc.ui.RichEdit.getIMEColor は変換されませんでした。

com.ms.wfc.ui.RichEdit.getIMEOptions は変換されませんでした。

com.ms.wfc.ui.RichEdit.getLeadPunctuation は変換されませんでした。

com.ms.wfc.ui.RichEdit.getOnHScroll を変換できませんでした。

com.ms.wfc.ui.RichEdit.getOnIMEChange を変換できませんでした。

com.ms.wfc.ui.RichEdit.getOnProtected を変換できませんでした。

com.ms.wfc.ui.RichEdit.getOnRequestResize を変換できませんでした。

com.ms.wfc.ui.RichEdit.getOnSelChange を変換できませんでした。

com.ms.wfc.ui.RichEdit.getOnVScroll を変換できませんでした。

com.ms.wfc.ui.RichEdit.getWordBreak は変換されませんでした。

com.ms.wfc.ui.RichEdit.getWordPunctuation は変換されませんでした。

com.ms.wfc.ui.RichEdit.moveInsertionPoint は変換されませんでした。

com.ms.wfc.ui.RichEdit.RICHEDIT_CLASS10A は変換されませんでした。

com.ms.wfc.ui.RichEdit.RICHEDIT_CLASSA は変換されませんでした。

com.ms.wfc.ui.RichEdit.RICHEDIT_CLASSW は変換されませんでした。

com.ms.wfc.ui.RichEdit.RICHEDIT_DLL10 は変換されませんでした。

com.ms.wfc.ui.RichEdit.RICHEDIT_DLL20 は変換されませんでした。

com.ms.wfc.ui.RichEdit.setFollowPunctuation は変換されませんでした。

com.ms.wfc.ui.RichEdit.setIMEColor は変換されませんでした。

com.ms.wfc.ui.RichEdit.setIMEOptions は変換されませんでした。

com.ms.wfc.ui.RichEdit.setLeadPunctuation は変換されませんでした。

com.ms.wfc.ui.RichEdit.setWordBreak は変換されませんでした。

com.ms.wfc.ui.RichEdit.setWordPunctuation は変換されませんでした。

com.ms.wfc.ui.RichEdit.span は変換されませんでした。

com.ms.wfc.ui.RichEdit.WC_RICHEDIT は変換されませんでした。

com.ms.wfc.ui.RichEditIMEColor は変換されませんでした。

com.ms.wfc.ui.RichEditIMEOptions は変換されませんでした。

com.ms.wfc.ui.SameParentReferenceEditor は変換されませんでした。

com.ms.wfc.ui.SaveFileDialog.runFileDialog は変換されませんでした。

com.ms.wfc.ui.SaveFileDialog.setCreatePrompt を変換できませんでした。

com.ms.wfc.ui.Screen.MonitorEnumProc は変換されませんでした。

com.ms.wfc.ui.Screen.MONITORINFO は変換されませんでした。

com.ms.wfc.ui.Screen.MONITORINFOEX は変換されませんでした。

com.ms.wfc.ui.ScrollBar.propertyChanged は変換されませんでした。

com.ms.wfc.ui.ScrollEvent.ScrollEvent は変換されませんでした。

com.ms.wfc.ui.ScrollEventHandler.ScrollEventHandler を変換できませんでした。

com.ms.wfc.ui.SelectionChangedEventHandler.SelectionChangedEventHandler を変換できませんでした。

com.ms.wfc.ui.SelectionRange.Editor を変換できませんでした。

com.ms.wfc.ui.SelectionRange.getConstructorArgs は変換されませんでした。

com.ms.wfc.ui.SelectionStyle は変換されませんでした。

com.ms.wfc.ui.Shortcut.Editor を変換できませんでした。

com.ms.wfc.ui.Splitter.postFilterMessage は変換されませんでした。

com.ms.wfc.ui.SplitterEventHandler.SplitterEventHandler は変換されませんでした。

com.ms.wfc.ui.State.save を変換できませんでした。

com.ms.wfc.ui.State.State を変換できませんでした。

com.ms.wfc.ui.StatusBarDrawItemEventHandler.StatusBarDrawItemEventHandler を変換できませんでした。

com.ms.wfc.ui.StatusBarPanelClickEventHandler.StatusBarPanelClickEventHandler を変換できませんでした。

com.ms.wfc.ui.TabBase.getTCITEM を変換できませんでした。

com.ms.wfc.ui.TabBase.propertyChanged は変換されませんでした。

com.ms.wfc.ui.TabBase.setSelectedIndex は変換されませんでした。

com.ms.wfc.ui.TabStrip.insertTab は変換されませんでした。

com.ms.wfc.ui.TextFormat.BOTTOM は変換されませんでした。

com.ms.wfc.ui.TextFormat.EDITCONTROL は変換されませんでした。

com.ms.wfc.ui.TextFormat.ENDELLIPSIS は変換されませんでした。

com.ms.wfc.ui.TextFormat.EXPANDTABS は変換されませんでした。

com.ms.wfc.ui.TextFormat.HORIZONTALCENTER は変換されませんでした。

com.ms.wfc.ui.TextFormat.LEFT は変換されませんでした。

com.ms.wfc.ui.TextFormat.NOPREFIX は変換されませんでした。

com.ms.wfc.ui.TextFormat.PATHELLIPSIS は変換されませんでした。

com.ms.wfc.ui.TextFormat.RIGHT は変換されませんでした。

com.ms.wfc.ui.TextFormat.RIGHTTOLEFT は変換されませんでした。

com.ms.wfc.ui.TextFormat.SINGLELINE は変換されませんでした。

com.ms.wfc.ui.TextFormat.TextFormat は変換されませんでした。

com.ms.wfc.ui.TextFormat.TOP は変換されませんでした。

com.ms.wfc.ui.TextFormat.valid は変換されませんでした。

com.ms.wfc.ui.TextFormat.VERTICALCENTER は変換されませんでした。

com.ms.wfc.ui.TextFormat.WORDBREAK は変換されませんでした。

com.ms.wfc.ui.Toolbar.propertyChanged は変換できませんでした。

com.ms.wfc.ui.ToolBarTextAlign.RIGHT を変換できませんでした。

com.ms.wfc.ui.ToolBarTextAlign.UNDERNEATH を変換できませんでした。

com.ms.wfc.ui.ToolTip.shouldPersistAutomaticDelay は変換されませんでした。

com.ms.wfc.ui.ToolTip.shouldPersistAutoPopDelay は変換されませんでした。

com.ms.wfc.ui.ToolTip.shouldPersistInitialDelay は変換されませんでした。

com.ms.wfc.ui.ToolTip.shouldPersistReshowDelay は変換されませんでした。

com.ms.wfc.ui.TrackBar.propertyChanged は変換されませんでした。

com.ms.wfc.ui.TreeNode.getConstructorArgs は変換されませんでした。

com.ms.wfc.ui.TreeNode.save は変換されませんでした。

com.ms.wfc.ui.TreeNode.TreeNode は変換されませんでした。

com.ms.wfc.ui.TreeNodesEditDialog は変換されませんでした。

com.ms.wfc.ui.TreeNodesEditor は変換されませんでした。

com.ms.wfc.ui.TreeView.shouldPersistIndent は変換されませんでした。

com.ms.wfc.ui.TreeViewCancelEventHandler.TreeViewCancelEventHandler を変換できませんでした。

com.ms.wfc.ui.TreeViewEventHandler.TreeViewEventHandler を変換できませんでした。

com.ms.wfc.ui.UpDown.getAcceleration は変換されませんでした。

com.ms.wfc.ui.UpDown.getAutoBuddy は変換されませんでした。

com.ms.wfc.ui.UpDown.getAutoSize は変換されませんでした。

com.ms.wfc.ui.UpDown.getBuddyControl は変換されませんでした。

com.ms.wfc.ui.UpDown.getHorizontal は変換されませんでした。

com.ms.wfc.ui.UpDown.getModifyBuddy は変換されませんでした。

com.ms.wfc.ui.UpDown.getRadix は変換されませんでした。

com.ms.wfc.ui.UpDown.getWrap は変換されませんでした。

com.ms.wfc.ui.UpDown.onCreateHandle は変換されませんでした。

com.ms.wfc.ui.UpDown.setAcceleration は変換されませんでした。

com.ms.wfc.ui.UpDown.setAutoBuddy は変換されませんでした。

com.ms.wfc.ui.UpDown.setAutoSize は変換されませんでした。

com.ms.wfc.ui.UpDown.setBuddyControl は変換されませんでした。

com.ms.wfc.ui.UpDown.setHorizontal は変換されませんでした。

com.ms.wfc.ui.UpDown.setModifyBuddy は変換されませんでした。

com.ms.wfc.ui.UpDown.setRadix は変換されませんでした。

com.ms.wfc.ui.UpDown.setWrap は変換されませんでした。

com.ms.wfc.ui.UpDown.shouldPersistBuddyControl は変換されませんでした。

com.ms.wfc.ui.UpDownAcceleration は変換されませんでした。

com.ms.wfc.ui.UpDownAcceleration.Editor を変換できませんでした。

com.ms.wfc.ui.UpDownAlignment.MANUAL は変換されませんでした。

com.ms.wfc.ui.UpDownChangedEvent.delta は変換されませんでした。

com.ms.wfc.ui.UpDownChangedEvent.value は変換されませんでした。

com.ms.wfc.ui.UpDownChangedEventHandler.UpDownChangedEventHandler を変換できませんでした。

Com.ms.wfc.util のエラー メッセージ

com.ms.wfc.util.ArrayEnumerator.hasMoreItems は変換されませんでした。

com.ms.wfc.util.Debug.addOnDisplayAssert は変換されませんでした。

com.ms.wfc.util.Debug.addOnDisplayMessage は変換されませんでした。

com.ms.wfc.util.Debug.Debug は変換されませんでした。

com.ms.wfc.util.Debug.displaySwitches は変換されませんでした。

com.ms.wfc.util.Debug.getSwitchListText は変換されませんでした。

com.ms.wfc.util.Debug.printEnumIf は変換されませんでした。

com.ms.wfc.util.Debug.printExceptionIf は変換されませんでした。

com.ms.wfc.util.Debug.printIf は変換されませんでした。

com.ms.wfc.util.Debug.printlnIf は変換されませんでした。

com.ms.wfc.util.Debug.printObjectIf は変換されませんでした。

com.ms.wfc.util.Debug.printSwitchList は変換されませんでした。

com.ms.wfc.util.Debug.printStackTraceIf は変換されませんでした。

com.ms.wfc.util.Debug.removeOnDisplayAssert は変換されませんでした。

com.ms.wfc.util.Debug.removeOnDisplayMessage は変換されませんでした。

com.ms.wfc.util.DebugMessageEventHandler は変換されませんでした。

com.ms.wfc.util.HandleCollector は変換されませんでした。

com.ms.wfc.util.HashTable.addSlot は変換されませんでした。

com.ms.wfc.util.HashTable.buckets は変換されませんでした。

com.ms.wfc.util.HashTable.findItem は変換されませんでした。

com.ms.wfc.util.HashTable.free は変換されませんでした。

com.ms.wfc.util.HashTable.getKeys は変換されませんでした。

com.ms.wfc.util.HashTable.getValues は変換されませんでした。

com.ms.wfc.util.HashTable.growHashTable は変換されませんでした。

com.ms.wfc.util.HashTable.HashTable は変換されませんでした。

com.ms.wfc.util.HashTable.hashValues は変換されませんでした。

com.ms.wfc.util.HashTable.iMax は変換されませんでした。

com.ms.wfc.util.HashTable.keys は変換されませんでした。

com.ms.wfc.util.HashTable.maxAverageDepth は変換されませんでした。

com.ms.wfc.util.HashTable.rehash は変換されませんでした。

com.ms.wfc.util.HashTable.removeSlot は変換されませんでした。

com.ms.wfc.util.HashTable.removeValue は変換されませんでした。

com.ms.wfc.util.HashTable.sizes は変換されませんでした。

com.ms.wfc.util.HashTable.values は変換されませんでした。

com.ms.wfc.util.IEnumerator.hasMoreItems は変換されませんでした。

com.ms.wfc.util.IEnumerator.nextItem は変換されませんでした。

com.ms.wfc.util.List.capInc は変換されませんでした。

com.ms.wfc.util.List.clone は変換されませんでした。

com.ms.wfc.util.List.getVersion は変換されませんでした。

com.ms.wfc.util.List.version は変換されませんでした。

com.ms.wfc.util.NumberFormat は変換されませんでした。

com.ms.wfc.util.Root.checkLeaks は変換されませんでした。

com.ms.wfc.util.Root.get は変換されませんでした。

com.ms.wfc.util.StringSorter.compare は変換されませんでした。

com.ms.wfc.util.StringSorter.DESCENDING は変換されませんでした。

com.ms.wfc.util.StringSorter.sort は変換されませんでした。

com.ms.wfc.util.StringSorter.sort(Locale, String,.Object, int, int, int) は変換されませんでした。

com.ms.wfc.util.Utils.getJavaIdentifier は変換されませんでした。

com.ms.wfc.util.Utils.getPrimitiveWord は変換されませんでした。

com.ms.wfc.util.Utils.getReservedWord は変換されませんでした。

com.ms.wfc.util.Utils.validateIdentifier は変換されませんでした。

com.ms.wfc.util.Value.format は変換されませんでした。

com.ms.wfc.util.Value.formatCurrency は変換されませんでした。

com.ms.wfc.util.Value.formatNumber は変換されませんでした。

com.ms.wfc.util.Value.SysFreeString は変換されませんでした。

Com.ms.wfc.win32 のエラー メッセージ

com.ms.wfc.win32.CharBuffer は変換されませんでした。

com.ms.wfc.win32.Windows.ACCELERATORHANDLE は変換されませんでした。

com.ms.wfc.win32.Windows.BeginPaint は変換されませんでした。

com.ms.wfc.win32.Windows.CloseEnhMetaFile は変換されませんでした。

com.ms.wfc.win32.Windows.CloseHandle は変換されませんでした。

com.ms.wfc.win32.Windows.CopyEnhMetaFile は変換されませんでした。

com.ms.wfc.win32.Windows.CopyImage は変換されませんでした。

com.ms.wfc.win32.Windows.CreateAcceleratorTable は変換されませんでした。

com.ms.wfc.win32.Windows.CreateBitmap は変換されませんでした。

com.ms.wfc.win32.Windows.CreateBitmap(int, int, int, int, int[]) は変換されませんでした。

com.ms.wfc.win32.Windows.CreateBitmap(int, int, int, int, short[]) は変換されませんでした。

com.ms.wfc.win32.Windows.CreateBrushIndirect は変換されませんでした。

com.ms.wfc.win32.Windows.CreateCompatibleBitmap は変換されませんでした。

com.ms.wfc.win32.Windows.CreateCompatibleDC は変換されませんでした。

com.ms.wfc.win32.Windows.CreateDC は変換されませんでした。

com.ms.wfc.win32.Windows.CreateDIBitmap は変換されませんでした。

com.ms.wfc.win32.Windows.CreateEllipticRgn は変換されませんでした。

com.ms.wfc.win32.Windows.CreateEllipticRgnIndirect は変換されませんでした。

com.ms.wfc.win32.Windows.CreateEnhMetaFile は変換されませんでした。

com.ms.wfc.win32.Windows.CreateFile は変換されませんでした。

com.ms.wfc.win32.Windows.CreateFont は変換されませんでした。

com.ms.wfc.win32.Windows.CreateFontIndirect は変換されませんでした。

com.ms.wfc.win32.Windows.CreateHalftonePalette は変換されませんでした。

com.ms.wfc.win32.Windows.CreateHatchBrush は変換されませんでした。

com.ms.wfc.win32.Windows.CreateIC は変換されませんでした。

com.ms.wfc.win32.Windows.CreateILockBytesOnHGlobal は変換されませんでした。

com.ms.wfc.win32.Windows.CreateMappedBitmap は変換されませんでした。

com.ms.wfc.win32.Windows.CreateMenu は変換されませんでした。

com.ms.wfc.win32.Windows.CreatePalette は変換されませんでした。

com.ms.wfc.win32.Windows.CreatePatternBrush は変換されませんでした。

com.ms.wfc.win32.Windows.CreatePen は変換されませんでした。

com.ms.wfc.win32.Windows.CreatePenIndirect は変換されませんでした。

com.ms.wfc.win32.Windows.CreatePolygonRgn は変換されませんでした。

com.ms.wfc.win32.Windows.CreatePolyPolygonRgn は変換されませんでした。

com.ms.wfc.win32.Windows.CreatePopupMenu は変換されませんでした。

com.ms.wfc.win32.Windows.CreateProcess は変換されませんでした。

com.ms.wfc.win32.Windows.CreateRectRgn は変換されませんでした。

com.ms.wfc.win32.Windows.CreateRectRgnIndirect は変換されませんでした。

com.ms.wfc.win32.Windows.CreateRoundRectRgn は変換されませんでした。

com.ms.wfc.win32.Windows.CreateSolidBrush は変換されませんでした。

com.ms.wfc.win32.Windows.CreateWindowEx は変換されませんでした。

com.ms.wfc.win32.Windows.CURSORHANDLE は変換されませんでした。

com.ms.wfc.win32.Windows.DeleteDC は変換されませんでした。

com.ms.wfc.win32.Windows.DeleteEnhMetaFile は変換されませんでした。

com.ms.wfc.win32.Windows.DeleteObject は変換されませんでした。

com.ms.wfc.win32.Windows.DestroyAcceleratorTable は変換されませんでした。

com.ms.wfc.win32.Windows.DestroyCursor は変換されませんでした。

com.ms.wfc.win32.Windows.DestroyIcon は変換されませんでした。

com.ms.wfc.win32.Windows.DestroyMenu は変換されませんでした。

com.ms.wfc.win32.Windows.DestroyWindow は変換されませんでした。

com.ms.wfc.win32.Windows.DoDragDrop は変換されませんでした。

com.ms.wfc.win32.Windows.DrawState は変換されませんでした。

com.ms.wfc.win32.Windows.DuplicateHandle は変換されませんでした。

com.ms.wfc.win32.Windows.EMFHANDLE は変換されませんでした。

com.ms.wfc.win32.Windows.EndPaint は変換されませんでした。

com.ms.wfc.win32.Windows.EnumObjects は変換されませんでした。

com.ms.wfc.win32.Windows.EnumThreadWindows は変換されませんでした。

com.ms.wfc.win32.Windows.FindClose は変換されませんでした。

com.ms.wfc.win32.Windows.FindFirstFile は変換されませんでした。

com.ms.wfc.win32.Windows.FINDHANDLE は変換されませんでした。

com.ms.wfc.win32.Windows.GDIHANDLE は変換されませんでした。

com.ms.wfc.win32.Windows.GetDC は変換されませんでした。

com.ms.wfc.win32.Windows.GetDCEx は変換されませんでした。

com.ms.wfc.win32.Windows.GetEnhMetaFile は変換されませんでした。

com.ms.wfc.win32.Windows.GetHGlobalFromILockBytes は変換されませんでした。

com.ms.wfc.win32.Windows.GetWindowDC は変換されませんでした。

com.ms.wfc.win32.Windows.HDCHANDLE は変換されませんでした。

com.ms.wfc.win32.Windows.ICONHANDLE は変換されませんでした。

com.ms.wfc.win32.Windows.InvalidateRect は変換されませんでした。

com.ms.wfc.win32.Windows.KERNELHANDLE は変換されませんでした。

com.ms.wfc.win32.Windows.MENUHANDLE は変換されませんでした。

com.ms.wfc.win32.Windows.OleCreatePictureIndirect は変換されませんでした。

com.ms.wfc.win32.Windows.PathToRegion は変換されませんでした。

com.ms.wfc.win32.Windows.RegisterDragDrop は変換されませんでした。

com.ms.wfc.win32.Windows.ReleaseDC は変換されませんでした。

com.ms.wfc.win32.Windows.SetEnhMetaFileBits は変換されませんでした。

com.ms.wfc.win32.Windows.SetMetaFileBitsEx は変換されませんでした。

com.ms.wfc.win32.Windows.SetWindowRgn は変換されませんでした。

com.ms.wfc.win32.Windows.SetWinMetaFileBits は変換されませんでした。

com.ms.wfc.win32.Windows.StgCreateDocfileOnILockBytes は変換されませんでした。

com.ms.wfc.win32.Windows.StgOpenStorageOnILockBytes は変換されませんでした。

com.ms.wfc.win32.Windows.WINDOWHANDLE は変換されませんでした。

Com.ms.win32 のエラー メッセージ

com.ms.win32.Kernel32.CreateFiber は変換されませんでした。

com.ms.win32.Ole32.CoCreateInstanceEx は変換されませんでした。

com.ms.win32.Ole32.CoGetInstanceFromFile は変換されませんでした。

com.ms.win32.Ole32.CoGetInstanceFromIStorage は変換されませんでした。

com.ms.win32.Ole32.CreateILockBytesOnHGlobal は変換されませんでした。

com.ms.win32.Ole32.GetHGlobalFromILockBytes は変換されませんでした。

com.ms.win32.Ole32.StgCreateDocfile は変換されませんでした。

com.ms.win32.Ole32.StgCreateStorageEx は変換されませんでした。

com.ms.win32.Ole32.StgOpenLayoutDocfile は変換されませんでした。

com.ms.win32.Ole32.StgOpenStorage は変換されませんでした。

com.ms.win32.Spoolss.EnumPrinters は変換されませんでした。

Com.sun.image.codec のエラー メッセージ

com.sun.image.codec.jpeg.ImageFormatException を変換できませんでした。

com.sun.image.codec.jpeg.JPEGCodec を変換できませんでした。

com.sun.image.codec.jpeg.JPEGCodec.createJPEGDecoder を変換できませんでした。

com.sun.image.codec.jpeg.JPEGCodec.createJPEGEncoder を変換できませんでした。

com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(BufferedImage) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(int,int) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(JPEGDecodeParam) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(JPEGEncodeParam) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(Raster, int) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGDecodeParam を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.addMarkerData を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setACHuffmanComponentMapping を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setACHuffmanTable を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setDCHuffmanComponentMapping を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setDCHuffmanTable を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setDensityUnit を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setHorizontalSubsampling を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setImageInfoValid を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setMarkerData を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setQTable を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setQTableComponentMapping を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setQuality を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setRestartInterval を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setTableInfoValid を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setVerticalSubsampling を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setXDensity を変換できませんでした。

com.sun.image.codec.jpeg.JPEGEncodeParam.setYDensity を変換できませんでした。

com.sun.image.codec.jpeg.JPEGHuffmanTable を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageDecoder を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageDecoder.decodeAsBufferedImage を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageDecoder.decodeAsRaster を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageDecoder.getInputStream を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageDecoder.getJPEGDecodeParam を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.encode(BufferedImage) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.encode(BufferedImage, JPEGEncodeParam) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.encode(Raster) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.encode(Raster, JPEGEncodeParam) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultColorId を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGDecodeParam) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGEncodeParam) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGEncodeParam, BufferedImage) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGEncodeParam, int, int) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGEncodeParam, Raster,int) を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.getJPEGEncodeParam を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.getOutputStream を変換できませんでした。

com.sun.image.codec.jpeg.JPEGImageEncoder.setJPEGEncodeParam を変換できませんでした。

com.sun.image.codec.jpeg.JPEGQTable を変換できませんでした。

com.sun.image.codec.jpeg.TruncatedFileException を変換できませんでした。

java.applet のエラー メッセージ

java.applet.Applet は変換されませんでした。

java.applet.Applet.destroy は変換されませんでした。

java.applet.Applet.getAccessibleContext は変換されませんでした。

java.applet.Applet.getAppletContext は変換されませんでした。

java.applet.Applet.getAppletInfo は変換されませんでした。

java.applet.Applet.getAudioClip は変換されませんでした。

java.applet.Applet.getCodeBase は変換されませんでした。

java.applet.Applet.getDocumentBase は変換されませんでした。

java.applet.Applet.getImage は変換されませんでした。

java.applet.Applet.getLocale は変換されませんでした。

java.applet.Applet.getParameter は変換されませんでした。

java.applet.Applet.getParameterInfo は変換されませんでした。

java.applet.Applet.init は変換されませんでした。

java.applet.Applet.isActive は変換されませんでした。

java.applet.Applet.newAudioClip は変換されませんでした。

java.applet.Applet.play は変換されませんでした。

java.applet.Applet.resize は変換されませんでした。

java.applet.Applet.setStub は変換されませんでした。

java.applet.Applet.showStatus は変換されませんでした。

java.applet.Applet.start は変換されませんでした。

java.applet.Applet.stop は変換されませんでした。

java.applet.AppletContext は変換されませんでした。

java.applet.AppletStub は変換されませんでした。

java.applet.AudioClip は変換されませんでした。

Java.awt のエラー メッセージ

java.awt.<ClassName>.add*Listener は変換されませんでした。

java.awt.<ClassName>.addNotify は変換されませんでした。

java.awt.<ClassName>.getPeer は変換されませんでした。

java.awt.<ClassName>.print* は変換されませんでした。

java.awt.<ClassName>.processActionEvent は変換されませんでした。

java.awt.<ClassName>.processEvent は変換されませんでした。

java.awt.<ClassName>.processItemEvent は変換されませんでした。

java.awt.<ClassName>.remove*Listener は変換されませんでした。

java.awt.<ClassName>.removeNotify は変換されませんでした。

java.awt.ActiveEvent は変換されませんでした。

java.awt.Adjustable は変換されませんでした。

java.awt.AlphaComposite は変換されませんでした。

java.awt.AWTError は変換されませんでした。

java.awt.AWTEvent は変換されませんでした。

java.awt.AWTEvent.<EventType> は変換されませんでした。

java.awt.AWTEvent.AWTEvent は変換されませんでした。

java.awt.AWTEvent.consume は変換されませんでした。

java.awt.AWTEvent.consumed は変換されませんでした。

java.awt.AWTEvent.getID は変換されませんでした。

java.awt.AWTEvent.id は変換されませんでした。

java.awt.AWTEvent.isConsumed は変換されませんでした。

java.awt.AWTEventMulticaster は変換されませんでした。

java.awt.AWTPermission.AWTPermission は変換されませんでした。

java.awt.BasicStroke.BasicStroke は変換されませんでした。

java.awt.BasicStroke.createStrokedShape は変換されませんでした。

java.awt.BorderLayout は変換されませんでした。

java.awt.Button.getActionCommand は変換されませんでした。

java.awt.Button.getListeners は変換されませんでした。

java.awt.Button.setActionCommand は変換されませんでした。

java.awt.Canvas は変換されませんでした。

java.awt.Canvas.Canvas は変換されませんでした。

java.awt.Canvas.paint は変換されませんでした。

java.awt.CardLayout は変換されませんでした。

java.awt.Checkbox.getListeners は変換されませんでした。

java.awt.CheckboxGroup.getCurrent は変換されませんでした。

java.awt.CheckboxGroup.getSelectedCheckbox は変換されませんでした。

java.awt.CheckboxMenuItem.getAccessibleContext は変換されませんでした。

java.awt.CheckboxMenuItem.getListeners は変換されませんでした。

java.awt.Choice は変換されませんでした。

java.awt.Choice.getListeners は変換されませんでした。

java.awt.Choice.getSelectedIndex は変換されませんでした。

java.awt.Choice.getSelectedItem は変換されませんでした。

java.awt.Color.Color(ColorSpace, float[], float) は変換されませんでした。

java.awt.Color.Color(float, float, float, float) は変換されませんでした。

java.awt.Color.Color(int) は変換されませんでした。

java.awt.Color.Color(int, boolean) は変換されませんでした。

java.awt.Color.Color(int, int, int,int) は変換されませんでした。

java.awt.Color.createContext は変換されませんでした。

java.awt.Color.decode は変換されませんでした。

java.awt.Color.getColor は変換されませんでした。

java.awt.Color.getColorComponents(ColorSpace, float[]) は変換されませんでした。

java.awt.Color.getColorComponents(float[]) は変換されませんでした。

java.awt.Color.getColorSpace は変換されませんでした。

java.awt.Color.getComponents(ColorSpace, float[]) は変換されませんでした。

java.awt.Color.getComponents(float[]) は変換されませんでした。

java.awt.Color.getHSBColor は変換されませんでした。

java.awt.Color.getRGBColorComponents は変換されませんでした。

java.awt.Color.getRGBComponents は変換されませんでした。

java.awt.Color.getTransparency は変換されませんでした。

java.awt.Color.HSBtoRGB は変換されませんでした。

java.awt.Color.RGBtoHSB は変換されませんでした。

java.awt.Component.action は変換されませんでした。

java.awt.Component.addHierarchyBoundsListener は変換されませんでした。

java.awt.Component.addHierarchyListener は変換されませんでした。

java.awt.Component.addInputMethodListener は変換されませんでした。

java.awt.Component.addPropertyChangeListener は変換されませんでした。

java.awt.Component.BOTTOM_ALIGNMENT は変換されませんでした。

java.awt.Component.CENTER_ALIGNMENT は変換されませんでした。

java.awt.Component.checkImage は変換されませんでした。

java.awt.Component.coalesceEvents は変換されませんでした。

java.awt.Component.createImage(ImageProducer) は変換されませんでした。

java.awt.Component.createImage(int, int) は変換されませんでした。

java.awt.Component.deliverEvent は変換されませんでした。

java.awt.Component.disable は変換されませんでした。

java.awt.Component.disableEvents は変換されませんでした。

java.awt.Component.dispatchEvent は変換されませんでした。

java.awt.Component.enableEvents は変換されませんでした。

java.awt.Component.enableInputMethods は変換されませんでした。

java.awt.Component.firePropertyChange は変換されませんでした。

java.awt.Component.getAlignmentX は変換されませんでした。

java.awt.Component.getAlignmentY は変換されませんでした。

java.awt.Component.getColorModel は変換されませんでした。

java.awt.Component.getDropTarget は変換されませんでした。

java.awt.Component.getGraphicsConfiguration は変換されませんでした。

java.awt.Component.getInputContext は変換されませんでした。

java.awt.Component.getInputMethodRequests は変換されませんでした。

java.awt.Component.getListeners は変換されませんでした。

java.awt.Component.getLocale は変換されませんでした。

java.awt.Component.getMaximumSize は変換されませんでした。

java.awt.Component.getMinimumSize は変換されませんでした。

java.awt.Component.getPreferredSize は変換されませんでした。

java.awt.Component.getSize は変換されませんでした。

java.awt.Component.getToolkit は変換されませんでした。

java.awt.Component.getTreeLock は変換されませんでした。

java.awt.Component.handleEvent は変換されませんでした。

java.awt.Component.hasFocus は変換されませんでした。

java.awt.Component.imageUpdate は変換されませんでした。

java.awt.Component.isDisplayable は変換されませんでした。

java.awt.Component.isDoubleBuffered は変換されませんでした。

java.awt.Component.isLightweight は変換されませんでした。

java.awt.Component.isOpaque は変換されませんでした。

java.awt.Component.isValid は変換されませんでした。

java.awt.Component.isVisible は変換されませんでした。

java.awt.Component.keyDown は変換されませんでした。

java.awt.Component.keyUp は変換されませんでした。

java.awt.Component.LEFT_ALIGNMENT は変換されませんでした。

java.awt.Component.lostFocus は変換されませんでした。

java.awt.Component.minimumSize は変換されませんでした。

java.awt.Component.mouseDown は変換されませんでした。

java.awt.Component.mouseDrag は変換されませんでした。

java.awt.Component.mouseEnter は変換されませんでした。

java.awt.Component.mouseExit は変換されませんでした。

java.awt.Component.mouseMove は変換されませんでした。

java.awt.Component.mouseUp は変換されませんでした。

java.awt.Component.move は変換されませんでした。

java.awt.Component.nextFocus は変換されませんでした。

java.awt.Component.paint は変換されませんでした。

java.awt.Component.paintAll は変換されませんでした。

java.awt.Component.postEvent は変換されませんでした。

java.awt.Component.preferredSize は変換されませんでした。

java.awt.Component.prepareImage は変換されませんでした。

java.awt.Component.process*Event は変換されませんでした。

java.awt.Component.remove は変換されませんでした。

java.awt.Component.removeInputMethodListener は変換されませんでした。

java.awt.Component.removePropertyChangeListener は変換されませんでした。

java.awt.Component.repaint は変換されませんでした。

java.awt.Component.repaint(int, int, int, int) は変換されませんでした。

java.awt.Component.repaint(long) は変換されませんでした。

java.awt.Component.repaint(long, int, int, int, int) は変換されませんでした。

java.awt.Component.RIGHT_ALIGNMENT は変換されませんでした。

java.awt.Component.setBounds は変換されませんでした。

java.awt.Component.setComponentOrientation は変換されませんでした。

java.awt.Component.setDropTarget は変換されませんでした。

java.awt.Component.setEnabled は変換されませんでした。

java.awt.Component.setLocale は変換されませんでした。

java.awt.Component.setLocation は変換されませんでした。

java.awt.Component.setSize は変換されませんでした。

java.awt.Component.setVisible は変換されませんでした。

java.awt.Component.TOP_ALIGNMENT は変換されませんでした。

java.awt.Component.transferFocus は変換されませんでした。

java.awt.Component.validate は変換されませんでした。

java.awt.ComponentOrientation.getOrientation(Locale) は変換されませんでした。

java.awt.ComponentOrientation.getOrientation(ResourceBundle) は変換されませんでした。

java.awt.ComponentOrientation.isHorizontal は変換されませんでした。

java.awt.Composite は変換されませんでした。

java.awt.CompositeContext は変換されませんでした。

java.awt.Container.add(Component) は変換されませんでした。

java.awt.Container.add(Component, int) は変換されませんでした。

java.awt.Container.add(Component, Object) は変換されませんでした。

java.awt.Container.add(Component, Object, int) は変換されませんでした。

java.awt.Container.add(String, Component) は変換されませんでした。

java.awt.Container.deliverEvent は変換されませんでした。

java.awt.Container.getComponent は変換されませんでした。

java.awt.Container.getComponentAt(int, int) は変換されませんでした。

java.awt.Container.getComponentAt(Point) は変換されませんでした。

java.awt.Container.getLayout は変換されませんでした。

java.awt.Container.getListeners は変換されませんでした。

java.awt.Container.paint* は変換されませんでした。

java.awt.Container.processContainerEvent は変換されませんでした。

java.awt.Container.setLayout は変換されませんでした。

java.awt.Container.validate は変換されませんでした。

java.awt.Container.validateTree は変換されませんでした。

java.awt.Cursor.Cursor は変換されませんでした。

java.awt.Cursor.CUSTOM_CURSOR は変換されませんでした。

java.awt.Cursor.getName は変換されませんでした。

java.awt.Cursor.getSystemCustomCursor は変換されませんでした。

java.awt.Cursor.name は変換されませんでした。

java.awt.Cursor.predefined は変換されませんでした。

java.awt.Dialog.Dialog は変換されませんでした。

java.awt.Dialog.hide は変換されませんでした。

java.awt.Dialog.setModal は変換されませんでした。

java.awt.Dialog.show は変換されませんでした。

java.awt.Dimension.setSize は変換されませんでした。

java.awt.Dimension.toString は変換されませんでした。

java.awt.Event は変換されませんでした。

java.awt.EventQueue は変換されませんでした。

java.awt.FileDialog.*FilenameFilter は変換されませんでした。

java.awt.FileDialog.FileDialog(Frame) は変換されませんでした。

java.awt.FileDialog.FileDialog(Frame, String) は変換されませんでした。

java.awt.FileDialog.FileDialog(Frame, String, int) は変換されませんでした。

java.awt.FileDialog.getFile は変換されませんでした。

java.awt.FileDialog.setMode は変換されませんでした。

java.awt.FlowLayout は変換されませんでした。

java.awt.FlowLayout.addLayoutComponent は変換されませんでした。

java.awt.FlowLayout.FlowLayout は変換されませんでした。

java.awt.FlowLayout.FlowLayout(int) は変換されませんでした。

java.awt.FlowLayout.FlowLayout(int, int, int) は変換されませんでした。

java.awt.FlowLayout.getAlignment は変換されませんでした。

java.awt.FlowLayout.getHgap は変換されませんでした。

java.awt.FlowLayout.getVgap は変換されませんでした。

java.awt.FlowLayout.layoutContainer は変換されませんでした。

java.awt.FlowLayout.minimumLayoutSize は変換されませんでした。

java.awt.FlowLayout.preferredLayoutSize は変換されませんでした。

java.awt.FlowLayout.removeLayoutComponent は変換されませんでした。

java.awt.FlowLayout.setAlignment は変換されませんでした。

java.awt.FlowLayout.setHgap は変換されませんでした。

java.awt.FlowLayout.setVgap は変換されませんでした。

java.awt.FlowLayout.toString は変換されませんでした。

java.awt.Font.canDisplay は変換されませんでした。

java.awt.Font.canDisplayUpTo は変換されませんでした。

java.awt.Font.CENTER_BASELINE は変換されませんでした。

java.awt.Font.createFont は変換されませんでした。

java.awt.Font.createGlyphVector は変換されませんでした。

java.awt.Font.deriveFont(AffineTransform) は変換されませんでした。

java.awt.Font.deriveFont(int, AffineTransform) は変換されませんでした。

java.awt.Font.deriveFont(Map) は変換されませんでした。

java.awt.Font.Font は変換されませんでした。

java.awt.Font.getAttributes は変換されませんでした。

java.awt.Font.getAvailableAttributes は変換されませんでした。

java.awt.Font.getBaselineFor は変換されませんでした。

java.awt.Font.getFont は変換されませんでした。

java.awt.Font.getFont(Map) は変換されませんでした。

java.awt.Font.getItalicAngle は変換されませんでした。

java.awt.Font.getLineMetrics は変換されませんでした。

java.awt.Font.getLineMetrics(CharacterIterator, int, int, FontRenderContext) は変換されませんでした。

java.awt.Font.getMaxCharBounds は変換されませんでした。

java.awt.Font.getMissingGlyphCode は変換されませんでした。

java.awt.Font.getNumGlyphs は変換されませんでした。

java.awt.Font.getPSName は変換されませんでした。

java.awt.Font.getStringBounds(char[], int, int, FontRenderContext) は変換されませんでした。

java.awt.Font.getStringBounds(CharacterIterator, int, int, FontRenderContext) は変換されませんでした。

java.awt.Font.getStringBounds(String, FontRenderContext) は変換されませんでした。

java.awt.Font.getStringBounds(String, int, int, FontRenderContext) は変換されませんでした。

java.awt.Font.getTransform は変換されませんでした。

java.awt.Font.HANGING_BASELINE は変換されませんでした。

java.awt.Font.hasUniformLineMetrics は変換されませんでした。

java.awt.Font.PLAIN は変換されませんでした。

java.awt.Font.ROMAN_BASELINE は変換されませんでした。

java.awt.Font.TRUETYPE_FONT は変換されませんでした。

java.awt.FontMetrics.*Width は変換されませんでした。

java.awt.FontMetrics.FontMetrics は変換されませんでした。

java.awt.FontMetrics.getLeading は変換されませんでした。

java.awt.FontMetrics.getLineMetrics は変換されませんでした。

java.awt.FontMetrics.getMaxAdvance は変換されませんでした。

java.awt.FontMetrics.getMaxCharBounds は変換されませんでした。

java.awt.FontMetrics.getStringBounds(char[], int, int, Graphics) は変換されませんでした。

java.awt.FontMetrics.getStringBounds(CharacterIterator, int, int, Graphics) は変換されませんでした。

java.awt.FontMetrics.getStringBounds(String, Graphics) は変換されませんでした。

java.awt.FontMetrics.getStringBounds(String, int, int, Graphics) は変換されませんでした。

java.awt.FontMetrics.getWidths は変換されませんでした。

java.awt.FontMetrics.hasUniformLineMetrics は変換されませんでした。

java.awt.Frame は変換されませんでした。

java.awt.Frame.Frame(GraphicsConfiguration) は変換されませんでした。

java.awt.Frame.Frame(String, GraphicsConfiguration) は変換されませんでした。

java.awt.Frame.getCursorType は変換されませんでした。

java.awt.Frame.getFrames は変換されませんでした。

java.awt.Frame.getIconImage は変換されませんでした。

java.awt.Frame.setIconImage は変換されませんでした。

java.awt.Frame.setState は変換されませんでした。

java.awt.GradientPaint.createContext は変換されませんでした。

java.awt.GradientPaint.getTransparency は変換されませんでした。

java.awt.GradientPaint.GradientPaint は変換されませんでした。

java.awt.GradientPaint.isCyclic は変換されませんでした。

java.awt.Graphics.copyArea は変換されませんでした。

java.awt.Graphics.drawBytes は変換されませんでした。

java.awt.Graphics.drawChars は変換されませんでした。

java.awt.Graphics.drawRoundRect は変換されませんでした。

java.awt.Graphics.drawString は変換されませんでした。

java.awt.Graphics.drawString(AttributedCharacterIterator, int, int) は変換されませんでした。

java.awt.Graphics.drawString(int, int) は変換されませんでした。

java.awt.Graphics.fill3DRect は変換されませんでした。

java.awt.Graphics.fillRoundRect は変換されませんでした。

java.awt.Graphics.finalize は変換されませんでした。

java.awt.Graphics.getClip は変換されませんでした。

java.awt.Graphics.setClip は変換されませんでした。

java.awt.Graphics.setPaintMode は変換されませんでした。

java.awt.Graphics.setXORMode は変換されませんでした。

java.awt.Graphics2D.addRenderingHints は変換されませんでした。

java.awt.Graphics2D.clip は変換されませんでした。

java.awt.Graphics2D.draw は変換されませんでした。

java.awt.Graphics2D.drawGlyphVector は変換されませんでした。

java.awt.Graphics2D.drawImage は変換されませんでした。

java.awt.Graphics2D.drawRenderableImage は変換されませんでした。

java.awt.Graphics2D.drawRenderedImage は変換されませんでした。

java.awt.Graphics2D.drawString は変換されませんでした。

java.awt.Graphics2D.fill は変換されませんでした。

java.awt.Graphics2D.fill3DRect は変換されませんでした。

java.awt.Graphics2D.getComposite は変換されませんでした。

java.awt.Graphics2D.getDeviceConfiguration は変換されませんでした。

java.awt.Graphics2D.getPaint は変換されませんでした。

java.awt.Graphics2D.getRenderingHint は変換されませんでした。

java.awt.Graphics2D.getRenderingHints は変換されませんでした。

java.awt.Graphics2D.getStroke は変換されませんでした。

java.awt.Graphics2D.Graphics2D は変換されませんでした。

java.awt.Graphics2D.hit は変換されませんでした。

java.awt.Graphics2D.rotate は変換されませんでした。

java.awt.Graphics2D.setComposite は変換されませんでした。

java.awt.Graphics2D.setPaint は変換されませんでした。

java.awt.Graphics2D.setRenderingHint は変換されませんでした。

java.awt.Graphics2D.setRenderingHints は変換されませんでした。

java.awt.Graphics2D.setStroke は変換されませんでした。

java.awt.Graphics2D.shear は変換されませんでした。

java.awt.Graphics2D.transform は変換されませんでした。

java.awt.GraphicsConfigTemplate は変換されませんでした。

java.awt.GraphicsConfiguration は変換されませんでした。

java.awt.GraphicsDevice は変換されませんでした。

java.awt.GraphicsEnvironment は変換されませんでした。

java.awt.GraphicsEnvironment.getAllFonts は変換されませんでした。

java.awt.GraphicsEnvironment.getAvailableFontFamilyNames は変換されませんでした。

java.awt.GraphicsEnvironment.getAvailableFontFamilyNames(Locale) は変換されませんでした。

java.awt.GraphicsEnvironment.getDefaultScreenDevice は変換されませんでした。

java.awt.GraphicsEnvironment.getScreenDevices は変換されませんでした。

java.awt.GridBagConstraints は変換されませんでした。

java.awt.GridBagLayout は変換されませんでした。

java.awt.GridLayout は変換されませんでした。

java.awt.GridLayout.addLayoutComponent は変換されませんでした。

java.awt.GridLayout.GridLayout は変換されませんでした。

java.awt.GridLayout.GridLayout(int, int) は変換されませんでした。

java.awt.GridLayout.GridLayout(int, int, int, int) は変換されませんでした。

java.awt.GridLayout.layoutContainer は変換されませんでした。

java.awt.GridLayout.minimumLayoutSize は変換されませんでした。

java.awt.GridLayout.preferredLayoutSize は変換されませんでした。

java.awt.GridLayout.removeLayoutComponent は変換されませんでした。

java.awt.Image.getProperty は変換されませんでした。

java.awt.Image.SCALE_<Algorithm> は変換されませんでした。

java.awt.Image.UndefinedProperty は変換されませんでした。

java.awt.ItemSelectable は変換されませんでした。

java.awt.JobAttributes.DestinationType は変換されませんでした。

java.awt.JobAttributes.DialogType は変換されませんでした。

java.awt.JobAttributes.getDefaultSelection は変換されませんでした。

java.awt.JobAttributes.getDestination は変換されませんでした。

java.awt.JobAttributes.getDialog は変換されませんでした。

java.awt.JobAttributes.getFileName は変換されませんでした。

java.awt.JobAttributes.getMultipleDocumentHandling は変換されませんでした。

java.awt.JobAttributes.getPageRanges は変換されませんでした。

java.awt.JobAttributes.getSides は変換されませんでした。

java.awt.JobAttributes.JobAttributes は変換されませんでした。

java.awt.JobAttributes.MultipleDocumentHandlingType は変換されませんでした。

java.awt.JobAttributes.setDefaultSelection は変換されませんでした。

java.awt.JobAttributes.setDestination は変換されませんでした。

java.awt.JobAttributes.setDialog は変換されませんでした。

java.awt.JobAttributes.setFileName は変換されませんでした。

java.awt.JobAttributes.setMultipleDocumentHandling は変換されませんでした。

java.awt.JobAttributes.setMultipleDocumentHandlingToDefault は変換されませんでした。

java.awt.JobAttributes.setPageRanges は変換されませんでした。

java.awt.JobAttributes.setSides は変換されませんでした。

java.awt.JobAttributes.setSidesToDefault は変換されませんでした。

java.awt.Label.setAlignment は変換されませんでした。

java.awt.LayoutManager は変換されませんでした。

java.awt.LayoutManager2 は変換されませんでした。

java.awt.List.delItems は変換されませんでした。

java.awt.List.getListeners は変換されませんでした。

java.awt.MediaTracker は変換されませんでした。

java.awt.Menu.getAccessibleContext は変換されませんでした。

java.awt.Menu.isTearOff は変換されませんでした。

java.awt.MenuBar.deleteShortcut は変換されませんでした。

java.awt.MenuBar.getAccessibleContext は変換されませんでした。

java.awt.MenuBar.getHelpMenu は変換されませんでした。

java.awt.MenuBar.getShortcutMenuItem は変換されませんでした。

java.awt.MenuBar.setHelpMenu は変換されませんでした。

java.awt.MenuBar.shortcuts は変換されませんでした。

java.awt.MenuComponent.dispatchEvent は変換されませんでした。

java.awt.MenuComponent.getAccessibleContext は変換されませんでした。

java.awt.MenuComponent.getTreeLock は変換されませんでした。

java.awt.MenuComponent.MenuComponent は変換されませんでした。

java.awt.MenuComponent.postEvent は変換されませんでした。

java.awt.MenuComponent.setFont は変換されませんでした。

java.awt.MenuContainer.postEvent は変換されませんでした。

java.awt.MenuItem.disableEvents は変換されませんでした。

java.awt.MenuItem.enableEvents は変換されませんでした。

java.awt.MenuItem.getAccessibleContext は変換されませんでした。

java.awt.MenuItem.getActionCommand は変換されませんでした。

java.awt.MenuItem.getListeners は変換されませんでした。

java.awt.MenuItem.setActionCommand は変換されませんでした。

java.awt.MenuShortcut.MenuShortcut は変換されませんでした。

java.awt.MenuShortcut.usesShiftModifier は変換されませんでした。

java.awt.PageAttributes.getOrigin は変換されませんでした。

java.awt.PageAttributes.MediaType.<PaperKind> は変換されませんでした。

java.awt.PageAttributes.PageAttributes は変換されませんでした。

java.awt.PageAttributes.setMedia は変換されませんでした。

java.awt.PageAttributes.setMediaToDefault は変換されませんでした。

java.awt.PageAttributes.setOrigin は変換されませんでした。

java.awt.PageAttributes.setPrinterResolution(int) は変換されませんでした。

java.awt.PageAttributes.setPrinterResolution(int[]) は変換されませんでした。

java.awt.PageAttributes.setPrintQuality(int) は変換されませんでした。

java.awt.PageAttributes.setPrintQuality(PageAttributesPrintQualityType) は変換されませんでした。

java.awt.Paint.createContext は変換されませんでした。

java.awt.PaintContext は変換されませんでした。

java.awt.Panel.Panel は変換されませんでした。

java.awt.Point.getX は変換されませんでした。

java.awt.Point.getY は変換されませんでした。

java.awt.Point.setLocation は変換されませんでした。

java.awt.Polygon.addPoint は変換されませんでした。

java.awt.Polygon.bounds は変換されませんでした。

java.awt.Polygon.contains(double, double) は変換されませんでした。

java.awt.Polygon.contains(double, double, double, double) は変換されませんでした。

java.awt.Polygon.contains(Rectangle2D) は変換されませんでした。

java.awt.Polygon.getPathIterator は変換されませんでした。

java.awt.Polygon.intersects(double, double, double, double) は変換されませんでした。

java.awt.Polygon.intersects(Rectangle2D) は変換されませんでした。

java.awt.Polygon.npoints は変換されませんでした。

java.awt.Polygon.Polygon は変換されませんでした。

java.awt.PopupMenu.getAccessibleContext は変換されませんでした。

java.awt.PopupMenu.show は変換されませんでした。

java.awt.PrintGraphics は変換されませんでした。

java.awt.PrintJob は変換されませんでした。

java.awt.Rectangle.outcode は変換されませんでした。

java.awt.Rectangle.setRect は変換されませんでした。

java.awt.RenderingHints.<PropertySetting> は変換されませんでした。

java.awt.RenderingHints.add は変換されませんでした。

java.awt.RenderingHints.Key は変換されませんでした。

java.awt.RenderingHints.putAll は変換されませんでした。

java.awt.RenderingHints.RenderingHints は変換されませんでした。

java.awt.Robot は変換されませんでした。

java.awt.Scrollbar は変換されませんでした。

java.awt.Scrollbar.addAdjustmentListener は変換されませんでした。

java.awt.Scrollbar.getListeners は変換されませんでした。

java.awt.Scrollbar.getVisible は変換されませんでした。

java.awt.Scrollbar.getVisibleAmount は変換されませんでした。

java.awt.Scrollbar.paramString は変換されませんでした。

java.awt.Scrollbar.processAdjustmentEvent は変換されませんでした。

java.awt.Scrollbar.Scrollbar は変換されませんでした。

java.awt.Scrollbar.Scrollbar は変換されませんでした。

java.awt.Scrollbar.setOrientation は変換されませんでした。

java.awt.Scrollbar.setVisibleAmount は変換されませんでした。

java.awt.ScrollPane.get*Adjustable は変換されませんでした。

java.awt.ScrollPane.printComponents は変換されませんでした。

java.awt.ScrollPane.ScrollPane は変換されませんでした。

java.awt.ScrollPane.setLayout は変換されませんでした。

java.awt.Shape は変換されませんでした。

java.awt.Shape.contains(double,double) は変換されませんでした。

java.awt.Shape.contains(double, double, double, double) は変換されませんでした。

java.awt.Shape.contains(Rectangle2D) は変換されませんでした。

java.awt.Shape.getPathIterator は変換されませんでした。

java.awt.Shape.intersects(double,double,double,double) は変換されませんでした。

java.awt.Shape.intersects(Rectangle2D) は変換されませんでした。

java.awt.Stroke は変換されませんでした。

java.awt.Stroke.createStrokedShape は変換されませんでした。

java.awt.SystemColor.createContext は変換されませんでした。

java.awt.SystemColor.NUM_COLORS は変換されませんでした。

java.awt.SystemColor.text は変換されませんでした。

java.awt.SystemColor.TEXT_HIGHLIGHT は変換されませんでした。

java.awt.SystemColor.TEXT_HIGHLIGHT_TEXT は変換されませんでした。

java.awt.SystemColor.TEXT_INACTIVE_TEXT は変換されませんでした。

java.awt.SystemColor.TEXT_TEXT は変換されませんでした。

java.awt.SystemColor.textHighlight は変換されませんでした。

java.awt.SystemColor.textHighlightText は変換されませんでした。

java.awt.SystemColor.textInactiveText は変換されませんでした。

java.awt.SystemColor.textText は変換されませんでした。

java.awt.TextArea.*Columns は変換されませんでした。

java.awt.TextArea.*Rows は変換されませんでした。

java.awt.TextComponent.enableInputMethods は変換されませんでした。

java.awt.TextComponent.getBackground は変換されませんでした。

java.awt.TextComponent.getListeners は変換されませんでした。

java.awt.TextComponent.processTextEvent は変換されませんでした。

java.awt.TextComponent.setText は変換されませんでした。

java.awt.TextComponent.textListener は変換されませんでした。

java.awt.TextField.*Columns は変換されませんでした。

java.awt.TextField.getColumns は変換されませんでした。

java.awt.TextField.getListeners は変換されませんでした。

java.awt.TextField.getPreferredSize は変換されませんでした。

java.awt.TextField.preferredSize は変換されませんでした。

java.awt.TextField.setColumns は変換されませんでした。

java.awt.TextField.TextField は変換されませんでした。

java.awt.TexturePaint.createContext は変換されませんでした。

java.awt.TexturePaint.getAnchorRect は変換されませんでした。

java.awt.TexturePaint.getTransparency は変換されませんでした。

java.awt.Toolkit は変換されませんでした。

java.awt.Transparency は変換されませんでした。

java.awt.Window.applyResourceBundle は変換されませんでした。

java.awt.Window.getGraphicsConfiguration は変換されませんでした。

java.awt.Window.getInputContext は変換されませんでした。

java.awt.Window.getListeners は変換されませんでした。

java.awt.Window.getToolkit は変換されませんでした。

java.awt.Window.getWarningString は変換されませんでした。

java.awt.Window.pack は変換されませんでした。

java.awt.Window.postEvent は変換されませんでした。

java.awt.Window.processWindowEvent は変換されませんでした。

java.awt.Window.Window は変換されませんでした。

Java.awt.color のエラー メッセージ

java.awt.color.ColorSpace は変換されませんでした。

java.awt.color.ICC_ColorSpace は変換されませんでした。

java.awt.color.ICC_Profile は変換されませんでした。

java.awt.color.ICC_ProfileGray は変換されませんでした。

java.awt.color.ICC_ProfileRGB は変換されませんでした。

Java.awt.datatransfer のエラー メッセージ

java.awt.datatransfer.Clipboard.Clipboard は変換されませんでした。

java.awt.datatransfer.Clipboard.contents は変換されませんでした。

java.awt.datatransfer.Clipboard.getName は変換されませんでした。

java.awt.datatransfer.Clipboard.owner は変換されませんでした。

java.awt.datatransfer.ClipboardOwner は変換されませんでした。

java.awt.datatransfer.DataFlavor.clone は変換されませんでした。

java.awt.datatransfer.DataFlavor.DataFlavor は変換されませんでした。

java.awt.datatransfer.DataFlavor.DataFlavor(Class, String) は変換できませんでした。

java.awt.datatransfer.DataFlavor.DataFlavor(DataFlavor) は変換されませんでした。

java.awt.datatransfer.DataFlavor.DataFlavor(String) は変換されませんでした。

java.awt.datatransfer.DataFlavor.DataFlavor(String, String) は変換できませんでした。

java.awt.datatransfer.DataFlavor.DataFlavor(String, String, ClassLoader) は変換できませんでした。

java.awt.datatransfer.DataFlavor.DataFlavor(String, String, MimeTypeParameterList, class, j) は変換できませんでした。

java.awt.datatransfer.DataFlavor.equals は変換されませんでした。

java.awt.datatransfer.DataFlavor.getDefaultRepresentationClass は変換されませんでした。

java.awt.datatransfer.DataFlavor.getDefaultRepresentationClassAsString は変換されませんでした。

java.awt.datatransfer.DataFlavor.getHumanPresentableName は変換されませんでした。

java.awt.datatransfer.DataFlavor.getMimeType は変換されませんでした。

java.awt.datatransfer.DataFlavor.getParameter は変換されませんでした。

java.awt.datatransfer.DataFlavor.getPrimaryType は変換されませんでした。

java.awt.datatransfer.DataFlavor.getReaderForText は変換されませんでした。

java.awt.datatransfer.DataFlavor.getRepresentationClass は変換されませんでした。

java.awt.datatransfer.DataFlavor.getSubType は変換されませんでした。

java.awt.datatransfer.DataFlavor.isFlavorJavaFileListType は変換されませんでした。

java.awt.datatransfer.DataFlavor.isFlavorRemoteObjectType は変換されませんでした。

java.awt.datatransfer.DataFlavor.isMimeTypeEqual は変換されませんでした。

java.awt.datatransfer.DataFlavor.isMimeTypeSerializedObject は変換されませんでした。

java.awt.datatransfer.DataFlavor.isRepresentationClassInputStream は変換されませんでした。

java.awt.datatransfer.DataFlavor.isRepresentationClassRemote は変換されませんでした。

java.awt.datatransfer.DataFlavor.isRepresentationClassSerializable は変換されませんでした。

java.awt.datatransfer.DataFlavor.javaFileListFlavor は変換されませんでした。

java.awt.datatransfer.DataFlavor.javaJVMLocalObjectMimeType は変換されませんでした。

java.awt.datatransfer.DataFlavor.javaRemoteObjectMimeType は変換されませんでした。

java.awt.datatransfer.DataFlavor.normalizeMimeType は変換されませんでした。

java.awt.datatransfer.DataFlavor.normalizeMimeTypeParameter は変換されませんでした。

java.awt.datatransfer.DataFlavor.readExternal は変換されませんでした。

java.awt.datatransfer.DataFlavor.selectBestTextFlavor は変換されませんでした。

java.awt.datatransfer.DataFlavor.setHumanPresentableName は変換されませんでした。

java.awt.datatransfer.DataFlavor.tryToLoadClass は変換されませんでした。

java.awt.datatransfer.DataFlavor.writeExternal は変換されませんでした。

java.awt.datatransfer.FlavorMap は変換されませんでした。

java.awt.datatransfer.StringSelection.getTransferDataFlavors は変換されませんでした。

java.awt.datatransfer.StringSelection.isDataFlavorSupported は変換されませんでした。

java.awt.datatransfer.StringSelection.lostOwnership は変換されませんでした。

java.awt.datatransfer.SystemFlavorMap は変換されませんでした。

java.awt.datatransfer.Transferable.getTransferDataFlavors は変換されませんでした。

java.awt.datatransfer.Transferable.isDataFlavorSupported は変換されませんでした。

Java.awt.dnd のエラー メッセージ

java.awt.dnd.Autoscroll は変換されませんでした。

java.awt.dnd.DragGestureEvent.DragGestureEvent は変換されませんでした。

java.awt.dnd.DragGestureEvent.getDragAction は変換されませんでした。

java.awt.dnd.DragGestureEvent.getDragSource は変換されませんでした。

java.awt.dnd.DragGestureEvent.getSourceAsDragGestureRecognizer は変換されませんでした。

java.awt.dnd.DragGestureEvent.getTriggerEvent は変換されませんでした。

java.awt.dnd.DragGestureEvent.iterator は変換されませんでした。

java.awt.dnd.DragGestureEvent.toArray は変換されませんでした。

java.awt.dnd.DragGestureRecognizer は変換されませんでした。

java.awt.dnd.DragSource.createDragSourceContext は変換されませんでした。

java.awt.dnd.DragSource.DefaultCopyDrop は変換されませんでした。

java.awt.dnd.DragSource.DefaultCopyNoDrop は変換されませんでした。

java.awt.dnd.DragSource.DefaultLinkDrop は変換されませんでした。

java.awt.dnd.DragSource.DefaultLinkNoDrop は変換されませんでした。

java.awt.dnd.DragSource.DefaultMoveDrop は変換されませんでした。

java.awt.dnd.DragSource.DefaultMoveNoDrop は変換されませんでした。

java.awt.dnd.DragSource.DragSource は変換されませんでした。

java.awt.dnd.DragSource.getDefaultDragSource は変換されませんでした。

java.awt.dnd.DragSource.getFlavorMap は変換されませんでした。

java.awt.dnd.DragSource.isDragImageSupported は変換されませんでした。

java.awt.dnd.DragSourceContext は変換されませんでした。

java.awt.dnd.DragSourceDragEvent.DragSourceDragEvent は変換されませんでした。

java.awt.dnd.DragSourceDropEvent.DragSourceDropEvent は変換されませんでした。

java.awt.dnd.DragSourceDropEvent.getDropSuccess は変換されませんでした。

java.awt.dnd.DragSourceEvent.DragSourceEvent は変換されませんでした。

java.awt.dnd.DragSourceEvent.getDragSourceContext は変換されませんでした。

java.awt.dnd.DragSourceListener.dropActionChanged は変換されませんでした。

java.awt.dnd.DropTarget は変換されませんでした。

java.awt.dnd.DropTarget.setComponent は変換されませんでした。

java.awt.dnd.DropTargetContext は変換されませんでした。

java.awt.dnd.DropTargetDragEvent.DropTargetDragEvent は変換されませんでした。

java.awt.dnd.DropTargetDragEvent.getCurrentDataFlavors は変換されませんでした。

java.awt.dnd.DropTargetDragEvent.getCurrentDataFlavorsAsList は変換されませんでした。

java.awt.dnd.DropTargetDragEvent.isDataFlavorSupported は変換されませんでした。

java.awt.dnd.DropTargetDragEvent.rejectDrag は変換されませんでした。

java.awt.dnd.DropTargetDropEvent.dropComplete は変換されませんでした。

java.awt.dnd.DropTargetDropEvent.DropTargetDropEvent(DropTargetContext, Point, int, int) は変換されませんでした。

java.awt.dnd.DropTargetDropEvent.DropTargetDropEvent(DropTargetContext, Point, int, int, boolean) は変換されませんでした。

java.awt.dnd.DropTargetDropEvent.getCurrentDataFlavors は変換されませんでした。

java.awt.dnd.DropTargetDropEvent.getCurrentDataFlavorsAsList は変換されませんでした。

java.awt.dnd.DropTargetDropEvent.isDataFlavorSupported は変換されませんでした。

java.awt.dnd.DropTargetDropEvent.isLocalTransfer は変換されませんでした。

java.awt.dnd.DropTargetDropEvent.rejectDrop は変換されませんでした。

java.awt.dnd.DropTargetEvent は変換されませんでした。

java.awt.dnd.InvalidDnDOperationException は変換されませんでした。

java.awt.dnd.MouseDragGestureRecognizer は変換されませんでした。

java.awt.dnd.peer.DragSourceContextPeer は変換されませんでした。

java.awt.dnd.peer.DropTargetContextPeer は変換されませんでした。

java.awt.dnd.peer.DropTargetPeer は変換されませんでした。

Java.awt.event のエラー メッセージ

java.awt.event.<ClassName>.<EventType>_FIRST は変換されませんでした。

java.awt.event.<ClassName>.<EventType>_LAST は変換されませんでした。

java.awt.event.ActionEvent.<KeyName>_MASK は変換されませんでした。

java.awt.event.ActionEvent.ACTION_PERFORMED は変換されませんでした。

java.awt.event.ActionEvent.getActionCommand は変換されませんでした。

java.awt.event.AWTEventListener は変換されませんでした。

java.awt.event.ComponentEvent.COMPONENT_<EventType> は変換されませんでした。

java.awt.event.FocusEvent.FOCUS_<EventType> は変換されませんでした。

java.awt.event.FocusEvent.isTemporary は変換されませんでした。

java.awt.event.HierarchyBoundsAdapter は変換されませんでした。

java.awt.event.HierarchyBoundsListener は変換されませんでした。

java.awt.event.HierarchyEvent は変換されませんでした。

java.awt.event.HierarchyListener は変換されませんでした。

java.awt.event.InputEvent は変換されませんでした。

java.awt.event.InputMethodEvent は変換されませんでした。

java.awt.event.InputMethodListener は変換されませんでした。

java.awt.event.InvocationEvent は変換されませんでした。

java.awt.event.ItemEvent.<EventType> は変換されませんでした。

java.awt.event.ItemEvent.getStateChange は変換されませんでした。

java.awt.event.KeyEvent.<VirtualKey> は変換されませんでした。

java.awt.event.KeyEvent.CHAR_UNDEFINED は変換されませんでした。

java.awt.event.KeyEvent.getKeyChar は変換されませんでした。

java.awt.event.KeyEvent.getKeyCode は変換されませんでした。

java.awt.event.KeyEvent.getKeyModifiersText は変換されませんでした。

java.awt.event.KeyEvent.getKeyText は変換されませんでした。

java.awt.event.KeyEvent.isActionKey は変換されませんでした。

java.awt.event.KeyEvent.KEY_<EventType> は変換されませんでした。

java.awt.event.KeyEvent.setKeyChar は変換されませんでした。

java.awt.event.KeyEvent.setKeyCode は変換されませんでした。

java.awt.event.KeyEvent.setModifiers は変換されませんでした。

java.awt.event.KeyEvent.setSource は変換されませんでした。

java.awt.event.KeyEvent.VK.<VirtualKey> は変換されませんでした。

java.awt.event.KeyEvent.VK_<CharacterName> は変換されませんでした。

java.awt.event.MouseEvent.getClickCount は変換されませんでした。

java.awt.event.MouseEvent.getPoint は変換されませんでした。

java.awt.event.MouseEvent.getX は変換されませんでした。

java.awt.event.MouseEvent.getY は変換されませんでした。

java.awt.event.MouseEvent.isPopupTrigger は変換されませんでした。

java.awt.event.MouseEvent.MOUSE_<EventType> は変換されませんでした。

java.awt.event.MouseEvent.translatePoint は変換されませんでした。

java.awt.event.PaintEvent.<EventType> は変換されませんでした。

java.awt.event.PaintEvent.setUpdateRect は変換されませんでした。

java.awt.event.TextEvent.TEXT_VALUE_CHANGED は変換されませんでした。

java.awt.event.WindowEvent.WINDOW_<EventType> は変換されませんでした。

Java.awt.font のエラー メッセージ

java.awt.font.FontRenderContext は変換されませんでした。

java.awt.font.FontRenderContext.FontRenderContext は変換されませんでした。

java.awt.font.FontRenderContext.getTransform は変換されませんでした。

java.awt.font.FontRenderContext.isAntiAliased は変換されませんでした。

java.awt.font.FontRenderContext.usesFractionalMetrics は変換されませんでした。

java.awt.font.GlyphJustificationInfo は変換されませんでした。

java.awt.font.GlyphMetrics は変換されませんでした。

java.awt.font.GlyphVector は変換されませんでした。

java.awt.font.GraphicAttribute は変換されませんでした。

java.awt.font.GraphicAttribute.<AttributeName> は変換されませんでした。

java.awt.font.GraphicAttribute.draw は変換されませんでした。

java.awt.font.GraphicAttribute.getAdvance は変換されませんでした。

java.awt.font.GraphicAttribute.getAlignment は変換されませんでした。

java.awt.font.GraphicAttribute.getAscent は変換されませんでした。

java.awt.font.GraphicAttribute.getBounds は変換されませんでした。

java.awt.font.GraphicAttribute.getDescent は変換されませんでした。

java.awt.font.GraphicAttribute.getJustificationInfo は変換されませんでした。

java.awt.font.GraphicAttribute.GraphicAttribute は変換されませんでした。

java.awt.font.ImageGraphicAttribute は変換されませんでした。

java.awt.font.ImageGraphicAttribute.draw は変換されませんでした。

java.awt.font.ImageGraphicAttribute.equals は変換されませんでした。

java.awt.font.ImageGraphicAttribute.getAdvance は変換されませんでした。

java.awt.font.ImageGraphicAttribute.getAscent は変換されませんでした。

java.awt.font.ImageGraphicAttribute.getBounds は変換されませんでした。

java.awt.font.ImageGraphicAttribute.getDescent は変換されませんでした。

java.awt.font.ImageGraphicAttribute.hashCode は変換されませんでした。

java.awt.font.ImageGraphicAttribute.ImageGraphicAttribute は変換されませんでした。

java.awt.font.LineBreakMeasurer は変換されませんでした。

java.awt.font.LineBreakMeasurer.deleteChar は変換されませんでした。

java.awt.font.LineBreakMeasurer.getPosition は変換されませんでした。

java.awt.font.LineBreakMeasurer.insertChar は変換されませんでした。

java.awt.font.LineBreakMeasurer.LineBreakMeasurer は変換されませんでした。

java.awt.font.LineBreakMeasurer.nextLayout は変換されませんでした。

java.awt.font.LineBreakMeasurer.nextOffset は変換されませんでした。

java.awt.font.LineBreakMeasurer.setPosition は変換されませんでした。

java.awt.font.LineMetrics は変換されませんでした。

java.awt.font.LineMetrics.getAscent は変換されませんでした。

java.awt.font.LineMetrics.getBaselineIndex は変換されませんでした。

java.awt.font.LineMetrics.getBaselineOffsets は変換されませんでした。

java.awt.font.LineMetrics.getDescent は変換されませんでした。

java.awt.font.LineMetrics.getHeight は変換されませんでした。

java.awt.font.LineMetrics.getLeading は変換されませんでした。

java.awt.font.LineMetrics.getNumChars は変換されませんでした。

java.awt.font.LineMetrics.getStrikethroughOffset は変換されませんでした。

java.awt.font.LineMetrics.getStrikethroughThickness は変換されませんでした。

java.awt.font.LineMetrics.getUnderlineOffset は変換されませんでした。

java.awt.font.LineMetrics.getUnderlineThickness は変換されませんでした。

java.awt.font.LineMetrics.LineMetrics は変換されませんでした。

java.awt.font.MultipleMaster は変換されませんでした。

java.awt.font.OpenType は変換されませんでした。

java.awt.font.OpenType.<TagType> は変換されませんでした。

java.awt.font.OpenType.getFontTable(int) は変換されませんでした。

java.awt.font.OpenType.getFontTable(int, int, int) は変換されませんでした。

java.awt.font.OpenType.getFontTable(String) は変換されませんでした。

java.awt.font.OpenType.getFontTable(String, int, int) は変換されませんでした。

java.awt.font.OpenType.getFontTableSize は変換されませんでした。

java.awt.font.OpenType.getVersion は変換されませんでした。

java.awt.font.ShapeGraphicAttribute は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.draw は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.equals は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.FILL は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.getAdvance は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.getAscent は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.getBounds は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.getDescent は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.hashCode は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.ShapeGraphicAttribute は変換されませんでした。

java.awt.font.ShapeGraphicAttribute.STROKE は変換されませんでした。

java.awt.font.TextAttribute は変換されませんでした。

java.awt.font.TextHitInfo は変換されませんでした。

java.awt.font.TextHitInfo.afterOffset は変換されませんでした。

java.awt.font.TextHitInfo.beforeOffset は変換されませんでした。

java.awt.font.TextHitInfo.equals は変換されませんでした。

java.awt.font.TextHitInfo.getCharIndex は変換されませんでした。

java.awt.font.TextHitInfo.getInsertionIndex は変換されませんでした。

java.awt.font.TextHitInfo.getOffsetHit は変換されませんでした。

java.awt.font.TextHitInfo.getOtherHit は変換されませんでした。

java.awt.font.TextHitInfo.hashCode は変換されませんでした。

java.awt.font.TextHitInfo.isLeadingEdge は変換されませんでした。

java.awt.font.TextHitInfo.leading は変換されませんでした。

java.awt.font.TextHitInfo.toString は変換されませんでした。

java.awt.font.TextHitInfo.trailing は変換されませんでした。

java.awt.font.TextLayout は変換されませんでした。

java.awt.font.TextLayout.CaretPolicy は変換されませんでした。

java.awt.font.TextLayout.CaretPolicy.CaretPolicy は変換されませんでした。

java.awt.font.TextLayout.CaretPolicy.getStrongCaret は変換されませんでした。

java.awt.font.TextLayout.clone は変換されませんでした。

java.awt.font.TextLayout.DEFAULT_CARET_POLICY は変換されませんでした。

java.awt.font.TextLayout.draw は変換されませんでした。

java.awt.font.TextLayout.equals は変換されませんでした。

java.awt.font.TextLayout.getAdvance は変換されませんでした。

java.awt.font.TextLayout.getAscent は変換されませんでした。

java.awt.font.TextLayout.getBaseline は変換されませんでした。

java.awt.font.TextLayout.getBaselineOffsets は変換されませんでした。

java.awt.font.TextLayout.getBlackBoxBounds は変換されませんでした。

java.awt.font.TextLayout.getBounds は変換されませんでした。

java.awt.font.TextLayout.getCaretInfo は変換されませんでした。

java.awt.font.TextLayout.getCaretShape は変換されませんでした。

java.awt.font.TextLayout.getCaretShapes は変換されませんでした。

java.awt.font.TextLayout.getCharacterCount は変換されませんでした。

java.awt.font.TextLayout.getCharacterLevel は変換されませんでした。

java.awt.font.TextLayout.getDescent は変換されませんでした。

java.awt.font.TextLayout.getJustifiedLayout は変換されませんでした。

java.awt.font.TextLayout.getLeading は変換されませんでした。

java.awt.font.TextLayout.getLogicalHighlightShape は変換されませんでした。

java.awt.font.TextLayout.getLogicalRangesForVisualSelection は変換されませんでした。

java.awt.font.TextLayout.getNextLeftHit は変換されませんでした。

java.awt.font.TextLayout.getNextRightHit は変換されませんでした。

java.awt.font.TextLayout.getOutline は変換されませんでした。

java.awt.font.TextLayout.getVisibleAdvance は変換されませんでした。

java.awt.font.TextLayout.getVisualHighlightShape は変換されませんでした。

java.awt.font.TextLayout.getVisualOtherHit は変換されませんでした。

java.awt.font.TextLayout.handleJustify は変換されませんでした。

java.awt.font.TextLayout.hashCode は変換されませんでした。

java.awt.font.TextLayout.hitTestChar は変換されませんでした。

java.awt.font.TextLayout.isLeftToRight は変換されませんでした。

java.awt.font.TextLayout.isVertical は変換されませんでした。

java.awt.font.TextLayout.TextLayout は変換されませんでした。

java.awt.font.TextLayout.toString は変換されませんでした。

java.awt.font.TextMeasurer は変換されませんでした。

java.awt.font.TransformAttribute は変換されませんでした。

Java.awt.geom のエラー メッセージ

java.awt.geom.AffineTransform.<Type> は変換されませんでした。

java.awt.geom.AffineTransform.clone は変換されませんでした。

java.awt.geom.AffineTransform.concatenate は変換されませんでした。

java.awt.geom.AffineTransform.createTransformedShape は変換されませんでした。

java.awt.geom.AffineTransform.deltaTransform(double[], int, double[], int, int) は変換されませんでした。

java.awt.geom.AffineTransform.deltaTransform(Point2D, Point2D) は変換されませんでした。

java.awt.geom.AffineTransform.getType は変換されませんでした。

java.awt.geom.AffineTransform.inverseTransform(double[], int, double[], int, int) は変換されませんでした。

java.awt.geom.AffineTransform.inverseTransform(Point2D, Point2D) は変換されませんでした。

java.awt.geom.AffineTransform.preConcatenate は変換されませんでした。

java.awt.geom.AffineTransform.setTransform(AffineTransform) は変換されませんでした。

java.awt.geom.AffineTransform.setTransform(double, double, double, double, double, double) は変換されませんでした。

java.awt.geom.AffineTransform.transform は変換されませんでした。

java.awt.geom.Arc2D.Arc2D は変換されませんでした。

java.awt.geom.Arc2D.containsAngle は変換されませんでした。

java.awt.geom.Arc2D.Double.Double は変換されませんでした。

java.awt.geom.Arc2D.Double.extent は変換されませんでした。

java.awt.geom.Arc2D.Double.getAngleExtent は変換されませんでした。

java.awt.geom.Arc2D.Double.getAngleStart は変換されませんでした。

java.awt.geom.Arc2D.Double.getHeight は変換されませんでした。

java.awt.geom.Arc2D.Double.getWidth は変換されませんでした。

java.awt.geom.Arc2D.Double.getX は変換されませんでした。

java.awt.geom.Arc2D.Double.getY は変換されませんでした。

java.awt.geom.Arc2D.Double.height は変換されませんでした。

java.awt.geom.Arc2D.Double.makeBounds は変換されませんでした。

java.awt.geom.Arc2D.Double.setAngleExtent は変換されませんでした。

java.awt.geom.Arc2D.Double.setAngleStart は変換されませんでした。

java.awt.geom.Arc2D.Double.start は変換されませんでした。

java.awt.geom.Arc2D.Double.width は変換されませんでした。

java.awt.geom.Arc2D.Double.x は変換されませんでした。

java.awt.geom.Arc2D.Double.y は変換されませんでした。

java.awt.geom.Arc2D.Float.extent は変換されませんでした。

java.awt.geom.Arc2D.Float.Float は変換されませんでした。

java.awt.geom.Arc2D.Float.getAngleExtent は変換されませんでした。

java.awt.geom.Arc2D.Float.getAngleStart は変換されませんでした。

java.awt.geom.Arc2D.Float.getHeight は変換されませんでした。

java.awt.geom.Arc2D.Float.getWidth は変換されませんでした。

java.awt.geom.Arc2D.Float.getX は変換されませんでした。

java.awt.geom.Arc2D.Float.getY は変換されませんでした。

java.awt.geom.Arc2D.Float.height は変換されませんでした。

java.awt.geom.Arc2D.Float.makeBounds は変換されませんでした。

java.awt.geom.Arc2D.Float.setAngleExtent は変換されませんでした。

java.awt.geom.Arc2D.Float.setAngleStart は変換されませんでした。

java.awt.geom.Arc2D.Float.start は変換されませんでした。

java.awt.geom.Arc2D.Float.width は変換されませんでした。

java.awt.geom.Arc2D.Float.x は変換されませんでした。

java.awt.geom.Arc2D.Float.y は変換されませんでした。

java.awt.geom.Arc2D.getAngleExtent は変換されませんでした。

java.awt.geom.Arc2D.getAngleStart は変換されませんでした。

java.awt.geom.Arc2D.getArcType は変換されませんでした。

java.awt.geom.Arc2D.makeBounds は変換されませんでした。

java.awt.geom.Arc2D.setAngleExtent は変換されませんでした。

java.awt.geom.Arc2D.setAngles(double, double, double, double) は変換されませんでした。

java.awt.geom.Arc2D.setAngles(Point2D) は変換されませんでした。

java.awt.geom.Arc2D.setAngleStart(double) は変換されませんでした。

java.awt.geom.Arc2D.setAngleStart(Point2D) は変換されませんでした。

java.awt.geom.Arc2D.setArcByCenter は変換されませんでした。

java.awt.geom.Arc2D.setArcByTangent は変換されませんでした。

java.awt.geom.Arc2D.setArcType は変換されませんでした。

java.awt.geom.Arc2D.setFrame は変換されませんでした。

java.awt.geom.Area.add は変換されませんでした。

java.awt.geom.Area.clone は変換されませんでした。

java.awt.geom.Area.contains は変換されませんでした。

java.awt.geom.Area.equals は変換されませんでした。

java.awt.geom.Area.exclusiveOr は変換されませんでした。

java.awt.geom.Area.getBounds は変換されませんでした。

java.awt.geom.Area.getBounds2D は変換されませんでした。

java.awt.geom.Area.getPathIterator は変換されませんでした。

java.awt.geom.Area.intersects は変換されませんでした。

java.awt.geom.Area.isEmpty は変換されませんでした。

java.awt.geom.Area.isPolygonal は変換されませんでした。

java.awt.geom.Area.isRectangular は変換されませんでした。

java.awt.geom.Area.isSingular は変換されませんでした。

java.awt.geom.Area.subtract は変換されませんでした。

java.awt.geom.CubicCurve2D.contains は変換されませんでした。

java.awt.geom.CubicCurve2D.CubicCurve2D は変換されませんでした。

java.awt.geom.CubicCurve2D.Double.Double は変換されませんでした。

java.awt.geom.CubicCurve2D.Float.Float は変換されませんでした。

java.awt.geom.CubicCurve2D.getBounds は変換されませんでした。

java.awt.geom.CubicCurve2D.getFlatness は変換されませんでした。

java.awt.geom.CubicCurve2D.getFlatnessSq は変換されませんでした。

java.awt.geom.CubicCurve2D.getPathIterator は変換されませんでした。

java.awt.geom.CubicCurve2D.solveCubic(double[]) は変換されませんでした。

java.awt.geom.CubicCurve2D.solveCubic(double[], double[]) は変換されませんでした。

java.awt.geom.CubicCurve2D.subdivide は変換されませんでした。

java.awt.geom.Dimension2D.clone は変換されませんでした。

java.awt.geom.Dimension2D.Dimension2D は変換されませんでした。

java.awt.geom.Ellipse2D.Double.Double は変換されませんでした。

java.awt.geom.Ellipse2D.Ellipse2D は変換されませんでした。

java.awt.geom.Ellipse2D.Float.Float は変換されませんでした。

java.awt.geom.FlatteningPathIterator は変換されませんでした。

java.awt.geom.GeneralPath.append は変換されませんでした。

java.awt.geom.GeneralPath.GeneralPath は変換されませんでした。

java.awt.geom.GeneralPath.GeneralPath(int) は変換されませんでした。

java.awt.geom.GeneralPath.GeneralPath(int, int) は変換されませんでした。

java.awt.geom.GeneralPath.GeneralPath(Shape) は変換されませんでした。

java.awt.geom.GeneralPath.getBounds は変換されませんでした。

java.awt.geom.GeneralPath.getBounds2D は変換されませんでした。

java.awt.geom.GeneralPath.getCurrentPoint は変換されませんでした。

java.awt.geom.GeneralPath.getPathIterator は変換されませんでした。

java.awt.geom.GeneralPath.moveTo は変換されませんでした。

java.awt.geom.GeneralPath.quadTo は変換されませんでした。

java.awt.geom.Line2D.Double.Double(double, double, double, double) は変換されませんでした。

java.awt.geom.Line2D.Double.Double(Point2D, Point2D) は変換されませんでした。

java.awt.geom.Line2D.Float.Float(float, float, float, float) は変換されませんでした。

java.awt.geom.Line2D.Float.Float(Point2D, Point2D) は変換されませんでした。

java.awt.geom.Line2D.getBounds は変換されませんでした。

java.awt.geom.Line2D.getPathIterator は変換されませんでした。

java.awt.geom.Line2D.intersectsLine は変換されませんでした。

java.awt.geom.Line2D.Line2D は変換されませんでした。

java.awt.geom.Line2D.linesIntersect は変換されませんでした。

java.awt.geom.Line2D.ptLineDist は変換されませんでした。

java.awt.geom.Line2D.ptLineDistSq は変換されませんでした。

java.awt.geom.Line2D.ptSegDist は変換されませんでした。

java.awt.geom.Line2D.ptSegDistSq は変換されませんでした。

java.awt.geom.Line2D.relativeCCW は変換されませんでした。

java.awt.geom.PathIterator は変換されませんでした。

java.awt.geom.PathIterator.<SegmentType> は変換されませんでした。

java.awt.geom.PathIterator.currentSegment は変換されませんでした。

java.awt.geom.PathIterator.getWindingRule は変換されませんでした。

java.awt.geom.PathIterator.isDone は変換されませんでした。

java.awt.geom.PathIterator.next は変換されませんでした。

java.awt.geom.Point2D.Point2D は変換されませんでした。

java.awt.geom.QuadCurve2D は変換されませんでした。

java.awt.geom.QuadCurve2D.Double は変換されませんでした。

java.awt.geom.QuadCurve2D.Float は変換されませんでした。

java.awt.geom.Rectangle2D.<Position> は変換されませんでした。

java.awt.geom.Rectangle2D.add は変換されませんでした。

java.awt.geom.Rectangle2D.createIntersection は変換されませんでした。

java.awt.geom.Rectangle2D.Double.createIntersection は変換されませんでした。

java.awt.geom.Rectangle2D.Double.outcode は変換されませんでした。

java.awt.geom.Rectangle2D.Float.createIntersection は変換されませんでした。

java.awt.geom.Rectangle2D.Float.outcode は変換されませんでした。

java.awt.geom.Rectangle2D.getPathIterator は変換されませんでした。

java.awt.geom.Rectangle2D.intersect は変換されませんでした。

java.awt.geom.Rectangle2D.intersectsLine は変換されませんでした。

java.awt.geom.Rectangle2D.outcode は変換されませんでした。

java.awt.geom.Rectangle2D.Rectangle2D は変換されませんでした。

java.awt.geom.RectangularShape.clone は変換されませんでした。

java.awt.geom.RectangularShape.getBounds は変換されませんでした。

java.awt.geom.RectangularShape.getPathIterator は変換されませんでした。

java.awt.geom.RectangularShape.setFrame(double, double, double, double) は変換されませんでした。

java.awt.geom.RectangularShape.setFrame(Point2D, Dimension2D) は変換されませんでした。

java.awt.geom.RectangularShape.setFrame(Rectangle2D) は変換されませんでした。

java.awt.geom.RectangularShape.setFrameFromCenter は変換されませんでした。

java.awt.geom.RectangularShape.setFrameFromDiagonal(double, double, double, double) は変換されませんでした。

java.awt.geom.RectangularShape.setFrameFromDiagonal(Point2D, Point2D) は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double.archeight は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double.arcwidth は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double.getArcHeight は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double.getArcWidth は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double.height は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double.setRoundRect は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double.width は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double.x は変換されませんでした。

java.awt.geom.RoundRectangle2D.Double.y は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float.archeight は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float.arcwidth は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float.getArcHeight は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float.getArcWidth は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float.height は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float.setRoundRect は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float.width は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float.x は変換されませんでした。

java.awt.geom.RoundRectangle2D.Float.y は変換されませんでした。

java.awt.geom.RoundRectangle2D.getArcHeight は変換されませんでした。

java.awt.geom.RoundRectangle2D.getArcWidth は変換されませんでした。

java.awt.geom.RoundRectangle2D.RoundRectangle2D は変換されませんでした。

java.awt.geom.RoundRectangle2D.setFrame は変換されませんでした。

java.awt.geom.RoundRectangle2D.setRoundRect は変換されませんでした。

Java.awt.im のエラー メッセージ

java.awt.im.InputContext は変換されませんでした。

java.awt.im.InputMethodHighlight は変換されませんでした。

java.awt.im.InputMethodRequests は変換されませんでした。

java.awt.im.InputSubset は変換されませんでした。

java.awt.im.spi.InputMethod は変換されませんでした。

java.awt.im.spi.InputMethodContext は変換されませんでした。

java.awt.im.spi.InputMethodDescriptor は変換されませんでした。

Java.awt.image のエラー メッセージ

java.awt.image.<ClassName>.*Consumer は変換されませんでした。

java.awt.image.<ClassName>.*TopDownLeftRight* は変換されませんでした。

java.awt.image.<ClassName>.imageComplete は変換されませんでした。

java.awt.image.<ClassName>.setHints は変換されませんでした。

java.awt.image.<ClassName>.setPixels は変換されませんでした。

java.awt.image.<ClassName>.setProperties は変換されませんでした。

java.awt.image.AffineTransformOp を変換できませんでした。

java.awt.image.BandCombineOp を変換できませんでした。

java.awt.image.BandedSampleModel を変換できませんでした。

java.awt.image.BandedSampleModel.BandedSampleModel を変換できませんでした。

java.awt.image.BandedSampleModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.BandedSampleModel.createSubsetSampleModel を変換できませんでした。

java.awt.image.BandedSampleModel.getDataElements を変換できませんでした。

java.awt.image.BandedSampleModel.getPixel を変換できませんでした。

java.awt.image.BandedSampleModel.getPixels を変換できませんでした。

java.awt.image.BandedSampleModel.getSample を変換できませんでした。

java.awt.image.BandedSampleModel.getSampleDouble を変換できませんでした。

java.awt.image.BandedSampleModel.getSampleFloat を変換できませんでした。

java.awt.image.BandedSampleModel.getSamples を変換できませんでした。

java.awt.image.BandedSampleModel.setDataElements を変換できませんでした。

java.awt.image.BandedSampleModel.setSample を変換できませんでした。

java.awt.image.BandedSampleModel.setSamples を変換できませんでした。

java.awt.image.BufferedImage.addTileObserver を変換できませんでした。

java.awt.image.BufferedImage.BufferedImage(ColorModel, Raster, Hashtable) を変換できませんでした。

java.awt.image.BufferedImage.BufferedImage(int, int, int) を変換できませんでした。

java.awt.image.BufferedImage.BufferedImage(int, int, int, ColorModel) を変換できませんでした。

java.awt.image.BufferedImage.coerceData を変換できませんでした。

java.awt.image.BufferedImage.copyData を変換できませんでした。

java.awt.image.BufferedImage.getAlphaRaster を変換できませんでした。

java.awt.image.BufferedImage.getColorModel を変換できませんでした。

java.awt.image.BufferedImage.getHeight を変換できませんでした。

java.awt.image.BufferedImage.getMinTileX を変換できませんでした。

java.awt.image.BufferedImage.getMinTileY を変換できませんでした。

java.awt.image.BufferedImage.getNumXTiles を変換できませんでした。

java.awt.image.BufferedImage.getNumYTiles を変換できませんでした。

java.awt.image.BufferedImage.getProperty を変換できませんでした。

java.awt.image.BufferedImage.getPropertyNames を変換できませんでした。

java.awt.image.BufferedImage.getRGB を変換できませんでした。

java.awt.image.BufferedImage.getSources を変換できませんでした。

java.awt.image.BufferedImage.getTile を変換できませんでした。

java.awt.image.BufferedImage.getTileGridXOffset を変換できませんでした。

java.awt.image.BufferedImage.getTileGridYOffset を変換できませんでした。

java.awt.image.BufferedImage.getTileHeight を変換できませんでした。

java.awt.image.BufferedImage.getTileWidth を変換できませんでした。

java.awt.image.BufferedImage.getWidth を変換できませんでした。

java.awt.image.BufferedImage.getWritableTile を変換できませんでした。

java.awt.image.BufferedImage.getWritableTileIndices を変換できませんでした。

java.awt.image.BufferedImage.hasTileWriters を変換できませんでした。

java.awt.image.BufferedImage.isAlphaPremultiplied を変換できませんでした。

java.awt.image.BufferedImage.isTileWritable を変換できませんでした。

java.awt.image.BufferedImage.releaseWritableTile を変換できませんでした。

java.awt.image.BufferedImage.removeTileObserver を変換できませんでした。

java.awt.image.BufferedImage.setRGB を変換できませんでした。

java.awt.image.BufferedImage.TYPE_BYTE_BINARY を変換できませんでした。

java.awt.image.BufferedImage.TYPE_BYTE_GRAY を変換できませんでした。

java.awt.image.BufferedImageFilter を変換できませんでした。

java.awt.image.BufferedImageOp を変換できませんでした。

java.awt.image.ByteLookupTable を変換できませんでした。

java.awt.image.ColorConvertOp を変換できませんでした。

java.awt.image.ConvolveOp を変換できませんでした。

java.awt.image.ColorModel.coerceData を変換できませんでした。

java.awt.image.ColorModel.ColorModel は変換されませんでした。

java.awt.image.ColorModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.ColorModel.createCompatibleWritableRaster を変換できませんでした。

java.awt.image.ColorModel.finalize を変換できませんでした。

java.awt.image.ColorModel.getAlpha を変換できませんでした。

java.awt.image.ColorModel.getAlphaRaster を変換できませんでした。

java.awt.image.ColorModel.getBlue を変換できませんでした。

java.awt.image.ColorModel.getColorSpace を変換できませんでした。

java.awt.image.ColorModel.getComponents(int, int[], int) を変換できませんでした。

java.awt.image.ColorModel.getComponents(object, int[], int) を変換できませんでした。

java.awt.image.ColorModel.getComponentSize を変換できませんでした。

java.awt.image.ColorModel.getComponentSize(int) を変換できませんでした。

java.awt.image.ColorModel.getDataElement を変換できませんでした。

java.awt.image.ColorModel.getDataElements(int, Object) を変換できませんでした。

java.awt.image.ColorModel.getDataElements(int[], int, Object) を変換できませんでした。

java.awt.image.ColorModel.getGreen を変換できませんでした。

java.awt.image.ColorModel.getNormalizedComponents を変換できませんでした。

java.awt.image.ColorModel.getNumColorComponents を変換できませんでした。

java.awt.image.ColorModel.getNumComponents を変換できませんでした。

java.awt.image.ColorModel.getRed を変換できませんでした。

java.awt.image.ColorModel.getRGB を変換できませんでした。

java.awt.image.ColorModel.getTransferType を変換できませんでした。

java.awt.image.ColorModel.getTransparency を変換できませんでした。

java.awt.image.ColorModel.getUnnormalizedComponents を変換できませんでした。

java.awt.image.ComponentColorModel.coerceData を変換できませんでした。

java.awt.image.ComponentColorModel.ComponentColorModel を変換できませんでした。

java.awt.image.ComponentColorModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.ComponentColorModel.createCompatibleWritableRaster を変換できませんでした。

java.awt.image.ComponentColorModel.getAlpha を変換できませんでした。

java.awt.image.ComponentColorModel.getAlphaRaster を変換できませんでした。

java.awt.image.ComponentColorModel.getBlue を変換できませんでした。

java.awt.image.ComponentColorModel.getComponents を変換できませんでした。

java.awt.image.ComponentColorModel.getDataElement を変換できませんでした。

java.awt.image.ComponentColorModel.getDataElements(int[], int) を変換できませんでした。

java.awt.image.ComponentColorModel.getDataElements(int, Object) を変換できませんでした。

java.awt.image.ComponentColorModel.getGreen を変換できませんでした。

java.awt.image.ComponentColorModel.getRed を変換できませんでした。

java.awt.image.ComponentColorModel.getRGB を変換できませんでした。

java.awt.image.ColorModel.isAlphaPremultiplied を変換できませんでした。

java.awt.image.ColorModel.isCompatibleRaster を変換できませんでした。

java.awt.image.ColorModel.isCompatibleSampleModel を変換できませんでした。

java.awt.image.ColorModel.transferType を変換できませんでした。

java.awt.image.ComponentColorModel.isCompatibleRaster を変換できませんでした。

java.awt.image.ComponentColorModel.isCompatibleSampleModel を変換できませんでした。

java.awt.image.ComponentSampleModel を変換できませんでした。

java.awt.image.ComponentSampleModel.bandOffsets を変換できませんでした。

java.awt.image.ComponentSampleModel.bankIndices を変換できませんでした。

java.awt.image.ComponentSampleModel.ComponentSampleModel を変換できませんでした。

java.awt.image.ComponentSampleModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.ComponentSampleModel.createSubsetSampleModel を変換できませんでした。

java.awt.image.ComponentSampleModel.getBandOffsets を変換できませんでした。

java.awt.image.ComponentSampleModel.getBankIndices を変換できませんでした。

java.awt.image.ComponentSampleModel.getDataElements を変換できませんでした。

java.awt.image.ComponentSampleModel.getNumDataElements を変換できませんでした。

java.awt.image.ComponentSampleModel.getOffset(int, int) を変換できませんでした。

java.awt.image.ComponentSampleModel.getOffset(int, int, int) を変換できませんでした。

java.awt.image.ComponentSampleModel.getPixel を変換できませんでした。

java.awt.image.ComponentSampleModel.getPixels を変換できませんでした。

java.awt.image.ComponentSampleModel.getSample を変換できませんでした。

java.awt.image.ComponentSampleModel.getSampleDouble を変換できませんでした。

java.awt.image.ComponentSampleModel.getSampleFloat を変換できませんでした。

java.awt.image.ComponentSampleModel.getSamples を変換できませんでした。

java.awt.image.ComponentSampleModel.getSampleSize を変換できませんでした。

java.awt.image.ComponentSampleModel.numBands を変換できませんでした。

java.awt.image.ComponentSampleModel.numBanks を変換できませんでした。

java.awt.image.ComponentSampleModel.pixelStride を変換できませんでした。

java.awt.image.ComponentSampleModel.scanlineStride を変換できませんでした。

java.awt.image.ComponentSampleModel.setDataElements を変換できませんでした。

java.awt.image.ComponentSampleModel.setSample を変換できませんでした。

java.awt.image.ComponentSampleModel.setSamples を変換できませんでした。

java.awt.image.DataBuffer.<DataType> を変換できませんでした。

java.awt.image.DataBuffer.DataBuffer(int, int, int, int) を変換できませんでした。

java.awt.image.DataBuffer.DataBuffer(int, int, int, int[]) を変換できませんでした。

java.awt.image.DataBuffer.dataType を変換できませんでした。

java.awt.image.DataBuffer.getDataType を変換できませんでした。

java.awt.image.DataBuffer.getDataTypeSize を変換できませんでした。

java.awt.image.DataBuffer.getElem を変換できませんでした。

java.awt.image.DataBuffer.getElemDouble を変換できませんでした。

java.awt.image.DataBuffer.getElemFloat を変換できませんでした。

java.awt.image.DataBuffer.getOffset を変換できませんでした。

java.awt.image.DataBuffer.getOffsets を変換できませんでした。

java.awt.image.DataBuffer.offset を変換できませんでした。

java.awt.image.DataBuffer.offsets を変換できませんでした。

java.awt.image.DataBuffer.setElem を変換できませんでした。

java.awt.image.DataBuffer.setElemDouble を変換できませんでした。

java.awt.image.DataBuffer.setElemFloat を変換できませんでした。

java.awt.image.DataBufferByte.DataBufferByte を変換できませんでした。

java.awt.image.DataBufferByte.getBankData を変換できませんでした。

java.awt.image.DataBufferInt.DataBufferInt を変換できませんでした。

java.awt.image.DataBufferInt.getBankData を変換できませんでした。

java.awt.image.DataBufferShort.DataBufferShort を変換できませんでした。

java.awt.image.DataBufferShort.getBankData を変換できませんでした。

java.awt.image.DataBufferUShort.DataBufferUShort を変換できませんでした。

java.awt.image.DataBufferUShort.getBankData を変換できませんでした。

java.awt.image.DirectColorModel.coerceData を変換できませんでした。

java.awt.image.DirectColorModel.createCompatibleWritableRaster を変換できませんでした。

java.awt.image.DirectColorModel.DirectColorModel(ColorSpace, int, int, int, int, int, boolean, int) を変換できませんでした。

java.awt.image.DirectColorModel.DirectColorModel(int, int, int, int) を変換できませんでした。

java.awt.image.DirectColorModel.DirectColorModel(int, int, int, int, int) を変換できませんでした。

java.awt.image.DirectColorModel.getAlpha は変換されませんでした。

java.awt.image.DirectColorModel.getBlue は変換されませんでした。

java.awt.image.DirectColorModel.getComponents を変換できませんでした。

java.awt.image.DirectColorModel.getDataElement を変換できませんでした。

java.awt.image.DirectColorModel.getDataElements(int, Object) を変換できませんでした。

java.awt.image.DirectColorModel.getDataElements(int[], int) を変換できませんでした。

java.awt.image.DirectColorModel.getGreen は変換されませんでした。

java.awt.image.DirectColorModel.getRed は変換されませんでした。

java.awt.image.DirectColorModel.getRGB は変換されませんでした。

java.awt.image.DirectColorModel.isCompatibleRaster を変換できませんでした。

java.awt.image.FilteredImageSource.startProduction は変換されませんでした。

java.awt.image.ImageConsumer は変換されませんでした。

java.awt.image.ImageFilter.setColorModel は変換されませんでした。

java.awt.image.ImageObserver は変換されませんでした。

java.awt.image.ImageProducer.startProduction は変換されませんでした。

java.awt.image.ImagingOpException を変換できませんでした。

java.awt.image.IndexColorModel.convertToIntDiscrete を変換できませんでした。

java.awt.image.IndexColorModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.IndexColorModel.createCompatibleWritableRaster を変換できませんでした。

java.awt.image.IndexColorModel.finalize を変換できませんでした。

java.awt.image.IndexColorModel.getComponents を変換できませんでした。

java.awt.image.IndexColorModel.getComponentSize を変換できませんでした。

java.awt.image.IndexColorModel.getDataElement を変換できませんでした。

java.awt.image.IndexColorModel.getDataElements を変換できませんでした。

java.awt.image.IndexColorModel.getRGBs を変換できませんでした。

java.awt.image.IndexColorModel.getTransparency を変換できませんでした。

java.awt.image.IndexColorModel.getValidPixels を変換できませんでした。

java.awt.image.IndexColorModel.IndexColorModel(int, int, int[], int, boolean,int,int) を変換できませんでした。

java.awt.image.IndexColorModel.IndexColorModel(int, int, int[], int, int, BigInteger) を変換できませんでした。

java.awt.image.IndexColorModel.isCompatibleRaster を変換できませんでした。

java.awt.image.IndexColorModel.isCompatibleSampleModel を変換できませんでした。

java.awt.image.Kernel を変換できませんでした。

java.awt.image.LookupOp を変換できませんでした。

java.awt.image.LookupTable を変換できませんでした。

java.awt.image.MemoryImageSource は変換されませんでした。

java.awt.image.MultiPixelPackedSampleModel を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.createSubsetSampleModel を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getBitOffset を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getDataBitOffset を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getDataElements を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getNumDataElements を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getOffset を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getPixel を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getSample を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getSampleSize を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getSampleSize(int) を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.getTransferType を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.MultiPixelPackedSampleModel(int, int, int, int) を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.MultiPixelPackedSampleModel(int, int, int, int, int, int) を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.setDataElements を変換できませんでした。

java.awt.image.MultiPixelPackedSampleModel.setSample を変換できませんでした。

java.awt.image.PackedColorModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.PackedColorModel.getAlphaRaster を変換できませんでした。

java.awt.image.PackedColorModel.getMask を変換できませんでした。

java.awt.image.PackedColorModel.getMasks を変換できませんでした。

java.awt.image.PackedColorModel.isCompatibleSampleModel を変換できませんでした。

java.awt.image.PackedColorModel.PackedColorModel を変換できませんでした。

java.awt.image.PixelGrabber.abortGrabbing は変換されませんでした。

java.awt.image.PixelGrabber.getStatus は変換されませんでした。

java.awt.image.PixelGrabber.setColorModel は変換されませんでした。

java.awt.image.PixelGrabber.startGrabbing は変換されませんでした。

java.awt.image.PixelGrabber.status は変換されませんでした。

java.awt.image.PixelInterleavedSampleModel を変換できませんでした。

java.awt.image.PixelInterleavedSampleModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.PixelInterleavedSampleModel.createSubsetSampleModel を変換できませんでした。

java.awt.image.PixelInterleavedSampleModel.PixelInterleavedSampleModel を変換できませんでした。

java.awt.image.Raster を変換できませんでした。

java.awt.image.Raster.createChild を変換できませんでした。

java.awt.image.Raster.createCompatibleWritableRaster を変換できませんでした。

java.awt.image.Raster.createRaster を変換できませんでした。

java.awt.image.Raster.createTranslatedChild を変換できませんでした。

java.awt.image.Raster.createWritableRaster を変換できませんでした。

java.awt.image.Raster.getDataBuffer を変換できませんでした。

java.awt.image.Raster.getDataElements(int, int, int, int, object) を変換できませんでした。

java.awt.image.Raster.getDataElements(int, int, object) を変換できませんでした。

java.awt.image.Raster.getNumBands を変換できませんでした。

java.awt.image.Raster.getNumDataElements を変換できませんでした。

java.awt.image.Raster.getParent を変換できませんでした。

java.awt.image.Raster.getPixel(int, int, double[]) を変換できませんでした。

java.awt.image.Raster.getPixel(int, int, float[]) を変換できませんでした。

java.awt.image.Raster.getPixels(int, int, int, int, double[]) を変換できませんでした。

java.awt.image.Raster.getPixels(int, int, int, int, float[]) を変換できませんでした。

java.awt.image.Raster.getPixels(int, int, int, int, int[]) を変換できませんでした。

java.awt.image.Raster.getPixel(int, int, int[]) を変換できませんでした。

java.awt.image.Raster.getSample を変換できませんでした。

java.awt.image.Raster.getSampleDouble を変換できませんでした。

java.awt.image.Raster.getSampleFloat を変換できませんでした。

java.awt.image.Raster.getSampleModelTranslateX を変換できませんでした。

java.awt.image.Raster.getSampleModelTranslateY を変換できませんでした。

java.awt.image.Raster.getSamples を変換できませんでした。

java.awt.image.Raster.getSamples(int, int, int, int, int, double[]) を変換できませんでした。

java.awt.image.Raster.getSamples(int, int, int, int, int, float[]) を変換できませんでした。

java.awt.image.Raster.getTransferType を変換できませんでした。

java.awt.image.Raster.numBands を変換できませんでした。

java.awt.image.Raster.numDataElements を変換できませんでした。

java.awt.image.Raster.parent を変換できませんでした。

java.awt.image.Raster.Raster(SampleModel, DataBuffer, Point) を変換できませんでした。

java.awt.image.Raster.Raster(SampleModel, DataBuffer, Rectangle, Point, Raster) を変換できませんでした。

java.awt.image.Raster.Raster(SampleModel, Point) を変換できませんでした。

java.awt.image.Raster.sampleModelTranslateX を変換できませんでした。

java.awt.image.Raster.sampleModelTranslateY を変換できませんでした。

java.awt.image.RasterOp を変換できませんでした。

java.awt.image.renderable.ContextualRenderedImageFactory を変換できませんでした。

java.awt.image.renderable.RenderableImage.createDefaultRendering を変換できませんでした。

java.awt.image.renderable.RenderableImage.createRendering を変換できませんでした。

java.awt.image.renderable.RenderableImage.createScaledRendering を変換できませんでした。

java.awt.image.renderable.RenderableImage.getMinX を変換できませんでした。

java.awt.image.renderable.RenderableImage.getMinY を変換できませんでした。

java.awt.image.renderable.RenderableImage.getProperty を変換できませんでした。

java.awt.image.renderable.RenderableImage.getPropertyNames を変換できませんでした。

java.awt.image.renderable.RenderableImage.getSources を変換できませんでした。

java.awt.image.renderable.RenderableImage.HINTS_OBSERVED を変換できませんでした。

java.awt.image.renderable.RenderableImage.isDynamic を変換できませんでした。

java.awt.image.renderable.RenderableImageOp を変換できませんでした。

java.awt.image.renderable.RenderableImageProducer.addConsumer を変換できませんでした。

java.awt.image.renderable.RenderableImageProducer.isConsumer を変換できませんでした。

java.awt.image.renderable.RenderableImageProducer.removeConsumer を変換できませんでした。

java.awt.image.renderable.RenderableImageProducer.RenderableImageProducer を変換できませんでした。

java.awt.image.renderable.RenderableImageProducer.requestTopDownLeftRightResend を変換できませんでした。

java.awt.image.renderable.RenderableImageProducer.run を変換できませんでした。

java.awt.image.renderable.RenderableImageProducer.setRenderContext を変換できませんでした。

java.awt.image.renderable.RenderableImageProducer.startProduction を変換できませんでした。

java.awt.image.renderable.RenderContext を変換できませんでした。

java.awt.image.renderable.RenderedImageFactory を変換できませんでした。

java.awt.image.RenderedImage.copyData を変換できませんでした。

java.awt.image.RenderedImage.getColorModel を変換できませんでした。

java.awt.image.RenderedImage.getMinTileX を変換できませんでした。

java.awt.image.RenderedImage.getMinTileY を変換できませんでした。

java.awt.image.RenderedImage.getNumXTiles を変換できませんでした。

java.awt.image.RenderedImage.getNumYTiles を変換できませんでした。

java.awt.image.RenderedImage.getProperty を変換できませんでした。

java.awt.image.RenderedImage.getPropertyNames を変換できませんでした。

java.awt.image.RenderedImage.getSources を変換できませんでした。

java.awt.image.RenderedImage.getTile を変換できませんでした。

java.awt.image.RenderedImage.getTileGridXOffset を変換できませんでした。

java.awt.image.RenderedImage.getTileGridYOffset を変換できませんでした。

java.awt.image.RenderedImage.getTileHeight を変換できませんでした。

java.awt.image.RenderedImage.getTileWidth を変換できませんでした。

java.awt.image.ReplicateScaleFilter.outpixbuf は変換されませんでした。

java.awt.image.ReplicateScaleFilter.src* は変換されませんでした。

java.awt.image.RescaleOp を変換できませんでした。

java.awt.image.RGBImageFilter は変換されませんでした。

java.awt.image.SampleModel を変換できませんでした。

java.awt.image.SampleModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.SampleModel.createSubsetSampleModel を変換できませんでした。

java.awt.image.SampleModel.dataType を変換できませんでした。

java.awt.image.SampleModel.getDataElements(int, int, int, int, Object, DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getDataElements(int, int, Object, DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getDataType を変換できませんでした。

java.awt.image.SampleModel.getNumBands を変換できませんでした。

java.awt.image.SampleModel.getNumDataElements を変換できませんでした。

java.awt.image.SampleModel.getPixel(int, int, double[], DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getPixel(int, int, float[], DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getPixel(int, int, int[], DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getPixels(int, int, int, int, double[], DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getPixels(int, int, int, int, float[], DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getPixels(int, int, int, int, int[], DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getSample を変換できませんでした。

java.awt.image.SampleModel.getSampleDouble を変換できませんでした。

java.awt.image.SampleModel.getSampleFloat を変換できませんでした。

java.awt.image.SampleModel.getSamples(int, int, int, int, int, double[], DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getSamples(int, int, int, int, int, float[], DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getSamples(int, int, int, int, int, int[], DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.getSampleSize を変換できませんでした。

java.awt.image.SampleModel.getSampleSize(int) を変換できませんでした。

java.awt.image.SampleModel.getTransferType を変換できませんでした。

java.awt.image.SampleModel.numBands を変換できませんでした。

java.awt.image.SampleModel.SampleModel を変換できませんでした。

java.awt.image.SampleModel.setDataElements(int, int, int, int, Object, DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.setDataElements(int, int, Object, DataBuffer) を変換できませんでした。

java.awt.image.SampleModel.setPixel を変換できませんでした。

java.awt.image.SampleModel.setSample を変換できませんでした。

java.awt.image.SampleModel.setSamples を変換できませんでした。

java.awt.image.ShortLookupTable を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.createCompatibleSampleModel を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.createSubsetSampleModel を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getBitMasks を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getBitOffsets を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getDataElements を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getNumDataElements を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getOffset を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getPixel を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getPixels を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getSample を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getSamples を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getSampleSize を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.getSampleSize(int) を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.setDataElements を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.setSample を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.setSamples を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.SinglePixelPackedSampleModel(int, int, int, int, int[]) を変換できませんでした。

java.awt.image.SinglePixelPackedSampleModel.SinglePixelPackedSampleModel(int, int, int, int[]) を変換できませんでした。

java.awt.image.TileObserver を変換できませんでした。

java.awt.Image.UndefinedProperty は変換されませんでした。

java.awt.image.WritableRaster を変換できませんでした。

java.awt.image.WritableRaster.createWritableChild を変換できませんでした。

java.awt.image.WritableRaster.createWritableTranslatedChild を変換できませんでした。

java.awt.image.WritableRaster.getWritableParent を変換できませんでした。

java.awt.image.WritableRaster.setDataElements(int, int, int, int, Object) を変換できませんでした。

java.awt.image.WritableRaster.setDataElements(int, int, Object) を変換できませんでした。

java.awt.image.WritableRaster.setDataElements(int, int, Raster) を変換できませんでした。

java.awt.image.WritableRaster.setPixel(int, int, int, double) を変換できませんでした。

java.awt.image.WritableRaster.setPixel(int, int, int, float) を変換できませんでした。

java.awt.image.WritableRaster.setPixel(int, int, int, int) を変換できませんでした。

java.awt.image.WritableRaster.setRect を変換できませんでした。

java.awt.image.WritableRaster.setSample(int, int, int, double) を変換できませんでした。

java.awt.image.WritableRaster.setSample(int, int, int, float) を変換できませんでした。

java.awt.image.WritableRaster.setSample(int, int, int, int) を変換できませんでした。

java.awt.image.WritableRaster.setSamples(int, int, int, int, int, double[]) を変換できませんでした。

java.awt.image.WritableRaster.setSamples(int, int, int, int, int, float[]) を変換できませんでした。

java.awt.image.WritableRaster.setSamples(int, int, int, int, int, int[]) を変換できませんでした。

java.awt.image.WritableRaster.WritableRaster(Rectangle, Point, WritableRaster) を変換できませんでした。

java.awt.image.WritableRaster.WritableRaster(SampleModel, DataBuffer, Point) を変換できませんでした。

java.awt.image.WritableRaster.WritableRaster(SampleModel, Point) を変換できませんでした。

java.awt.image.WritableRenderedImage を変換できませんでした。

Java.awt.peer のエラー メッセージ

java.awt.peer.CanvasPeer を変換できませんでした。

java.awt.peer.ComponentPeer.checkImage を変換できませんでした。

java.awt.peer.ComponentPeer.coalescePaintEvent を変換できませんでした。

java.awt.peer.ComponentPeer.createImage(ImageProducer) を変換できませんでした。

java.awt.peer.ComponentPeer.createImage(int, int) を変換できませんでした。

java.awt.peer.ComponentPeer.getColorModel を変換できませんでした。

java.awt.peer.ComponentPeer.getGraphicsConfiguration を変換できませんでした。

java.awt.peer.ComponentPeer.getMinimumSize を変換できませんでした。

java.awt.peer.ComponentPeer.getPreferredSize を変換できませんでした。

java.awt.peer.ComponentPeer.getToolkit を変換できませんでした。

java.awt.peer.ComponentPeer.handleEvent を変換できませんでした。

java.awt.peer.ComponentPeer.minimumSize を変換できませんでした。

java.awt.peer.ComponentPeer.paint を変換できませんでした。

java.awt.peer.ComponentPeer.preferredSize を変換できませんでした。

java.awt.peer.ComponentPeer.prepareImage を変換できませんでした。

java.awt.peer.ComponentPeer.print を変換できませんでした。

java.awt.peer.ComponentPeer.setBounds を変換できませんでした。

java.awt.peer.ComponentPeer.setVisible を変換できませんでした。

java.awt.peer.ContainerPeer.beginValidate を変換できませんでした。

java.awt.peer.ContainerPeer.endValidate を変換できませんでした。

java.awt.peer.FileDialogPeer.setFilenameFilter を変換できませんでした。

java.awt.peer.FramePeer.setIconImage を変換できませんでした。

java.awt.peer.LightweightPeer を変換できませんでした。

java.awt.peer.ListPeer.delItems を変換できませんでした。

java.awt.peer.ListPeer.makeVisible を変換できませんでした。

java.awt.peer.MenuBarPeer.addHelpMenu を変換できませんでした。

java.awt.peer.PopupMenuPeer.show を変換できませんでした。

java.awt.peer.RobotPeer を変換できませんでした。

java.awt.peer.ScrollbarPeer.setValues を変換できませんでした。

java.awt.peer.ScrollPanePeer.setUnitIncrement を変換できませんでした。

java.awt.peer.ScrollPanePeer.setValue を変換できませんでした。

java.awt.peer.TextComponentPeer.filterEvents を変換できませんでした。

java.awt.peer.TextComponentPeer.getCharacterBounds を変換できませんでした。

java.awt.peer.TextComponentPeer.getIndexAtPoint を変換できませんでした。

java.awt.peer.TextFieldPeer.getPreferredSize を変換できませんでした。

java.awt.peer.TextFieldPeer.preferredSize を変換できませんでした。

java.awt.peer.WindowPeer.handleFocusTraversalEvent を変換できませんでした。

Java.awt.print のエラー メッセージ

java.awt.print.Book を変換できませんでした。

java.awt.print.Pageable を変換できませんでした。

java.awt.print.PageFormat.<Format> を変換できませんでした。

java.awt.print.PageFormat.getMatrix を変換できませんでした。

java.awt.print.PageFormat.setOrientation を変換できませんでした。

java.awt.print.Paper.clone を変換できませんでした。

java.awt.print.Paper.getImageableHeight を変換できませんでした。

java.awt.print.Paper.getImageableWidth を変換できませんでした。

java.awt.print.Paper.getImageableX を変換できませんでした。

java.awt.print.Paper.getImageableY を変換できませんでした。

java.awt.print.Paper.Paper を変換できませんでした。

java.awt.print.Paper.setImageableArea を変換できませんでした。

java.awt.print.Paper.setSize を変換できませんでした。

java.awt.print.Printable を変換できませんでした。

java.awt.print.PrinterGraphics を変換できませんでした。

java.awt.print.PrinterJob.cancel を変換できませんでした。

java.awt.print.PrinterJob.defaultPage を変換できませんでした。

java.awt.print.PrinterJob.isCancelled を変換できませんでした。

java.awt.print.PrinterJob.setPageable を変換できませんでした。

java.awt.print.PrinterJob.setPrintable(Printable) を変換できませんでした。

java.awt.print.PrinterJob.setPrintable(Printable, PageFormat) を変換できませんでした。

java.awt.print.PrinterJob.validatePage を変換できませんでした。

Java.beans のエラー メッセージ

java.beans.AppletInitializer は変換されませんでした。

java.beans.BeanDescriptor は変換されませんでした。

java.beans.BeanDescriptor.BeanDescriptor は変換されませんでした。

java.beans.BeanDescriptor.getCustomizerClass は変換されませんでした。

java.beans.BeanInfo は変換されませんでした。

java.beans.BeanInfo.ICON_COLOR_16x16 は変換されませんでした。

java.beans.BeanInfo.ICON_COLOR_32x32 は変換されませんでした。

java.beans.BeanInfo.ICON_MONO_16x16 は変換されませんでした。

java.beans.BeanInfo.ICON_MONO_32x32 は変換されませんでした。

java.beans.Beans は変換されませんでした。

java.beans.Customizer は変換されませんでした。

java.beans.DesignMode は変換されませんでした。

java.beans.EventSetDescriptor は変換されませんでした。

java.beans.EventSetDescriptor.EventSetDescriptor は変換されませんでした。

java.beans.EventSetDescriptor.getAddListenerMethod は変換されませんでした。

java.beans.EventSetDescriptor.getListenerMethodDescriptors は変換されませんでした。

java.beans.EventSetDescriptor.getListenerMethods は変換されませんでした。

java.beans.EventSetDescriptor.getListenerType は変換されませんでした。

java.beans.EventSetDescriptor.getRemoveListenerMethod は変換されませんでした。

java.beans.EventSetDescriptor.isInDefaultEventSet は変換されませんでした。

java.beans.EventSetDescriptor.isUnicast は変換されませんでした。

java.beans.EventSetDescriptor.setInDefaultEventSet は変換されませんでした。

java.beans.EventSetDescriptor.setUnicast は変換されませんでした。

java.beans.FeatureDescriptor は変換されませんでした。

java.beans.FeatureDescriptor.attributeNames は変換されませんでした。

java.beans.FeatureDescriptor.FeatureDescriptor は変換されませんでした。

java.beans.FeatureDescriptor.getDisplayName は変換されませんでした。

java.beans.FeatureDescriptor.getName は変換されませんでした。

java.beans.FeatureDescriptor.getShortDescription は変換されませんでした。

java.beans.FeatureDescriptor.getValue は変換されませんでした。

java.beans.FeatureDescriptor.isExpert は変換されませんでした。

java.beans.FeatureDescriptor.isHidden は変換されませんでした。

java.beans.FeatureDescriptor.isPreferred は変換されませんでした。

java.beans.FeatureDescriptor.setDisplayName は変換されませんでした。

java.beans.FeatureDescriptor.setExpert は変換されませんでした。

java.beans.FeatureDescriptor.setHidden は変換されませんでした。

java.beans.FeatureDescriptor.setName は変換されませんでした。

java.beans.FeatureDescriptor.setPreferred は変換されませんでした。

java.beans.FeatureDescriptor.setShortDescription は変換されませんでした。

java.beans.FeatureDescriptor.setValue は変換されませんでした。

java.beans.IndexedPropertyDescriptor は変換されませんでした。

java.beans.IntrospectionException は変換されませんでした。

java.beans.Introspector は変換されませんでした。

java.beans.MethodDescriptor は変換されませんでした。

java.beans.ParameterDescriptor は変換されませんでした。

java.beans.PropertyChangeEvent.getPropagationId は変換されませんでした。

java.beans.PropertyChangeEvent.PropertyChangeEvent は変換されませんでした。

java.beans.PropertyChangeEvent.setPropagationId は変換されませんでした。

java.beans.PropertyChangeListener は変換されませんでした。

java.beans.PropertyChangeSupport は変換されませんでした。

java.beans.PropertyChangeSupport.addPropertyChangeListener は変換されませんでした。

java.beans.PropertyChangeSupport.firePropertyChange は変換されませんでした。

java.beans.PropertyChangeSupport.hasListeners は変換されませんでした。

java.beans.PropertyChangeSupport.PropertyChangeSupport は変換されませんでした。

java.beans.PropertyChangeSupport.removePropertyChangeListener は変換されませんでした。

java.beans.PropertyDescriptor は変換されませんでした。

java.beans.PropertyDescriptor.getPropertyEditorClass は変換されませんでした。

java.beans.PropertyDescriptor.getReadMethod は変換されませんでした。

java.beans.PropertyDescriptor.getWriteMethod は変換されませんでした。

java.beans.PropertyDescriptor.isBound は変換されませんでした。

java.beans.PropertyDescriptor.isConstrained は変換されませんでした。

java.beans.PropertyDescriptor.PropertyDescriptor は変換されませんでした。

java.beans.PropertyDescriptor.setBound は変換されませんでした。

java.beans.PropertyDescriptor.setConstrained は変換されませんでした。

java.beans.PropertyDescriptor.setPropertyEditorClass は変換されませんでした。

java.beans.PropertyDescriptor.setReadMethod は変換されませんでした。

java.beans.PropertyDescriptor.setWriteMethod は変換されませんでした。

java.beans.PropertyEditor は変換されませんでした。

java.beans.PropertyEditorManager は変換されませんでした。

java.beans.PropertyEditorSupport は変換されませんでした。

java.beans.PropertyVetoException.getPropertyChangeEvent は変換されませんでした。

java.beans.SimpleBeanInfo は変換されませんでした。

java.beans.VetoableChangeListener は変換されませんでした。

java.beans.VetoableChangeSupport は変換されませんでした。

java.beans.VetoableChangeSupport.addVetoableChangeListener は変換されませんでした。

java.beans.VetoableChangeSupport.fireVetoableChange は変換されませんでした。

java.beans.VetoableChangeSupport.hasListeners は変換されませんでした。

java.beans.VetoableChangeSupport.removeVetoableChangeListener は変換されませんでした。

java.beans.VetoableChangeSupport.VetoableChangeSupport は変換されませんでした。

java.beans.Visibility は変換されませんでした。

Java.io のエラー メッセージ

java.io.BufferedInputStream.available は変換されませんでした。

java.io.BufferedInputStream.buf は変換されませんでした。

java.io.BufferedInputStream.marklimit は変換されませんでした。

java.io.BufferedInputStream.markpos は変換されませんでした。

java.io.BufferedOutputStream.buf は変換されませんでした。

java.io.BufferedReader.BufferedReader は変換されませんでした。

java.io.BufferedReader.mark は変換されませんでした。

java.io.BufferedReader.reset は変換されませんでした。

java.io.BufferedWriter.BufferedWriter は変換されませんでした。

java.io.BufferedWriter.write は変換されませんでした。

java.io.ByteArrayInputStream.buf は変換されませんでした。

java.io.ByteArrayInputStream.mark~ は変換されませんでした。

java.io.ByteArrayOutputStream.buf は変換されませんでした。

java.io.ByteArrayOutputStream.reset は変換されませんでした。

java.io.ByteArrayOutputStream.toString は変換されませんでした。

java.io.CharArrayReader.buf は変換されませんでした。

java.io.CharArrayReader.count は変換されませんでした。

java.io.CharArrayReader.mark は変換されませんでした。

java.io.CharArrayReader.markedPos は変換されませんでした。

java.io.CharArrayReader.pos は変換されませんでした。

java.io.CharArrayReader.reset は変換されませんでした。

java.io.CharArrayReader.skip は変換されませんでした。

java.io.CharArrayWriter.reset は変換されませんでした。

java.io.CharArrayWriter.size は変換されませんでした。

java.io.CharArrayWriter.toCharArray は変換されませんでした。

java.io.CharArrayWriter.toString は変換されませんでした。

java.io.CharArrayWriter.write は変換されませんでした。

java.io.CharArrayWriter.writeTo は変換されませんでした。

java.io.DataInput は変換されませんでした。

java.io.DataInput.readLine は変換されませんでした。

java.io.DataInput.readUTF は変換されませんでした。

java.io.DataInputStream は変換されませんでした。

java.io.DataInputStream.readLine は変換されませんでした。

java.io.DataInputStream.readUTF は変換されませんでした。

java.io.DataOutput は変換されませんでした。

java.io.DataOutput.writeBytes は変換されませんでした。

java.io.DataOutput.writeChars は変換されませんでした。

java.io.DataOutput.writeUTF は変換されませんでした。

java.io.DataOutputStream は変換されませんでした。

java.io.DataOutputStream.writeBytes は変換されませんでした。

java.io.DataOutputStream.writeChars は変換されませんでした。

java.io.DataOutputStream.writeUTF は変換されませんでした。

java.io.DataOutputStream.written は変換されませんでした。

java.io.Externalizable は変換されませんでした。

java.io.File.canRead は変換されませんでした。

java.io.File.createTempFile(String, String) は変換されませんでした。

java.io.File.createTempFile(String, String, File) は変換されませんでした。

java.io.File.deleteOnExit は変換されませんでした。

java.io.File.isAbsolute は変換されませんでした。

java.io.File.list は変換されませんでした。

java.io.File.listFiles は変換されませんでした。

java.io.File.mkdir は変換されませんでした。

java.io.File.mkdirs は変換されませんでした。

java.io.File.renameTo は変換されませんでした。

java.io.File.setLastModified は変換されませんでした。

java.io.FileDescriptor は変換されませんでした。

java.io.FileFilter は変換されませんでした。

java.io.FileInputStream.available は変換されませんでした。

java.io.FileInputStream.FileInputStream は変換されませんでした。

java.io.FileInputStream.getFD は変換されませんでした。

java.io.FilenameFilter は変換されませんでした。

java.io.FileOutputStream.FileOutputStream は変換されませんでした。

java.io.FileOutputStream.getFD は変換されませんでした。

java.io.FileOutputStream.write は変換されませんでした。

java.io.FilePermission.FilePermission は変換されませんでした。

java.io.FileReader.FileReader は変換されませんでした。

java.io.FileWriter.FileWriter は変換されませんでした。

java.io.FilterInputStream.available は変換されませんでした。

java.io.FilterInputStream.close は変換されませんでした。

java.io.FileInputStream.FileInputStream は変換されませんでした。

java.io.FilterInputStream.mark は変換されませんでした。

java.io.FilterInputStream.markSupported は変換されませんでした。

java.io.FilterInputStream.reset は変換されませんでした。

java.io.FilterReader.FilterReader は変換されませんでした。

java.io.FilterReader.in は変換されませんでした。

java.io.FilterReader.mark は変換されませんでした。

java.io.FilterReader.markSupported は変換されませんでした。

java.io.FilterReader.reset は変換されませんでした。

java.io.FileWriter は変換されませんでした。

java.io.FileWriter.FileWriter は変換されませんでした。

java.io.FilterWriter.write は変換されませんでした。

java.io.InputStream.available は変換されませんでした。

java.io.InputStream.InputStream は変換されませんでした。

java.io.InputStream.mark は変換されませんでした。

java.io.InputStream.markSupported は変換されませんでした。

java.io.InputStream.read は変換されませんでした。

java.io.InputStream.reset は変換されませんでした。

java.io.InputStreamReader.InputStreamReader は変換されませんでした。

java.io.InputStreamReader.read は変換されませんでした。

java.io.InputStreamReader.read(char[], int, int) は変換されませんでした。

java.io.InterruptedIOException.bytesTransferred は変換されませんでした。

java.io.InvalidClassException.classname は変換されませんでした。

java.io.InvalidClassException.InvalidClassException は変換されませんでした。

java.io.LineNumberInputStream.available は変換されませんでした。

java.io.LineNumberInputStream.getLineNumber は変換されませんでした。

java.io.LineNumberInputStream.mark は変換されませんでした。

java.io.LineNumberInputStream.reset は変換されませんでした。

java.io.LineNumberInputStream.setLineNumber は変換されませんでした。

java.io.LineNumberReader.getLineNumber は変換されませんでした。

java.io.LineNumberReader.LineNumberReader は変換されませんでした。

java.io.LineNumberReader.mark は変換されませんでした。

java.io.LineNumberReader.reset は変換されませんでした。

java.io.LineNumberReader.setLineNumber は変換されませんでした。

java.io.ObjectInput は変換されませんでした。

java.io.ObjectInput.available は変換されませんでした。

java.io.ObjectInput.readObject は変換されませんでした。

java.io.ObjectInputStream は変換されませんでした。

java.io.ObjectInputStream.available は変換されませんでした。

java.io.ObjectInputStream.defaultReadObject は変換されませんでした。

java.io.ObjectInputStream.enableResolveObject は変換されませんでした。

java.io.ObjectInputStream.GetField は変換されませんでした。

java.io.ObjectInputStream.ObjectInputStream は変換されませんでした。

java.io.ObjectInputStream.readClassDescriptor は変換されませんでした。

java.io.ObjectInputStream.readFields は変換されませんでした。

java.io.ObjectInputStream.readLine は変換されませんでした。

java.io.ObjectInputStream.readObjectOverride は変換されませんでした。

java.io.ObjectInputStream.readStreamHeader は変換されませんでした。

java.io.ObjectInputStream.readUTF は変換されませんでした。

java.io.ObjectInputStream.registerValidation は変換されませんでした。

java.io.ObjectInputStream.resolveClass は変換されませんでした。

java.io.ObjectInputStream.resolveObject は変換されませんでした。

java.io.ObjectInputStream.resolveProxyClass は変換されませんでした。

java.io.ObjectInputValidation は変換されませんでした。

java.io.ObjectOutput は変換されませんでした。

java.io.ObjectOutput.writeObject は変換されませんでした。

java.io.ObjectOutputStream は変換されませんでした。

java.io.ObjectOutputStream.annotateClass は変換されませんでした。

java.io.ObjectOutputStream.annotateProxyClass は変換されませんでした。

java.io.ObjectOutputStream.baseWireHandle は変換されませんでした。

java.io.ObjectOutputStream.defaultWriteObject は変換されませんでした。

java.io.ObjectOutputStream.enableReplaceObject は変換されませんでした。

java.io.ObjectOutputStream.ObjectOutputStream は変換されませんでした。

java.io.ObjectOutputStream.PutField は変換されませんでした。

java.io.ObjectOutputStream.putFields は変換されませんでした。

java.io.ObjectOutputStream.replaceObject は変換されませんでした。

java.io.ObjectOutputStream.reset は変換されませんでした。

java.io.ObjectOutputStream.SC_EXTERNALIZABLE は変換されませんでした。

java.io.ObjectOutputStream.SC_SERIALIZABLE は変換されませんでした。

java.io.ObjectOutputStream.SC_WRITE_METHOD は変換されませんでした。

java.io.ObjectOutputStream.useProtocolVersion は変換されませんでした。

java.io.ObjectOutputStream.writeBytes は変換されませんでした。

java.io.ObjectOutputStream.writeChars は変換されませんでした。

java.io.ObjectOutputStream.writeClassDescriptor は変換されませんでした。

java.io.ObjectOutputStream.writeFields は変換されませんでした。

java.io.ObjectOutputStream.writeObject は変換されませんでした。

java.io.ObjectOutputStream.writeObjectOverride は変換されませんでした。

java.io.ObjectOutputStream.writeStreamHeader は変換されませんでした。

java.io.ObjectOutputStream.writeUTF は変換されませんでした。

java.io.ObjectStreamClass は変換されませんでした。

java.io.OptionalDataException は変換されませんでした。

java.io.OutputStreamWriter.getEncoding は変換されませんでした。

java.io.OutputStreamWriter.OutputStreamWriter は変換されませんでした。

java.io.OutputStreamWriter.write は変換されませんでした。

java.io.PipedInputStream.available は変換されませんでした。

java.io.PipedInputStream.buffer は変換されませんでした。

java.io.PipedInputStream.connect は変換されませんでした。

java.io.PipedInputStream.in は変換されませんでした。

java.io.PipedInputStream.out は変換されませんでした。

java.io.PipedInputStream.PIPE_SIZE は変換されませんでした。

java.io.PipedInputStream.PipedInputStream は変換されませんでした。

java.io.PipedInputStream.read は変換されませんでした。

java.io.PipedInputStream.receive は変換されませんでした。

java.io.PipedOutputStream.connect は変換されませんでした。

java.io.PipedOutputStream.PipedOutputStream は変換されませんでした。

java.io.PipedOutputStream.write は変換されませんでした。

java.io.PipedReader.connect は変換されませんでした。

java.io.PipedWriter.connect は変換されませんでした。

java.io.PipedWriter.write は変換されませんでした。

java.io.PrintStream.checkError は変換されませんでした。

java.io.PrintStream.print は変換されませんでした。

java.io.PrintStream.println は変換されませんでした。

java.io.PrintStream.setError は変換されませんでした。

java.io.PrintStream.write は変換されませんでした。

java.io.PrintWriter.checkError は変換されませんでした。

java.io.PrintWriter.print は変換されませんでした。

java.io.PrintWriter.PrintWriter は変換されませんでした。

java.io.PrintWriter.setError は変換されませんでした。

java.io.PrintWriter.write は変換されませんでした。

java.io.PushbackInputStream.available は変換されませんでした。

java.io.RandomAccessFile は変換されませんでした。

java.io.RandomAccessFile.getFD は変換されませんでした。

java.io.RandomAccessFile.RandomAccessFile(File, String) は変換されませんでした。

java.io.RandomAccessFile.RandomAccessFile(File, String, String) は変換されませんでした。

java.io.RandomAccessFile.readUTF は変換されませんでした。

java.io.RandomAccessFile.writeUTF は変換されませんでした。

java.io.Reader.lock は変換されませんでした。

java.io.Reader.mark は変換されませんでした。

java.io.Reader.markSupported は変換されませんでした。

java.io.Reader.read は変換されませんでした。

java.io.Reader.Reader は変換されませんでした。

java.io.Reader.reset は変換されませんでした。

java.io.SequenceInputStream.available は変換されませんでした。

java.io.SequenceInputStream.SequenceInputStream は変換されませんでした。

java.io.SerializablePermission は変換されませんでした。

java.io.StreamTokenizer は変換されませんでした。

java.io.StreamTokenizer.StreamTokenizer は変換されませんでした。

java.io.StringBufferInputStream.available は変換されませんでした。

java.io.StringBufferInputStream.buffer は変換されませんでした。

java.io.StringBufferInputStream.count は変換されませんでした。

java.io.StringBufferInputStream.pos は変換されませんでした。

java.io.StringBufferInputStream.read は変換されませんでした。

java.io.StringBufferInputStream.reset は変換されませんでした。

java.io.StringBufferInputStream.skip は変換されませんでした。

java.io.StringReader.mark は変換されませんでした。

java.io.StringReader.markSupported は変換されませんでした。

java.io.StringReader.ready は変換されませんでした。

java.io.StringReader.reset は変換されませんでした。

java.io.StringReader.skip は変換されませんでした。

java.io.StringWriter.getBuffer は変換されませんでした。

java.io.StringWriter.write は変換されませんでした。

java.io.Writer.lock は変換されませんでした。

java.io.Writer.write は変換されませんでした。

Java.lang のエラー メッセージ

java.lang.Boolean.getBoolean は変換されませんでした。

java.lang.Character.forDigit は変換されませんでした。

java.lang.Character.isDefined は変換されませんでした。

java.lang.Character.isIdentifierIgnorable は変換されませんでした。

java.lang.Character.isJavaIdentifierPart は変換されませんでした。

java.lang.Character.isUnicodeIdentifierPart は変換されませんでした。

java.lang.Character.isUnicodeIdentifierStart は変換されませんでした。

java.lang.Class.forName は変換されませんでした。

java.lang.Class.getClassLoader は変換されませんでした。

java.lang.Class.getDeclaredMethod は変換されませんでした。

java.lang.Class.getModifiers は変換されませんでした。

java.lang.Class.getPackage は変換されませんでした。

java.lang.Class.getProtectionDomain は変換されませんでした。

java.lang.Class.getResource は変換されませんでした。

java.lang.Class.getResourceAsStream は変換されませんでした。

java.lang.Class.getSigners は変換されませんでした。

java.lang.Class.newInstance は変換されませんでした。

java.lang.ClassLoader は変換されませんでした。

java.lang.ClassNotFoundException.printStackTrace は変換されませんでした。

java.lang.Compiler は変換されませんでした。

java.lang.Double.doubleToLongBits は変換されませんでした。

java.lang.Double.doubleToRawLongBits は変換されませんでした。

java.lang.Double.longBitsToDouble は変換されませんでした。

java.lang.ExceptionInInitializerError.getException は変換されませんでした。

java.lang.ExceptionInInitializerError.printStackTrace は変換されませんでした。

java.lang.Float.floatToIntBits は変換されませんでした。

java.lang.Float.floatToRawIntBits は変換されませんでした。

java.lang.Float.intBitsToFloat は変換されませんでした。

java.lang.InheritableThreadLocal は変換されませんでした。

java.lang.Integer.getInteger は変換されませんでした。

java.lang.Integer.TYPE は変換されませんでした。

java.lang.Long.getLong は変換されませんでした。

java.lang.Math.round は変換されませんでした。

java.lang.Number は変換されませんでした。

java.lang.Number.Number は変換されませんでした。

java.lang.Object.class は変換されませんでした。

java.lang.Object.clone は変換されませんでした。

java.lang.Package は変換されませんでした。

java.lang.ref.PhantomReference は変換されませんでした。

java.lang.ref.Reference は変換されませんでした。

java.lang.ref.ReferenceQueue は変換されませんでした。

java.lang.ref.SoftReference は変換されませんでした。

java.lang.ref.WeakReference.WeakReference は変換されませんでした。

java.lang.reflect.AccessibleObject は変換されませんでした。

java.lang.reflect.Constructor.getExceptionTypes は変換されませんでした。

java.lang.reflect.Constructor.newInstance は変換されませんでした。

java.lang.reflect.Field.getModifiers は変換されませんでした。

java.lang.reflect.Field.setByte は変換されませんでした。

java.lang.reflect.Field.setChar は変換されませんでした。

java.lang.reflect.Field.setShort は変換されませんでした。

java.lang.reflect.InvocationHandler は変換されませんでした。

java.lang.reflect.InvocationTargetException.InvocationTargetException は変換されませんでした。

java.lang.reflect.Member.DECLARED は変換されませんでした。

java.lang.reflect.Member.getModifiers は変換されませんでした。

java.lang.reflect.Member.PUBLIC は変換されませんでした。

java.lang.reflect.Method.getExceptionTypes は変換されませんでした。

java.lang.reflect.Method.getModifiers は変換されませんでした。

java.lang.reflect.Modifier は変換されませんでした。

java.lang.reflect.Proxy は変換されませんでした。

java.lang.reflect.ReflectPermission は変換されませんでした。

java.lang.Runtime.addShutdownHook は変換されませんでした。

java.lang.Runtime.exec は変換されませんでした。

java.lang.Runtime.freeMemory は変換されませんでした。

java.lang.Runtime.getLocalizedInputStream は変換されませんでした。

java.lang.Runtime.getLocalizedOutputStream は変換されませんでした。

java.lang.Runtime.halt は変換されませんでした。

java.lang.Runtime.load は変換されませんでした。

java.lang.Runtime.loadLibrary は変換されませんでした。

java.lang.Runtime.removeShutdownHook は変換されませんでした。

java.lang.Runtime.runFinalizersOnExit は変換されませんでした。

java.lang.Runtime.Runtime は変換されませんでした。

java.lang.Runtime.totalMemory は変換されませんでした。

java.lang.Runtime.traceInstructions は変換されませんでした。

java.lang.Runtime.traceMethodCalls は変換されませんでした。

java.lang.RuntimePermission は変換されませんでした。

java.lang.SecurityManager.checkAccess は変換されませんでした。

java.lang.SecurityManager.checkAwtEventQueueAccess は変換されませんでした。

java.lang.SecurityManager.checkConnect は変換されませんでした。

java.lang.SecurityManager.checkCreateClassLoader は変換されませんでした。

java.lang.SecurityManager.checkExec は変換されませんでした。

java.lang.SecurityManager.checkExit は変換されませんでした。

java.lang.SecurityManager.checkLink は変換されませんでした。

java.lang.SecurityManager.checkListen は変換されませんでした。

java.lang.SecurityManager.checkMemberAccess は変換されませんでした。

java.lang.SecurityManager.checkMulticast は変換されませんでした。

java.lang.SecurityManager.checkPackageAccess は変換されませんでした。

java.lang.SecurityManager.checkPackageDefinition は変換されませんでした。

java.lang.SecurityManager.checkPermission(Permission) は変換されませんでした。

java.lang.SecurityManager.checkPermission(Permission, Object) は変換されませんでした。

java.lang.SecurityManager.checkPrintJobAccess は変換されませんでした。

java.lang.SecurityManager.checkPropertiesAccess は変換されませんでした。

java.lang.SecurityManager.checkPropertyAccess は変換されませんでした。

java.lang.SecurityManager.checkRead は変換されませんでした。

java.lang.SecurityManager.checkSecurityAccess は変換されませんでした。

java.lang.SecurityManager.checkSetFactory は変換されませんでした。

java.lang.SecurityManager.checkTopLevelWindow は変換されませんでした。

java.lang.SecurityManager.checkWrite は変換されませんでした。

java.lang.SecurityManager.classDepth は変換されませんでした。

java.lang.SecurityManager.classLoaderDepth は変換されませんでした。

java.lang.SecurityManager.currentClassLoader は変換されませんでした。

java.lang.SecurityManager.currentLoadedClass は変換されませんでした。

java.lang.SecurityManager.getClassContext は変換されませんでした。

java.lang.SecurityManager.getInCheck は変換されませんでした。

java.lang.SecurityManager.getSecurityContext は変換されませんでした。

java.lang.SecurityManager.getThreadGroup は変換されませんでした。

java.lang.SecurityManager.inCheck は変換されませんでした。

java.lang.SecurityManager.inClass は変換されませんでした。

java.lang.SecurityManager.inClassLoader は変換されませんでした。

java.lang.SecurityManager.SecurityManager は変換されませんでした。

java.lang.StackOverflowError は変換されませんでした。

java.lang.StrictMath.round は変換されませんでした。

java.lang.String.CASE_INSENSITIVE_ORDER は変換されませんでした。

java.lang.String.getBytes は変換されませんでした。

java.lang.String.String(byte[]) は変換されませんでした。

java.lang.String.String(byte[]), int, int, int) は変換されませんでした。

java.lang.String.String(byte[], int, int, String) は変換されませんでした。

java.lang.String.String(byte[], String) は変換されませんでした。

java.lang.System は変換されませんでした。

java.lang.System.currentTimeMillis は変換されませんでした。

java.lang.System.getProperties は変換されませんでした。

java.lang.System.getProperty は変換されませんでした。

java.lang.System.getSecurityManager は変換されませんでした。

java.lang.System.load は変換されませんでした。

java.lang.System.loadLibrary は変換されませんでした。

java.lang.System.mapLibraryName は変換されませんでした。

java.lang.System.runFinalization は変換されませんでした。

java.lang.System.runFinalizersOnExit は変換されませんでした。

java.lang.System.setErr は変換されませんでした。

java.lang.System.setIn は変換されませんでした。

java.lang.System.setOut は変換されませんでした。

java.lang.System.setProperty は変換されませんでした。

java.lang.System.setProperties は変換されませんでした。

java.lang.System.setSecurityManager は変換されませんでした。

java.lang.Thread.activeCount は変換されませんでした。

java.lang.Thread.checkAccess は変換されませんでした。

java.lang.Thread.countStackFrames は変換されませんでした。

java.lang.Thread.destroy は変換されませんでした。

java.lang.Thread.enumerate は変換されませんでした。

java.lang.Thread.getContextClassLoader は変換されませんでした。

java.lang.Thread.getThreadGroup は変換されませんでした。

java.lang.Thread.interrupted は変換されませんでした。

java.lang.Thread.isInterrupted は変換されませんでした。

java.lang.Thread.run は変換されませんでした。

java.lang.Thread.setContextClassLoader は変換されませんでした。

java.lang.Thread.sleep(long) は変換されませんでした。

java.lang.Thread.sleep(long, int) は変換されませんでした。

java.lang.Thread.Thread は変換されませんでした。

java.lang.Thread.yield は変換されませんでした。

java.lang.ThreadGroup は変換されませんでした。

java.lang.ThreadLocal.initialValue は変換されませんでした。

java.lang.Throwable.fillInStackTrace は変換されませんでした。

java.lang.UnsupportedClassVersionError は変換されませんでした。

Java.math のエラー メッセージ

java.math.BigDecimal は変換されませんでした。

java.math.BigDecimal.BigDecimal は変換されませんでした。

java.math.BigDecimal.divide は変換されませんでした。

java.math.BigDecimal.movePointLeft は変換されませんでした。

java.math.BigDecimal.movePointRight は変換されませんでした。

java.math.BigDecimal.ROUND_CEILING は変換されませんでした。

java.math.BigDecimal.ROUND_DOWN は変換されませんでした。

java.math.BigDecimal.ROUND_FLOOR は変換されませんでした。

java.math.BigDecimal.ROUND_HALF_DOWN は変換されませんでした。

java.math.BigDecimal.ROUND_HALF_EVEN は変換されませんでした。

java.math.BigDecimal.ROUND_HALF_UP は変換されませんでした。

java.math.BigDecimal.ROUND_UNNECESSARY は変換されませんでした。

java.math.BigDecimal.ROUND_UP は変換されませんでした。

java.math.BigDecimal.scale は変換されませんでした。

java.math.BigDecimal.setScale は変換されませんでした。

java.math.BigDecimal.toBigInteger は変換されませんでした。

java.math.BigDecimal.unscaledValue は変換されませんでした。

java.math.BigDecimal.valueOf は変換されませんでした。

java.math.BigInteger は変換されませんでした。

java.math.BigInteger.and は変換されませんでした。

java.math.BigInteger.andNot は変換されませんでした。

java.math.BigInteger.BigInteger(byte[]) は変換されませんでした。

java.math.BigInteger.BigInteger(int, byte[]) は変換されませんでした。

java.math.BigInteger.BigInteger(int, int, Random) は変換されませんでした。

java.math.BigInteger.BigInteger(int, Random) は変換されませんでした。

java.math.BigInteger.BigInteger(String, int) は変換されませんでした。

java.math.BigInteger.bitCount は変換されませんでした。

java.math.BigInteger.bitLength は変換されませんでした。

java.math.BigInteger.clearBit は変換されませんでした。

java.math.BigInteger.flipBit は変換されませんでした。

java.math.BigInteger.gcd は変換されませんでした。

java.math.BigInteger.getLowestSetBit は変換されませんでした。

java.math.BigInteger.isProbablePrime は変換されませんでした。

java.math.BigInteger.modInverse は変換されませんでした。

java.math.BigInteger.modPow は変換されませんでした。

java.math.BigInteger.not は変換されませんでした。

java.math.BigInteger.or は変換されませんでした。

java.math.BigInteger.pow は変換されませんでした。

java.math.BigInteger.setBit は変換されませんでした。

java.math.BigInteger.shiftLeft は変換されませんでした。

java.math.BigInteger.shiftRight は変換されませんでした。

java.math.BigInteger.testBit は変換されませんでした。

java.math.BigInteger.toByteArray は変換されませんでした。

java.math.BigInteger.toString は変換されませんでした。

java.math.BigInteger.xor は変換されませんでした。

Java.net のエラー メッセージ

java.net.Authenticator は変換されませんでした。

java.net.ContentHandler は変換されませんでした。

java.net.ContentHandlerFactory は変換されませんでした。

java.net.DatagramPacket.DatagramPacket は変換されませんでした。

java.net.DatagramPacket.getOffset は変換されませんでした。

java.net.DatagramSocket.getLocalAddress は変換されませんでした。

java.net.DatagramSocket.getLocalPort は変換されませんでした。

java.net.DatagramSocket.getReceiveBufferSize は変換されませんでした。

java.net.DatagramSocket.getSendBufferSize は変換されませんでした。

java.net.DatagramSocket.getSoTimeout は変換されませんでした。

java.net.DatagramSocket.setDatagramSocketImplFactory は変換されませんでした。

java.net.DatagramSocket.setReceiveBufferSize は変換されませんでした。

java.net.DatagramSocket.setSendBufferSize は変換されませんでした。

java.net.DatagramSocket.setSoTimeout は変換されませんでした。

java.net.DatagramSocketImpl は変換されませんでした。

java.net.DatagramSocketImplFactory は変換されませんでした。

java.net.FileNameMap は変換されませんでした。

java.net.HttpURLConnection.getErrorStream は変換されませんでした。

java.net.HttpURLConnection.getFollowRedirects は変換されませんでした。

java.net.HttpURLConnection.getPermission は変換されませんでした。

java.net.HttpURLConnection.getResponseCode は変換されませんでした。

java.net.HttpURLConnection.getResponseMessage は変換されませんでした。

java.net.HttpURLConnection.HTTP_MOVED_TEMP は変換されませんでした。

java.net.HttpURLConnection.responseCode は変換されませんでした。

java.net.HttpURLConnection.responseMessage は変換されませんでした。

java.net.HttpURLConnection.usingProxy は変換されませんでした。

java.net.InetAddress.getAddress は変換されませんでした。

java.net.InetAddress.getAllByName は変換されませんでした。

java.net.InetAddress.getByName は変換されませんでした。

java.net.InetAddress.getLocalHost は変換されませんでした。

java.net.InetAddress.isMulticastAddress は変換されませんでした。

java.net.JarURLConnection は変換されませんでした。

java.net.MulticastSocket.getInterface は変換されませんでした。

java.net.MulticastSocket.getTimeToLive は変換されませんでした。

java.net.MulticastSocket.getTTL は変換されませんでした。

java.net.MulticastSocket.send は変換されませんでした。

java.net.MulticastSocket.setInterface は変換されませんでした。

java.net.MulticastSocket.setTimeToLive は変換されませんでした。

java.net.MulticastSocket.setTTL は変換されませんでした。

java.net.NetPermission.NetPermission は変換されませんでした。

java.net.NoRouteToHostException.NoRouteToHostException は変換されませんでした。

java.net.PlainDatagramSocketImpl は変換されませんでした。

java.net.PlainSocketImpl は変換されませんでした。

java.net.ServerSocket.getSoTimeout は変換されませんでした。

java.net.ServerSocket.implAccept は変換されませんでした。

java.net.ServerSocket.ServerSocket は変換されませんでした。

java.net.ServerSocket.setSocketFactory は変換されませんでした。

java.net.ServerSocket.setSoTimeout は変換されませんでした。

java.net.Socket.getInetAddress は変換されませんでした。

java.net.Socket.getKeepAlive は変換されませんでした。

java.net.Socket.getLocalAddress は変換されませんでした。

java.net.Socket.getLocalPort は変換されませんでした。

java.net.Socket.getPort は変換されませんでした。

java.net.Socket.setKeepAlive は変換されませんでした。

java.net.Socket.setSocketImplFactory は変換されませんでした。

java.net.Socket.shutdownInput は変換されませんでした。

java.net.Socket.shutdownOutput は変換されませんでした。

java.net.Socket.Socket は変換されませんでした。

java.net.SocketException.SocketException は変換されませんでした。

java.net.SocketImpl は変換されませんでした。

java.net.SocketImplFactory は変換されませんでした。

java.net.SocketInputStream は変換されませんでした。

java.net.SocketOptions は変換されませんでした。

java.net.SocketOutputStream は変換されませんでした。

java.net.SocketPermission.SocketPermission は変換されませんでした。

java.net.UnknownContentHandler は変換されませんでした。

java.net.URL.getContent は変換されませんでした。

java.net.URL.getContent(Class[]) は変換されませんでした。

java.net.URL.getPort は変換されませんでした。

java.net.URL.openStream は変換されませんでした。

java.net.URL.set は変換されませんでした。

java.net.URL.setURLStreamHandlerFactory は変換されませんでした。

java.net.URL.URL は変換されませんでした。

java.net.URLClassLoader は変換されませんでした。

java.net.URLConnection.allowUserInteraction は変換されませんでした。

java.net.URLConnection.connect は変換されませんでした。

java.net.URLConnection.doInput は変換されませんでした。

java.net.URLConnection.doOutput は変換されませんでした。

java.net.URLConnection.fileNameMap は変換されませんでした。

java.net.URLConnection.getAllowUserInteraction は変換されませんでした。

java.net.URLConnection.getContent は変換されませんでした。

java.net.URLConnection.getContentEncoding は変換されませんでした。

java.net.URLConnection.getDate は変換されませんでした。

java.net.URLConnection.getDefaultAllowUserInteraction は変換されませんでした。

java.net.URLConnection.getDefaultRequestProperty は変換されませんでした。

java.net.URLConnection.getDefaultUseCaches は変換されませんでした。

java.net.URLConnection.getDoInput は変換されませんでした。

java.net.URLConnection.getDoOutput は変換されませんでした。

java.net.URLConnection.getFileNameMap は変換されませんでした。

java.net.URLConnection.getHeaderField は変換されませんでした。

java.net.URLConnection.getHeaderFieldKey は変換されませんでした。

java.net.URLConnection.getIfModifiedSince は変換されませんでした。

java.net.URLConnection.getLastModified は変換されませんでした。

java.net.URLConnection.getPermission は変換されませんでした。

java.net.URLConnection.getRequestProperty は変換されませんでした。

java.net.URLConnection.getUseCaches は変換されませんでした。

java.net.URLConnection.guessContentTypeFromName は変換されませんでした。

java.net.URLConnection.guessContentTypeFromStream は変換されませんでした。

java.net.URLConnection.setAllowUserInteraction は変換されませんでした。

java.net.URLConnection.setContentHandlerFactory は変換されませんでした。

java.net.URLConnection.setDefaultAllowUserInteraction は変換されませんでした。

java.net.URLConnection.setDefaultRequestProperty は変換されませんでした。

java.net.URLConnection.setDefaultUseCaches は変換されませんでした。

java.net.URLConnection.setDoInput は変換されませんでした。

java.net.URLConnection.setDoOutput は変換されませんでした。

java.net.URLConnection.setFileNameMap は変換されませんでした。

java.net.URLConnection.setUseCaches は変換されませんでした。

java.net.URLConnection.toString は変換されませんでした。

java.net.URLConnection.url は変換されませんでした。

java.net.URLConnection.useCaches は変換されませんでした。

java.net.URLDecoder は変換されませんでした。

java.net.URLEncoder は変換されませんでした。

java.net.URLStreamHandler は変換されませんでした。

java.net.URLStreamHandlerFactory は変換されませんでした。

Java.rmi のエラー メッセージ

java.rmi.AccessException は変換されませんでした。

java.rmi.activation.Activatable.Activatable は変換されませんでした。

java.rmi.activation.Activatable.exportObject は変換されませんでした。

java.rmi.activation.Activatable.getID は変換されませんでした。

java.rmi.activation.Activatable.inactive は変換されませんでした。

java.rmi.activation.Activatable.register は変換されませんでした。

java.rmi.activation.Activatable.unexportObject は変換されませんでした。

java.rmi.activation.Activatable.unregister は変換されませんでした。

java.rmi.activation.ActivateFailedException は変換されませんでした。

java.rmi.activation.ActivationDesc は変換されませんでした。

java.rmi.activation.ActivationException は変換されませんでした。

java.rmi.activation.ActivationGroup は変換されませんでした。

java.rmi.activation.ActivationGroup_Stub は変換されませんでした。

java.rmi.activation.ActivationGroupDesc は変換されませんでした。

java.rmi.activation.ActivationGroupDesc.CommandEnvironment は変換されませんでした。

java.rmi.activation.ActivationGroupID は変換されませんでした。

java.rmi.activation.ActivationID は変換されませんでした。

java.rmi.activation.ActivationInstantiator は変換されませんでした。

java.rmi.activation.ActivationMonitor は変換されませんでした。

java.rmi.activation.ActivationSystem は変換されませんでした。

java.rmi.activation.Activator.activate は変換されませんでした。

java.rmi.activation.CommandEnvironment.CommandEnvironment は変換されませんでした。

java.rmi.activation.CommandEnvironment.equals は変換されませんでした。

java.rmi.activation.CommandEnvironment.getCommandOptions は変換されませんでした。

java.rmi.activation.CommandEnvironment.getCommandPath は変換されませんでした。

java.rmi.activation.CommandEnvironment.hashCode は変換されませんでした。

java.rmi.activation.UnknownGroupException は変換されませんでした。

java.rmi.activation.UnknownObjectException は変換されませんでした。

java.rmi.AlreadyBoundException は変換されませんでした。

java.rmi.ConnectException は変換されませんでした。

java.rmi.ConnectIOException は変換されませんでした。

java.rmi.dgc.DGC は変換されませんでした。

java.rmi.dgc.Lease.getVMID は変換されませんでした。

java.rmi.dgc.VMID は変換されませんでした。

java.rmi.MarshalException は変換されませんでした。

java.rmi.MarshalledObject は変換されませんでした。

java.rmi.Naming は変換されませんでした。

java.rmi.Naming.bind は変換されませんでした。

java.rmi.Naming.lookup は変換されませんでした。

java.rmi.Naming.rebind は変換されませんでした。

java.rmi.Naming.unbind は変換されませんでした。

java.rmi.NoSuchObjectException は変換されませんでした。

java.rmi.NotBoundException は変換されませんでした。

java.rmi.registry.LocateRegistry は変換されませんでした。

java.rmi.registry.Registry は変換されませんでした。

java.rmi.registry.Registry.bind は変換されませんでした。

java.rmi.registry.Registry.lookup は変換されませんでした。

java.rmi.registry.Registry.rebind は変換されませんでした。

java.rmi.registry.Registry.REGISTRY_PORT は変換されませんでした。

java.rmi.registry.Registry.unbind は変換されませんでした。

java.rmi.registry.RegistryHandler は変換されませんでした。

java.rmi.RemoteException.detail は変換されませんでした。

java.rmi.RMISecurityException は変換されませんでした。

java.rmi.RMISecurityManager は変換されませんでした。

java.rmi.RMISecurityManager.RMISecurityManager は変換されませんでした。

java.rmi.server.ExportException は変換されませんでした。

java.rmi.server.LoaderHandler は変換されませんでした。

java.rmi.server.LogStream は変換されませんでした。

java.rmi.server.ObjID は変換されませんでした。

java.rmi.server.Operation は変換されませんでした。

java.rmi.server.RemoteCall は変換されませんでした。

java.rmi.server.RemoteRef は変換されませんでした。

java.rmi.server.RemoteServer は変換されませんでした。

java.rmi.server.RemoteStub は変換されませんでした。

java.rmi.server.RMIClassLoader は変換されませんでした。

java.rmi.server.RMIFailureHandler は変換されませんでした。

java.rmi.server.RMISocketFactory.getDefaultSocketFactory は変換されませんでした。

java.rmi.server.RMISocketFactory.getFailureHandler は変換されませんでした。

java.rmi.server.RMISocketFactory.getSocketFactory は変換されませんでした。

java.rmi.server.RMISocketFactory.setFailureHandler は変換されませんでした。

java.rmi.server.RMISocketFactory.setSocketFactory は変換されませんでした。

java.rmi.server.RemoteObject.getRef は変換されませんでした。

java.rmi.server.RemoteObject.ref は変換されませんでした。

java.rmi.server.RemoteObject.RemoteObject は変換されませんでした。

java.rmi.server.RemoteObject.toStub は変換されませんでした。

java.rmi.server.RemoteServer.getClientHost は変換されませんでした。

java.rmi.server.RemoteServer.getLog は変換されませんでした。

java.rmi.server.RemoteServer.RemoteServer は変換されませんでした。

java.rmi.server.ServerCloneException は変換されませんでした。

java.rmi.server.ServerNotActiveException は変換されませんでした。

java.rmi.server.ServerRef は変換されませんでした。

java.rmi.server.Skeleton は変換されませんでした。

java.rmi.server.SkeletonMismatchException は変換されませんでした。

java.rmi.server.SkeletonNotFoundException は変換されませんでした。

java.rmi.server.SocketSecurityException は変換されませんでした。

java.rmi.server.UID は変換されませんでした。

java.rmi.server.UnicastRemoteObject.clone は変換されませんでした。

java.rmi.server.UnicastRemoteObject.exportObject は変換されませんでした。

java.rmi.server.UnicastRemoteObject.unexportObject は変換されませんでした。

java.rmi.server.UnicastRemoteObject.UnicastRemoteObject は変換されませんでした。

java.rmi.server.Unreferenced は変換されませんでした。

java.rmi.ServerRuntimeException は変換されませんでした。

java.rmi.StubNotFoundException は変換されませんでした。

java.rmi.UnexpectedException は変換されませんでした。

java.rmi.UnknownHostException は変換されませんでした。

java.rmi.UnmarshalException は変換されませんでした。

Java.security のエラー メッセージ

java.security.AccessControlContext は変換されませんでした。

java.security.AccessControlException.getPermission は変換されませんでした。

java.security.AccessController は変換されませんでした。

java.security.acl は変換されませんでした。

java.security.acl.Acl は変換されませんでした。

java.security.acl.AclEntry は変換されませんでした。

java.security.acl.AclNotFoundException は変換されませんでした。

java.security.acl.Group は変換されませんでした。

java.security.acl.LastOwnerException は変換されませんでした。

java.security.acl.NotOwnerException は変換されませんでした。

java.security.acl.Owner は変換されませんでした。

java.security.acl.Permission は変換されませんでした。

java.security.AlgorithmParameterGenerator は変換されませんでした。

java.security.AlgorithmParameterGeneratorSpi は変換されませんでした。

java.security.AlgorithmParameters は変換されませんでした。

java.security.AlgorithmParametersSpi は変換されませんでした。

java.security.AllPermission は変換されませんでした。

java.security.BasicPermission.newPermissionCollection は変換されませんでした。

java.security.cert.Certificate.Certificate は変換されませんでした。

java.security.cert.Certificate.getEncoded は変換されませんでした。

java.security.cert.Certificate.verify(PublicKey) は変換されませんでした。

java.security.cert.Certificate.verify(PublicKey, String) は変換されませんでした。

java.security.cert.Certificate.writeReplace は変換されませんでした。

java.security.cert.CertificateFactory.CertificateFactory は変換されませんでした。

java.security.cert.CertificateFactory.generateCertificate は変換されませんでした。

java.security.cert.CertificateFactory.generateCertificates は変換されませんでした。

java.security.cert.CertificateFactory.generateCRL は変換されませんでした。

java.security.cert.CertificateFactory.generateCRLs は変換されませんでした。

java.security.cert.CertificateFactory.getInstance は変換されませんでした。

java.security.cert.CertificateFactory.getProvider は変換されませんでした。

java.security.cert.CertificateFactorySpi は変換されませんでした。

java.security.cert.CRL は変換されませんでした。

java.security.cert.X509Certificate.getBasicConstraints は変換されませんでした。

java.security.cert.X509Certificate.getCriticalExtensionOIDs は変換されませんでした。

java.security.cert.X509Certificate.getExtensionValue は変換されませんでした。

java.security.cert.X509Certificate.getIssuerUniqueID は変換されませんでした。

java.security.cert.X509Certificate.getKeyUsage は変換されませんでした。

java.security.cert.X509Certificate.getNonCriticalExtensionOIDs は変換されませんでした。

java.security.cert.X509Certificate.getNotAfter は変換されませんでした。

java.security.cert.X509Certificate.getNotBefore は変換されませんでした。

java.security.cert.X509CRLEntry.getRevocationDate は変換されませんでした。

java.security.cert.X509Certificate.getSigAlgName は変換されませんでした。

java.security.cert.X509Certificate.getSignature は変換されませんでした。

java.security.cert.X509Certificate.getSubjectUniqueID は変換されませんでした。

java.security.cert.X509Certificate.getTBSCertificate は変換されませんでした。

java.security.cert.X509Certificate.getVersion は変換されませんでした。

java.security.cert.X509Certificate.hasUnsupportedCriticalExtension は変換されませんでした。

java.security.cert.X509CRL は変換されませんでした。

java.security.cert.X509CRLEntry.getCriticalExtensionOIDs は変換されませんでした。

java.security.cert.X509CRLEntry.getEncoded は変換されませんでした。

java.security.cert.X509CRLEntry.getExtensionValue は変換されませんでした。

java.security.cert.X509CRLEntry.getNonCriticalExtensionOIDs は変換されませんでした。

java.security.cert.X509CRLEntry.hasExtensions は変換されませんでした。

java.security.cert.X509CRLEntry.hasUnsupportedCriticalExtension は変換されませんでした。

java.security.cert.X509Extension は変換されませんでした。

java.security.Certificate は変換されませんでした。

java.security.CodeSource は変換されませんでした。

java.security.DomainCombiner は変換されませんでした。

java.security.Guard は変換されませんでした。

java.security.GuardedObject は変換されませんでした。

java.security.Identity は変換されませんでした。

java.security.IdentityScope は変換されませんでした。

java.security.interfaces.DSAKeyPairGenerator は変換されませんでした。

java.security.interfaces.DSAParams.getG は変換されませんでした。

java.security.interfaces.DSAParams.getP は変換されませんでした。

java.security.interfaces.DSAParams.getQ は変換されませんでした。

java.security.interfaces.DSAPrivateKey は変換されませんでした。

java.security.interfaces.DSAPublicKey は変換されませんでした。

java.security.interfaces.RSAPrivateCrtKey は変換されませんでした。

java.security.interfaces.RSAPrivateKey は変換されませんでした。

java.security.interfaces.RSAPublicKey は変換されませんでした。

java.security.Key.getFormat は変換されませんでした。

java.security.KeyFactory は変換されませんでした。

java.security.KeyFactorySpi は変換されませんでした。

java.security.KeyPairGenerator は変換されませんでした。

java.security.KeyPairGeneratorSpi は変換されませんでした。

java.security.KeyStore は変換されませんでした。

java.security.KeyStoreSpi は変換されませんでした。

java.security.MessageDigest.clone は変換されませんでした。

java.security.MessageDigest.getInstance は変換されませんでした。

java.security.MessageDigestSpi.clone は変換されませんでした。

java.security.MessageDigestSpi.MessageDigestSpi は変換されませんでした。

java.security.Permission.checkGuard は変換されませんでした。

java.security.Permission.getName は変換されませんでした。

java.security.PermissionCollection.implies は変換されませんでした。

java.security.PermissionCollection.setReadOnly は変換されませんでした。

java.security.Policy は変換されませんでした。

java.security.ProtectionDomain は変換されませんでした。

java.security.Provider は変換されませんでした。

java.security.SecureClassLoader は変換されませんでした。

java.security.SecureRandom.getProvider は変換されませんでした。

java.security.SecureRandom.next は変換されませんでした。

java.security.SecureRandomSpi.SecureRandomSpi は変換されませんでした。

java.security.Security は変換されませんでした。

java.security.SecurityPermission.SecurityPermission は変換されませんでした。

java.security.Signature.clone は変換されませんでした。

java.security.Signature.engineGetParameter は変換されませんでした。

java.security.Signature.engineInitSign は変換されませんでした。

java.security.Signature.engineInitVerify は変換されませんでした。

java.security.Signature.engineSetParameter は変換されませんでした。

java.security.Signature.engineSign は変換されませんでした。

java.security.Signature.engineVerify は変換されませんでした。

java.security.Signature.getInstance は変換されませんでした。

java.security.Signature.getParameter は変換されませんでした。

java.security.Signature.initSign は変換されませんでした。

java.security.Signature.initVerify は変換されませんでした。

java.security.Signature.setParameter は変換されませんでした。

java.security.Signature.sign は変換されませんでした。

java.security.Signature.SIGN~ は変換されませんでした。

java.security.Signature.state は変換されませんでした。

java.security.Signature.UNINITIALIZED は変換されませんでした。

java.security.Signature.verify は変換されませんでした。

java.security.Signature.VERIFY~ は変換されませんでした。

java.security.SignatureSpi.appRandom は変換されませんでした。

java.security.SignatureSpi.clone は変換されませんでした。

java.security.SignatureSpi.engineGetParameter は変換されませんでした。

java.security.SignatureSpi.engineInitSign は変換されませんでした。

java.security.SignatureSpi.engineSetParameter は変換されませんでした。

java.security.SignatureSpi.engineSign は変換されませんでした。

java.security.SignatureSpi.engineSign(byte[], int, int) は変換されませんでした。

java.security.SignatureSpi.engineVerify は変換されませんでした。

java.security.SecureRandomSpi.SecureRandomSpi は変換されませんでした。

java.security.SignedObject は変換されませんでした。

java.security.Signer は変換されませんでした。

java.security.spec.DSAParameterSpec.DSAParameterSpec は変換されませんでした。

java.security.spec.DSAPrivateKeySpec.DSAPrivateKeySpec は変換されませんでした。

java.security.spec.DSAPublicKeySpec.DSAPublicKeySpec は変換されませんでした。

java.security.spec.EncodedKeySpec は変換されませんでした。

java.security.spec.PKCS8EncodedKeySpec は変換されませんでした。

java.security.spec.RSAKeyGenParameterSpec は変換されませんでした。

java.security.spec.RSAPrivateCrtKeySpec は変換されませんでした。

java.security.spec.RSAPrivateKeySpec.RSAPrivateKeySpec は変換されませんでした。

java.security.spec.RSAPublicKeySpec.RSAPublicKeySpec は変換されませんでした。

java.security.spec.X509EncodedKeySpec は変換されませんでした。

java.security.UnresolvedPermission は変換されませんでした。

Java.sql のエラー メッセージ

java.sql.Array を変換できませんでした。

java.sql.BatchUpdateException は変換されませんでした。

java.sql.Blob は変換されませんでした。

java.sql.Blob.getBinaryStream を変換できませんでした。

java.sql.Blob.getBytes は変換されませんでした。

java.sql.Blob.position は変換されませんでした。

java.sql.CallableStatement.getArray は変換されませんでした。

java.sql.CallableStatement.getBigDecimal は変換されませんでした。

java.sql.CallableStatement.getBlob は変換されませんでした。

java.sql.CallableStatement.getClob は変換されませんでした。

java.sql.CallableStatement.getDate は変換されませんでした。

java.sql.CallableStatement.getObject は変換されませんでした。

java.sql.CallableStatement.getRef は変換されませんでした。

java.sql.CallableStatement.getTime は変換されませんでした。

java.sql.CallableStatement.getTimestamp は変換されませんでした。

java.sql.CallableStatement.registerOutParameter は変換されませんでした。

java.sql.CallableStatement.wasNull は変換されませんでした。

java.sql.Clob は変換されませんでした。

java.sql.Clob.getAsciiStream は変換されませんでした。

java.sql.Clob.getCharacterStream は変換されませんでした。

java.sql.Clob.getSubString は変換されませんでした。

java.sql.Clob.position は変換されませんでした。

java.sql.Connection.clearWarnings は変換されませんでした。

java.sql.Connection.createStatement は変換されませんでした。

java.sql.Connection.getTypeMap は変換されませんでした。

java.sql.Connection.getWarnings は変換されませんでした。

java.sql.Connection.isReadOnly は変換されませんでした。

java.sql.Connection.nativeSQL は変換されませんでした。

java.sql.Connection.prepareCall は変換されませんでした。

java.sql.Connection.prepareStatement は変換されませんでした。

java.sql.Connection.setCatalog は変換されませんでした。

java.sql.Connection.setReadOnly は変換されませんでした。

java.sql.Connection.setTypeMap は変換されませんでした。

java.sql.DatabaseMetaData.allProceduresAreCallable は変換されませんでした。

java.sql.DatabaseMetaData.allTablesAreSelectable は変換されませんでした。

java.sql.DatabaseMetaData.bestRowNotPseudo は変換されませんでした。

java.sql.DatabaseMetaData.bestRowPseudo は変換されませんでした。

java.sql.DatabaseMetaData.bestRowSession は変換されませんでした。

java.sql.DatabaseMetaData.bestRowTemporary は変換されませんでした。

java.sql.DatabaseMetaData.bestRowTransaction は変換されませんでした。

java.sql.DatabaseMetaData.bestRowUnknown は変換されませんでした。

java.sql.DatabaseMetaData.columnNoNulls は変換されませんでした。

java.sql.DatabaseMetaData.columnNullable は変換されませんでした。

java.sql.DatabaseMetaData.columnNullableUnknown は変換されませんでした。

java.sql.DatabaseMetaData.dataDefinitionCausesTransactionCommit は変換されませんでした。

java.sql.DatabaseMetaData.dataDefinitionIgnoredInTransactions は変換されませんでした。

java.sql.DatabaseMetaData.deletesAreDetected は変換されませんでした。

java.sql.DatabaseMetaData.doesMaxRowSizeIncludeBlobs は変換されませんでした。

java.sql.DatabaseMetaData.getBestRowIdentifier は変換されませんでした。

java.sql.DatabaseMetaData.getCatalogTerm は変換されませんでした。

java.sql.DatabaseMetaData.getColumnPrivileges は変換されませんでした。

java.sql.DatabaseMetaData.getColumns は変換されませんでした。

java.sql.DatabaseMetaData.getCrossReference は変換されませんでした。

java.sql.DatabaseMetaData.getDatabaseProductName は変換されませんでした。

java.sql.DatabaseMetaData.getDriverMajorVersion は変換されませんでした。

java.sql.DatabaseMetaData.getDriverMinorVersion は変換されませんでした。

java.sql.DatabaseMetaData.getDriverName は変換されませんでした。

java.sql.DatabaseMetaData.getDriverVersion は変換されませんでした。

java.sql.DatabaseMetaData.getExportedKeys は変換されませんでした。

java.sql.DatabaseMetaData.getExtraNameCharacters は変換されませんでした。

java.sql.DatabaseMetaData.getIdentifierQuoteString は変換されませんでした。

java.sql.DatabaseMetaData.getImportedKeys は変換されませんでした。

java.sql.DatabaseMetaData.getIndexInfo は変換されませんでした。

java.sql.DatabaseMetaData.getMaxColumnsInGroupBy は変換されませんでした。

java.sql.DatabaseMetaData.getMaxColumnsInIndex は変換されませんでした。

java.sql.DatabaseMetaData.getMaxColumnsInOrderBy は変換されませんでした。

java.sql.DatabaseMetaData.getMaxColumnsInSelect は変換されませんでした。

java.sql.DatabaseMetaData.getMaxColumnsInTable は変換されませんでした。

java.sql.DatabaseMetaData.getMaxConnections は変換されませんでした。

java.sql.DatabaseMetaData.getMaxIndexLength は変換されませんでした。

java.sql.DatabaseMetaData.getMaxRowSize は変換されませんでした。

java.sql.DatabaseMetaData.getMaxStatementLength は変換されませんでした。

java.sql.DatabaseMetaData.getMaxStatements は変換されませんでした。

java.sql.DatabaseMetaData.getMaxTablesInSelect は変換されませんでした。

java.sql.DatabaseMetaData.getNumericFunctions は変換されませんでした。

java.sql.DatabaseMetaData.getPrimaryKeys は変換されませんでした。

java.sql.DatabaseMetaData.getProcedureColumns は変換されませんでした。

java.sql.DatabaseMetaData.getProcedures は変換されませんでした。

java.sql.DatabaseMetaData.getProcedureTerm は変換されませんでした。

java.sql.DatabaseMetaData.getSchemaTerm は変換されませんでした。

java.sql.DatabaseMetaData.getSearchStringEscape は変換されませんでした。

java.sql.DatabaseMetaData.getSQLKeywords は変換されませんでした。

java.sql.DatabaseMetaData.getStringFunctions は変換されませんでした。

java.sql.DatabaseMetaData.getSystemFunctions は変換されませんでした。

java.sql.DatabaseMetaData.getTablePrivileges は変換されませんでした。

java.sql.DatabaseMetaData.getTables は変換されませんでした。

java.sql.DatabaseMetaData.getTimeDateFunctions は変換されませんでした。

java.sql.DatabaseMetaData.getUDTs を変換できませんでした。

java.sql.DatabaseMetaData.getURL は変換されませんでした。

java.sql.DatabaseMetaData.getUserName は変換されませんでした。

java.sql.DatabaseMetaData.getVersionColumns は変換されませんでした。

java.sql.DatabaseMetaData.importedKeyCascade は変換されませんでした。

java.sql.DatabaseMetaData.importedKeyInitiallyDeferred は変換されませんでした。

java.sql.DatabaseMetaData.importedKeyInitiallyImmediate は変換されませんでした。

java.sql.DatabaseMetaData.importedKeyNoAction は変換されませんでした。

java.sql.DatabaseMetaData.importedKeyNotDeferrable は変換されませんでした。

java.sql.DatabaseMetaData.importedKeyRestrict は変換されませんでした。

java.sql.DatabaseMetaData.importedKeySetDefault は変換されませんでした。

java.sql.DatabaseMetaData.importedKeySetNull は変換されませんでした。

java.sql.DatabaseMetaData.insertsAreDetected は変換されませんでした。

java.sql.DatabaseMetaData.isCatalogAtStart は変換されませんでした。

java.sql.DatabaseMetaData.isReadOnly は変換されませんでした。

java.sql.DatabaseMetaData.nullPlusNonNullIsNull は変換されませんでした。

java.sql.DatabaseMetaData.nullsAreSortedAtEnd は変換されませんでした。

java.sql.DatabaseMetaData.nullsAreSortedAtStart は変換されませんでした。

java.sql.DatabaseMetaData.nullsAreSortedHigh は変換されませんでした。

java.sql.DatabaseMetaData.nullsAreSortedLow は変換されませんでした。

java.sql.DatabaseMetaData.othersDeletesAreVisible は変換されませんでした。

java.sql.DatabaseMetaData.othersInsertsAreVisible は変換されませんでした。

java.sql.DatabaseMetaData.othersUpdatesAreVisible は変換されませんでした。

java.sql.DatabaseMetaData.ownDeletesAreVisible は変換されませんでした。

java.sql.DatabaseMetaData.ownInsertsAreVisible は変換されませんでした。

java.sql.DatabaseMetaData.ownUpdatesAreVisible は変換されませんでした。

java.sql.DatabaseMetaData.procedureColumnIn は変換されませんでした。

java.sql.DatabaseMetaData.procedureColumnInOut は変換されませんでした。

java.sql.DatabaseMetaData.procedureColumnOut は変換されませんでした。

java.sql.DatabaseMetaData.procedureColumnResult は変換されませんでした。

java.sql.DatabaseMetaData.procedureColumnReturn は変換されませんでした。

java.sql.DatabaseMetaData.procedureColumnUnknown は変換されませんでした。

java.sql.DatabaseMetaData.procedureNoNulls は変換されませんでした。

java.sql.DatabaseMetaData.procedureNoResult は変換されませんでした。

java.sql.DatabaseMetaData.procedureNullable は変換されませんでした。

java.sql.DatabaseMetaData.procedureNullableUnknown は変換されませんでした。

java.sql.DatabaseMetaData.procedureResultUnknown は変換されませんでした。

java.sql.DatabaseMetaData.procedureReturnsResult は変換されませんでした。

java.sql.DatabaseMetaData.storesLowerCaseIdentifiers は変換されませんでした。

java.sql.DatabaseMetaData.storesLowerCaseQuotedIdentifiers は変換されませんでした。

java.sql.DatabaseMetaData.storesMixedCaseIdentifiers は変換されませんでした。

java.sql.DatabaseMetaData.storesMixedCaseQuotedIdentifiers は変換されませんでした。

java.sql.DatabaseMetaData.storesUpperCaseIdentifiers は変換されませんでした。

java.sql.DatabaseMetaData.storesUpperCaseQuotedIdentifiers は変換されませんでした。

java.sql.DatabaseMetaData.supportsAlterTableWithAddColumn は変換されませんでした。

java.sql.DatabaseMetaData.supportsAlterTableWithDropColumn は変換されませんでした。

java.sql.DatabaseMetaData.supportsANSI92EntryLevelSQL は変換されませんでした。

java.sql.DatabaseMetaData.supportsANSI92FullSQL は変換されませんでした。

java.sql.DatabaseMetaData.supportsANSI92IntermediateSQL は変換されませんでした。

java.sql.DatabaseMetaData.supportsBatchUpdates は変換されませんでした。

java.sql.DatabaseMetaData.supportsCatalogsInDataManipulation は変換されませんでした。

java.sql.DatabaseMetaData.supportsCatalogsInIndexDefinitions は変換されませんでした。

java.sql.DatabaseMetaData.supportsCatalogsInPrivilegeDefinitions は変換されませんでした。

java.sql.DatabaseMetaData.supportsCatalogsInProcedureCalls は変換されませんでした。

java.sql.DatabaseMetaData.supportsCatalogsInTableDefinitions は変換されませんでした。

java.sql.DatabaseMetaData.supportsColumnAliasing は変換されませんでした。

java.sql.DatabaseMetaData.supportsConvert は変換されませんでした。

java.sql.DatabaseMetaData.supportsCoreSQLGrammar は変換されませんでした。

java.sql.DatabaseMetaData.supportsCorrelatedSubqueries は変換されませんでした。

java.sql.DatabaseMetaData.supportsDataDefinitionAndDataManipulationTransactions は変換されませんでした。

java.sql.DatabaseMetaData.supportsDataManipulationTransactionsOnly は変換されませんでした。

java.sql.DatabaseMetaData.supportsDifferentTableCorrelationNames は変換されませんでした。

java.sql.DatabaseMetaData.supportsExpressionsInOrderBy は変換されませんでした。

java.sql.DatabaseMetaData.supportsExtendedSQLGrammar は変換されませんでした。

java.sql.DatabaseMetaData.supportsFullOuterJoins は変換されませんでした。

java.sql.DatabaseMetaData.supportsGroupBy は変換されませんでした。

java.sql.DatabaseMetaData.supportsGroupByBeyondSelect は変換されませんでした。

java.sql.DatabaseMetaData.supportsGroupByUnrelated は変換されませんでした。

java.sql.DatabaseMetaData.supportsIntegrityEnhancementFacility は変換されませんでした。

java.sql.DatabaseMetaData.supportsLikeEscapeClause は変換されませんでした。

java.sql.DatabaseMetaData.supportsLimitedOuterJoins は変換されませんでした。

java.sql.DatabaseMetaData.supportsMinimumSQLGrammar は変換されませんでした。

java.sql.DatabaseMetaData.supportsMixedCaseIdentifiers は変換されませんでした。

java.sql.DatabaseMetaData.supportsMixedCaseQuotedIdentifiers は変換されませんでした。

java.sql.DatabaseMetaData.supportsMultipleResultSets は変換されませんでした。

java.sql.DatabaseMetaData.supportsMultipleTransactions は変換されませんでした。

java.sql.DatabaseMetaData.supportsNonNullableColumns は変換されませんでした。

java.sql.DatabaseMetaData.supportsOpenCursorsAcrossCommit は変換されませんでした。

java.sql.DatabaseMetaData.supportsOpenCursorsAcrossRollback は変換されませんでした。

java.sql.DatabaseMetaData.supportsOpenStatementsAcrossCommit は変換されませんでした。

java.sql.DatabaseMetaData.supportsOpenStatementsAcrossRollback は変換されませんでした。

java.sql.DatabaseMetaData.supportsOrderByUnrelated は変換されませんでした。

java.sql.DatabaseMetaData.supportsOuterJoins は変換されませんでした。

java.sql.DatabaseMetaData.supportsPositionedDelete は変換されませんでした。

java.sql.DatabaseMetaData.supportsPositionedUpdate は変換されませんでした。

java.sql.DatabaseMetaData.supportsResultSetConcurrency は変換されませんでした。

java.sql.DatabaseMetaData.supportsResultSetType は変換されませんでした。

java.sql.DatabaseMetaData.supportsSchemasInDataManipulation は変換されませんでした。

java.sql.DatabaseMetaData.supportsSchemasInIndexDefinitions は変換されませんでした。

java.sql.DatabaseMetaData.supportsSchemasInPrivilegeDefinitions は変換されませんでした。

java.sql.DatabaseMetaData.supportsSchemasInProcedureCalls は変換されませんでした。

java.sql.DatabaseMetaData.supportsSchemasInTableDefinitions は変換されませんでした。

java.sql.DatabaseMetaData.supportsSelectForUpdate は変換されませんでした。

java.sql.DatabaseMetaData.supportsStoredProcedures は変換されませんでした。

java.sql.DatabaseMetaData.supportsSubqueriesInComparisons は変換されませんでした。

java.sql.DatabaseMetaData.supportsSubqueriesInExists は変換されませんでした。

java.sql.DatabaseMetaData.supportsSubqueriesInIns は変換されませんでした。

java.sql.DatabaseMetaData.supportsSubqueriesInQuantifieds は変換されませんでした。

java.sql.DatabaseMetaData.supportsTableCorrelationNames は変換されませんでした。

java.sql.DatabaseMetaData.supportsTransactionIsolationLevel は変換されませんでした。

java.sql.DatabaseMetaData.supportsTransactions は変換されませんでした。

java.sql.DatabaseMetaData.supportsUnion は変換されませんでした。

java.sql.DatabaseMetaData.supportsUnionAll は変換されませんでした。

java.sql.DatabaseMetaData.tableIndexClustered は変換されませんでした。

java.sql.DatabaseMetaData.tableIndexHashed は変換されませんでした。

java.sql.DatabaseMetaData.tableIndexOther は変換されませんでした。

java.sql.DatabaseMetaData.tableIndexStatistic は変換されませんでした。

java.sql.DatabaseMetaData.typeNoNulls は変換されませんでした。

java.sql.DatabaseMetaData.typeNullable は変換されませんでした。

java.sql.DatabaseMetaData.typeNullableUnknown は変換されませんでした。

java.sql.DatabaseMetaData.typePredBasic は変換されませんでした。

java.sql.DatabaseMetaData.typePredChar は変換されませんでした。

java.sql.DatabaseMetaData.typePredNone は変換されませんでした。

java.sql.DatabaseMetaData.typeSearchable は変換されませんでした。

java.sql.DatabaseMetaData.updatesAreDetected を変換できませんでした。

java.sql.DatabaseMetaData.usesLocalFilePerTable は変換されませんでした。

java.sql.DatabaseMetaData.usesLocalFiles は変換されませんでした。

java.sql.DatabaseMetaData.versionColumnNotPseudo は変換されませんでした。

java.sql.DatabaseMetaData.versionColumnPseudo は変換されませんでした。

java.sql.DatabaseMetaData.versionColumnUnknown は変換されませんでした。

java.sql.DataTruncation は変換されませんでした。

java.sql.Date.Date は変換されませんでした。

java.sql.Date.getHours は変換されませんでした。

java.sql.Date.getMinutes は変換されませんでした。

java.sql.Date.getSeconds は変換されませんでした。

java.sql.Date.setHours は変換されませんでした。

java.sql.Date.setMinutes は変換されませんでした。

java.sql.Date.setSeconds は変換されませんでした。

java.sql.DriverInfo は変換されませんでした。

java.sql.DriverManager は変換されませんでした。

java.sql.DriverPropertyInfo は変換されませんでした。

java.sql.PreparedStatement.addBatch は変換されませんでした。

java.sql.PreparedStatement.getMetaData は変換されませんでした。

java.sql.PreparedStatement.setArray は変換されませんでした。

java.sql.PreparedStatement.setBlob を変換できませんでした。

java.sql.PreparedStatement.setClob は変換されませんでした。

java.sql.PreparedStatement.setDate は変換されませんでした。

java.sql.PreparedStatement.setNull は変換されませんでした。

java.sql.PreparedStatement.setRef を変換できませんでした。

java.sql.PreparedStatement.setTime は変換されませんでした。

java.sql.PreparedStatement.setTimestamp は変換されませんでした。

java.sql.Ref は変換されませんでした。

java.sql.ResultSet は変換されませんでした。

java.sql.ResultSet.<ConcurrencyMode> は変換されませんでした。

java.sql.ResultSet.absolute を変換できませんでした。

java.sql.ResultSet.afterLast は変換されませんでした。

java.sql.ResultSet.beforeFirst は変換されませんでした。

java.sql.ResultSet.cancelRowUpdates は変換されませんでした。

java.sql.ResultSet.clearWarnings は変換されませんでした。

java.sql.ResultSet.close は変換されませんでした。

java.sql.ResultSet.deleteRow は変換されませんでした。

java.sql.ResultSet.findColumn は変換されませんでした。

java.sql.ResultSet.first は変換されませんでした。

java.sql.ResultSet.getArray は変換されませんでした。

java.sql.ResultSet.getAsciiStream は変換されませんでした。

java.sql.ResultSet.getBigDecimal は変換されませんでした。

java.sql.ResultSet.getBinaryStream は変換されませんでした。

java.sql.ResultSet.getBlob は変換されませんでした。

java.sql.ResultSet.getBoolean は変換されませんでした。

java.sql.ResultSet.getByte は変換されませんでした。

java.sql.ResultSet.getBytes は変換されませんでした。

java.sql.ResultSet.getCharacterStream は変換されませんでした。

java.sql.ResultSet.getClob は変換されませんでした。

java.sql.ResultSet.getConcurrency は変換されませんでした。

java.sql.ResultSet.getCursorName は変換されませんでした。

java.sql.ResultSet.getDate は変換されませんでした。

java.sql.ResultSet.getDouble は変換されませんでした。

java.sql.ResultSet.getFetchDirection は変換されませんでした。

java.sql.ResultSet.getFetchSize は変換されませんでした。

java.sql.ResultSet.getFloat は変換されませんでした。

java.sql.ResultSet.getInt を変換できませんでした。

java.sql.ResultSet.getLong は変換されませんでした。

java.sql.ResultSet.getMetaData は変換されませんでした。

java.sql.ResultSet.getObject は変換されませんでした。

java.sql.ResultSet.getRef は変換されませんでした。

java.sql.ResultSet.getRow は変換されませんでした。

java.sql.ResultSet.getShort は変換されませんでした。

java.sql.ResultSet.getStatement は変換されませんでした。

java.sql.ResultSet.getString は変換されませんでした。

java.sql.ResultSet.getTime は変換されませんでした。

java.sql.ResultSet.getTimestamp は変換されませんでした。

java.sql.ResultSet.getType は変換されませんでした。

java.sql.ResultSet.getUnicodeStream は変換されませんでした。

java.sql.ResultSet.getWarnings は変換されませんでした。

java.sql.ResultSet.insertRow は変換されませんでした。

java.sql.ResultSet.isAfterLast は変換されませんでした。

java.sql.ResultSet.isBeforeFirst は変換されませんでした。

java.sql.ResultSet.isFirst は変換されませんでした。

java.sql.ResultSet.isLast は変換されませんでした。

java.sql.ResultSet.last は変換されませんでした。

java.sql.ResultSet.moveToCurrentRow は変換されませんでした。

java.sql.ResultSet.moveToInsertRow は変換されませんでした。

java.sql.ResultSet.next は変換されませんでした。

java.sql.ResultSet.previous は変換されませんでした。

java.sql.ResultSet.refreshRow は変換されませんでした。

java.sql.ResultSet.relative は変換されませんでした。

java.sql.ResultSet.rowDeleted は変換されませんでした。

java.sql.ResultSet.rowInserted は変換されませんでした。

java.sql.ResultSet.rowUpdated は変換されませんでした。

java.sql.ResultSet.setFetchDirection は変換されませんでした。

java.sql.ResultSet.setFetchSize を変換できませんでした。

java.sql.ResultSet.TYPE_FORWARD_ONLY は変換されませんでした。

java.sql.ResultSet.updateAsciiStream は変換されませんでした。

java.sql.ResultSet.updateBigDecimal は変換されませんでした。

java.sql.ResultSet.updateBinaryStream は変換されませんでした。

java.sql.ResultSet.updateBoolean は変換されませんでした。

java.sql.ResultSet.updateByte は変換されませんでした。

java.sql.ResultSet.updateBytes は変換されませんでした。

java.sql.ResultSet.updateCharacterStream は変換されませんでした。

java.sql.ResultSet.updateDate は変換されませんでした。

java.sql.ResultSet.updateDouble は変換されませんでした。

java.sql.ResultSet.updateFloat は変換されませんでした。

java.sql.ResultSet.updateInt を変換できませんでした。

java.sql.ResultSet.updateLong は変換されませんでした。

java.sql.ResultSet.updateNull は変換されませんでした。

java.sql.ResultSet.updateObject は変換されませんでした。

java.sql.ResultSet.updateRow は変換されませんでした。

java.sql.ResultSet.updateShort は変換されませんでした。

java.sql.ResultSet.updateString は変換されませんでした。

java.sql.ResultSet.updateTime は変換されませんでした。

java.sql.ResultSet.updateTimestamp は変換されませんでした。

java.sql.ResultSet.wasNull は変換されませんでした。

java.sql.ResultSetMetaData.columnNoNulls は変換されませんでした。

java.sql.ResultSetMetaData.columnNullable は変換されませんでした。

java.sql.ResultSetMetaData.columnNullableUnknown は変換されませんでした。

java.sql.ResultSetMetaData.getPrecision は変換されませんでした。

java.sql.ResultSetMetaData.isCaseSensitive は変換されませんでした。

java.sql.ResultSetMetaData.isCurrency は変換されませんでした。

java.sql.ResultSetMetaData.isDefinitelyWritable は変換されませんでした。

java.sql.ResultSetMetaData.isSearchable は変換されませんでした。

java.sql.ResultSetMetaData.isSigned は変換されませんでした。

java.sql.ResultSetMetaData.isWritable は変換されませんでした。

javax.sql.RowSetMetaData.setNullable は変換されませんでした。

java.sql.SQLData は変換されませんでした。

java.sql.SQLException.getNextException は変換されませんでした。

java.sql.SQLException.setNextException は変換されませんでした。

java.sql.SQLException.SQLException は変換されませんでした。

java.sql.SQLInput は変換されませんでした。

java.sql.SQLOutput は変換されませんでした。

java.sql.SQLPermission を変換できませんでした。

java.sql.SQLWarning は変換されませんでした。

java.sql.Statement.clearWarnings は変換されませんでした。

java.sql.Statement.close は変換されませんでした。

java.sql.Statement.execute は変換されませんでした。

java.sql.Statement.executeBatch は変換されませんでした。

java.sql.Statement.getFetchDirection は変換されませんでした。

java.sql.Statement.getFetchSize は変換されませんでした。

java.sql.Statement.getMaxFieldSize は変換されませんでした。

java.sql.Statement.getMaxRows は変換されませんでした。

java.sql.Statement.getMoreResults は変換されませんでした。

java.sql.Statement.getResultSet は変換されませんでした。

java.sql.Statement.getResultSetConcurrency は変換されませんでした。

java.sql.Statement.getResultSetType は変換されませんでした。

java.sql.Statement.getUpdateCount は変換されませんでした。

java.sql.Statement.getWarnings は変換されませんでした。

java.sql.Statement.setCursorName は変換されませんでした。

java.sql.Statement.setEscapeProcessing は変換されませんでした。

java.sql.Statement.setFetchDirection は変換されませんでした。

java.sql.Statement.setFetchSize は変換されませんでした。

java.sql.Statement.setMaxFieldSize は変換されませんでした。

java.sql.Statement.setMaxRows は変換されませんでした。

java.sql.Struct は変換されませんでした。

java.sql.Timestamp.getNanos は変換されませんでした。

java.sql.Timestamp.setNanos は変換されませんでした。

java.sql.Timestamp.Timestamp は変換されませんでした。

java.sql.Types.<TypeName> は変換されませんでした。

Java.text のエラー メッセージ

java.text.Annotation を変換できませんでした。

java.text.AttributedCharacterIterator を変換できませんでした。

java.text.AttributedCharacterIterator.Attribute を変換できませんでした。

java.text.AttributedString を変換できませんでした。

java.text.BreakDictionary を変換できませんでした。

java.text.BreakIterator は変換されませんでした。

java.text.CharacterIterator.clone は変換されませんでした。

java.text.ChoiceFormat は変換されませんでした。

java.text.CollationElementIterator は変換されませんでした。

java.text.CollationKey.compareTo を変換できませんでした。

java.text.Collator.CANONICAL_DECOMPOSITION は変換されませんでした。

java.text.Collator.Collator は変換されませんでした。

java.text.Collator.compare は変換されませんでした。

java.text.Collator.FULL_DECOMPOSITION は変換されませんでした。

java.text.Collator.getDecomposition は変換されませんでした。

java.text.Collator.getStrength は変換されませんでした。

java.text.Collator.IDENTICAL は変換されませんでした。

java.text.Collator.NO_DECOMPOSITION は変換されませんでした。

java.text.Collator.PRIMARY は変換されませんでした。

java.text.Collator.SECONDARY は変換されませんでした。

java.text.Collator.setDecomposition は変換されませんでした。

java.text.Collator.setStrength は変換されませんでした。

java.text.Collator.TERTIARY は変換されませんでした。

java.text.DateFormat.format は変換されませんでした。

java.text.DateFormat.getNumberFormat は変換されませんでした。

java.text.DateFormat.getTimeZone は変換されませんでした。

java.text.DateFormat.isLenient は変換されませんでした。

java.text.DateFormat.numberFormat は変換されませんでした。

java.text.DateFormat.parse は変換されませんでした。

java.text.DateFormat.parseObject は変換されませんでした。

java.text.DateFormat.setLenient は変換されませんでした。

java.text.DateFormat.setNumberFormat は変換されませんでした。

java.text.DateFormat.setTimeZone は変換されませんでした。

java.text.DateFormatSymbols.DateFormatSymbols を変換できませんでした。

java.text.DateFormatSymbols.getEras は変換されませんでした。

java.text.DateFormatSymbols.getLocalPatternChars は変換されませんでした。

java.text.DateFormatSymbols.getZoneStrings は変換されませんでした。

java.text.DateFormatSymbols.setEras は変換されませんでした。

java.text.DateFormatSymbols.setLocalPatternChars は変換されませんでした。

java.text.DateFormatSymbols.setZoneStrings は変換されませんでした。

java.text.DecimalFormat は変換されませんでした。

java.text.DecimalFormatSymbols.DecimalFormatSymbols は変換されませんでした。

java.text.DecimalFormatSymbols.getDigit は変換されませんでした。

java.text.DecimalFormatSymbols.getInternationalCurrencySymbol を変換できませんでした。

java.text.DecimalFormatSymbols.getPatternSeparator は変換されませんでした。

java.text.DecimalFormatSymbols.getZeroDigit は変換されませんでした。

java.text.DecimalFormatSymbols.setDigit は変換されませんでした。

java.text.DecimalFormatSymbols.setInternationalCurrencySymbol を変換できませんでした。

java.text.DecimalFormatSymbols.setPatternSeparator は変換されませんでした。

java.text.DecimalFormatSymbols.setZeroDigit は変換されませんでした。

java.text.DictionaryBasedBreakIterator を変換できませんでした。

java.text.FieldPosition を変換できませんでした。

java.text.Format は変換されませんでした。

java.text.Format.format を変換できませんでした。

java.text.MessageFormat は変換されませんでした。

java.text.NumberFormat.format は変換されませんでした。

java.text.NumberFormat.FRACTION_FIELD は変換されませんでした。

java.text.NumberFormat.INTEGER_FIELD は変換されませんでした。

java.text.NumberFormat.isParseIntegerOnly は変換されませんでした。

java.text.NumberFormat.parse は変換されませんでした。

java.text.NumberFormat.parseObject は変換されませんでした。

java.text.NumberFormat.setParseIntegerOnly は変換されませんでした。

java.text.ParseException を変換できませんでした。

java.text.ParseException.getErrorOffset は変換されませんでした。

java.text.ParsePosition.getErrorIndex を変換できませんでした。

java.text.ParsePosition.setErrorIndex を変換できませんでした。

java.text.ParsePosition.toString を変換できませんでした。

java.text.RuleBasedBreakIterator を変換できませんでした。

java.text.RuleBasedCollator は変換されませんでした。

java.text.SimpleDateFormat は変換されませんでした。

java.text.StringCharacterIterator.clone は変換されませんでした。

java.text.StringCharacterIterator.setText は変換されませんでした。

Java.text.resources のエラー メッセージ

java.text.resources.BreakIteratorRules.BreakIteratorRules は変換されませんでした。

java.text.resources.BreakIteratorRules.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData.DateFormatZoneData は変換されませんでした。

java.text.resources.DateFormatZoneData.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData.getKeys は変換されませんでした。

java.text.resources.DateFormatZoneData_ar は変換されませんでした。

java.text.resources.DateFormatZoneData_ar.DateFormatZoneData_ar は変換されませんでした。

java.text.resources.DateFormatZoneData_ar.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_be は変換されませんでした。

java.text.resources.DateFormatZoneData_be.DateFormatZoneData_be は変換されませんでした。

java.text.resources.DateFormatZoneData_be.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_bg は変換されませんでした。

java.text.resources.DateFormatZoneData_bg.DateFormatZoneData_bg は変換されませんでした。

java.text.resources.DateFormatZoneData_bg.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_ca は変換されませんでした。

java.text.resources.DateFormatZoneData_ca.DateFormatZoneData_ca は変換されませんでした。

java.text.resources.DateFormatZoneData_ca.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_cs.DateFormatZoneData_cs は変換されませんでした。

java.text.resources.DateFormatZoneData_cs.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_da.DateFormatZoneData_da は変換されませんでした。

java.text.resources.DateFormatZoneData_da.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_de は変換されませんでした。

java.text.resources.DateFormatZoneData_de.DateFormatZoneData_de は変換されませんでした。

java.text.resources.DateFormatZoneData_de.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_de_AT は変換されませんでした。

java.text.resources.DateFormatZoneData_de_AT.DateFormatZoneData_de_AT は変換されませんでした。

java.text.resources.DateFormatZoneData_de_AT.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_de_CH は変換されませんでした。

java.text.resources.DateFormatZoneData_de_CH.DateFormatZoneData_de_CH は変換されませんでした。

java.text.resources.DateFormatZoneData_de_CH.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_el は変換されませんでした。

java.text.resources.DateFormatZoneData_el.DateFormatZoneData_el は変換されませんでした。

java.text.resources.DateFormatZoneData_el.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_en は変換されませんでした。

java.text.resources.DateFormatZoneData_en.DateFormatZoneData_en は変換されませんでした。

java.text.resources.DateFormatZoneData_en.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_en_CA は変換されませんでした。

java.text.resources.DateFormatZoneData_en_CA.DateFormatZoneData_en_CA は変換されませんでした。

java.text.resources.DateFormatZoneData_en_CA.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_en_GB は変換されませんでした。

java.text.resources.DateFormatZoneData_en_GB.DateFormatZoneData_en_GB は変換されませんでした。

java.text.resources.DateFormatZoneData_en_GB.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_en_IE は変換されませんでした。

java.text.resources.DateFormatZoneData_en_IE.DateFormatZoneData_en_IE は変換されませんでした。

java.text.resources.DateFormatZoneData_en_IE.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_es は変換されませんでした。

java.text.resources.DateFormatZoneData_es.DateFormatZoneData_es は変換されませんでした。

java.text.resources.DateFormatZoneData_es.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_et は変換されませんでした。

java.text.resources.DateFormatZoneData_et.DateFormatZoneData_et は変換されませんでした。

java.text.resources.DateFormatZoneData_et.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_fi は変換されませんでした。

java.text.resources.DateFormatZoneData_fi.DateFormatZoneData_fi は変換されませんでした。

java.text.resources.DateFormatZoneData_fi.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_fr は変換されませんでした。

java.text.resources.DateFormatZoneData_fr.DateFormatZoneData_fr は変換されませんでした。

java.text.resources.DateFormatZoneData_fr.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_fr_BE は変換されませんでした。

java.text.resources.DateFormatZoneData_fr_BE.DateFormatZoneData_fr_BE は変換されませんでした。

java.text.resources.DateFormatZoneData_fr_BE.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_fr_CA は変換されませんでした。

java.text.resources.DateFormatZoneData_fr_CA.DateFormatZoneData_fr_CA は変換されませんでした。

java.text.resources.DateFormatZoneData_fr_CA.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_fr_CH は変換されませんでした。

java.text.resources.DateFormatZoneData_fr_CH.DateFormatZoneData_fr_CH は変換されませんでした。

java.text.resources.DateFormatZoneData_fr_CH.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_hr は変換されませんでした。

java.text.resources.DateFormatZoneData_hr.DateFormatZoneData_hr は変換されませんでした。

java.text.resources.DateFormatZoneData_hr.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_hu は変換されませんでした。

java.text.resources.DateFormatZoneData_hu.DateFormatZoneData_hu は変換されませんでした。

java.text.resources.DateFormatZoneData_hu.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_is は変換されませんでした。

java.text.resources.DateFormatZoneData_is.DateFormatZoneData_is は変換されませんでした。

java.text.resources.DateFormatZoneData_is.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_it は変換されませんでした。

java.text.resources.DateFormatZoneData_it.DateFormatZoneData_it は変換されませんでした。

java.text.resources.DateFormatZoneData_it.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_it_CH は変換されませんでした。

java.text.resources.DateFormatZoneData_it_CH.DateFormatZoneData_it_CH は変換されませんでした。

java.text.resources.DateFormatZoneData_it_CH.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_iw は変換されませんでした。

java.text.resources.DateFormatZoneData_iw.DateFormatZoneData_iw は変換されませんでした。

java.text.resources.DateFormatZoneData_iw.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_ja は変換されませんでした。

java.text.resources.DateFormatZoneData_ja.DateFormatZoneData_ja は変換されませんでした。

java.text.resources.DateFormatZoneData_ja.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_ko は変換されませんでした。

java.text.resources.DateFormatZoneData_ko.DateFormatZoneData_ko は変換されませんでした。

java.text.resources.DateFormatZoneData_ko.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_lt は変換されませんでした。

java.text.resources.DateFormatZoneData_lt.DateFormatZoneData_lt は変換されませんでした。

java.text.resources.DateFormatZoneData_lt.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_lv は変換されませんでした。

java.text.resources.DateFormatZoneData_lv.DateFormatZoneData_lv は変換されませんでした。

java.text.resources.DateFormatZoneData_lv.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_mk は変換されませんでした。

java.text.resources.DateFormatZoneData_mk.DateFormatZoneData_mk は変換されませんでした。

java.text.resources.DateFormatZoneData_mk.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_nl は変換されませんでした。

java.text.resources.DateFormatZoneData_nl.DateFormatZoneData_nl は変換されませんでした。

java.text.resources.DateFormatZoneData_nl.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_nl_BE は変換されませんでした。

java.text.resources.DateFormatZoneData_nl_BE.DateFormatZoneData_nl_BE は変換されませんでした。

java.text.resources.DateFormatZoneData_nl_BE.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_no は変換されませんでした。

java.text.resources.DateFormatZoneData_no.DateFormatZoneData_no は変換されませんでした。

java.text.resources.DateFormatZoneData_no.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_no_NO_NY は変換されませんでした。

java.text.resources.DateFormatZoneData_no_NO_NY.DateFormatZoneData_no_NO_NY は変換されませんでした。

java.text.resources.DateFormatZoneData_no_NO_NY.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_pl は変換されませんでした。

java.text.resources.DateFormatZoneData_pl.DateFormatZoneData_pl は変換されませんでした。

java.text.resources.DateFormatZoneData_pl.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_pt は変換されませんでした。

java.text.resources.DateFormatZoneData_pt.DateFormatZoneData_pt は変換されませんでした。

java.text.resources.DateFormatZoneData_pt.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_ro は変換されませんでした。

java.text.resources.DateFormatZoneData_ro.DateFormatZoneData_ro は変換されませんでした。

java.text.resources.DateFormatZoneData_ro.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_ru は変換されませんでした。

java.text.resources.DateFormatZoneData_ru.DateFormatZoneData_ru は変換されませんでした。

java.text.resources.DateFormatZoneData_ru.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_sh は変換されませんでした。

java.text.resources.DateFormatZoneData_sh.DateFormatZoneData_sh は変換されませんでした。

java.text.resources.DateFormatZoneData_sh.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_sk は変換されませんでした。

java.text.resources.DateFormatZoneData_sk.DateFormatZoneData_sk は変換されませんでした。

java.text.resources.DateFormatZoneData_sk.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_sl は変換されませんでした。

java.text.resources.DateFormatZoneData_sl.DateFormatZoneData_sl は変換されませんでした。

java.text.resources.DateFormatZoneData_sl.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_sq は変換されませんでした。

java.text.resources.DateFormatZoneData_sq.DateFormatZoneData_sq は変換されませんでした。

java.text.resources.DateFormatZoneData_sq.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_sr は変換されませんでした。

java.text.resources.DateFormatZoneData_sr.DateFormatZoneData_sr は変換されませんでした。

java.text.resources.DateFormatZoneData_sr.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_sv は変換されませんでした。

java.text.resources.DateFormatZoneData_sv.DateFormatZoneData_sv は変換されませんでした。

java.text.resources.DateFormatZoneData_sv.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_th は変換されませんでした。

java.text.resources.DateFormatZoneData_th.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_tr は変換されませんでした。

java.text.resources.DateFormatZoneData_tr.DateFormatZoneData_tr は変換されませんでした。

java.text.resources.DateFormatZoneData_tr.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_uk は変換されませんでした。

java.text.resources.DateFormatZoneData_uk.DateFormatZoneData_uk は変換されませんでした。

java.text.resources.DateFormatZoneData_uk.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_zh は変換されませんでした。

java.text.resources.DateFormatZoneData_zh.DateFormatZoneData_zh は変換されませんでした。

java.text.resources.DateFormatZoneData_zh.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_zh_HK は変換されませんでした。

java.text.resources.DateFormatZoneData_zh_HK.getContents は変換されませんでした。

java.text.resources.DateFormatZoneData_zh_TW は変換されませんでした。

java.text.resources.DateFormatZoneData_zh_TW.DateFormatZoneData_zh_TW は変換されませんでした。

java.text.resources.DateFormatZoneData_zh_TW.getContents は変換されませんでした。

java.text.resources.LocaleData.getKeys は変換されませんでした。

java.text.resources.LocaleData.init は変換されませんでした。

java.text.resources.LocaleData.LocaleData は変換されませんでした。

java.text.resources.LocaleElements.getContents は変換されませんでした。

java.text.resources.LocaleElements.LocaleElements は変換されませんでした。

java.text.resources.LocaleElements_ar は変換されませんでした。

java.text.resources.LocaleElements_ar.LocaleElements_ar は変換されませんでした。

java.text.resources.LocaleElements_ar_AE は変換されませんでした。

java.text.resources.LocaleElements_ar_AE.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_AE.LocaleElements_ar_AE は変換されませんでした。

java.text.resources.LocaleElements_ar_BH は変換されませんでした。

java.text.resources.LocaleElements_ar_BH.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_BH.LocaleElements_ar_BH は変換されませんでした。

java.text.resources.LocaleElements_ar_DZ は変換されませんでした。

java.text.resources.LocaleElements_ar_DZ.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_DZ.LocaleElements_ar_DZ は変換されませんでした。

java.text.resources.LocaleElements_ar_EG は変換されませんでした。

java.text.resources.LocaleElements_ar_EG.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_EG.LocaleElements_ar_EG は変換されませんでした。

java.text.resources.LocaleElements_ar_IQ は変換されませんでした。

java.text.resources.LocaleElements_ar_IQ.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_IQ.LocaleElements_ar_IQ は変換されませんでした。

java.text.resources.LocaleElements_ar_JO は変換されませんでした。

java.text.resources.LocaleElements_ar_JO.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_JO.LocaleElements_ar_JO は変換されませんでした。

java.text.resources.LocaleElements_ar_KW は変換されませんでした。

java.text.resources.LocaleElements_ar_KW.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_KW.LocaleElements_ar_KW は変換されませんでした。

java.text.resources.LocaleElements_ar_LB は変換されませんでした。

java.text.resources.LocaleElements_ar_LB.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_LB.LocaleElements_ar_LB は変換されませんでした。

java.text.resources.LocaleElements_ar_LY.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_LY.LocaleElements_ar_LY は変換されませんでした。

java.text.resources.LocaleElements_ar_MA は変換されませんでした。

java.text.resources.LocaleElements_ar_MA.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_MA.LocaleElements_ar_MA は変換されませんでした。

java.text.resources.LocaleElements_ar_OM は変換されませんでした。

java.text.resources.LocaleElements_ar_OM.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_OM.LocaleElements_ar_OM は変換されませんでした。

java.text.resources.LocaleElements_ar_QA は変換されませんでした。

java.text.resources.LocaleElements_ar_QA.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_QA.LocaleElements_ar_QA は変換されませんでした。

java.text.resources.LocaleElements_ar_SA は変換されませんでした。

java.text.resources.LocaleElements_ar_SA.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_SA.LocaleElements_ar_SA は変換されませんでした。

java.text.resources.LocaleElements_ar_SD は変換されませんでした。

java.text.resources.LocaleElements_ar_SD.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_SD.LocaleElements_ar_SD は変換されませんでした。

java.text.resources.LocaleElements_ar_SY は変換されませんでした。

java.text.resources.LocaleElements_ar_SY.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_SY.LocaleElements_ar_SY は変換されませんでした。

java.text.resources.LocaleElements_ar_TN は変換されませんでした。

java.text.resources.LocaleElements_ar_TN.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_TN.LocaleElements_ar_TN は変換されませんでした。

java.text.resources.LocaleElements_ar_YE は変換されませんでした。

java.text.resources.LocaleElements_ar_YE.getContents は変換されませんでした。

java.text.resources.LocaleElements_ar_YE.LocaleElements_ar_YE は変換されませんでした。

java.text.resources.LocaleElements_be は変換されませんでした。

java.text.resources.LocaleElements_be.LocaleElements_be は変換されませんでした。

java.text.resources.LocaleElements_be_BY は変換されませんでした。

java.text.resources.LocaleElements_be_BY.getContents は変換されませんでした。

java.text.resources.LocaleElements_be_BY.LocaleElements_be_BY は変換されませんでした。

java.text.resources.LocaleElements_bg は変換されませんでした。

java.text.resources.LocaleElements_bg.LocaleElements_bg は変換されませんでした。

java.text.resources.LocaleElements_bg_BG は変換されませんでした。

java.text.resources.LocaleElements_bg_BG.getContents は変換されませんでした。

java.text.resources.LocaleElements_bg_BG.LocaleElements_bg_BG は変換されませんでした。

java.text.resources.LocaleElements_ca は変換されませんでした。

java.text.resources.LocaleElements_ca.LocaleElements_ca は変換されませんでした。

java.text.resources.LocaleElements_ca_ES は変換されませんでした。

java.text.resources.LocaleElements_ca_ES.getContents は変換されませんでした。

java.text.resources.LocaleElements_ca_ES.LocaleElements_ca_ES は変換されませんでした。

java.text.resources.LocaleElements_ca_ES_EURO は変換されませんでした。

java.text.resources.LocaleElements_ca_ES_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_ca_ES_EURO.LocaleElements_ca_ES_EURO は変換されませんでした。

java.text.resources.LocaleElements_cs は変換されませんでした。

java.text.resources.LocaleElements_cs.LocaleElements_cs は変換されませんでした。

java.text.resources.LocaleElements_cs_CZ は変換されませんでした。

java.text.resources.LocaleElements_cs_CZ.getContents は変換されませんでした。

java.text.resources.LocaleElements_cs_CZ.LocaleElements_cs_CZ は変換されませんでした。

java.text.resources.LocaleElements_da は変換されませんでした。

java.text.resources.LocaleElements_da.LocaleElements_da は変換されませんでした。

java.text.resources.LocaleElements_da_DK は変換されませんでした。

java.text.resources.LocaleElements_da_DK.getContents は変換されませんでした。

java.text.resources.LocaleElements_da_DK.LocaleElements_da_DK は変換されませんでした。

java.text.resources.LocaleElements_de は変換されませんでした。

java.text.resources.LocaleElements_de.LocaleElements_de は変換されませんでした。

java.text.resources.LocaleElements_de_AT は変換されませんでした。

java.text.resources.LocaleElements_de_AT.LocaleElements_de_AT は変換されませんでした。

java.text.resources.LocaleElements_de_AT_EURO は変換されませんでした。

java.text.resources.LocaleElements_de_AT_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_de_AT_EURO.LocaleElements_de_AT_EURO は変換されませんでした。

java.text.resources.LocaleElements_de_CH は変換されませんでした。

java.text.resources.LocaleElements_de_CH.LocaleElements_de_CH は変換されませんでした。

java.text.resources.LocaleElements_de_DE は変換されませんでした。

java.text.resources.LocaleElements_de_DE.getContents は変換されませんでした。

java.text.resources.LocaleElements_de_DE.LocaleElements_de_DE は変換されませんでした。

java.text.resources.LocaleElements_de_DE_EURO は変換されませんでした。

java.text.resources.LocaleElements_de_DE_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_de_DE_EURO.LocaleElements_de_DE_EURO は変換されませんでした。

java.text.resources.LocaleElements_de_LU は変換されませんでした。

java.text.resources.LocaleElements_de_LU.getContents は変換されませんでした。

java.text.resources.LocaleElements_de_LU.LocaleElements_de_LU は変換されませんでした。

java.text.resources.LocaleElements_de_LU_EURO は変換されませんでした。

java.text.resources.LocaleElements_de_LU_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_de_LU_EURO.LocaleElements_de_LU_EURO は変換されませんでした。

java.text.resources.LocaleElements_el は変換されませんでした。

java.text.resources.LocaleElements_el.LocaleElements_el は変換されませんでした。

java.text.resources.LocaleElements_el_GR は変換されませんでした。

java.text.resources.LocaleElements_el_GR.getContents は変換されませんでした。

java.text.resources.LocaleElements_el_GR.LocaleElements_el_GR は変換されませんでした。

java.text.resources.LocaleElements_en は変換されませんでした。

java.text.resources.LocaleElements_en.getContents は変換されませんでした。

java.text.resources.LocaleElements_en.LocaleElements_en は変換されませんでした。

java.text.resources.LocaleElements_en_AU は変換されませんでした。

java.text.resources.LocaleElements_en_AU.LocaleElements_en_AU は変換されませんでした。

java.text.resources.LocaleElements_en_CA は変換されませんでした。

java.text.resources.LocaleElements_en_CA.LocaleElements_en_CA は変換されませんでした。

java.text.resources.LocaleElements_en_GB は変換されませんでした。

java.text.resources.LocaleElements_en_GB.LocaleElements_en_GB は変換されませんでした。

java.text.resources.LocaleElements_en_IE は変換されませんでした。

java.text.resources.LocaleElements_en_IE.LocaleElements_en_IE は変換されませんでした。

java.text.resources.LocaleElements_en_IE_EURO は変換されませんでした。

java.text.resources.LocaleElements_en_IE_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_en_IE_EURO.LocaleElements_en_IE_EURO は変換されませんでした。

java.text.resources.LocaleElements_en_IR は変換されませんでした。

java.text.resources.LocaleElements_en_IR.LocaleElements_en_IR は変換されませんでした。

java.text.resources.LocaleElements_en_NZ は変換されませんでした。

java.text.resources.LocaleElements_en_NZ.LocaleElements_en_NZ は変換されませんでした。

java.text.resources.LocaleElements_en_US は変換されませんでした。

java.text.resources.LocaleElements_en_US.getContents は変換されませんでした。

java.text.resources.LocaleElements_en_US.LocaleElements_en_US は変換されませんでした。

java.text.resources.LocaleElements_en_ZA は変換されませんでした。

java.text.resources.LocaleElements_en_ZA.LocaleElements_en_ZA は変換されませんでした。

java.text.resources.LocaleElements_es は変換されませんでした。

java.text.resources.LocaleElements_es.LocaleElements_es は変換されませんでした。

java.text.resources.LocaleElements_es_AR は変換されませんでした。

java.text.resources.LocaleElements_es_AR.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_AR.LocaleElements_es_AR は変換されませんでした。

java.text.resources.LocaleElements_es_BO は変換されませんでした。

java.text.resources.LocaleElements_es_BO.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_BO.LocaleElements_es_BO は変換されませんでした。

java.text.resources.LocaleElements_es_CL は変換されませんでした。

java.text.resources.LocaleElements_es_CL.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_CL.LocaleElements_es_CL は変換されませんでした。

java.text.resources.LocaleElements_es_CO は変換されませんでした。

java.text.resources.LocaleElements_es_CO.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_CO.LocaleElements_es_CO は変換されませんでした。

java.text.resources.LocaleElements_es_CR は変換されませんでした。

java.text.resources.LocaleElements_es_CR.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_CR.LocaleElements_es_CR は変換されませんでした。

java.text.resources.LocaleElements_es_DO は変換されませんでした。

java.text.resources.LocaleElements_es_DO.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_DO.LocaleElements_es_DO は変換されませんでした。

java.text.resources.LocaleElements_es_EC は変換されませんでした。

java.text.resources.LocaleElements_es_EC.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_EC.LocaleElements_es_EC は変換されませんでした。

java.text.resources.LocaleElements_es_ES は変換されませんでした。

java.text.resources.LocaleElements_es_ES.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_ES.LocaleElements_es_ES は変換されませんでした。

java.text.resources.LocaleElements_es_ES_EURO は変換されませんでした。

java.text.resources.LocaleElements_es_ES_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_ES_EURO.LocaleElements_es_ES_EURO は変換されませんでした。

java.text.resources.LocaleElements_es_GT は変換されませんでした。

java.text.resources.LocaleElements_es_GT.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_GT.LocaleElements_es_GT は変換されませんでした。

java.text.resources.LocaleElements_es_HN は変換されませんでした。

java.text.resources.LocaleElements_es_HN.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_HN.LocaleElements_es_HN は変換されませんでした。

java.text.resources.LocaleElements_es_MX は変換されませんでした。

java.text.resources.LocaleElements_es_MX.LocaleElements_es_MX は変換されませんでした。

java.text.resources.LocaleElements_es_NI は変換されませんでした。

java.text.resources.LocaleElements_es_NI.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_NI.LocaleElements_es_NI は変換されませんでした。

java.text.resources.LocaleElements_es_PA は変換されませんでした。

java.text.resources.LocaleElements_es_PA.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_PA.LocaleElements_es_PA は変換されませんでした。

java.text.resources.LocaleElements_es_PE は変換されませんでした。

java.text.resources.LocaleElements_es_PE.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_PE.LocaleElements_es_PE は変換されませんでした。

java.text.resources.LocaleElements_es_PR は変換されませんでした。

java.text.resources.LocaleElements_es_PR.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_PR.LocaleElements_es_PR は変換されませんでした。

java.text.resources.LocaleElements_es_PY は変換されませんでした。

java.text.resources.LocaleElements_es_PY.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_PY.LocaleElements_es_PY は変換されませんでした。

java.text.resources.LocaleElements_es_SV は変換されませんでした。

java.text.resources.LocaleElements_es_SV.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_SV.LocaleElements_es_SV は変換されませんでした。

java.text.resources.LocaleElements_es_UY は変換されませんでした。

java.text.resources.LocaleElements_es_UY.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_UY.LocaleElements_es_UY は変換されませんでした。

java.text.resources.LocaleElements_es_VE は変換されませんでした。

java.text.resources.LocaleElements_es_VE.getContents は変換されませんでした。

java.text.resources.LocaleElements_es_VE.LocaleElements_es_VE は変換されませんでした。

java.text.resources.LocaleElements_et は変換されませんでした。

java.text.resources.LocaleElements_et.LocaleElements_et は変換されませんでした。

java.text.resources.LocaleElements_et_EE は変換されませんでした。

java.text.resources.LocaleElements_et_EE.getContents は変換されませんでした。

java.text.resources.LocaleElements_et_EE.LocaleElements_et_EE は変換されませんでした。

java.text.resources.LocaleElements_fi は変換されませんでした。

java.text.resources.LocaleElements_fi.LocaleElements_fi は変換されませんでした。

java.text.resources.LocaleElements_fi_FI は変換されませんでした。

java.text.resources.LocaleElements_fi_FI.getContents は変換されませんでした。

java.text.resources.LocaleElements_fi_FI.LocaleElements_fi_FI は変換されませんでした。

java.text.resources.LocaleElements_fi_FI_EURO は変換されませんでした。

java.text.resources.LocaleElements_fi_FI_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_fi_FI_EURO.LocaleElements_fi_FI_EURO は変換されませんでした。

java.text.resources.LocaleElements_fr は変換されませんでした。

java.text.resources.LocaleElements_fr.LocaleElements_fr は変換されませんでした。

java.text.resources.LocaleElements_fr_BE は変換されませんでした。

java.text.resources.LocaleElements_fr_BE.LocaleElements_fr_BE は変換されませんでした。

java.text.resources.LocaleElements_fr_BE_EURO は変換されませんでした。

java.text.resources.LocaleElements_fr_BE_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_fr_BE_EURO.LocaleElements_fr_BE_EURO は変換されませんでした。

java.text.resources.LocaleElements_fr_CA は変換されませんでした。

java.text.resources.LocaleElements_fr_CA.LocaleElements_fr_CA は変換されませんでした。

java.text.resources.LocaleElements_fr_CH は変換されませんでした。

java.text.resources.LocaleElements_fr_CH.LocaleElements_fr_CH は変換されませんでした。

java.text.resources.LocaleElements_fr_FR は変換されませんでした。

java.text.resources.LocaleElements_fr_FR.getContents は変換されませんでした。

java.text.resources.LocaleElements_fr_FR.LocaleElements_fr_FR は変換されませんでした。

java.text.resources.LocaleElements_fr_FR_EURO は変換されませんでした。

java.text.resources.LocaleElements_fr_FR_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_fr_FR_EURO.LocaleElements_fr_FR_EURO は変換されませんでした。

java.text.resources.LocaleElements_fr_LU は変換されませんでした。

java.text.resources.LocaleElements_fr_LU.getContents は変換されませんでした。

java.text.resources.LocaleElements_fr_LU.LocaleElements_fr_LU は変換されませんでした。

java.text.resources.LocaleElements_fr_LU_EURO は変換されませんでした。

java.text.resources.LocaleElements_fr_LU_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_fr_LU_EURO.LocaleElements_fr_LU_EURO は変換されませんでした。

java.text.resources.LocaleElements_he は変換されませんでした。

java.text.resources.LocaleElements_he.LocaleElements_he は変換されませんでした。

java.text.resources.LocaleElements_hr は変換されませんでした。

java.text.resources.LocaleElements_hr.LocaleElements_hr は変換されませんでした。

java.text.resources.LocaleElements_hr_HR は変換されませんでした。

java.text.resources.LocaleElements_hr_HR.getContents は変換されませんでした。

java.text.resources.LocaleElements_hr_HR.LocaleElements_hr_HR は変換されませんでした。

java.text.resources.LocaleElements_hu は変換されませんでした。

java.text.resources.LocaleElements_hu.LocaleElements_hu は変換されませんでした。

java.text.resources.LocaleElements_hu_HU は変換されませんでした。

java.text.resources.LocaleElements_hu_HU.getContents は変換されませんでした。

java.text.resources.LocaleElements_hu_HU.LocaleElements_hu_HU は変換されませんでした。

java.text.resources.LocaleElements_is は変換されませんでした。

java.text.resources.LocaleElements_is.LocaleElements_is は変換されませんでした。

java.text.resources.LocaleElements_is_IS は変換されませんでした。

java.text.resources.LocaleElements_is_IS.getContents は変換されませんでした。

java.text.resources.LocaleElements_is_IS.LocaleElements_is_IS は変換されませんでした。

java.text.resources.LocaleElements_it は変換されませんでした。

java.text.resources.LocaleElements_it.LocaleElements_it は変換されませんでした。

java.text.resources.LocaleElements_it_CH は変換されませんでした。

java.text.resources.LocaleElements_it_CH.LocaleElements_it_CH は変換されませんでした。

java.text.resources.LocaleElements_it_IT は変換されませんでした。

java.text.resources.LocaleElements_it_IT.getContents は変換されませんでした。

java.text.resources.LocaleElements_it_IT.LocaleElements_it_IT は変換されませんでした。

java.text.resources.LocaleElements_it_IT_EURO は変換されませんでした。

java.text.resources.LocaleElements_it_IT_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_it_IT_EURO.LocaleElements_it_IT_EURO は変換されませんでした。

java.text.resources.LocaleElements_iw は変換されませんでした。

java.text.resources.LocaleElements_iw.LocaleElements_iw は変換されませんでした。

java.text.resources.LocaleElements_iw_IL は変換されませんでした。

java.text.resources.LocaleElements_iw_IL.getContents は変換されませんでした。

java.text.resources.LocaleElements_iw_IL.LocaleElements_iw_IL は変換されませんでした。

java.text.resources.LocaleElements_ja は変換されませんでした。

java.text.resources.LocaleElements_ja.LocaleElements_ja は変換されませんでした。

java.text.resources.LocaleElements_ja_JP は変換されませんでした。

java.text.resources.LocaleElements_ja_JP.getContents は変換されませんでした。

java.text.resources.LocaleElements_ja_JP.LocaleElements_ja_JP は変換されませんでした。

java.text.resources.LocaleElements_ko は変換されませんでした。

java.text.resources.LocaleElements_ko.LocaleElements_ko は変換されませんでした。

java.text.resources.LocaleElements_ko_KR は変換されませんでした。

java.text.resources.LocaleElements_ko_KR.getContents は変換されませんでした。

java.text.resources.LocaleElements_ko_KR.LocaleElements_ko_KR は変換されませんでした。

java.text.resources.LocaleElements_lt は変換されませんでした。

java.text.resources.LocaleElements_lt.LocaleElements_lt は変換されませんでした。

java.text.resources.LocaleElements_lt_LT は変換されませんでした。

java.text.resources.LocaleElements_lt_LT.getContents は変換されませんでした。

java.text.resources.LocaleElements_lt_LT.LocaleElements_lt_LT は変換されませんでした。

java.text.resources.LocaleElements_lv は変換されませんでした。

java.text.resources.LocaleElements_lv.LocaleElements_lv は変換されませんでした。

java.text.resources.LocaleElements_lv_LV は変換されませんでした。

java.text.resources.LocaleElements_lv_LV.getContents は変換されませんでした。

java.text.resources.LocaleElements_lv_LV.LocaleElements_lv_LV は変換されませんでした。

java.text.resources.LocaleElements_mk は変換されませんでした。

java.text.resources.LocaleElements_mk.LocaleElements_mk は変換されませんでした。

java.text.resources.LocaleElements_mk_MK は変換されませんでした。

java.text.resources.LocaleElements_mk_MK.getContents は変換されませんでした。

java.text.resources.LocaleElements_mk_MK.LocaleElements_mk_MK は変換されませんでした。

java.text.resources.LocaleElements_nl は変換されませんでした。

java.text.resources.LocaleElements_nl.LocaleElements_nl は変換されませんでした。

java.text.resources.LocaleElements_nl_BE は変換されませんでした。

java.text.resources.LocaleElements_nl_BE.LocaleElements_nl_BE は変換されませんでした。

java.text.resources.LocaleElements_nl_BE_EURO は変換されませんでした。

java.text.resources.LocaleElements_nl_BE_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_nl_BE_EURO.LocaleElements_nl_BE_EURO は変換されませんでした。

java.text.resources.LocaleElements_nl_NL は変換されませんでした。

java.text.resources.LocaleElements_nl_NL.getContents は変換されませんでした。

java.text.resources.LocaleElements_nl_NL.LocaleElements_nl_NL は変換されませんでした。

java.text.resources.LocaleElements_nl_NL_EURO は変換されませんでした。

java.text.resources.LocaleElements_nl_NL_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_nl_NL_EURO.LocaleElements_nl_NL_EURO は変換されませんでした。

java.text.resources.LocaleElements_no は変換されませんでした。

java.text.resources.LocaleElements_no.LocaleElements_no は変換されませんでした。

java.text.resources.LocaleElements_no_NO は変換されませんでした。

java.text.resources.LocaleElements_no_NO.getContents は変換されませんでした。

java.text.resources.LocaleElements_no_NO.LocaleElements_no_NO は変換されませんでした。

java.text.resources.LocaleElements_no_NO_NY は変換されませんでした。

java.text.resources.LocaleElements_no_NO_NY.LocaleElements_no_NO_NY は変換されませんでした。

java.text.resources.LocaleElements_pl は変換されませんでした。

java.text.resources.LocaleElements_pl.LocaleElements_pl は変換されませんでした。

java.text.resources.LocaleElements_pl_PL は変換されませんでした。

java.text.resources.LocaleElements_pl_PL.getContents は変換されませんでした。

java.text.resources.LocaleElements_pl_PL.LocaleElements_pl_PL は変換されませんでした。

java.text.resources.LocaleElements_pt は変換されませんでした。

java.text.resources.LocaleElements_pt.LocaleElements_pt は変換されませんでした。

java.text.resources.LocaleElements_pt_BR は変換されませんでした。

java.text.resources.LocaleElements_pt_BR.getContents は変換されませんでした。

java.text.resources.LocaleElements_pt_BR.LocaleElements_pt_BR は変換されませんでした。

java.text.resources.LocaleElements_pt_BZ は変換されませんでした。

java.text.resources.LocaleElements_pt_BZ.LocaleElements_pt_BZ は変換されませんでした。

java.text.resources.LocaleElements_pt_PT は変換されませんでした。

java.text.resources.LocaleElements_pt_PT.getContents は変換されませんでした。

java.text.resources.LocaleElements_pt_PT.LocaleElements_pt_PT は変換されませんでした。

java.text.resources.LocaleElements_pt_PT_EURO は変換されませんでした。

java.text.resources.LocaleElements_pt_PT_EURO.getContents は変換されませんでした。

java.text.resources.LocaleElements_pt_PT_EURO.LocaleElements_pt_PT_EURO は変換されませんでした。

java.text.resources.LocaleElements_ro は変換されませんでした。

java.text.resources.LocaleElements_ro.LocaleElements_ro は変換されませんでした。

java.text.resources.LocaleElements_ro_RO は変換されませんでした。

java.text.resources.LocaleElements_ro_RO.getContents は変換されませんでした。

java.text.resources.LocaleElements_ro_RO.LocaleElements_ro_RO は変換されませんでした。

java.text.resources.LocaleElements_ru は変換されませんでした。

java.text.resources.LocaleElements_ru.LocaleElements_ru は変換されませんでした。

java.text.resources.LocaleElements_ru_RU は変換されませんでした。

java.text.resources.LocaleElements_ru_RU.getContents は変換されませんでした。

java.text.resources.LocaleElements_ru_RU.LocaleElements_ru_RU は変換されませんでした。

java.text.resources.LocaleElements_sh は変換されませんでした。

java.text.resources.LocaleElements_sh.LocaleElements_sh は変換されませんでした。

java.text.resources.LocaleElements_sh_YU は変換されませんでした。

java.text.resources.LocaleElements_sh_YU.getContents は変換されませんでした。

java.text.resources.LocaleElements_sh_YU.LocaleElements_sh_YU は変換されませんでした。

java.text.resources.LocaleElements_sk は変換されませんでした。

java.text.resources.LocaleElements_sk.LocaleElements_sk は変換されませんでした。

java.text.resources.LocaleElements_sk_SK は変換されませんでした。

java.text.resources.LocaleElements_sk_SK.getContents は変換されませんでした。

java.text.resources.LocaleElements_sk_SK.LocaleElements_sk_SK は変換されませんでした。

java.text.resources.LocaleElements_sl は変換されませんでした。

java.text.resources.LocaleElements_sl.LocaleElements_sl は変換されませんでした。

java.text.resources.LocaleElements_sl_SI は変換されませんでした。

java.text.resources.LocaleElements_sl_SI.getContents は変換されませんでした。

java.text.resources.LocaleElements_sl_SI.LocaleElements_sl_SI は変換されませんでした。

java.text.resources.LocaleElements_sq は変換されませんでした。

java.text.resources.LocaleElements_sq.LocaleElements_sq は変換されませんでした。

java.text.resources.LocaleElements_sq_AL は変換されませんでした。

java.text.resources.LocaleElements_sq_AL.getContents は変換されませんでした。

java.text.resources.LocaleElements_sq_AL.LocaleElements_sq_AL は変換されませんでした。

java.text.resources.LocaleElements_sr は変換されませんでした。

java.text.resources.LocaleElements_sr.LocaleElements_sr は変換されませんでした。

java.text.resources.LocaleElements_sr_YU は変換されませんでした。

java.text.resources.LocaleElements_sr_YU.getContents は変換されませんでした。

java.text.resources.LocaleElements_sr_YU.LocaleElements_sr_YU は変換されませんでした。

java.text.resources.LocaleElements_sv は変換されませんでした。

java.text.resources.LocaleElements_sv.LocaleElements_sv は変換されませんでした。

java.text.resources.LocaleElements_sv_SE は変換されませんでした。

java.text.resources.LocaleElements_sv_SE.getContents は変換されませんでした。

java.text.resources.LocaleElements_sv_SE.LocaleElements_sv_SE は変換されませんでした。

java.text.resources.LocaleElements_th は変換されませんでした。

java.text.resources.LocaleElements_th.getContents は変換されませんでした。

java.text.resources.LocaleElements_th.LocaleElements_th は変換されませんでした。

java.text.resources.LocaleElements_th_TH は変換されませんでした。

java.text.resources.LocaleElements_th_TH.getContents は変換されませんでした。

java.text.resources.LocaleElements_th_TH.LocaleElements_th_TH は変換されませんでした。

java.text.resources.LocaleElements_tr は変換されませんでした。

java.text.resources.LocaleElements_tr.LocaleElements_tr は変換されませんでした。

java.text.resources.LocaleElements_tr_TR は変換されませんでした。

java.text.resources.LocaleElements_tr_TR.getContents は変換されませんでした。

java.text.resources.LocaleElements_tr_TR.LocaleElements_tr_TR は変換されませんでした。

java.text.resources.LocaleElements_uk は変換されませんでした。

java.text.resources.LocaleElements_uk.LocaleElements_uk は変換されませんでした。

java.text.resources.LocaleElements_uk_UA は変換されませんでした。

java.text.resources.LocaleElements_uk_UA.getContents は変換されませんでした。

java.text.resources.LocaleElements_uk_UA.LocaleElements_uk_UA は変換されませんでした。

java.text.resources.LocaleElements_zh は変換されませんでした。

java.text.resources.LocaleElements_zh.LocaleElements_zh は変換されませんでした。

java.text.resources.LocaleElements_zh_CN は変換されませんでした。

java.text.resources.LocaleElements_zh_CN.getContents は変換されませんでした。

java.text.resources.LocaleElements_zh_CN.LocaleElements_zh_CN は変換されませんでした。

java.text.resources.LocaleElements_zh_HK は変換されませんでした。

java.text.resources.LocaleElements_zh_HK.getContents は変換されませんでした。

java.text.resources.LocaleElements_zh_HK.LocaleElements_zh_HK は変換されませんでした。

java.text.resources.LocaleElements_zh_SG は変換されませんでした。

java.text.resources.LocaleElements_zh_SG.LocaleElements_zh_SG は変換されませんでした。

java.text.resources.LocaleElements_zh_TW は変換されませんでした。

java.text.resources.LocaleElements_zh_TW.LocaleElements_zh_TW は変換されませんでした。

Java.util のエラー メッセージ

java.util.AbstractCollection.AbstractCollection は変換されませんでした。

java.util.AbstractCollection.add は変換されませんでした。

java.util.AbstractCollection.addAll は変換されませんでした。

java.util.AbstractCollection.contains は変換されませんでした。

java.util.AbstractCollection.containsAll は変換されませんでした。

java.util.AbstractCollection.iterator は変換されませんでした。

java.util.AbstractCollection.remove は変換されませんでした。

java.util.AbstractCollection.removeAll は変換されませんでした。

java.util.AbstractCollection.retainAll は変換されませんでした。

java.util.AbstractCollection.toArray は変換されませんでした。

java.util.AbstractCollection.toArray(Object[]) は変換されませんでした。

java.util.AbstractList.AbstractList は変換されませんでした。

java.util.AbstractList.add は変換されませんでした。

java.util.AbstractList.addAll は変換されませんでした。

java.util.AbstractList.lastIndexOf は変換されませんでした。

java.util.AbstractList.listIterator は変換されませんでした。

java.util.AbstractList.listIterator(int) は変換されませんでした。

java.util.AbstractList.modCount は変換されませんでした。

java.util.AbstractList.remove は変換されませんでした。

java.util.AbstractList.removeRange は変換されませんでした。

java.util.AbstractList.subList は変換されませんでした。

java.util.AbstractMap は変換されませんでした。

java.util.AbstractMap.AbstractMap は変換されませんでした。

java.util.AbstractMap.clear は変換されませんでした。

java.util.AbstractMap.containsKey は変換されませんでした。

java.util.AbstractMap.containsValue は変換されませんでした。

java.util.AbstractMap.entrySet は変換されませんでした。

java.util.AbstractMap.equals は変換されませんでした。

java.util.AbstractMap.get は変換されませんでした。

java.util.AbstractMap.hashCode は変換されませんでした。

java.util.AbstractMap.keySet は変換されませんでした。

java.util.AbstractMap.put は変換されませんでした。

java.util.AbstractMap.putAll は変換されませんでした。

java.util.AbstractMap.remove は変換されませんでした。

java.util.AbstractMap.size は変換されませんでした。

java.util.AbstractMap.values は変換されませんでした。

java.util.AbstractSequentialList.AbstractSequentialList は変換されませんでした。

java.util.AbstractSequentialList.addAll は変換されませんでした。

java.util.AbstractSequentialList.listIterator は変換されませんでした。

java.util.AbstractSequentialList.remove は変換されませんでした。

java.util.AbstractSequentialList.set は変換されませんでした。

java.util.AbstractSet.AbstractSet は変換されませんでした。

java.util.AbstractSet.removeAll は変換されませんでした。

java.util.ArrayList.add は変換されませんでした。

java.util.ArrayList.toArray は変換されませんでした。

java.util.Arrays.asList は変換されませんでした。

java.util.Arrays.sort は変換されませんでした。

java.util.BitSet.andNot は変換されませんでした。

java.util.BitSet.equals は変換されませんでした。

java.util.BitSet.length は変換されませんでした。

java.util.Calendar.add は変換されませんでした。

java.util.Calendar.after は変換されませんでした。

java.util.Calendar.AM は変換されませんでした。

java.util.Calendar.AM_PM は変換されませんでした。

java.util.Calendar.APRIL は変換されませんでした。

java.util.Calendar.areFieldsSet は変換されませんでした。

java.util.Calendar.AUGUST は変換されませんでした。

java.util.Calendar.before は変換されませんでした。

java.util.Calendar.Calendar は変換されませんでした。

java.util.Calendar.clear は変換されませんでした。

java.util.Calendar.clone は変換されませんでした。

java.util.Calendar.complete は変換されませんでした。

java.util.Calendar.computeFields は変換されませんでした。

java.util.Calendar.computeTime は変換されませんでした。

java.util.Calendar.DAY_OF_MONTH は変換されませんでした。

java.util.Calendar.DAY_OF_WEEK は変換されませんでした。

java.util.Calendar.DAY_OF_WEEK_IN_MONTH は変換されませんでした。

java.util.Calendar.DAY_OF_YEAR は変換されませんでした。

java.util.Calendar.DECEMBER は変換されませんでした。

java.util.Calendar.DST_OFFSET は変換されませんでした。

java.util.Calendar.ERA は変換されませんでした。

java.util.Calendar.FEBRUARY は変換されませんでした。

java.util.Calendar.FIELD_COUNT は変換されませんでした。

java.util.Calendar.fields は変換されませんでした。

java.util.Calendar.get は変換されませんでした。

java.util.Calendar.getActualMaximum は変換されませんでした。

java.util.Calendar.getActualMinimum は変換されませんでした。

java.util.Calendar.getAvailableLocales は変換されませんでした。

java.util.Calendar.getGreatestMinimum は変換されませんでした。

java.util.Calendar.getInstance は変換されませんでした。

java.util.Calendar.getLeastMaximum は変換されませんでした。

java.util.Calendar.getMaximum は変換されませんでした。

java.util.Calendar.getMinimalDaysInFirstWeek は変換されませんでした。

java.util.Calendar.getMinimum は変換されませんでした。

java.util.Calendar.getTimeInMillis は変換されませんでした。

java.util.Calendar.getTimeZone は変換されませんでした。

java.util.Calendar.HOUR は変換されませんでした。

java.util.Calendar.HOUR_OF_DAY は変換されませんでした。

java.util.Calendar.internalGet は変換されませんでした。

java.util.Calendar.isLenient は変換されませんでした。

java.util.Calendar.isSet は変換されませんでした。

java.util.Calendar.isSet~ は変換されませんでした。

java.util.Calendar.isTimeSet は変換されませんでした。

java.util.Calendar.JANUARY は変換されませんでした。

java.util.Calendar.JULY は変換されませんでした。

java.util.Calendar.JUNE は変換されませんでした。

java.util.Calendar.MARCH は変換されませんでした。

java.util.Calendar.MAY は変換されませんでした。

java.util.Calendar.NOVEMBER は変換されませんでした。

java.util.Calendar.OCTOBER は変換されませんでした。

java.util.Calendar.PM は変換されませんでした。

java.util.Calendar.roll は変換されませんでした。

java.util.Calendar.roll(int, boolean) は変換されませんでした。

java.util.Calendar.roll(int, int) は変換されませんでした。

java.util.Calendar.SEPTEMBER は変換されませんでした。

java.util.Calendar.setLenient は変換されませんでした。

java.util.Calendar.setMinimalDaysInFirstWeek は変換されませんでした。

java.util.Calendar.setTimeInMillis は変換されませんでした。

java.util.Calendar.setTimeZone は変換されませんでした。

java.util.Calendar.time は変換されませんでした。

java.util.Calendar.UNDECIMBER は変換されませんでした。

java.util.Calendar.WEEK_OF_MONTH は変換されませんでした。

java.util.Calendar.WEEK_OF_YEAR は変換されませんでした。

java.util.Calendar.ZONE_OFFSET は変換されませんでした。

java.util.Collection.add は変換されませんでした。

java.util.Collection.addAll は変換されませんでした。

java.util.Collection.clear は変換されませんでした。

java.util.Collection.contains は変換されませんでした。

java.util.Collection.containsAll は変換されませんでした。

java.util.Collection.remove は変換されませんでした。

java.util.Collection.removeAll は変換されませんでした。

java.util.Collection.retainAll は変換されませんでした。

java.util.Collection.toArray は変換されませんでした。

java.util.Collections は変換されませんでした。

java.util.Collections.EMPTY_MAP は変換されませんでした。

java.util.Collections.EMPTY_SET は変換されませんでした。

java.util.Collections.max(Collection) は変換されませんでした。

java.util.Collections.max(Collection, Comparator) は変換されませんでした。

java.util.Collections.min(Collection) は変換されませんでした。

java.util.Collections.min(Collection, Comparator) は変換されませんでした。

java.util.Collections.reverseOrder は変換されませんでした。

java.util.Collections.sort(Comparator) は変換されませんでした。

java.util.Collections.sort(List) は変換されませんでした。

java.util.Collections.synchronizedMap は変換されませんでした。

java.util.Collections.synchronizedSortedSet は変換されませんでした。

java.util.Collections.unmodifiableMap は変換されませんでした。

java.util.Collections.unmodifiableSortedMap は変換されませんでした。

java.util.Collections.unmodifiableSortedSet は変換されませんでした。

java.util.Date.clone は変換されませんでした。

java.util.Date.Date(int, int, int) は変換されませんでした。

java.util.Date.Date(int, int, int, int, int) は変換されませんでした。

java.util.Date.Date(int, int, int, int, int, int) は変換されませんでした。

java.util.Date.Date(long) は変換されませんでした。

java.util.Date.Date(String) は変換されませんでした。

java.util.Date.getMonth は変換されませんでした。

java.util.Date.getTime は変換されませんでした。

java.util.Date.getTimezoneOffset は変換されませんでした。

java.util.Date.getYear は変換されませんでした。

java.util.Date.hashCode は変換されませんでした。

java.util.Date.parse は変換されませんでした。

java.util.Date.setTime は変換されませんでした。

java.util.Date.toGMTString は変換されませんでした。

java.util.Date.toLocaleString は変換されませんでした。

java.util.Date.toString は変換されませんでした。

java.util.Date.UTC は変換されませんでした。

java.util.Dictionary.Dictionary は変換されませんでした。

java.util.Dictionary.get は変換されませんでした。

java.util.Dictionary.isEmpty は変換されませんでした。

java.util.Dictionary.keys は変換されませんでした。

java.util.Dictionary.put は変換されませんでした。

java.util.Dictionary.remove は変換されませんでした。

java.util.Enumeration.hasMoreElements は変換されませんでした。

java.util.Enumeration.nextElement は変換されませんでした。

java.util.Entry.setValue は変換されませんでした。

java.util.EventListener は変換されませんでした。

java.util.EventObject は変換されませんでした。

java.util.EventObject.source は変換されませんでした。

java.util.GregorianCalendar.add は変換されませんでした。

java.util.GregorianCalendar.after は変換されませんでした。

java.util.GregorianCalendar.before は変換されませんでした。

java.util.GregorianCalendar.computeFields は変換されませんでした。

java.util.GregorianCalendar.computeTime は変換されませんでした。

java.util.GregorianCalendar.getActualMaximum は変換されませんでした。

java.util.GregorianCalendar.getActualMinimum は変換されませんでした。

java.util.GregorianCalendar.getGreatestMinimum は変換されませんでした。

java.util.GregorianCalendar.getGregorianChange は変換されませんでした。

java.util.GregorianCalendar.getLeastMaximum は変換されませんでした。

java.util.GregorianCalendar.getMaximum は変換されませんでした。

java.util.GregorianCalendar.getMinimum は変換されませんでした。

java.util.GregorianCalendar.GregorianCalendar は変換されませんでした。

java.util.GregorianCalendar.roll は変換されませんでした。

java.util.GregorianCalendar.roll(int, boolean) は変換されませんでした。

java.util.GregorianCalendar.setGregorianChange は変換されませんでした。

java.util.HashMap は変換されませんでした。

java.util.HashMap.entrySet は変換されませんでした。

java.util.HashMap.get は変換されませんでした。

java.util.HashMap.keySet は変換されませんでした。

java.util.HashMap.putAll は変換されませんでした。

java.util.HashSet は変換されませんでした。

java.util.HashSet.add は変換されませんでした。

java.util.HashSet.clear は変換されませんでした。

java.util.HashSet.clone は変換されませんでした。

java.util.HashSet.contains は変換されませんでした。

java.util.HashSet.HashSet は変換されませんでした。

java.util.HashSet.HashSet(Collection) は変換されませんでした。

java.util.HashSet.HashSet(int) は変換されませんでした。

java.util.HashSet.HashSet(int, float) は変換されませんでした。

java.util.HashSet.remove は変換されませんでした。

java.util.Hashtable.entrySet は変換されませんでした。

java.util.Hashtable.putAll は変換されませんでした。

java.util.Hashtable.rehash は変換されませんでした。

java.util.HashtableEntry は変換されませんでした。

java.util.HashtableEnumerator は変換されませんでした。

java.util.Iterator.hasNext は変換されませんでした。

java.util.Iterator.next は変換されませんでした。

java.util.Iterator.remove は変換されませんでした。

java.util.LinkedList は変換されませんでした。

java.util.LinkedList.add は変換されませんでした。

java.util.LinkedList.listIterator は変換されませんでした。

java.util.List.add は変換されませんでした。

java.util.List.addAll は変換されませんでした。

java.util.List.containsAll は変換されませんでした。

java.util.List.lastIndexOf は変換されませんでした。

java.util.List.listIterator は変換されませんでした。

java.util.List.remove は変換されませんでした。

java.util.List.removeAll は変換されませんでした。

java.util.List.retainAll は変換されませんでした。

java.util.List.subList は変換されませんでした。

java.util.List.toArray は変換されませんでした。

java.util.ListIterator.add は変換されませんでした。

java.util.ListIterator.hasNext は変換されませんでした。

java.util.ListIterator.hasPrevious は変換されませんでした。

java.util.ListIterator.next は変換されませんでした。

java.util.ListIterator.nextIndex は変換されませんでした。

java.util.ListIterator.previous は変換されませんでした。

java.util.ListIterator.previousIndex は変換されませんでした。

java.util.ListIterator.remove は変換されませんでした。

java.util.ListIterator.set は変換されませんでした。

java.util.ListResourceBundle.getContents は変換されませんでした。

java.util.ListResourceBundle.getKeys は変換されませんでした。

java.util.ListResourceBundle.handleGetObject は変換されませんでした。

java.util.ListResourceBundle.ListResourceBundle は変換されませんでした。

java.util.Locale.CHINESE は変換されませんでした。

java.util.Locale.getDefault は変換されませんでした。

java.util.Locale.getDisplayName は変換されませんでした。

java.util.Locale.getDisplayVariant は変換されませんでした。

java.util.Locale.getISOCountries は変換されませんでした。

java.util.Locale.getLanguage は変換されませんでした。

java.util.Locale.getVariant は変換されませんでした。

java.util.Locale.hashCode は変換されませんでした。

java.util.Locale.setDefault は変換されませんでした。

java.util.Map.containsValue は変換されませんでした。

java.util.Map.entrySet は変換されませんでした。

java.util.Map.get は変換されませんでした。

java.util.Map.isEmpty は変換されませんでした。

java.util.Map.keySet は変換されませんでした。

java.util.Map.putAll は変換されませんでした。

java.util.MissingResourceException.getClassName は変換されませんでした。

java.util.MissingResourceException.getKey は変換されませんでした。

java.util.MissingResourceException.MissingResourceException は変換されませんでした。

java.util.Properties.defaults は変換されませんでした。

java.util.Properties.list は変換されませんでした。

java.util.Properties.load は変換されませんでした。

java.util.Properties.save は変換されませんでした。

java.util.Properties.setProperty は変換されませんでした。

java.util.Properties.store は変換されませんでした。

java.util.PropertyPermission は変換されませんでした。

java.util.PropertyResourceBundle.getKeys は変換されませんでした。

java.util.PropertyResourceBundle.PropertyResourceBundle は変換されませんでした。

java.util.Random.nextLong は変換されませんでした。

java.util.Random.nextBoolean は変換されませんでした。

java.util.Random.nextGaussian は変換されませんでした。

java.util.ResourceBundle.getKeys は変換されませんでした。

java.util.ResourceBundle.getObject は変換されませんでした。

java.util.ResourceBundle.getString は変換されませんでした。

java.util.ResourceBundle.getStringArray は変換されませんでした。

java.util.ResourceBundle.handleGetObject は変換されませんでした。

java.util.ResourceBundle.parent は変換されませんでした。

java.util.ResourceBundle.ResourceBundle は変換されませんでした。

java.util.ResourceBundle.setParent は変換されませんでした。

java.util.Set.add は変換されませんでした。

java.util.Set.addAll は変換されませんでした。

java.util.Set.clear は変換されませんでした。

java.util.Set.contains は変換されませんでした。

java.util.Set.containsAll は変換されませんでした。

java.util.Set.remove は変換されませんでした。

java.util.Set.removeAll は変換されませんでした。

java.util.Set.retainAll は変換されませんでした。

java.util.Set.toArray は変換されませんでした。

java.util.Set.toArray(Object[]) は変換されませんでした。

java.util.SimpleTimeZone は変換されませんでした。

java.util.SimpleTimeZone.clone は変換されませんでした。

java.util.SimpleTimeZone.getRawOffset は変換されませんでした。

java.util.SimpleTimeZone.setEndRule は変換されませんでした。

java.util.SimpleTimeZone.setRawOffset は変換されませんでした。

java.util.SimpleTimeZone.setStartRule は変換されませんでした。

java.util.SimpleTimeZone.setStartYear は変換されませんでした。

java.util.SimpleTimeZone.SimpleTimeZone は変換されませんでした。

java.util.SortedMap は変換されませんでした。

java.util.SortedMap.comparator は変換されませんでした。

java.util.SortedMap.headMap は変換されませんでした。

java.util.SortedMap.subMap は変換されませんでした。

java.util.SortedMap.tailMap は変換されませんでした。

java.util.SortedSet.comparator は変換されませんでした。

java.util.Stack.addElement は変換されませんでした。

java.util.Stack.elements は変換されませんでした。

java.util.StringTokenizer.StringTokenizer は変換されませんでした。

java.util.SystemClassLoader は変換されませんでした。

java.util.Timer.cancel は変換されませんでした。

java.util.Timer.schedule(TimerTask, Date) は変換されませんでした。

java.util.Timer.schedule(TimerTask, Date, long) は変換されませんでした。

java.util.Timer.schedule(TimerTask, long) は変換されませんでした。

java.util.Timer.schedule(TimerTask, long, long) は変換されませんでした。

java.util.Timer.scheduleAtFixedRate(TimerTask, Date, long) は変換されませんでした。

java.util.Timer.scheduleAtFixedRate(TimerTask, long, long) は変換されませんでした。

java.util.Timer.Timer は変換されませんでした。

java.util.TimerTask は変換されませんでした。

java.util.TimeZone.clone は変換されませんでした。

java.util.TimeZone.getAvailableIDs は変換されませんでした。

java.util.TimeZone.getDisplayName(boolean, int) は変換されませんでした。

java.util.TimeZone.getDisplayName(boolean, int, Locale) は変換されませんでした。

java.util.TimeZone.getDisplayName(Locale) は変換されませんでした。

java.util.TimeZone.getRawOffset は変換されませんでした。

java.util.TimeZone.getTimeZone は変換されませんでした。

java.util.TimeZone.hasSameRules は変換されませんでした。

java.util.TimeZone.setDefault は変換されませんでした。

java.util.TimeZone.setID は変換されませんでした。

java.util.TimeZone.setRawOffset は変換されませんでした。

java.util.TimeZone.TimeZone は変換されませんでした。

java.util.TreeMap.comparator は変換されませんでした。

java.util.TreeMap.entrySet は変換されませんでした。

java.util.TreeMap.headMap は変換されませんでした。

java.util.TreeMap.keySet は変換されませんでした。

java.util.TreeMap.putAll は変換されませんでした。

java.util.TreeMap.subMap は変換されませんでした。

java.util.TreeMap.tailMap は変換されませんでした。

java.util.TreeMap.TreeMap は変換されませんでした。

java.util.TreeMap.TreeMap(Map) は変換されませんでした。

java.util.TreeMap.TreeMap(SortedMap) は変換されませんでした。

java.util.TreeMap.values は変換されませんでした。

java.util.TreeSet は変換されませんでした。

java.util.TreeSet.comparator は変換されませんでした。

java.util.TreeSet.TreeSet(Comparator) は変換されませんでした。

java.util.TreeSet.TreeSet(SortedMap) は変換されませんでした。

java.util.TreeSet.TreeSet(SortedSet) は変換されませんでした。

java.util.Vector.add は変換されませんでした。

java.util.Vector.capacityIncrement は変換されませんでした。

java.util.Vector.containsAll は変換されませんでした。

java.util.Vector.elementData は変換されませんでした。

java.util.Vector.removeAll は変換されませんでした。

java.util.Vector.retainAll は変換されませんでした。

java.util.Vector.toArray は変換されませんでした。

java.util.Vector.Vector は変換されませんでした。

java.util.VectorEnumerator は変換されませんでした。

java.util.WeakHashMap は変換されませんでした。

java.util.WeakHashMap.entrySet は変換されませんでした。

java.util.WeakHashMap.WeakHashMap は変換されませんでした。

Java.util.cab のエラー メッセージ

com.ms.util.cab.CabConstants は変換されませんでした。

com.ms.util.cab.CabCorruptException は変換されませんでした。

com.ms.util.cab.CabCreator は変換されませんでした。

com.ms.util.cab.CabDecoder は変換されませんでした。

com.ms.util.cab.CabDecoderInterface は変換されませんでした。

Java.util.jar のエラー メッセージ

java.util.jar.Attributes は変換されませんでした。

java.util.jar.Attributes.Name は変換されませんでした。

java.util.jar.JarEntry は変換されませんでした。

java.util.jar.JarException は変換されませんでした。

java.util.jar.JarFile は変換されませんでした。

java.util.jar.JarInputStream は変換されませんでした。

java.util.jar.JarOutputStream は変換されませんでした。

java.util.jar.Manifest は変換されませんでした。

Java.util.zip のエラー メッセージ

java.util.zip.Adler32 は変換されませんでした。

java.util.zip.CheckedInputStream は変換されませんでした。

java.util.zip.CheckedInputStream.CheckedInputStream は変換されませんでした。

java.util.zip.CheckedInputStream.getChecksum は変換されませんでした。

java.util.zip.CheckedOutputStream は変換されませんでした。

java.util.zip.CheckedOutputStream.CheckedOutputStream は変換されませんでした。

java.util.zip.CheckedOutputStream.getChecksum は変換されませんでした。

java.util.zip.Checksum は変換されませんでした。

java.util.zip.CRC32 は変換されませんでした。

java.util.zip.DataFormatException は変換されませんでした。

java.util.zip.Deflater は変換されませんでした。

java.util.zip.DeflaterOutputStream は変換されませんでした。

java.util.zip.GZIPInputStream は変換されませんでした。

java.util.zip.GZIPOutputStream は変換されませんでした。

java.util.zip.Inflater は変換されませんでした。

java.util.zip.InflaterInputStream は変換されませんでした。

java.util.zip.ZipEntry は変換されませんでした。

java.util.zip.ZipFile は変換されませんでした。

java.util.zip.ZipInputStream は変換されませんでした。

java.util.zip.ZipOutputStream は変換されませんでした。

Javax.accessibility のエラー メッセージ

javax.accessibility.AccessibleAction は変換されませんでした。

javax.accessibility.AccessibleAction.doAccessibleAction は変換されませんでした。

javax.accessibility.AccessibleAction.getAccessibleActionCount は変換されませんでした。

javax.accessibility.AccessibleAction.getAccessibleActionDescription は変換されませんでした。

javax.accessibility.AccessibleBundle は変換されませんでした。

javax.accessibility.AccessibleComponent は変換されませんでした。

javax.accessibility.AccessibleComponent.addFocusListener は変換されませんでした。

javax.accessibility.AccessibleComponent.isVisible は変換されませんでした。

javax.accessibility.AccessibleComponent.removeFocusListener は変換されませんでした。

javax.accessibility.AccessibleComponent.setBounds は変換されませんでした。

javax.accessibility.AccessibleContext.<PropertyChangeEvent> は変換されませんでした。

javax.accessibility.AccessibleContext.accessibleDescription は変換されませんでした。

javax.accessibility.AccessibleContext.accessibleName は変換されませんでした。

javax.accessibility.AccessibleContext.accessibleParent は変換されませんでした。

javax.accessibility.AccessibleContext.addPropertyChangeListener は変換されませんでした。

javax.accessibility.AccessibleContext.firePropertyChange は変換されませんでした。

javax.accessibility.AccessibleContext.getAccessibleIcon は変換されませんでした。

javax.accessibility.AccessibleContext.getAccessibleIndexInParent は変換されませんでした。

javax.accessibility.AccessibleContext.getAccessibleParent は変換されませんでした。

javax.accessibility.AccessibleContext.getAccessibleRelationSet は変換されませんでした。

javax.accessibility.AccessibleContext.getAccessibleSelection は変換されませんでした。

javax.accessibility.AccessibleContext.getAccessibleTable は変換されませんでした。

javax.accessibility.AccessibleContext.getAccessibleText は変換されませんでした。

javax.accessibility.AccessibleContext.removePropertyChangeListener は変換されませんでした。

javax.accessibility.AccessibleContext.setAccessibleParent は変換されませんでした。

javax.accessibility.AccessibleHyperlink は変換されませんでした。

javax.accessibility.AccessibleHypertext は変換されませんでした。

javax.accessibility.AccessibleIcon は変換されませんでした。

javax.accessibility.AccessibleRelation は変換されませんでした。

javax.accessibility.AccessibleRelationSet は変換されませんでした。

javax.accessibility.AccessibleResourceBundle は変換されませんでした。

javax.accessibility.AccessibleRole.AccessibleRole は変換されませんでした。

javax.accessibility.AccessibleSelection は変換されませんでした。

javax.accessibility.AccessibleSelection.addAccessibleSelection は変換されませんでした。

javax.accessibility.AccessibleSelection.clearAccessibleSelection は変換されませんでした。

javax.accessibility.AccessibleSelection.getAccessibleSelection は変換されませんでした。

javax.accessibility.AccessibleSelection.getAccessibleSelectionCount は変換されませんでした。

javax.accessibility.AccessibleSelection.isAccessibleChildSelected は変換されませんでした。

javax.accessibility.AccessibleSelection.removeAccessibleSelection は変換されませんでした。

javax.accessibility.AccessibleSelection.selectAllAccessibleSelection は変換されませんでした。

javax.accessibility.AccessibleState.<StateName> は変換されませんでした。

javax.accessibility.AccessibleState.AccessibleState は変換されませんでした。

javax.accessibility.AccessibleStateSet.states は変換されませんでした。

javax.accessibility.AccessibleStateSet.toArray は変換されませんでした。

javax.accessibility.AccessibleTable は変換されませんでした。

javax.accessibility.AccessibleTableModelChange は変換されませんでした。

javax.accessibility.AccessibleText は変換されませんでした。

javax.accessibility.AccessibleText.<TextType> は変換されませんでした。

javax.accessibility.AccessibleText.getAfterIndex は変換されませんでした。

javax.accessibility.AccessibleText.getAtIndex は変換されませんでした。

javax.accessibility.AccessibleText.getBeforeIndex は変換されませんでした。

javax.accessibility.AccessibleText.getCaretPosition は変換されませんでした。

javax.accessibility.AccessibleText.getCharacterAttribute は変換されませんでした。

javax.accessibility.AccessibleText.getCharacterBounds は変換されませんでした。

javax.accessibility.AccessibleText.getCharCount は変換されませんでした。

javax.accessibility.AccessibleText.getIndexAtPoint は変換されませんでした。

javax.accessibility.AccessibleText.getSelectedText は変換されませんでした。

javax.accessibility.AccessibleText.getSelectionEnd は変換されませんでした。

javax.accessibility.AccessibleText.getSelectionStart は変換されませんでした。

javax.accessibility.AccessibleValue は変換されませんでした。

javax.accessibility.AccessibleValue.getCurrentAccessibleValue は変換されませんでした。

javax.accessibility.AccessibleValue.getMaximumAccessibleValue は変換されませんでした。

javax.accessibility.AccessibleValue.getMinimumAccessibleValue は変換されませんでした。

javax.accessibility.AccessibleValue.setCurrentAccessibleValue は変換されませんでした。

Javax.crypto のエラー メッセージ

javax.crypto.Cipher.<KeyType> は変換されませんでした。

javax.crypto.Cipher.Cipher は変換されませんでした。

javax.crypto.Cipher.doFinal は変換されませんでした。

javax.crypto.Cipher.getExemptionMechanism は変換されませんでした。

javax.crypto.Cipher.getInstance は変換されませんでした。

javax.crypto.Cipher.getParameters は変換されませんでした。

javax.crypto.Cipher.getProvider は変換されませんでした。

javax.crypto.Cipher.init は変換されませんでした。

javax.crypto.Cipher.t は変換されませんでした。

javax.crypto.Cipher.unwrap は変換されませんでした。

javax.crypto.Cipher.UNWRAP_MODE は変換されませんでした。

javax.crypto.Cipher.wrap は変換されませんでした。

javax.crypto.Cipher.WRAP_MODE は変換されませんでした。

javax.crypto.CipherInputStream.available は変換されませんでした。

javax.crypto.CipherInputStream.CipherInputStream(Stream) は変換されませんでした。

javax.crypto.CipherInputStream.CipherInputStream(Stream, Cipher) は変換されませんでした。

javax.crypto.CipherInputStream.skip は変換されませんでした。

javax.crypto.CipherOutputStream.CipherOutputStream(OutputStream) は変換されませんでした。

javax.crypto.CipherOutputStream.CipherOutputStream(OutputStream, Cipher) は変換されませんでした。

javax.crypto.CipherSpi は変換されませんでした。

javax.crypto.ExemptionMechanism は変換されませんでした。

javax.crypto.ExemptionMechanismSpi は変換されませんでした。

javax.crypto.interfaces.DHKey は変換されませんでした。

javax.crypto.interfaces.DHPrivateKey は変換されませんでした。

javax.crypto.interfaces.DHPublicKey は変換されませんでした。

javax.crypto.KeyAgreement は変換されませんでした。

javax.crypto.KeyAgreementSpi は変換されませんでした。

javax.crypto.KeyGenerator.generateKey は変換されませんでした。

javax.crypto.KeyGenerator.getAlgorithm は変換されませんでした。

javax.crypto.KeyGenerator.getProvider は変換されませんでした。

javax.crypto.KeyGenerator.init は変換されませんでした。

javax.crypto.KeyGeneratorSpi.engineGenerateKey は変換されませんでした。

javax.crypto.KeyGeneratorSpi.engineInit は変換されませんでした。

javax.crypto.Mac.clone は変換されませんでした。

javax.crypto.Mac.doFinal は変換されませんでした。

javax.crypto.Mac.getInstance は変換されませんでした。

javax.crypto.Mac.getProvider は変換されませんでした。

javax.crypto.Mac.Mac は変換されませんでした。

javax.crypto.MacSpi.clone は変換されませんでした。

javax.crypto.MacSpi.engineDoFinal は変換されませんでした。

javax.crypto.MacSpi.MacSpi は変換されませんでした。

javax.crypto.NullCipher.NullCipher は変換されませんでした。

javax.crypto.SealedObject は変換されませんでした。

javax.crypto.SealedObject.encodedParams は変換されませんでした。

javax.crypto.SealedObject.getAlgorithm は変換されませんでした。

javax.crypto.SealedObject.getObject は変換されませんでした。

javax.crypto.SealedObject.SealedObject は変換されませんでした。

javax.crypto.SecretKey は変換されませんでした。

javax.crypto.SecretKeyFactory.generateSecret は変換されませんでした。

javax.crypto.SecretKeyFactory.getAlgorithm は変換されませんでした。

javax.crypto.SecretKeyFactory.getKeySpec は変換されませんでした。

javax.crypto.SecretKeyFactory.getProvider は変換されませんでした。

javax.crypto.SecretKeyFactory.translateKey は変換されませんでした。

javax.crypto.SecretKeyFactorySpi.engineGenerateSecret は変換されませんでした。

javax.crypto.SecretKeyFactorySpi.engineGetKeySpec は変換されませんでした。

javax.crypto.SecretKeyFactorySpi.engineTranslateKey は変換されませんでした。

javax.crypto.ShortBufferException.a は変換されませんでした。

javax.crypto.spec.DESedeKeySpec.isParityAdjusted は変換されませんでした。

javax.crypto.spec.DESKeySpec.isParityAdjusted は変換されませんでした。

javax.crypto.spec.DESKeySpec.isWeak は変換されませんでした。

javax.crypto.spec.DHGenParameterSpec は変換されませんでした。

javax.crypto.spec.DHParameterSpec は変換されませんでした。

javax.crypto.spec.DHPrivateKeySpec は変換されませんでした。

javax.crypto.spec.DHPublicKeySpec は変換されませんでした。

javax.crypto.spec.IvParameterSpec.IvParameterSpec は変換されませんでした。

javax.crypto.spec.PBEKeySpec.getPassword は変換されませんでした。

javax.crypto.spec.PBEParameterSpec は変換されませんでした。

javax.crypto.spec.RC5ParameterSpec は変換されませんでした。

javax.crypto.spec.SecretKeySpec.getAlgorithm は変換されませんでした。

Javax.ejb のエラー メッセージ

javax.ejb.deployment.AccessControlEntry は変換されませんでした。

javax.ejb.deployment.ControlDescriptor は変換されませんでした。

javax.ejb.deployment.DeploymentDescriptor は変換されませんでした。

javax.ejb.deployment.EntityDescriptor は変換されませんでした。

javax.ejb.deployment.SessionDescriptor は変換されませんでした。

javax.ejb.EJBContext は変換されませんでした。

javax.ejb.EJBContext.getCallerIdentity は変換されませんでした。

javax.ejb.EJBContext.getCallerPrincipal は変換されませんでした。

javax.ejb.EJBContext.getEJBHome は変換されませんでした。

javax.ejb.EJBContext.getEJBLocalHome は変換されませんでした。

javax.ejb.EJBContext.getEnvironment は変換されませんでした。

javax.ejb.EJBContext.getRollbackOnly は変換されませんでした。

javax.ejb.EJBContext.getUserTransaction は変換されませんでした。

javax.ejb.EJBContext.isCallerInRole は変換されませんでした。

javax.ejb.EJBContext.setRollbackOnly は変換されませんでした。

javax.ejb.EJBHome は変換されませんでした。

javax.ejb.EJBHome.getEJBMetaData は変換されませんでした。

javax.ejb.EJBHome.getHomeHandle は変換されませんでした。

javax.ejb.EJBLocalHome は変換されませんでした。

javax.ejb.EJBLocalObject は変換されませんでした。

javax.ejb.EJBLocalObject.getEJBLocalHome は変換されませんでした。

javax.ejb.EJBMetaData は変換されませんでした。

javax.ejb.EJBObject は変換されませんでした。

javax.ejb.EJBObject.getEJBHome は変換されませんでした。

javax.ejb.EJBObject.getHandle は変換されませんでした。

javax.ejb.EntityBean.setEntityContext は変換されませんでした。

javax.ejb.EntityBean.unsetEntityContext は変換されませんでした。

javax.ejb.EntityContext は変換されませんでした。

javax.ejb.EntityContext.getEJBLocalObject は変換されませんでした。

javax.ejb.EntityContext.getEJBObject は変換されませんでした。

javax.ejb.Handle は変換されませんでした。

javax.ejb.HomeHandle は変換されませんでした。

javax.ejb.MessageDrivenBean.setMessageDrivenContext は変換されませんでした。

javax.ejb.MessageDrivenContext は変換されませんでした。

javax.ejb.SessionBean.setSessionContext は変換されませんでした。

javax.ejb.SessionContext は変換されませんでした。

javax.ejb.SessionContext.getEJBLocalObject は変換されませんでした。

javax.ejb.SessionContext.getEJBObject は変換されませんでした。

javax.ejb.SessionSynchronization は変換されませんでした。

javax.ejb.spi.HandleDelegate は変換されませんでした。

Javax.jms のエラー メッセージ

javax.jms.BytesMessage.readBytes は変換されませんでした。

javax.jms.BytesMessage.readUTF は変換されませんでした。

javax.jms.BytesMessage.reset は変換されませんでした。

javax.jms.BytesMessage.writeBytes は変換されませんでした。

javax.jms.BytesMessage.writeObject は変換されませんでした。

javax.jms.BytesMessage.writeUTF は変換されませんでした。

javax.jms.Connection は変換されませんでした。

javax.jms.Connection.close は変換されませんでした。

javax.jms.Connection.getClientID は変換されませんでした。

javax.jms.Connection.getExceptionListener は変換されませんでした。

javax.jms.Connection.getMetaData は変換されませんでした。

javax.jms.Connection.setClientID は変換されませんでした。

javax.jms.Connection.setExceptionListener は変換されませんでした。

javax.jms.Connection.start は変換されませんでした。

javax.jms.Connection.stop は変換されませんでした。

javax.jms.ConnectionConsumer は変換されませんでした。

javax.jms.ConnectionFactory は変換されませんでした。

javax.jms.ConnectionMetaData は変換されませんでした。

javax.jms.DeliveryMode は変換されませんでした。

javax.jms.ExceptionListener は変換されませんでした。

javax.jms.IllegalStateException.IllegalStateException は変換されませんでした。

javax.jms.InvalidClientIDException は変換されませんでした。

javax.jms.InvalidDestinationException.InvalidDestinationException(String) は変換されませんでした。

javax.jms.InvalidDestinationException.InvalidDestinationException(String, String) は変換されませんでした。

javax.jms.InvalidSelectorException は変換されませんでした。

javax.jms.JMSException.getLinkedException は変換されませんでした。

javax.jms.JMSException.JMSException(String) は変換されませんでした。

javax.jms.JMSException.JMSException(String, String) は変換されませんでした。

javax.jms.JMSException.setLinkedException は変換されませんでした。

javax.jms.JMSSecurityException.JMSSecurityException(String) は変換されませんでした。

javax.jms.JMSSecurityException.JMSSecurityException(String, String) は変換されませんでした。

javax.jms.MapMessage.getFloat は変換されませんでした。

javax.jms.MapMessage.setBytes は変換されませんでした。

javax.jms.Message.acknowledge は変換されませんでした。

javax.jms.Message.clearProperties は変換されませんでした。

javax.jms.Message.getBooleanProperty は変換されませんでした。

javax.jms.Message.getByteProperty は変換されませんでした。

javax.jms.Message.getDoubleProperty は変換されませんでした。

javax.jms.Message.getFloatProperty は変換されませんでした。

javax.jms.Message.getIntProperty は変換されませんでした。

javax.jms.Message.getJMSCorrelationIDAsBytes は変換されませんでした。

javax.jms.Message.getJMSDestination は変換されませんでした。

javax.jms.Message.getJMSPriority は変換されませんでした。

javax.jms.Message.getJMSRedelivered は変換されませんでした。

javax.jms.Message.getJMSReplyTo は変換されませんでした。

javax.jms.Message.getJMSTimestamp は変換されませんでした。

javax.jms.Message.getJMSType は変換されませんでした。

javax.jms.Message.getLongProperty は変換されませんでした。

javax.jms.Message.getObjectProperty は変換されませんでした。

javax.jms.Message.getPropertyNames は変換されませんでした。

javax.jms.Message.getShortProperty は変換されませんでした。

javax.jms.Message.getStringProperty は変換されませんでした。

javax.jms.Message.propertyExists は変換されませんでした。

javax.jms.Message.setBooleanProperty は変換されませんでした。

javax.jms.Message.setByteProperty は変換されませんでした。

javax.jms.Message.setDoubleProperty は変換されませんでした。

javax.jms.Message.setFloatProperty は変換されませんでした。

javax.jms.Message.setIntProperty は変換されませんでした。

javax.jms.Message.setJMSCorrelationID は変換されませんでした。

javax.jms.Message.setJMSCorrelationIDAsBytes は変換されませんでした。

javax.jms.Message.setJMSDestination は変換されませんでした。

javax.jms.Message.setJMSMessageID は変換されませんでした。

javax.jms.Message.setJMSPriority は変換されませんでした。

javax.jms.Message.setJMSRedelivered は変換されませんでした。

javax.jms.Message.setJMSTimestamp は変換されませんでした。

javax.jms.Message.setJMSType は変換されませんでした。

javax.jms.Message.setLongProperty は変換されませんでした。

javax.jms.Message.setObjectProperty は変換されませんでした。

javax.jms.Message.setShortProperty は変換されませんでした。

javax.jms.Message.setStringProperty は変換されませんでした。

javax.jms.MessageConsumer.setMessageListener は変換されませんでした。

javax.jms.MessageConsumer.getMessageSelector は変換されませんでした。

javax.jms.MessageEOFException.MessageEOFException は変換されませんでした。

javax.jms.MessageFormatException は変換されませんでした。

javax.jms.MessageListener は変換されませんでした。

javax.jms.MessageNotReadableException は変換されませんでした。

javax.jms.MessageNotWriteableException は変換されませんでした。

javax.jms.MessageProducer.getDisableMessageID は変換されませんでした。

javax.jms.MessageProducer.getDisableMessageTimestamp は変換されませんでした。

javax.jms.MessageProducer.setDeliveryMode は変換されませんでした。

javax.jms.MessageProducer.setDisableMessageID は変換されませんでした。

javax.jms.MessageProducer.setDisableMessageTimestamp は変換されませんでした。

javax.jms.MessageProducer.setPriority は変換されませんでした。

javax.jms.MessageProducer.setTimeToLive は変換されませんでした。

javax.jms.QueueConnection は変換されませんでした。

javax.jms.QueueConnection.createConnectionConsumer は変換されませんでした。

javax.jms.QueueConnection.createQueueSession は変換されませんでした。

javax.jms.QueueConnectionFactory は変換されませんでした。

javax.jms.QueueConnectionFactory.createQueueConnection は変換されませんでした。

javax.jms.QueueConnectionFactory.createQueueConnection(String, String) は変換されませんでした。

javax.jms.QueueRequestor は変換されませんでした。

javax.jms.QueueSession は変換されませんでした。

javax.jms.QueueSession.createBrowser は変換されませんでした。

javax.jms.QueueSession.createReceiver は変換されませんでした。

javax.jms.QueueBrowser.getMessageSelector は変換されませんでした。

javax.jms.QueueConnection は変換されませんでした。

javax.jms.QueueConnection.createQueueSession は変換されませんでした。

javax.jms.QueueConnectionFactory は変換されませんでした。

javax.jms.QueueConnectionFactory.createQueueConnection は変換されませんでした。

javax.jms.QueueConnectionFactory.createQueueConnection(String, String) は変換されませんでした。

javax.jms.QueueSession は変換されませんでした。

javax.jms.ResourceAllocationException.ResourceAllocationException は変換されませんでした。

javax.jms.ServerSession は変換されませんでした。

javax.jms.ServerSessionPool は変換されませんでした。

javax.jms.Session は変換されませんでした。

javax.jms.Session.AUTO_ACKNOWLEDGE は変換されませんでした。

javax.jms.Session.CLIENT_ACKNOWLEDGE は変換されませんでした。

javax.jms.Session.close は変換されませんでした。

javax.jms.Session.commit は変換されませんでした。

javax.jms.Session.DUPS_OK_ACKNOWLEDGE は変換されませんでした。

javax.jms.Session.getMessageListener は変換されませんでした。

javax.jms.Session.getTransacted は変換されませんでした。

javax.jms.Session.recover は変換されませんでした。

javax.jms.Session.rollback は変換されませんでした。

javax.jms.Session.run は変換されませんでした。

javax.jms.Session.setMessageListener は変換されませんでした。

javax.jms.StreamMessage は変換されませんでした。

javax.jms.StreamMessage.readBytes は変換されませんでした。

javax.jms.StreamMessage.readObject は変換されませんでした。

javax.jms.StreamMessage.reset は変換されませんでした。

javax.jms.StreamMessage.writeBytes は変換されませんでした。

javax.jms.StreamMessage.writeObject は変換されませんでした。

javax.jms.TemporaryQueue は変換されませんでした。

javax.jms.TemporaryTopic は変換されませんでした。

javax.jms.TopicConnection は変換されませんでした。

javax.jms.TopicConnection.createConnectionConsumer は変換されませんでした。

javax.jms.TopicConnection.createDurableConnectionConsumer は変換されませんでした。

javax.jms.TopicConnection.createTopicSession は変換されませんでした。

javax.jms.TopicConnectionFactory は変換されませんでした。

javax.jms.TopicConnectionFactory.createTopicConnection は変換されませんでした。

javax.jms.TopicRequestor は変換されませんでした。

javax.jms.TopicSession は変換されませんでした。

javax.jms.TopicSession.createDurableSubscriber は変換されませんでした。

javax.jms.TopicSession.createSubscriber は変換されませんでした。

javax.jms.TopicSession.createTemporaryTopic は変換されませんでした。

javax.jms.TopicSession.unsubscribe は変換されませんでした。

javax.jms.TopicSubscriber.getNoLocal は変換されませんでした。

javax.jms.TransactionInProgressException.TransactionInProgressException は変換されませんでした。

javax.jms.TransactionRolledBackException.TransactionRolledBackException は変換されませんでした。

javax.jms.XAConnection は変換されませんでした。

javax.jms.XAConnectionFactory は変換されませんでした。

javax.jms.XAQueueConnection は変換されませんでした。

javax.jms.XAQueueConnectionFactory は変換されませんでした。

javax.jms.XAQueueSession は変換されませんでした。

javax.jms.XASession は変換されませんでした。

javax.jms.XATopicConnection は変換されませんでした。

javax.jms.XATopicConnectionFactory は変換されませんでした。

javax.jms.XATopicSession は変換されませんでした。

Javax.mail のエラー メッセージ

javax.mail.Address.getType は変換されませんでした。

javax.mail.AuthenticationFailedException は変換されませんでした。

javax.mail.Authenticator は変換されませんでした。

javax.mail.BodyPart.addHeader は変換されませんでした。

javax.mail.BodyPart.BodyPart は変換されませんでした。

javax.mail.BodyPart.getAllHeaders は変換されませんでした。

javax.mail.BodyPart.getContent は変換されませんでした。

javax.mail.BodyPart.getContentType は変換されませんでした。

javax.mail.BodyPart.getDataHandler は変換されませんでした。

javax.mail.BodyPart.getDescription は変換されませんでした。

javax.mail.BodyPart.getDisposition は変換されませんでした。

javax.mail.BodyPart.getHeader は変換されませんでした。

javax.mail.BodyPart.getLineCount は変換されませんでした。

javax.mail.BodyPart.getMatchingHeaders は変換されませんでした。

javax.mail.BodyPart.getParent は変換されませんでした。

javax.mail.BodyPart.getSize は変換されませんでした。

javax.mail.BodyPart.isMimeType は変換されませんでした。

javax.mail.BodyPart.parent は変換されませんでした。

javax.mail.BodyPart.removeHeader は変換されませんでした。

javax.mail.BodyPart.setContent(Multipart) は変換されませんでした。

javax.mail.BodyPart.setContent(Object, String) は変換されませんでした。

javax.mail.BodyPart.setDataHandler は変換されませんでした。

javax.mail.BodyPart.setDescription は変換されませんでした。

javax.mail.BodyPart.setDisposition は変換されませんでした。

javax.mail.BodyPart.setFileName は変換されませんでした。

javax.mail.BodyPart.setHeader は変換されませんでした。

javax.mail.BodyPart.writeTo は変換されませんでした。

javax.mail.event.ConnectionAdapter は変換されませんでした。

javax.mail.event.ConnectionEvent は変換されませんでした。

javax.mail.event.ConnectionListener は変換されませんでした。

javax.mail.event.FolderAdapter は変換されませんでした。

javax.mail.event.FolderEvent は変換されませんでした。

javax.mail.event.FolderListener は変換されませんでした。

javax.mail.event.MailEvent は変換されませんでした。

javax.mail.event.MessageChangedEvent は変換されませんでした。

javax.mail.event.MessageChangedListener は変換されませんでした。

javax.mail.event.MessageCountAdapter は変換されませんでした。

javax.mail.event.MessageCountEvent は変換されませんでした。

javax.mail.event.MessageCountListener は変換されませんでした。

javax.mail.event.StoreEvent は変換されませんでした。

javax.mail.event.StoreListener は変換されませんでした。

javax.mail.event.TransportAdapter は変換されませんでした。

javax.mail.event.TransportEvent は変換されませんでした。

javax.mail.event.TransportListener は変換されませんでした。

javax.mail.FetchProfile は変換されませんでした。

javax.mail.FetchProfile.Item は変換されませんでした。

javax.mail.Flags は変換されませんでした。

javax.mail.Flags.Flag は変換されませんでした。

javax.mail.Folder は変換されませんでした。

javax.mail.FolderClosedException は変換されませんでした。

javax.mail.FolderNotFoundException は変換されませんでした。

javax.mail.Header.Header は変換されませんでした。

javax.mail.IllegalWriteException は変換されませんでした。

javax.mail.internet.AddressException は変換されませんでした。

javax.mail.internet.AddressException.getPos は変換されませんでした。

javax.mail.internet.AddressException.getRef は変換されませんでした。

javax.mail.internet.AddressException.pos は変換されませんでした。

javax.mail.internet.AddressException.ref は変換されませんでした。

javax.mail.internet.ContentDisposition は変換されませんでした。

javax.mail.internet.ContentType は変換されませんでした。

javax.mail.internet.HeaderTokenizer は変換されませんでした。

javax.mail.internet.HeaderTokenizer.Token は変換されませんでした。

javax.mail.internet.InternetAddress.clone は変換されませんでした。

javax.mail.internet.InternetAddress.encodedPersonal は変換されませんでした。

javax.mail.internet.InternetHeaders.getHeader は変換されませんでした。

javax.mail.internet.InternetAddress.getLocalAddress は変換されませんでした。

javax.mail.internet.InternetAddress.getPersonal は変換されませんでした。

javax.mail.internet.InternetAddress.InternetAddress は変換されませんでした。

javax.mail.internet.InternetAddress.personal は変換されませんでした。

javax.mail.internet.InternetAddress.setAddress は変換されませんでした。

javax.mail.internet.InternetAddress.setPersonal は変換されませんでした。

javax.mail.internet.InternetHeaders.addHeader は変換されませんでした。

javax.mail.internet.InternetHeaders.getMatchingHeaderLines は変換されませんでした。

javax.mail.internet.InternetHeaders.getMatchingHeaders は変換されませんでした。

javax.mail.internet.InternetHeaders.getNonMatchingHeaderLines は変換されませんでした。

javax.mail.internet.InternetHeaders.getNonMatchingHeaders は変換されませんでした。

javax.mail.internet.InternetHeaders.InternetHeaders は変換されませんでした。

javax.mail.internet.InternetHeaders.load は変換されませんでした。

javax.mail.internet.MailDateFormat は変換されませんでした。

javax.mail.internet.MimeBodyPart.addHeader は変換されませんでした。

javax.mail.internet.MimeBodyPart.addHeaderLine は変換されませんでした。

javax.mail.internet.MimeBodyPart.content は変換されませんでした。

javax.mail.internet.MimeBodyPart.contentStream は変換されませんでした。

javax.mail.internet.MimeBodyPart.dh は変換されませんでした。

javax.mail.internet.MimeBodyPart.getAllHeaderLines は変換されませんでした。

javax.mail.internet.MimeBodyPart.getAllHeaders は変換されませんでした。

javax.mail.internet.MimeBodyPart.getContent は変換されませんでした。

javax.mail.internet.MimeBodyPart.getContentID は変換されませんでした。

javax.mail.internet.MimeBodyPart.getContentLanguage は変換されませんでした。

javax.mail.internet.MimeBodyPart.getContentMD5 は変換されませんでした。

javax.mail.internet.MimeBodyPart.getContentStream は変換されませんでした。

javax.mail.internet.MimeBodyPart.getContentType は変換されませんでした。

javax.mail.internet.MimeBodyPart.getDataHandler は変換されませんでした。

javax.mail.internet.MimeBodyPart.getDescription は変換されませんでした。

javax.mail.internet.MimeBodyPart.getDisposition は変換されませんでした。

javax.mail.internet.MimeBodyPart.getEncoding は変換されませんでした。

javax.mail.internet.MimeBodyPart.getHeader(String) は変換されませんでした。

javax.mail.internet.MimeBodyPart.getHeader(String, String) は変換されませんでした。

javax.mail.internet.MimeBodyPart.getLineCount は変換されませんでした。

javax.mail.internet.MimeBodyPart.getMatchingHeaderLines は変換されませんでした。

javax.mail.internet.MimeBodyPart.getMatchingHeaders は変換されませんでした。

javax.mail.internet.MimeBodyPart.getNonMatchingHeaderLines は変換されませんでした。

javax.mail.internet.MimeBodyPart.getNonMatchingHeaders は変換されませんでした。

javax.mail.internet.MimeBodyPart.getRawInputStream は変換されませんでした。

javax.mail.internet.MimeBodyPart.getSize は変換されませんでした。

javax.mail.internet.MimeBodyPart.headers は変換されませんでした。

javax.mail.internet.MimeBodyPart.isMimeType は変換されませんでした。

javax.mail.internet.MimeBodyPart.MimeBodyPart は変換されませんでした。

javax.mail.internet.MimeBodyPart.MimeBodyPart(InputStream) は変換されませんでした。

javax.mail.internet.MimeBodyPart.MimeBodyPart(InternetHeaders, byte[]) は変換されませんでした。

javax.mail.internet.MimeBodyPart.removeHeader は変換されませんでした。

javax.mail.internet.MimeBodyPart.setContent(Multipart) は変換されませんでした。

javax.mail.internet.MimeBodyPart.setContent(Object, String) は変換されませんでした。

javax.mail.internet.MimeBodyPart.setContentLanguage は変換されませんでした。

javax.mail.internet.MimeBodyPart.setContentMD5 は変換されませんでした。

javax.mail.internet.MimeBodyPart.setDataHandler は変換されませんでした。

javax.mail.internet.MimeBodyPart.setDescription は変換されませんでした。

javax.mail.internet.MimeBodyPart.setDisposition は変換されませんでした。

javax.mail.internet.MimeBodyPart.setHeader は変換されませんでした。

javax.mail.internet.MimeBodyPart.setText は変換されませんでした。

javax.mail.internet.MimeBodyPart.updateHeaders は変換されませんでした。

javax.mail.internet.MimeBodyPart.writeTo は変換されませんでした。

javax.mail.internet.MimeMessage.addHeader は変換されませんでした。

javax.mail.internet.MimeMessage.addHeaderLine は変換されませんでした。

javax.mail.internet.MimeMessage.addRecipients は変換されませんでした。

javax.mail.internet.MimeMessage.content は変換されませんでした。

javax.mail.internet.MimeMessage.contentStream は変換されませんでした。

javax.mail.internet.MimeMessage.createInternetHeaders は変換されませんでした。

javax.mail.internet.MimeMessage.dh は変換されませんでした。

javax.mail.internet.MimeMessage.flags は変換されませんでした。

javax.mail.internet.MimeMessage.getAllHeaderLines は変換されませんでした。

javax.mail.internet.MimeMessage.getAllRecipients は変換されませんでした。

javax.mail.internet.MimeMessage.getContentID は変換されませんでした。

javax.mail.internet.MimeMessage.getContentLanguage は変換されませんでした。

javax.mail.internet.MimeMessage.getContentMD5 は変換されませんでした。

javax.mail.internet.MimeMessage.getContentStream は変換されませんでした。

javax.mail.internet.MimeMessage.getContentType は変換されませんでした。

javax.mail.internet.MimeMessage.getDataHandler は変換されませんでした。

javax.mail.internet.MimeMessage.getDescription は変換されませんでした。

javax.mail.internet.MimeMessage.getDisposition は変換されませんでした。

javax.mail.internet.MimeMessage.getEncoding は変換されませんでした。

javax.mail.internet.MimeMessage.getFileName は変換されませんでした。

javax.mail.internet.MimeMessage.getFlags は変換されませんでした。

javax.mail.internet.MimeMessage.getFrom は変換されませんでした。

javax.mail.internet.MimeMessage.getHeader は変換されませんでした。

javax.mail.internet.MimeMessage.getInputStream は変換されませんでした。

javax.mail.internet.MimeMessage.getLineCount は変換されませんでした。

javax.mail.internet.MimeMessage.getMatchingHeaderLines は変換されませんでした。

javax.mail.internet.MimeMessage.getMatchingHeaders は変換されませんでした。

javax.mail.internet.MimeMessage.getMessageID は変換されませんでした。

javax.mail.internet.MimeMessage.getNonMatchingHeaderLines は変換されませんでした。

javax.mail.internet.MimeMessage.getRawInputStream は変換されませんでした。

javax.mail.internet.MimeMessage.getReceivedDate は変換されませんでした。

javax.mail.internet.MimeMessage.getRecipients は変換されませんでした。

javax.mail.internet.MimeMessage.getReplyTo は変換されませんでした。

javax.mail.internet.MimeMessage.getSentDate は変換されませんでした。

javax.mail.internet.MimeMessage.getSize は変換されませんでした。

javax.mail.internet.MimeMessage.headers は変換されませんでした。

javax.mail.internet.MimeMessage.isMimeType は変換されませんでした。

javax.mail.internet.MimeMessage.isSet は変換されませんでした。

javax.mail.internet.MimeMessage.MimeMessage は変換されませんでした。

javax.mail.internet.MimeMessage.modified は変換されませんでした。

javax.mail.internet.MimeMessage.parse は変換されませんでした。

javax.mail.internet.MimeMessage.RecipientType は変換されませんでした。

javax.mail.internet.MimeMessage.reply は変換されませんでした。

javax.mail.internet.MimeMessage.saveChanges は変換されませんでした。

javax.mail.internet.MimeMessage.saved は変換されませんでした。

javax.mail.internet.MimeMessage.setContent は変換されませんでした。

javax.mail.internet.MimeMessage.setContentID は変換されませんでした。

javax.mail.internet.MimeMessage.setContentLanguage は変換されませんでした。

javax.mail.internet.MimeMessage.setContentMD5 は変換されませんでした。

javax.mail.internet.MimeMessage.setDataHandler は変換されませんでした。

javax.mail.internet.MimeMessage.setDescription は変換されませんでした。

javax.mail.internet.MimeMessage.setDisposition は変換されませんでした。

javax.mail.internet.MimeMessage.setFileName は変換されませんでした。

javax.mail.internet.MimeMessage.setFlags は変換されませんでした。

javax.mail.internet.MimeMessage.setFrom は変換されませんでした。

javax.mail.internet.MimeMessage.setRecipients は変換されませんでした。

javax.mail.internet.MimeMessage.setSentDate は変換されませんでした。

javax.mail.internet.MimeMessage.setSubject は変換されませんでした。

javax.mail.internet.MimeMessage.setText は変換されませんでした。

javax.mail.internet.MimeMessage.updateHeaders は変換されませんでした。

javax.mail.internet.MimeMessage.writeTo は変換されませんでした。

javax.mail.internet.MimeMultipart.createInternetHeaders は変換されませんでした。

javax.mail.internet.MimeMultipart.createMimeBodyPart は変換されませんでした。

javax.mail.internet.MimeMultipart.ds は変換されませんでした。

javax.mail.internet.MimeMultipart.getBodyPart は変換されませんでした。

javax.mail.internet.MimeMultipart.MimeMultipart(DataSource) は変換されませんでした。

javax.mail.internet.MimeMultipart.MimeMultipart(String) は変換されませんでした。

javax.mail.internet.MimeMultipart.parse は変換されませんでした。

javax.mail.internet.MimeMultipart.parsed は変換されませんでした。

javax.mail.internet.MimeMultipart.setSubType は変換されませんでした。

javax.mail.internet.MimeMultipart.updateHeaders は変換されませんでした。

javax.mail.internet.MimeMultipart.writeTo は変換されませんでした。

javax.mail.internet.MimePart.getContentID は変換されませんでした。

javax.mail.internet.MimePart.getContentLanguage は変換されませんでした。

javax.mail.internet.MimePart.getContentMD5 は変換されませんでした。

javax.mail.internet.MimePart.getMatchingHeaderLines は変換されませんでした。

javax.mail.internet.MimePart.getNonMatchingHeaderLines は変換されませんでした。

javax.mail.internet.MimePart.setContentLanguage は変換されませんでした。

javax.mail.internet.MimePart.setContentMD5 は変換されませんでした。

javax.mail.internet.MimePart.setText は変換されませんでした。

javax.mail.internet.MimePartDataSource は変換されませんでした。

javax.mail.internet.MimeUtility は変換されませんでした。

javax.mail.internet.NewsAddress は変換されませんでした。

javax.mail.internet.ParameterList は変換されませんでした。

javax.mail.internet.ParseException は変換されませんでした。

javax.mail.internet.SharedInputStream は変換されませんでした。

javax.mail.Message.addHeader は変換されませんでした。

javax.mail.Message.expunged は変換されませんでした。

javax.mail.Message.folder は変換されませんでした。

javax.mail.Message.getAllRecipients は変換されませんでした。

javax.mail.Message.getContentType は変換されませんでした。

javax.mail.Message.getDataHandler は変換されませんでした。

javax.mail.Message.getDescription は変換されませんでした。

javax.mail.Message.getDisposition は変換されませんでした。

javax.mail.Message.getFileName は変換されませんでした。

javax.mail.Message.getFlags は変換されませんでした。

javax.mail.Message.getFolder は変換されませんでした。

javax.mail.Message.getFrom は変換されませんでした。

javax.mail.Message.getInputStream は変換されませんでした。

javax.mail.Message.getLineCount は変換されませんでした。

javax.mail.Message.getMatchingHeaders は変換されませんでした。

javax.mail.Message.getMessageNumber は変換されませんでした。

javax.mail.Message.getNonMatchingHeaders は変換されませんでした。

javax.mail.Message.getReceivedDate は変換されませんでした。

javax.mail.Message.getRecipients は変換されませんでした。

javax.mail.Message.getReplyTo は変換されませんでした。

javax.mail.Message.getSentDate は変換されませんでした。

javax.mail.Message.getSize は変換されませんでした。

javax.mail.Message.isExpunged は変換されませんでした。

javax.mail.Message.isMimeType は変換されませんでした。

javax.mail.Message.isSet は変換されませんでした。

javax.mail.Message.match は変換されませんでした。

javax.mail.Message.Message は変換されませんでした。

javax.mail.Message.msgnum は変換されませんでした。

javax.mail.Message.RecipientType.readResolve は変換されませんでした。

javax.mail.Message.RecipientType.type は変換されませんでした。

javax.mail.Message.reply は変換されませんでした。

javax.mail.Message.saveChanges は変換されませんでした。

javax.mail.Message.session は変換されませんでした。

javax.mail.Message.setContent は変換されませんでした。

javax.mail.Message.setDataHandler は変換されませんでした。

javax.mail.Message.setDescription は変換されませんでした。

javax.mail.Message.setDisposition は変換されませんでした。

javax.mail.Message.setExpunged は変換されませんでした。

javax.mail.Message.setFileName は変換されませんでした。

javax.mail.Message.setFlag は変換されませんでした。

javax.mail.Message.setFlags は変換されませんでした。

javax.mail.Message.setFrom は変換されませんでした。

javax.mail.Message.setMessageNumber は変換されませんでした。

javax.mail.Message.setSentDate は変換されませんでした。

javax.mail.Message.writeTo は変換されませんでした。

javax.mail.MessageAware は変換されませんでした。

javax.mail.MessageContext は変換されませんでした。

javax.mail.MessageRemovedException は変換されませんでした。

javax.mail.MessagingException.setNextException は変換されませんでした。

javax.mail.MethodNotSupportedException は変換されませんでした。

javax.mail.Multipart.contentType は変換されませんでした。

javax.mail.Multipart.getContentType は変換されませんでした。

javax.mail.Multipart.getParent は変換されませんでした。

javax.mail.Multipart.parent は変換されませんでした。

javax.mail.Multipart.setMultipartDataSource は変換されませんでした。

javax.mail.Multipart.setParent は変換されませんでした。

javax.mail.Multipart.writeTo は変換されませんでした。

javax.mail.MultipartDataSource は変換されませんでした。

javax.mail.NoSuchProviderException は変換されませんでした。

javax.mail.Part.addHeader は変換されませんでした。

javax.mail.Part.ATTACHMENT は変換されませんでした。

javax.mail.Part.getContentType は変換されませんでした。

javax.mail.Part.getDataHandler は変換されませんでした。

javax.mail.Part.getDescription は変換されませんでした。

javax.mail.Part.getDisposition は変換されませんでした。

javax.mail.Part.getFileName は変換されませんでした。

javax.mail.Part.getInputStream は変換されませんでした。

javax.mail.Part.getLineCount は変換されませんでした。

javax.mail.Part.getMatchingHeaders は変換されませんでした。

javax.mail.Part.getNonMatchingHeaders は変換されませんでした。

javax.mail.Part.getSize は変換されませんでした。

javax.mail.Part.INLINE は変換されませんでした。

javax.mail.Part.isMimeType は変換されませんでした。

javax.mail.Part.setContent は変換されませんでした。

javax.mail.Part.setDataHandler は変換されませんでした。

javax.mail.Part.setDescription は変換されませんでした。

javax.mail.Part.setDisposition は変換されませんでした。

javax.mail.Part.setFileName は変換されませんでした。

javax.mail.Part.writeTo は変換されませんでした。

javax.mail.PasswordAuthentication は変換されませんでした。

javax.mail.Provider は変換されませんでした。

javax.mail.Provider.Type は変換されませんでした。

javax.mail.ReadOnlyFolderException は変換されませんでした。

javax.mail.search.AddressStringTerm は変換されませんでした。

javax.mail.search.AddressTerm は変換されませんでした。

javax.mail.search.AndTerm は変換されませんでした。

javax.mail.search.BodyTerm は変換されませんでした。

javax.mail.search.ComparisonTerm は変換されませんでした。

javax.mail.search.DateTerm は変換されませんでした。

javax.mail.search.FlagTerm は変換されませんでした。

javax.mail.search.FromStringTerm は変換されませんでした。

javax.mail.search.FromTerm は変換されませんでした。

javax.mail.search.HeaderTerm は変換されませんでした。

javax.mail.search.IntegerComparisonTerm は変換されませんでした。

javax.mail.search.MessageIDTerm は変換されませんでした。

javax.mail.search.MessageNumberTerm は変換されませんでした。

javax.mail.search.NotTerm は変換されませんでした。

javax.mail.search.OrTerm は変換されませんでした。

javax.mail.search.ReceivedDateTerm は変換されませんでした。

javax.mail.search.RecipientStringTerm は変換されませんでした。

javax.mail.search.RecipientTerm は変換されませんでした。

javax.mail.search.SearchException は変換されませんでした。

javax.mail.search.SearchTerm は変換されませんでした。

javax.mail.search.SentDateTerm は変換されませんでした。

javax.mail.search.SizeTerm は変換されませんでした。

javax.mail.search.StringTerm は変換されませんでした。

javax.mail.search.SubjectTerm は変換されませんでした。

javax.mail.SendFailedException.getInvalidAddresses は変換されませんでした。

javax.mail.SendFailedException.getValidSentAddresses は変換されませんでした。

javax.mail.SendFailedException.getValidUnsentAddresses は変換されませんでした。

javax.mail.SendFailedException.invalid は変換されませんでした。

javax.mail.SendFailedException.validSent は変換されませんでした。

javax.mail.SendFailedException.validUnsent は変換されませんでした。

javax.mail.Service は変換されませんでした。

javax.mail.Session は変換されませんでした。

javax.mail.Store は変換されませんでした。

javax.mail.StoreClosedException は変換されませんでした。

javax.mail.Transport.addTransportListener は変換されませんでした。

javax.mail.Transport.notifyTransportListeners は変換されませんでした。

javax.mail.Transport.removeTransportListener は変換されませんでした。

javax.mail.Transport.send は変換されませんでした。

javax.mail.Transport.Transport は変換されませんでした。

javax.mail.UIDFolder は変換されませんでした。

javax.mail.UIDFolder.FetchProfileItem は変換されませんでした。

javax.mail.URLName.fullURL は変換されませんでした。

javax.mail.URLName.getFile は変換されませんでした。

javax.mail.URLName.getRef は変換されませんでした。

javax.mail.URLName.parseString は変換されませんでした。

javax.mail.URLName.URLName は変換されませんでした。

Javax.naming のエラー メッセージ

javax.naming.BinaryRefAddr は変換されませんでした。

javax.naming.BinaryRefAddr.BinaryRefAddr(String, byte[]) は変換されませんでした。

javax.naming.BinaryRefAddr.BinaryRefAddr(String, byte[], int, int) は変換されませんでした。

javax.naming.Binding は変換されませんでした。

javax.naming.Binding.Binding(String, Object) は変換されませんでした。

javax.naming.Binding.Binding(String, Object, boolean) は変換されませんでした。

javax.naming.Binding.Binding(String, String, Object) は変換されませんでした。

javax.naming.Binding.Binding(String, String, Object, boolean) は変換されませんでした。

javax.naming.Binding.setObject は変換されませんでした。

javax.naming.CannotProceedException.altName は変換されませんでした。

javax.naming.CannotProceedException.altNameCtx は変換されませんでした。

javax.naming.CannotProceedException.environment は変換されませんでした。

javax.naming.CannotProceedException.getAltName は変換されませんでした。

javax.naming.CannotProceedException.getAltNameCtx は変換されませんでした。

javax.naming.CannotProceedException.getEnvironment は変換されませんでした。

javax.naming.CannotProceedException.getRemainingNewName は変換されませんでした。

javax.naming.CannotProceedException.remainingNewName は変換されませんでした。

javax.naming.CannotProceedException.setAltName は変換されませんでした。

javax.naming.CannotProceedException.setAltNameCtx は変換されませんでした。

javax.naming.CannotProceedException.setEnvironment は変換されませんでした。

javax.naming.CannotProceedException.setRemainingNewName は変換されませんでした。

javax.naming.CompositeName は変換されませんでした。

javax.naming.CompositeName.CompositeName は変換されませんでした。

javax.naming.CompoundName は変換されませんでした。

javax.naming.CompoundName.CompoundName(Enumeration, Properties) は変換されませんでした。

javax.naming.CompoundName.CompoundName(String, Properties) は変換されませんでした。

javax.naming.CompoundName.impl は変換されませんでした。

javax.naming.CompoundName.mySyntax は変換されませんでした。

javax.naming.Context.<EnvironmentProperty> は変換されませんでした。

javax.naming.Context.addToEnvironment は変換されませんでした。

javax.naming.Context.bind(Name, Object) は変換されませんでした。

javax.naming.Context.bind(String, Object) は変換されませんでした。

javax.naming.Context.composeName は変換されませんでした。

javax.naming.Context.createSubcontext は変換されませんでした。

javax.naming.Context.destroySubcontext は変換されませんでした。

javax.naming.Context.getEnvironment は変換されませんでした。

javax.naming.Context.getNameInNamespace は変換されませんでした。

javax.naming.Context.getNameParser は変換されませんでした。

javax.naming.Context.list は変換されませんでした。

javax.naming.Context.listBindings は変換されませんでした。

javax.naming.Context.lookup は変換されませんでした。

javax.naming.Context.lookupLink は変換されませんでした。

javax.naming.Context.rebind は変換されませんでした。

javax.naming.Context.removeFromEnvironment は変換されませんでした。

javax.naming.Context.rename は変換されませんでした。

javax.naming.Context.unbind は変換されませんでした。

javax.naming.directory.Attribute は変換されませんでした。

javax.naming.directory.Attribute.clone は変換されませんでした。

javax.naming.directory.Attribute.get は変換されませんでした。

javax.naming.directory.Attribute.getAttributeDefinition は変換されませんでした。

javax.naming.directory.Attribute.getAttributeSyntaxDefinition は変換されませんでした。

javax.naming.directory.Attribute.getID は変換されませんでした。

javax.naming.directory.Attribute.isOrdered は変換されませんでした。

javax.naming.directory.Attribute.serialVersionUID は変換されませんでした。

javax.naming.directory.AttributeModificationException.getUnexecutedModifications は変換されませんでした。

javax.naming.directory.AttributeModificationException.setUnexecutedModifications は変換されませんでした。

javax.naming.directory.Attributes は変換されませんでした。

javax.naming.directory.Attributes.clone は変換されませんでした。

javax.naming.directory.Attributes.isCaseIgnored は変換されませんでした。

javax.naming.directory.Attributes.put は変換されませんでした。

javax.naming.directory.Attributes.remove は変換されませんでした。

javax.naming.directory.BasicAttribute は変換されませんでした。

javax.naming.directory.BasicAttribute.attrID は変換されませんでした。

javax.naming.directory.BasicAttribute.BasicAttribute(String) は変換されませんでした。

javax.naming.directory.BasicAttribute.BasicAttribute(String, boolean) は変換されませんでした。

javax.naming.directory.BasicAttribute.BasicAttribute(String, Object) は変換されませんでした。

javax.naming.directory.BasicAttribute.BasicAttribute(String, Object, boolean) は変換されませんでした。

javax.naming.directory.BasicAttribute.clone は変換されませんでした。

javax.naming.directory.BasicAttribute.get は変換されませんでした。

javax.naming.directory.BasicAttribute.getAttributeDefinition は変換されませんでした。

javax.naming.directory.BasicAttribute.getAttributeSyntaxDefinition は変換されませんでした。

javax.naming.directory.BasicAttribute.getID は変換されませんでした。

javax.naming.directory.BasicAttribute.isOrdered は変換されませんでした。

javax.naming.directory.BasicAttribute.ordered は変換されませんでした。

javax.naming.directory.BasicAttribute.values は変換されませんでした。

javax.naming.directory.BasicAttributes は変換されませんでした。

javax.naming.directory.BasicAttributes.BasicAttributes は変換されませんでした。

javax.naming.directory.BasicAttributes.clone は変換されませんでした。

javax.naming.directory.BasicAttributes.isCaseIgnored は変換されませんでした。

javax.naming.directory.BasicAttributes.put は変換されませんでした。

javax.naming.directory.BasicAttributes.remove は変換されませんでした。

javax.naming.directory.DirContext は変換されませんでした。

javax.naming.directory.DirContext.<OperationType> は変換されませんでした。

javax.naming.directory.DirContext.bind(Name, Object, Attributes) は変換されませんでした。

javax.naming.directory.DirContext.bind(String, Object, Attributes) は変換されませんでした。

javax.naming.directory.DirContext.createSubcontext は変換されませんでした。

javax.naming.directory.DirContext.getAttributes は変換されませんでした。

javax.naming.directory.DirContext.getSchema は変換されませんでした。

javax.naming.directory.DirContext.getSchemaClassDefinition は変換されませんでした。

javax.naming.directory.DirContext.modifyAttributes は変換されませんでした。

javax.naming.directory.DirContext.rebind は変換されませんでした。

javax.naming.directory.DirContext.search は変換されませんでした。

javax.naming.directory.InitialDirContext.bind は変換されませんでした。

javax.naming.directory.InitialDirContext.createSubcontext は変換されませんでした。

javax.naming.directory.InitialDirContext.getAttributes は変換されませんでした。

javax.naming.directory.InitialDirContext.getSchema は変換されませんでした。

javax.naming.directory.InitialDirContext.getSchemaClassDefinition は変換されませんでした。

javax.naming.directory.InitialDirContext.InitialDirContext(boolean) は変換されませんでした。

javax.naming.directory.InitialDirContext.InitialDirContext(Hashtable) は変換されませんでした。

javax.naming.directory.InitialDirContext.modifyAttributes は変換されませんでした。

javax.naming.directory.InitialDirContext.rebind は変換されませんでした。

javax.naming.directory.InitialDirContext.search は変換されませんでした。

javax.naming.directory.SearchControls.getDerefLinkFlag は変換されませんでした。

javax.naming.directory.SearchControls.getReturningObjFlag は変換されませんでした。

javax.naming.directory.SearchControls.SearchControls は変換されませんでした。

javax.naming.directory.SearchControls.setDerefLinkFlag は変換されませんでした。

javax.naming.directory.SearchControls.setReturningObjFlag は変換されませんでした。

javax.naming.directory.SearchResult.getAttributes は変換されませんでした。

javax.naming.directory.SearchResult.SearchResult(String, Object, Attributes) は変換されませんでした。

javax.naming.directory.SearchResult.SearchResult(String, Object, Attributes, boolean) は変換されませんでした。

javax.naming.directory.SearchResult.SearchResult(String, String, Object, Attributes) は変換されませんでした。

javax.naming.directory.SearchResult.SearchResult(String, String, Object, Attributes, boolean) は変換されませんでした。

javax.naming.directory.SearchResult.setAttributes は変換されませんでした。

javax.naming.event.EventContext は変換されませんでした。

javax.naming.event.EventDirContext は変換されませんでした。

javax.naming.event.NamespaceChangeListener は変換されませんでした。

javax.naming.event.NamingEvent は変換されませんでした。

javax.naming.event.NamingExceptionEvent は変換されませんでした。

javax.naming.event.NamingListener は変換されませんでした。

javax.naming.event.ObjectChangeListener は変換されませんでした。

javax.naming.InitialContext.addToEnvironment は変換されませんでした。

javax.naming.InitialContext.bind は変換されませんでした。

javax.naming.InitialContext.composeName は変換されませんでした。

javax.naming.InitialContext.createSubcontext は変換されませんでした。

javax.naming.InitialContext.destroySubcontext は変換されませんでした。

javax.naming.InitialContext.getDefaultInitCtx は変換されませんでした。

javax.naming.InitialContext.getEnvironment は変換されませんでした。

javax.naming.InitialContext.getNameInNamespace は変換されませんでした。

javax.naming.InitialContext.getNameParser は変換されませんでした。

javax.naming.InitialContext.getURLOrDefaultInitCtx は変換されませんでした。

javax.naming.InitialContext.gotDefault は変換されませんでした。

javax.naming.InitialContext.init は変換されませんでした。

javax.naming.InitialContext.InitialContext は変換されませんでした。

javax.naming.InitialContext.InitialContext(boolean) は変換されませんでした。

javax.naming.InitialContext.InitialContext(Hashtable) は変換されませんでした。

javax.naming.InitialContext.list は変換されませんでした。

javax.naming.InitialContext.listBindings は変換されませんでした。

javax.naming.InitialContext.lookup は変換されませんでした。

javax.naming.InitialContext.lookupLink は変換されませんでした。

javax.naming.InitialContext.myProps は変換されませんでした。

javax.naming.InitialContext.rebind は変換されませんでした。

javax.naming.InitialContext.removeFromEnvironment は変換されませんでした。

javax.naming.InitialContext.rename は変換されませんでした。

javax.naming.InitialContext.unbind は変換されませんでした。

javax.naming.ldap.Control は変換されませんでした。

javax.naming.ldap.ControlFactory は変換されませんでした。

javax.naming.ldap.ExtendedRequest は変換されませんでした。

javax.naming.ldap.ExtendedResponse は変換されませんでした。

javax.naming.ldap.HasControls は変換されませんでした。

javax.naming.ldap.InitialLdapContext は変換されませんでした。

javax.naming.ldap.InitialLdapContext.extendedOperation は変換されませんでした。

javax.naming.ldap.InitialLdapContext.getConnectControls は変換されませんでした。

javax.naming.ldap.InitialLdapContext.getRequestControls は変換されませんでした。

javax.naming.ldap.InitialLdapContext.getResponseControls は変換されませんでした。

javax.naming.ldap.InitialLdapContext.InitialLdapContext は変換されませんでした。

javax.naming.ldap.InitialLdapContext.newInstance は変換されませんでした。

javax.naming.ldap.InitialLdapContext.reconnect は変換されませんでした。

javax.naming.ldap.InitialLdapContext.setRequestControls は変換されませんでした。

javax.naming.ldap.LdapContext は変換されませんでした。

javax.naming.ldap.LdapContext.CONTROL_FACTORIES は変換されませんでした。

javax.naming.ldap.LdapContext.extendedOperation は変換されませんでした。

javax.naming.ldap.LdapContext.getConnectControls は変換されませんでした。

javax.naming.ldap.LdapContext.getRequestControls は変換されませんでした。

javax.naming.ldap.LdapContext.getResponseControls は変換されませんでした。

javax.naming.ldap.LdapContext.newInstance は変換されませんでした。

javax.naming.ldap.LdapContext.reconnect は変換されませんでした。

javax.naming.ldap.LdapContext.setRequestControls は変換されませんでした。

javax.naming.ldap.LdapReferralException は変換されませんでした。

javax.naming.ldap.UnsolicitedNotification は変換されませんでした。

javax.naming.ldap.UnsolicitedNotificationEvent は変換されませんでした。

javax.naming.ldap.UnsolicitedNotificationListener は変換されませんでした。

javax.naming.LinkException.getLinkRemainingName は変換されませんでした。

javax.naming.LinkException.getLinkResolvedName は変換されませんでした。

javax.naming.LinkException.getLinkResolvedObj は変換されませんでした。

javax.naming.LinkException.linkRemainingName は変換されませんでした。

javax.naming.LinkException.linkResolvedName は変換されませんでした。

javax.naming.LinkException.linkResolvedObj は変換されませんでした。

javax.naming.LinkException.setLinkRemainingName は変換されませんでした。

javax.naming.LinkException.setLinkResolvedName は変換されませんでした。

javax.naming.LinkException.setLinkResolvedObj は変換されませんでした。

javax.naming.LinkException.toString は変換されませんでした。

javax.naming.LinkRef は変換されませんでした。

javax.naming.Name は変換されませんでした。

javax.naming.NameClassPair.isRelative は変換されませんでした。

javax.naming.NameClassPair.NameClassPair(String, String) は変換されませんでした。

javax.naming.NameClassPair.NameClassPair(String, String, boolean) は変換されませんでした。

javax.naming.NameClassPair.setClassName は変換されませんでした。

javax.naming.NameClassPair.setRelative は変換されませんでした。

javax.naming.NameParser は変換されませんでした。

javax.naming.NamingEnumeration.close は変換されませんでした。

javax.naming.NamingEnumeration.hasMore は変換されませんでした。

javax.naming.NamingEnumeration.next は変換されませんでした。

javax.naming.NamingException.appendRemainingComponent は変換されませんでした。

javax.naming.NamingException.appendRemainingName は変換されませんでした。

javax.naming.NamingException.getRemainingName は変換されませんでした。

javax.naming.NamingException.getResolvedName は変換されませんでした。

javax.naming.NamingException.getResolvedObj は変換されませんでした。

javax.naming.NamingException.printStackTrace は変換されませんでした。

javax.naming.NamingException.remainingName は変換されませんでした。

javax.naming.NamingException.resolvedName は変換されませんでした。

javax.naming.NamingException.resolvedObj は変換されませんでした。

javax.naming.NamingException.rootException は変換されませんでした。

javax.naming.NamingException.setRemainingName は変換されませんでした。

javax.naming.NamingException.setResolvedName は変換されませんでした。

javax.naming.NamingException.setResolvedObj は変換されませんでした。

javax.naming.NamingException.setRootCause は変換されませんでした。

javax.naming.NamingException.toString は変換されませんでした。

javax.naming.RefAddr は変換されませんでした。

javax.naming.RefAddr.addrType は変換されませんでした。

javax.naming.RefAddr.RefAddr は変換されませんでした。

javax.naming.Reference.addrs は変換されませんでした。

javax.naming.Reference.classFactory は変換されませんでした。

javax.naming.Reference.classFactoryLocation は変換されませんでした。

javax.naming.Reference.className は変換されませんでした。

javax.naming.Reference.get は変換されませんでした。

javax.naming.Reference.getClassName は変換されませんでした。

javax.naming.Reference.getFactoryClassLocation は変換されませんでした。

javax.naming.Reference.getFactoryClassName は変換されませんでした。

javax.naming.Reference.Reference は変換されませんでした。

javax.naming.Referenceable は変換されませんでした。

javax.naming.ReferralException.getReferralContext は変換されませんでした。

javax.naming.ReferralException.getReferralInfo は変換されませんでした。

javax.naming.ReferralException.retryReferral は変換されませんでした。

javax.naming.ReferralException.skipReferral は変換されませんでした。

javax.naming.spi.DirectoryManager は変換されませんでした。

javax.naming.spi.DirObjectFactory は変換されませんでした。

javax.naming.spi.DirStateFactory は変換されませんでした。

javax.naming.spi.DirStateFactory.Result は変換されませんでした。

javax.naming.spi.InitialContextFactory は変換されませんでした。

javax.naming.spi.InitialContextFactoryBuilder は変換されませんでした。

javax.naming.spi.NamingManager は変換されませんでした。

javax.naming.spi.ObjectFactory は変換されませんでした。

javax.naming.spi.ObjectFactoryBuilder は変換されませんでした。

javax.naming.spi.Resolver は変換されませんでした。

javax.naming.spi.ResolveResult は変換されませんでした。

javax.naming.spi.StateFactory は変換されませんでした。

javax.naming.StringRefAddr は変換されませんでした。

javax.naming.StringRefAddr.StringRefAddr は変換されませんでした。

Javax.rmi のエラー メッセージ

javax.rmi.CORBA.ClassDesc は変換されませんでした。

javax.rmi.CORBA.PortableRemoteObjectDelegate は変換されませんでした。

javax.rmi.CORBA.Stub は変換されませんでした。

javax.rmi.CORBA.StubDelegate は変換されませんでした。

javax.rmi.CORBA.Tie は変換されませんでした。

javax.rmi.CORBA.Util は変換されませんでした。

javax.rmi.CORBA.UtilDelegate は変換されませんでした。

javax.rmi.CORBA.ValueHandler は変換されませんでした。

javax.rmi.PortableRemoteObject は変換されませんでした。

javax.rmi.PortableRemoteObject.narrow は変換されませんでした。

Javax.security のエラー メッセージ

javax.security.auth.AuthPermission は変換されませんでした。

javax.security.auth.callback.ChoiceCallback.ChoiceCallback は変換されませんでした。

javax.security.auth.callback.ChoiceCallback.getPrompt は変換されませんでした。

javax.security.auth.callback.LanguageCallback.LanguageCallback は変換されませんでした。

javax.security.auth.callback.NameCallback.getDefaultName は変換されませんでした。

javax.security.auth.callback.NameCallback.getPrompt は変換されませんでした。

javax.security.auth.callback.NameCallback.NameCallback(String) は変換されませんでした。

javax.security.auth.callback.NameCallback.NameCallback(String, String) は変換されませんでした。

javax.security.auth.callback.PasswordCallback.getPrompt は変換されませんでした。

javax.security.auth.callback.PasswordCallback.isEchoOn は変換されませんでした。

javax.security.auth.callback.PasswordCallback.PasswordCallback は変換されませんでした。

javax.security.auth.callback.UnsupportedCallbackException は変換されませんでした。

javax.security.auth.callback.UnsupportedCallbackException.getCallback は変換されませんでした。

javax.security.auth.callback.UnsupportedCallbackException.UnsupportedCallbackException は変換されませんでした。

javax.security.auth.Destroyable.isDestroyed は変換されませんでした。

javax.security.auth.DestroyFailedException は変換されませんでした。

javax.security.auth.DestroyFailedException.DestroyFailedException は変換されませんでした。

javax.security.auth.login.AccountExpiredException は変換されませんでした。

javax.security.auth.login.AccountExpiredException.AccountExpiredException は変換されませんでした。

javax.security.auth.login.AppConfigurationEntry は変換されませんでした。

javax.security.auth.login.AppConfigurationEntry.AppConfigurationEntry は変換されませんでした。

javax.security.auth.login.AppConfigurationEntry.getControlFlag は変換されませんでした。

javax.security.auth.login.AppConfigurationEntry.getLoginModuleName は変換されませんでした。

javax.security.auth.login.Configuration は変換されませんでした。

javax.security.auth.login.Configuration.Configuration は変換されませんでした。

javax.security.auth.login.Configuration.refresh は変換されませんでした。

javax.security.auth.login.Configuration.setConfiguration は変換されませんでした。

javax.security.auth.login.CredentialExpiredException は変換されませんでした。

javax.security.auth.login.CredentialExpiredException.CredentialExpiredException は変換されませんでした。

javax.security.auth.login.FailedLoginException は変換されませんでした。

javax.security.auth.login.FailedLoginException.FailedLoginException は変換されませんでした。

javax.security.auth.login.LoginContext.getSubject は変換されませんでした。

javax.security.auth.login.LoginContext.login は変換されませんでした。

javax.security.auth.login.LoginContext.LoginContext は変換されませんでした。

javax.security.auth.login.LoginContext.logout は変換されませんでした。

javax.security.auth.login.LoginException は変換されませんでした。

javax.security.auth.login.LoginException.LoginException は変換されませんでした。

javax.security.auth.Policy は変換されませんでした。

javax.security.auth.Policy.getPolicy は変換されませんでした。

javax.security.auth.Policy.Policy は変換されませんでした。

javax.security.auth.Policy.refresh は変換されませんでした。

javax.security.auth.Policy.setPolicy は変換されませんでした。

javax.security.auth.PrivateCredentialPermission は変換されませんでした。

javax.security.auth.PrivateCredentialPermission.getActions は変換されませんでした。

javax.security.auth.PrivateCredentialPermission.getCredentialClass は変換されませんでした。

javax.security.auth.PrivateCredentialPermission.getPrincipals は変換されませんでした。

javax.security.auth.PrivateCredentialPermission.implies は変換されませんでした。

javax.security.auth.PrivateCredentialPermission.newPermissionCollection は変換されませんでした。

javax.security.auth.PrivateCredentialPermission.PrivateCredentialPermission は変換されませんでした。

javax.security.auth.Refreshable は変換されませんでした。

javax.security.auth.RefreshFailedException は変換されませんでした。

javax.security.auth.RefreshFailedException.RefreshFailedException は変換されませんでした。

javax.security.auth.spi.LoginModule.abort は変換されませんでした。

javax.security.auth.spi.LoginModule.commit は変換されませんでした。

javax.security.auth.spi.LoginModule.initialize は変換されませんでした。

javax.security.auth.spi.LoginModule.login は変換されませんでした。

javax.security.auth.spi.LoginModule.logout は変換されませんでした。

javax.security.auth.Subject.doAs は変換されませんでした。

javax.security.auth.Subject.doAsPrivileged は変換されませんでした。

javax.security.auth.Subject.getPrincipals は変換されませんでした。

javax.security.auth.Subject.getPrivateCredentials は変換されませんでした。

javax.security.auth.Subject.getPublicCredentials は変換されませんでした。

javax.security.auth.Subject.getSubject は変換されませんでした。

javax.security.auth.Subject.isReadOnly は変換されませんでした。

javax.security.auth.Subject.setReadOnly は変換されませんでした。

javax.security.auth.Subject.Subject は変換されませんでした。

javax.security.auth.SubjectDomainCombiner は変換されませんでした。

javax.security.cert.Certificate.Certificate は変換されませんでした。

javax.security.cert.Certificate.getEncoded は変換されませんでした。

javax.security.cert.Certificate.verify(PublicKey) は変換されませんでした。

javax.security.cert.Certificate.verify(PublicKey, String) は変換されませんでした。

javax.security.cert.X509Certificate.getInstance は変換されませんでした。

javax.security.cert.X509Certificate.getNotAfter は変換されませんでした。

javax.security.cert.X509Certificate.getNotBefore は変換されませんでした。

javax.security.cert.X509Certificate.getSigAlgName は変換されませんでした。

javax.security.cert.X509Certificate.getVersion は変換されませんでした。

javax.security.cert.X509Certificate.X509Certificate は変換されませんでした。

Javax.servlet のエラー メッセージ

javax.servlet.FilterConfig は変換されませんでした。

javax.servlet.GenericServlet.destroy は変換されませんでした。

javax.servlet.GenericServlet.getInitParameter は変換されませんでした。

javax.servlet.GenericServlet.getInitParameterNames は変換されませんでした。

javax.servlet.GenericServlet.getServletConfig は変換されませんでした。

javax.servlet.GenericServlet.getServletContext は変換されませんでした。

javax.servlet.GenericServlet.getServletInfo は変換されませんでした。

javax.servlet.GenericServlet.getServletName は変換されませんでした。

javax.servlet.GenericServlet.init は変換されませんでした。

javax.servlet.GenericServlet.log(String) は変換されませんでした。

javax.servlet.GenericServlet.log(String, Throwable) は変換されませんでした。

javax.servlet.GenericServlet.service は変換されませんでした。

javax.servlet.HttpDummyBase.Config は変換されませんでした。

javax.servlet.HttpDummyBase.HttpDummyBase は変換されませんでした。

javax.servlet.RequestDispatcher は変換されませんでした。

javax.servlet.RequestDispatcher.forward は変換されませんでした。

javax.servlet.RequestDispatcher.include は変換されませんでした。

javax.servlet.Servlet.destroy は変換されませんでした。

javax.servlet.Servlet.getServletConfig は変換されませんでした。

javax.servlet.Servlet.getServletInfo は変換されませんでした。

javax.servlet.Servlet.init は変換されませんでした。

javax.servlet.Servlet.service は変換されませんでした。

javax.servlet.ServletConfig は変換されませんでした。

javax.servlet.ServletContext.getContext は変換されませんでした。

javax.servlet.ServletContext.getInitParameter は変換されませんでした。

javax.servlet.ServletContext.getInitParameterNames は変換されませんでした。

javax.servlet.ServletContext.getMajorVersion は変換されませんでした。

javax.servlet.ServletContext.getMimeType は変換されませんでした。

javax.servlet.ServletContext.getMinorVersion は変換されませんでした。

javax.servlet.ServletContext.getNamedDispatcher は変換されませんでした。

javax.servlet.ServletContext.getRequestDispatcher は変換されませんでした。

javax.servlet.ServletContext.getResource は変換されませんでした。

javax.servlet.ServletContext.getResourceAsStream は変換されませんでした。

javax.servlet.ServletContext.getResourcePaths は変換されませんでした。

javax.servlet.ServletContext.getServerInfo は変換されませんでした。

javax.servlet.ServletContext.getServlet は変換されませんでした。

javax.servlet.ServletContext.getServletContextName は変換されませんでした。

javax.servlet.ServletContext.getServletNames は変換されませんでした。

javax.servlet.ServletContext.getServlets は変換されませんでした。

javax.servlet.ServletContext.log は変換されませんでした。

javax.servlet.ServletContext.setAttribute は変換されませんでした。

javax.servlet.ServletContextAttributeEvent は変換されませんでした。

javax.servlet.ServletContextAttributeListener は変換されませんでした。

javax.servlet.ServletContextEvent は変換されませんでした。

javax.servlet.ServletContextListener は変換されませんでした。

javax.servlet.ServletInputStream は変換されませんでした。

javax.servlet.ServletInputStream.readLine は変換されませんでした。

javax.servlet.ServletRequest.getAttribute は変換されませんでした。

javax.servlet.ServletRequest.getAttributeNames は変換されませんでした。

javax.servlet.ServletRequest.getLocale は変換されませんでした。

javax.servlet.ServletRequest.getLocales は変換されませんでした。

javax.servlet.ServletRequest.getParameter は変換されませんでした。

javax.servlet.ServletRequest.getParameterMap は変換されませんでした。

javax.servlet.ServletRequest.getParameterNames は変換されませんでした。

javax.servlet.ServletRequest.getParameterValues は変換されませんでした。

javax.servlet.ServletRequest.getProtocol は変換されませんでした。

javax.servlet.ServletRequest.getReader は変換されませんでした。

javax.servlet.ServletRequest.getRealPath は変換されませんでした。

javax.servlet.ServletRequest.getRequestDispatcher は変換されませんでした。

javax.servlet.ServletRequest.getScheme は変換されませんでした。

javax.servlet.ServletRequest.getServerPort は変換されませんでした。

javax.servlet.ServletRequest.removeAttribute は変換されませんでした。

javax.servlet.ServletRequest.setAttribute は変換されませんでした。

javax.servlet.ServletRequest.setCharacterEncoding は変換されませんでした。

javax.servlet.ServletRequestWrapper.getAttribute は変換されませんでした。

javax.servlet.ServletRequestWrapper.getAttributeNames は変換されませんでした。

javax.servlet.ServletRequestWrapper.getLocale は変換されませんでした。

javax.servlet.ServletRequestWrapper.getLocales は変換されませんでした。

javax.servlet.ServletRequestWrapper.getParameter は変換されませんでした。

javax.servlet.ServletRequestWrapper.getParameterMap は変換されませんでした。

javax.servlet.ServletRequestWrapper.getReader は変換されませんでした。

javax.servlet.ServletRequestWrapper.getRequestDispatcher は変換されませんでした。

javax.servlet.ServletRequestWrapper.getScheme は変換されませんでした。

javax.servlet.ServletResponse は変換されませんでした。

javax.servlet.ServletResponse.flushBuffer は変換されませんでした。

javax.servlet.ServletResponse.getBufferSize は変換されませんでした。

javax.servlet.ServletResponse.getLocale は変換されませんでした。

javax.servlet.ServletResponse.isCommitted は変換されませんでした。

javax.servlet.ServletResponse.resetBuffer は変換されませんでした。

javax.servlet.ServletResponse.setBufferSize は変換されませんでした。

javax.servlet.ServletResponse.setContentLength は変換されませんでした。

javax.servlet.ServletResponseWrapper.flushBuffer は変換されませんでした。

javax.servlet.ServletResponseWrapper.getBufferSize は変換されませんでした。

javax.servlet.ServletResponseWrapper.getLocale は変換されませんでした。

javax.servlet.ServletResponseWrapper.isCommitted は変換されませんでした。

javax.servlet.ServletResponseWrapper.setBufferSize は変換されませんでした。

javax.servlet.ServletResponseWrapper.setContentLength は変換されませんでした。

javax.servlet.ServletResponseWrapper.setLocale は変換されませんでした。

javax.servlet.UnavailableException.getServlet は変換されませんでした。

javax.servlet.UnavailableException.getUnavailableSeconds は変換されませんでした。

javax.servlet.UnavailableException.isPermanent は変換されませんでした。

javax.servlet.UnavailableException.UnavailableException は変換されませんでした。

Javax.servlet.http のエラー メッセージ

javax.servlet.http.Cookie.clone は変換されませんでした。

javax.servlet.http.Cookie.getComment は変換されませんでした。

javax.servlet.http.Cookie.getMaxAge は変換されませんでした。

javax.servlet.http.Cookie.getSecure は変換されませんでした。

javax.servlet.http.Cookie.getVersion は変換されませんでした。

javax.servlet.http.Cookie.setComment は変換されませんでした。

javax.servlet.http.Cookie.setMaxAge は変換されませんでした。

javax.servlet.http.Cookie.setSecure は変換されませんでした。

javax.servlet.http.Cookie.setVersion は変換されませんでした。

javax.servlet.http.HttpServlet.getLastModified は変換されませんでした。

javax.servlet.http.HttpServlet.HttpServlet は変換されませんでした。

javax.servlet.http.HttpServlet.service(HttpServletRequest, HttpServletResponse) は変換されませんでした。

javax.servlet.http.HttpServlet.service(ServletRequest, ServletResponse) は変換されませんでした。

javax.servlet.http.HttpServletRequest.BASIC_AUTH は変換されませんでした。

javax.servlet.http.HttpServletRequest.CLIENT_CERT_AUTH は変換されませんでした。

javax.servlet.http.HttpServletRequest.DIGEST_AUTH は変換されませんでした。

javax.servlet.http.HttpServletRequest.FORM_AUTH は変換されませんでした。

javax.servlet.http.HttpServletRequest.getQueryString は変換されませんでした。

javax.servlet.http.HttpServletRequest.getSession は変換されませんでした。

javax.servlet.http.HttpServletRequest.isRequestedSessionIdFromCookie は変換されませんでした。

javax.servlet.http.HttpServletRequest.isRequestedSessionIdFromURL は変換されませんでした。

javax.servlet.http.HttpServletRequest.isRequestedSessionIdValid は変換されませんでした。

javax.servlet.http.HttpServletRequestWrapper.getSession は変換されませんでした。

javax.servlet.http.HttpServletRequestWrapper.isRequestedSessionIdFromCookie は変換されませんでした。

javax.servlet.http.HttpServletRequestWrapper.isRequestedSessionIdFromURL は変換されませんでした。

javax.servlet.http.HttpServletRequestWrapper.isRequestedSessionIdValid は変換されませんでした。

javax.servlet.http.HttpServletResponse.addDateHeader は変換されませんでした。

javax.servlet.http.HttpServletResponse.containsHeader は変換されませんでした。

javax.servlet.http.HttpServletResponse.encodeRedirectURL は変換されませんでした。

javax.servlet.http.HttpServletResponse.encodeURL は変換されませんでした。

javax.servlet.http.HttpServletResponse.SC_TEMPORARY_REDIRECT は変換されませんでした。

javax.servlet.http.HttpServletResponse.setDateHeader は変換されませんでした。

javax.servlet.http.HttpServletResponseWrapper.containsHeader は変換されませんでした。

javax.servlet.http.HttpServletResponseWrapper.setDateHeader は変換されませんでした。

javax.servlet.http.HttpServletResponseWrapper.setHeader は変換されませんでした。

javax.servlet.http.HttpServletResponseWrapper.setIntHeader は変換されませんでした。

javax.servlet.http.HttpSession.getCreationTime は変換されませんでした。

javax.servlet.http.HttpSession.getLastAccessedTime は変換されませんでした。

javax.servlet.http.HttpSession.getMaxInactiveInterval は変換されませんでした。

javax.servlet.http.HttpSession.getSessionContext は変換されませんでした。

javax.servlet.http.HttpSession.getValue は変換されませんでした。

javax.servlet.http.HttpSession.invalidate は変換されませんでした。

javax.servlet.http.HttpSession.putValue は変換されませんでした。

javax.servlet.http.HttpSession.removeValue は変換されませんでした。

javax.servlet.http.HttpSession.setMaxInactiveInterval は変換されませんでした。

javax.servlet.http.HttpSessionActivationListener は変換されませんでした。

javax.servlet.http.HttpSessionAttributeListener は変換されませんでした。

javax.servlet.http.HttpSessionBindingEvent は変換されませんでした。

javax.servlet.http.HttpSessionBindingListener は変換されませんでした。

javax.servlet.http.HttpSessionContext は変換されませんでした。

javax.servlet.http.HttpSessionEvent は変換されませんでした。

javax.servlet.http.HttpSessionListener は変換されませんでした。

javax.servlet.http.HttpUtils は変換されませんでした。

Javax.servlet.jsp のエラー メッセージ

javax.servlet.jsp.HttpJspPage は変換されませんでした。

javax.servlet.jsp.JspEngineInfo は変換されませんでした。

javax.servlet.jsp.JspFactory は変換されませんでした。

javax.servlet.jsp.JspPage は変換されませんでした。

javax.servlet.jsp.JspWriter.bufferSize は変換されませんでした。

javax.servlet.jsp.JspWriter.DEFAULT_BUFFER は変換されませんでした。

javax.servlet.jsp.JspWriter.getBufferSize は変換されませんでした。

javax.servlet.jsp.JspWriter.getRemaining は変換されませんでした。

javax.servlet.jsp.JspWriter.NO_BUFFER は変換されませんでした。

javax.servlet.jsp.JspWriter.UNBOUNDED_BUFFER は変換されませんでした。

javax.servlet.jsp.PageContext.APPLICATION は変換されませんでした。

javax.servlet.jsp.PageContext.APPLICATION_SCOPE は変換されませんでした。

javax.servlet.jsp.PageContext.CONFIG は変換されませんでした。

javax.servlet.jsp.PageContext.EXCEPTION は変換されませんでした。

javax.servlet.jsp.PageContext.findAttribute は変換されませんでした。

javax.servlet.jsp.PageContext.getAttribute は変換されませんでした。

javax.servlet.jsp.PageContext.getAttributeNamesInScope は変換されませんでした。

javax.servlet.jsp.PageContext.getAttributesScope は変換されませんでした。

javax.servlet.jsp.PageContext.getServletConfig は変換されませんでした。

javax.servlet.jsp.PageContext.handlePageException は変換されませんでした。

javax.servlet.jsp.PageContext.initialize は変換されませんでした。

javax.servlet.jsp.PageContext.OUT は変換されませんでした。

javax.servlet.jsp.PageContext.PAGE は変換されませんでした。

javax.servlet.jsp.PageContext.PageContext は変換されませんでした。

javax.servlet.jsp.PageContext.PAGECONTEXT は変換されませんでした。

javax.servlet.jsp.PageContext.PAGE_SCOPE は変換されませんでした。

javax.servlet.jsp.PageContext.popBody は変換されませんでした。

javax.servlet.jsp.PageContext.pushBody は変換されませんでした。

javax.servlet.jsp.PageContext.release は変換されませんでした。

javax.servlet.jsp.PageContext.removeAttribute は変換されませんでした。

javax.servlet.jsp.PageContext.REQUEST は変換されませんでした。

javax.servlet.jsp.PageContext.REQUEST_SCOPE は変換されませんでした。

javax.servlet.jsp.PageContext.RESPONSE は変換されませんでした。

javax.servlet.jsp.PageContext.SESSION は変換されませんでした。

javax.servlet.jsp.PageContext.SESSION_SCOPE は変換されませんでした。

javax.servlet.jsp.PageContext.setAttribute は変換されませんでした。

javax.servlet.jsp.tagext.BodyContent.flush は変換されませんでした。

javax.servlet.jsp.tagext.PageData は変換されませんでした。

javax.servlet.jsp.tagext.Tag.setParent は変換されませんでした。

javax.servlet.jsp.tagext.TagAttributeInfo は変換されませんでした。

javax.servlet.jsp.tagext.TagData は変換されませんでした。

javax.servlet.jsp.tagext.TagExtraInfo は変換されませんでした。

javax.servlet.jsp.tagext.TagInfo は変換されませんでした。

javax.servlet.jsp.tagext.TagLibraryInfo は変換されませんでした。

javax.servlet.jsp.tagext.TagLibraryValidator は変換されませんでした。

javax.servlet.jsp.tagext.TagSupport.setParent は変換されませんでした。

javax.servlet.jsp.tagext.TagVariableInfo は変換されませんでした。

javax.servlet.jsp.tagext.TryCatchFinally は変換されませんでした。

javax.servlet.jsp.tagext.ValidationMessage は変換されませんでした。

javax.servlet.jsp.tagext.VariableInfo は変換されませんでした。

Javax.sound のエラー メッセージ

javax.sound.midi.ControllerEventListener は変換されませんでした。

javax.sound.midi.Instrument は変換されませんでした。

javax.sound.midi.InvalidMidiDataException は変換されませんでした。

javax.sound.midi.MetaEventListener は変換されませんでした。

javax.sound.midi.MetaMessage は変換されませんでした。

javax.sound.midi.MidiChannel は変換されませんでした。

javax.sound.midi.MidiDevice は変換されませんでした。

javax.sound.midi.MidiDevice.Info は変換されませんでした。

javax.sound.midi.MidiEvent は変換されませんでした。

javax.sound.midi.MidiFileFormat は変換されませんでした。

javax.sound.midi.MidiMessage は変換されませんでした。

javax.sound.midi.MidiSystem は変換されませんでした。

javax.sound.midi.MidiUnavailableException は変換されませんでした。

javax.sound.midi.Patch は変換されませんでした。

javax.sound.midi.Receiver は変換されませんでした。

javax.sound.midi.Sequence は変換されませんでした。

javax.sound.midi.Sequencer は変換されませんでした。

javax.sound.midi.Sequencer.SyncMode は変換されませんでした。

javax.sound.midi.ShortMessage は変換されませんでした。

javax.sound.midi.Soundbank は変換されませんでした。

javax.sound.midi.SoundbankResource は変換されませんでした。

javax.sound.midi.spi.MidiDeviceProvider は変換されませんでした。

javax.sound.midi.spi.MidiFileReader は変換されませんでした。

javax.sound.midi.spi.MidiFileWriter は変換されませんでした。

javax.sound.midi.spi.SoundbankReader は変換されませんでした。

javax.sound.midi.Synthesizer は変換されませんでした。

javax.sound.midi.SysexMessage は変換されませんでした。

javax.sound.midi.Track は変換されませんでした。

javax.sound.midi.Transmitter は変換されませんでした。

javax.sound.midi.VoiceStatus は変換されませんでした。

javax.sound.sampled.AudioFileFormat は変換されませんでした。

javax.sound.sampled.AudioFileFormat.Type は変換されませんでした。

javax.sound.sampled.AudioFormat.bigEndian は変換されませんでした。

javax.sound.sampled.AudioFormat.Encoding.<EncodingType> は変換されませんでした。

javax.sound.sampled.AudioFormat.Encoding.Encoding は変換されませんでした。

javax.sound.sampled.AudioFormat.isBigEndian は変換されませんでした。

javax.sound.sampled.AudioFormat.matches は変換されませんでした。

javax.sound.sampled.AudioInputStream.AudioInputStream は変換されませんでした。

javax.sound.sampled.AudioInputStream.available は変換されませんでした。

javax.sound.sampled.AudioInputStream.format は変換されませんでした。

javax.sound.sampled.AudioInputStream.frameLength は変換されませんでした。

javax.sound.sampled.AudioInputStream.framePos は変換されませんでした。

javax.sound.sampled.AudioInputStream.frameSize は変換されませんでした。

javax.sound.sampled.AudioInputStream.getFormat は変換されませんでした。

javax.sound.sampled.AudioInputStream.getFrameLength は変換されませんでした。

javax.sound.sampled.AudioInputStream.mark は変換されませんでした。

javax.sound.sampled.AudioInputStream.markSupported は変換されませんでした。

javax.sound.sampled.AudioPermission は変換されませんでした。

javax.sound.sampled.AudioSystem は変換されませんでした。

javax.sound.sampled.BooleanControl は変換されませんでした。

javax.sound.sampled.BooleanControl.Type は変換されませんでした。

javax.sound.sampled.Clip.getFrameLength は変換されませんでした。

javax.sound.sampled.Clip.getMicrosecondLength は変換されませんでした。

javax.sound.sampled.Clip.loop は変換されませんでした。

javax.sound.sampled.Clip.open は変換されませんでした。

javax.sound.sampled.Clip.setLoopPoints は変換されませんでした。

javax.sound.sampled.Clip.setMicrosecondPosition は変換されませんでした。

javax.sound.sampled.CompoundControl は変換されませんでした。

javax.sound.sampled.CompoundControl.Type は変換されませんでした。

javax.sound.sampled.Control は変換されませんでした。

javax.sound.sampled.Control.Type は変換されませんでした。

javax.sound.sampled.DataLine.available は変換されませんでした。

javax.sound.sampled.DataLine.drain は変換されませんでした。

javax.sound.sampled.DataLine.flush は変換されませんでした。

javax.sound.sampled.DataLine.getMicrosecondPosition は変換されませんでした。

javax.sound.sampled.DataLine.Info は変換されませんでした。

javax.sound.sampled.DataLine.start は変換されませんでした。

javax.sound.sampled.EnumControl は変換されませんでした。

javax.sound.sampled.EnumControl.Type は変換されませんでした。

javax.sound.sampled.FloatControl は変換されませんでした。

javax.sound.sampled.FloatControl.Type は変換されませんでした。

javax.sound.sampled.Line は変換されませんでした。

javax.sound.sampled.Line.Info は変換されませんでした。

javax.sound.sampled.LineEvent は変換されませんでした。

javax.sound.sampled.LineEvent.Type は変換されませんでした。

javax.sound.sampled.LineListener は変換されませんでした。

javax.sound.sampled.Mixer は変換されませんでした。

javax.sound.sampled.Mixer.Info は変換されませんでした。

javax.sound.sampled.Port は変換されませんでした。

javax.sound.sampled.Port.Info は変換されませんでした。

javax.sound.sampled.ReverbType は変換されませんでした。

javax.sound.sampled.SourceDataLine は変換されませんでした。

javax.sound.sampled.SourceDataLine.open は変換されませんでした。

javax.sound.sampled.SourceDataLine.write は変換されませんでした。

javax.sound.sampled.spi.AudioFileReader は変換されませんでした。

javax.sound.sampled.spi.AudioFileWriter は変換されませんでした。

javax.sound.sampled.spi.FormatConversionProvider は変換されませんでした。

javax.sound.sampled.spi.MixerProvider は変換されませんでした。

javax.sound.sampled.TargetDataLine は変換されませんでした。

javax.sound.sampled.TargetDataLine.open は変換されませんでした。

javax.sound.sampled.TargetDataLine.read は変換されませんでした。

Javax.sql のエラー メッセージ

javax.sql.ConnectionEvent は変換されませんでした。

javax.sql.ConnectionEventListener は変換されませんでした。

javax.sql.ConnectionPoolDataSource は変換されませんでした。

javax.sql.DataSource は変換されませんでした。

javax.sql.PooledConnection.addConnectionEventListener は変換されませんでした。

javax.sql.PooledConnection.removeConnectionEventListener は変換されませんでした。

javax.sql.RowSet.addRowSetListener は変換されませんでした。

javax.sql.RowSet.getEscapeProcessing は変換されませんでした。

javax.sql.RowSet.getMaxFieldSize は変換されませんでした。

javax.sql.RowSet.getMaxRows は変換されませんでした。

javax.sql.RowSet.getPassword は変換されませんでした。

javax.sql.RowSet.getTypeMap は変換されませんでした。

javax.sql.RowSet.getUrl は変換されませんでした。

javax.sql.RowSet.getUsername は変換されませんでした。

javax.sql.RowSet.isReadOnly は変換されませんでした。

javax.sql.RowSet.removeRowSetListener は変換されませんでした。

javax.sql.RowSet.setAsciiStream は変換されませんでした。

javax.sql.RowSet.setBinaryStream は変換されませんでした。

javax.sql.RowSet.setBlob は変換されませんでした。

javax.sql.RowSet.setCharacterStream は変換されませんでした。

javax.sql.RowSet.setClob は変換されませんでした。

javax.sql.RowSet.setConcurrency は変換されませんでした。

javax.sql.RowSet.setDataSourceName は変換されませんでした。

javax.sql.RowSet.setEscapeProcessing は変換されませんでした。

javax.sql.RowSet.setMaxFieldSize は変換されませんでした。

javax.sql.RowSet.setMaxRows は変換されませんでした。

javax.sql.RowSet.setObject は変換されませんでした。

javax.sql.RowSet.setPassword は変換されませんでした。

javax.sql.RowSet.setReadOnly は変換されませんでした。

javax.sql.RowSet.setType は変換されませんでした。

javax.sql.RowSet.setTypeMap は変換されませんでした。

javax.sql.RowSet.setUrl は変換されませんでした。

javax.sql.RowSet.setUsername は変換されませんでした。

javax.sql.RowSetEvent は変換されませんでした。

javax.sql.RowSetInternal は変換されませんでした。

javax.sql.RowSetListener は変換されませんでした。

javax.sql.RowSetMetaData.setCatalogName は変換されませんでした。

javax.sql.RowSetMetaData.setColumnCount は変換されませんでした。

javax.sql.RowSetMetaData.setCurrency は変換されませんでした。

javax.sql.RowSetMetaData.setPrecision は変換されませんでした。

javax.sql.RowSetMetaData.setScale は変換されませんでした。

javax.sql.RowSetMetaData.setSearchable は変換されませんでした。

javax.sql.RowSetMetaData.setSigned は変換されませんでした。

javax.sql.RowSetReader は変換されませんでした。

javax.sql.RowSetWriter は変換されませんでした。

javax.sql.XAConnection は変換されませんでした。

javax.sql.XAConnection.getXAResource は変換されませんでした。

javax.sql.XADataSource は変換されませんでした。

Javax.swing のエラー メッセージ

javax.swing.<ClassName>.addNotify は変換されませんでした。

javax.swing.<ClassName>.process*Event は変換されませんでした。

javax.swing.<ClassName>.removeNotify は変換されませんでした。

javax.swing.<ClassName>.setLayout は変換されませんでした。

javax.swing.<ClassName>BeanInfo は変換されませんでした。

javax.swing.AbstractAction.AbstractAction(String) は変換されませんでした。

javax.swing.AbstractAction.AbstractAction(String, Icon) は変換されませんでした。

javax.swing.AbstractAction.addPropertyChangeListener は変換されませんでした。

javax.swing.AbstractAction.changeSupport は変換されませんでした。

javax.swing.AbstractAction.enabled は変換されませんでした。

javax.swing.AbstractAction.firePropertyChange は変換されませんでした。

javax.swing.AbstractAction.getKeys は変換されませんでした。

javax.swing.AbstractAction.getValue は変換されませんでした。

javax.swing.AbstractAction.isEnabled は変換されませんでした。

javax.swing.AbstractAction.putValue は変換されませんでした。

javax.swing.AbstractAction.removePropertyChangeListener は変換されませんでした。

javax.swing.AbstractAction.setEnabled は変換されませんでした。

javax.swing.AbstractButton.<ChangedProperty> は変換されませんでした。

javax.swing.AbstractButton.AbstractButton は変換されませんでした。

javax.swing.AbstractButton.actionListener は変換されませんでした。

javax.swing.AbstractButton.addChangeListener は変換されませんでした。

javax.swing.AbstractButton.changeEvent は変換されませんでした。

javax.swing.AbstractButton.changeListener は変換されませんでした。

javax.swing.AbstractButton.checkHorizontalKey は変換されませんでした。

javax.swing.AbstractButton.checkVerticalKey は変換されませんでした。

javax.swing.AbstractButton.configurePropertiesFromAction は変換されませんでした。

javax.swing.AbstractButton.createActionListener は変換されませんでした。

javax.swing.AbstractButton.createActionPropertyChangeListener は変換されませんでした。

javax.swing.AbstractButton.createChangeListener は変換されませんでした。

javax.swing.AbstractButton.createItemListener は変換されませんでした。

javax.swing.AbstractButton.doClick は変換されませんでした。

javax.swing.AbstractButton.fireItemStateChanged は変換されませんでした。

javax.swing.AbstractButton.fireStateChanged は変換されませんでした。

javax.swing.AbstractButton.get<State>Icon は変換されませんでした。

javax.swing.AbstractButton.getAction は変換されませんでした。

javax.swing.AbstractButton.getActionCommand は変換されませんでした。

javax.swing.AbstractButton.getHorizontalAlignment は変換されませんでした。

javax.swing.AbstractButton.getHorizontalTextPosition は変換されませんでした。

javax.swing.AbstractButton.getIcon は変換されませんでした。

javax.swing.AbstractButton.getMargin は変換されませんでした。

javax.swing.AbstractButton.getMnemonic は変換されませんでした。

javax.swing.AbstractButton.getModel は変換されませんでした。

javax.swing.AbstractButton.getSelectedObjects は変換されませんでした。

javax.swing.AbstractButton.getUI は変換されませんでした。

javax.swing.AbstractButton.getVerticalAlignment は変換されませんでした。

javax.swing.AbstractButton.getVerticalTextPosition は変換されませんでした。

javax.swing.AbstractButton.imageUpdate は変換されませんでした。

javax.swing.AbstractButton.init は変換されませんでした。

javax.swing.AbstractButton.isBorderPainted は変換されませんでした。

javax.swing.AbstractButton.isContentAreaFilled は変換されませんでした。

javax.swing.AbstractButton.isFocusPainted は変換されませんでした。

javax.swing.AbstractButton.isRolloverEnabled は変換されませんでした。

javax.swing.AbstractButton.itemListener は変換されませんでした。

javax.swing.AbstractButton.model は変換されませんでした。

javax.swing.AbstractButton.paintBorder は変換されませんでした。

javax.swing.AbstractButton.removeChangeListener は変換されませんでした。

javax.swing.AbstractButton.set<State>Icon は変換されませんでした。

javax.swing.AbstractButton.setActionCommand は変換されませんでした。

javax.swing.AbstractButton.setBorderPainted は変換されませんでした。

javax.swing.AbstractButton.setContentAreaFilled は変換されませんでした。

javax.swing.AbstractButton.setFocusPainted は変換されませんでした。

javax.swing.AbstractButton.setHorizontalAlignment は変換されませんでした。

javax.swing.AbstractButton.setHorizontalTextPosition は変換されませんでした。

javax.swing.AbstractButton.setIcon は変換されませんでした。

javax.swing.AbstractButton.setMargin は変換されませんでした。

javax.swing.AbstractButton.setMnemonic は変換されませんでした。

javax.swing.AbstractButton.setModel は変換されませんでした。

javax.swing.AbstractButton.setRolloverEnabled は変換されませんでした。

javax.swing.AbstractButton.setSelected は変換されませんでした。

javax.swing.AbstractButton.setUI は変換されませんでした。

javax.swing.AbstractButton.setVerticalAlignment は変換されませんでした。

javax.swing.AbstractButton.setVerticalTextPosition は変換されませんでした。

javax.swing.AbstractButton.updateUI は変換されませんでした。

javax.swing.AbstractCellEditor は変換されませんでした。

javax.swing.AbstractListModel は変換されませんでした。

javax.swing.AbstractListModel.addListDataListener は変換されませんでした。

javax.swing.AbstractListModel.fireContentsChanged は変換されませんでした。

javax.swing.AbstractListModel.fireIntervalAdded は変換されませんでした。

javax.swing.AbstractListModel.fireIntervalRemoved は変換されませんでした。

javax.swing.AbstractListModel.getListeners は変換されませんでした。

javax.swing.AbstractListModel.listenerList は変換されませんでした。

javax.swing.AbstractListModel.removeListDataListener は変換されませんでした。

javax.swing.Action.ACCELERATOR_KEY は変換されませんでした。

javax.swing.Action.ACTION_COMMAND_KEY は変換されませんでした。

javax.swing.Action.addPropertyChangeListener は変換されませんでした。

javax.swing.Action.DEFAULT は変換されませんでした。

javax.swing.Action.getValue は変換されませんでした。

javax.swing.Action.isEnabled は変換されませんでした。

javax.swing.Action.LONG_DESCRIPTION は変換されませんでした。

javax.swing.Action.MNEMONIC_KEY は変換されませんでした。

javax.swing.Action.NAME は変換されませんでした。

javax.swing.Action.putValue は変換されませんでした。

javax.swing.Action.removePropertyChangeListener は変換されませんでした。

javax.swing.Action.setEnabled は変換されませんでした。

javax.swing.Action.SHORT_DESCRIPTION は変換されませんでした。

javax.swing.Action.SMALL_ICON は変換されませんでした。

javax.swing.ActionMap は変換されませんでした。

javax.swing.ActionMap.allKeys は変換されませんでした。

javax.swing.ActionMap.get は変換されませんでした。

javax.swing.ActionMap.getParent は変換されませんでした。

javax.swing.ActionMap.keys は変換されませんでした。

javax.swing.ActionMap.setParent は変換されませんでした。

javax.swing.ActionMap.size は変換されませんでした。

javax.swing.BorderFactory は変換されませんでした。

javax.swing.BorderFactory.BorderFactory は変換されませんでした。

javax.swing.BorderFactory.createCompoundBorder は変換されませんでした。

javax.swing.BorderFactory.createEtchedBorder は変換されませんでした。

javax.swing.BorderFactory.createMatteBorder は変換されませんでした。

javax.swing.BorderFactory.createTitledBorder は変換されませんでした。

javax.swing.BoundedRangeModel は変換されませんでした。

javax.swing.Box は変換されませんでした。

javax.swing.Box.Filler は変換されませんでした。

javax.swing.BoxLayout は変換されませんでした。

javax.swing.ButtonGroup は変換されませんでした。

javax.swing.ButtonGroup.add は変換されませんでした。

javax.swing.ButtonModel は変換されませんでした。

javax.swing.ButtonModel.addActionListener は変換されませんでした。

javax.swing.ButtonModel.addChangeListener は変換されませんでした。

javax.swing.ButtonModel.addItemListener は変換されませんでした。

javax.swing.ButtonModel.getActionCommand は変換されませんでした。

javax.swing.ButtonModel.getMnemonic は変換されませんでした。

javax.swing.ButtonModel.isArmed は変換されませんでした。

javax.swing.ButtonModel.isPressed は変換されませんでした。

javax.swing.ButtonModel.isRollover は変換されませんでした。

javax.swing.ButtonModel.isSelected は変換されませんでした。

javax.swing.ButtonModel.removeActionListener は変換されませんでした。

javax.swing.ButtonModel.removeChangeListener は変換されませんでした。

javax.swing.ButtonModel.removeItemListener は変換されませんでした。

javax.swing.ButtonModel.setActionCommand は変換されませんでした。

javax.swing.ButtonModel.setArmed は変換されませんでした。

javax.swing.ButtonModel.setGroup は変換されませんでした。

javax.swing.ButtonModel.setMnemonic は変換されませんでした。

javax.swing.ButtonModel.setPressed は変換されませんでした。

javax.swing.ButtonModel.setRollover は変換されませんでした。

javax.swing.ButtonModel.setSelected は変換されませんでした。

javax.swing.CellEditor は変換されませんでした。

javax.swing.CellRendererPane は変換されませんでした。

javax.swing.ComboBoxEditor は変換されませんでした。

javax.swing.ComboBoxModel.getSelectedItem は変換されませんでした。

javax.swing.ComboBoxModel.setSelectedItem は変換されませんでした。

javax.swing.ComponentInputMap は変換されませんでした。

javax.swing.DebugGraphics は変換されませんでした。

javax.swing.DebugGraphics.BUFFERED_OPTION は変換されませんでした。

javax.swing.DebugGraphics.clearRect は変換されませんでした。

javax.swing.DebugGraphics.clipRect は変換されませんでした。

javax.swing.DebugGraphics.copyArea は変換されませんでした。

javax.swing.DebugGraphics.create は変換されませんでした。

javax.swing.DebugGraphics.DebugGraphics は変換されませんでした。

javax.swing.DebugGraphics.DebugGraphics(Graphics) は変換されませんでした。

javax.swing.DebugGraphics.DebugGraphics (Graphics, IComponent) は変換されませんでした。

javax.swing.DebugGraphics.dispose は変換されませんでした。

javax.swing.DebugGraphics.draw3DRect は変換されませんでした。

javax.swing.DebugGraphics.drawArc は変換されませんでした。

javax.swing.DebugGraphics.drawBytes は変換されませんでした。

javax.swing.DebugGraphics.drawChars は変換されませんでした。

javax.swing.DebugGraphics.drawImage は変換されませんでした。

javax.swing.DebugGraphics.drawLine は変換されませんでした。

javax.swing.DebugGraphics.drawOval は変換されませんでした。

javax.swing.DebugGraphics.drawPolygon は変換されませんでした。

javax.swing.DebugGraphics.drawPolyline は変換されませんでした。

javax.swing.DebugGraphics.drawRect は変換されませんでした。

javax.swing.DebugGraphics.drawRoundRect は変換されませんでした。

javax.swing.DebugGraphics.drawString(AttributedCharacterIterator, int, int) は変換されませんでした。

javax.swing.DebugGraphics.drawString(String, int, int) は変換されませんでした。

javax.swing.DebugGraphics.fill3DRect は変換されませんでした。

javax.swing.DebugGraphics.fillArc は変換されませんでした。

javax.swing.DebugGraphics.fillOval は変換されませんでした。

javax.swing.DebugGraphics.fillPolygon は変換されませんでした。

javax.swing.DebugGraphics.fillRect は変換されませんでした。

javax.swing.DebugGraphics.fillRoundRect は変換されませんでした。

javax.swing.DebugGraphics.FLASH_OPTION は変換されませんでした。

javax.swing.DebugGraphics.flashColor は変換されませんでした。

javax.swing.DebugGraphics.flashCount は変換されませんでした。

javax.swing.DebugGraphics.flashTime は変換されませんでした。

javax.swing.DebugGraphics.getClip は変換されませんでした。

javax.swing.DebugGraphics.getClipBounds は変換されませんでした。

javax.swing.DebugGraphics.getColor は変換されませんでした。

javax.swing.DebugGraphics.getDebugOptions は変換されませんでした。

javax.swing.DebugGraphics.getFont は変換されませんでした。

javax.swing.DebugGraphics.getFontMetrics は変換されませんでした。

javax.swing.DebugGraphics.isDrawingBuffer は変換されませんでした。

javax.swing.DebugGraphics.LOG_OPTION は変換されませんでした。

javax.swing.DebugGraphics.logStream は変換されませんでした。

javax.swing.DebugGraphics.NONE_OPTION は変換されませんでした。

javax.swing.DebugGraphics.setClip は変換されませんでした。

javax.swing.DebugGraphics.setColor は変換されませんでした。

javax.swing.DebugGraphics.setDebugOptions は変換されませんでした。

javax.swing.DebugGraphics.setFlashColor は変換されませんでした。

javax.swing.DebugGraphics.setFlashCount は変換されませんでした。

javax.swing.DebugGraphics.setFlashTime は変換されませんでした。

javax.swing.DebugGraphics.setFont は変換されませんでした。

javax.swing.DebugGraphics.setLogStream は変換されませんでした。

javax.swing.DebugGraphics.setPaintMode は変換されませんでした。

javax.swing.DebugGraphics.setXORMode は変換されませんでした。

javax.swing.DebugGraphics.translate は変換されませんでした。

javax.swing.DefaultBoundedRangeModel は変換されませんでした。

javax.swing.DefaultButtonModel は変換されませんでした。

javax.swing.DefaultButtonModel.actionCommand は変換されませんでした。

javax.swing.DefaultButtonModel.addActionListener は変換されませんでした。

javax.swing.DefaultButtonModel.addChangeListener は変換されませんでした。

javax.swing.DefaultButtonModel.addItemListener は変換されませんでした。

javax.swing.DefaultButtonModel.ARMED は変換されませんでした。

javax.swing.DefaultButtonModel.changeEvent は変換されませんでした。

javax.swing.DefaultButtonModel.DefaultButtonModel は変換されませんでした。

javax.swing.DefaultButtonModel.ENABLED は変換されませんでした。

javax.swing.DefaultButtonModel.fireActionPerformed は変換されませんでした。

javax.swing.DefaultButtonModel.fireItemStateChanged は変換されませんでした。

javax.swing.DefaultButtonModel.fireStateChanged は変換されませんでした。

javax.swing.DefaultButtonModel.getActionCommand は変換されませんでした。

javax.swing.DefaultButtonModel.getGroup は変換されませんでした。

javax.swing.DefaultButtonModel.getListeners は変換されませんでした。

javax.swing.DefaultButtonModel.getMnemonic は変換されませんでした。

javax.swing.DefaultButtonModel.getSelectedObjects は変換されませんでした。

javax.swing.DefaultButtonModel.group は変換されませんでした。

javax.swing.DefaultButtonModel.isArmed は変換されませんでした。

javax.swing.DefaultButtonModel.isPressed は変換されませんでした。

javax.swing.DefaultButtonModel.isRollover は変換されませんでした。

javax.swing.DefaultButtonModel.isSelected は変換されませんでした。

javax.swing.DefaultButtonModel.listenerList は変換されませんでした。

javax.swing.DefaultButtonModel.mnemonic は変換されませんでした。

javax.swing.DefaultButtonModel.PRESSED は変換されませんでした。

javax.swing.DefaultButtonModel.removeActionListener は変換されませんでした。

javax.swing.DefaultButtonModel.removeChangeListener は変換されませんでした。

javax.swing.DefaultButtonModel.removeItemListener は変換されませんでした。

javax.swing.DefaultButtonModel.ROLLOVER は変換されませんでした。

javax.swing.DefaultButtonModel.SELECTED は変換されませんでした。

javax.swing.DefaultButtonModel.setActionCommand は変換されませんでした。

javax.swing.DefaultButtonModel.setArmed は変換されませんでした。

javax.swing.DefaultButtonModel.setGroup は変換されませんでした。

javax.swing.DefaultButtonModel.setMnemonic は変換されませんでした。

javax.swing.DefaultButtonModel.setPressed は変換されませんでした。

javax.swing.DefaultButtonModel.setRollover は変換されませんでした。

javax.swing.DefaultButtonModel.setSelected は変換されませんでした。

javax.swing.DefaultButtonModel.stateMask は変換されませんでした。

javax.swing.DefaultCellEditor は変換されませんでした。

javax.swing.DefaultComboBoxModel は変換されませんでした。

javax.swing.DefaultComboBoxModel.DefaultComboBoxModel は変換されませんでした。

javax.swing.DefaultComboBoxModel.DefaultComboBoxModel(Object[]) は変換されませんでした。

javax.swing.DefaultComboBoxModel.DefaultComboBoxModel(Vector) は変換されませんでした。

javax.swing.DefaultComboBoxModel.getSelectedItem は変換されませんでした。

javax.swing.DefaultComboBoxModel.setSelectedItem は変換されませんでした。

javax.swing.DefaultDesktopManager は変換されませんでした。

javax.swing.DefaultDesktopManager.deiconifyFrame は変換されませんでした。

javax.swing.DefaultDesktopManager.iconifyFrame は変換されませんでした。

javax.swing.DefaultDesktopManager.maximizeFrame は変換されませんでした。

javax.swing.DefaultDesktopManager.minimizeFrame は変換されませんでした。

javax.swing.DefaultFocusManager は変換されませんでした。

javax.swing.DefaultListCellRenderer は変換されませんでした。

javax.swing.DefaultListCellRenderer.UIResource は変換されませんでした。

javax.swing.DefaultListModel は変換されませんでした。

javax.swing.DefaultListModel.capacity は変換されませんでした。

javax.swing.DefaultListModel.ensureCapacity は変換されませんでした。

javax.swing.DefaultListModel.setSize は変換されませんでした。

javax.swing.DefaultListModel.trimToSize は変換されませんでした。

javax.swing.DefaultListSelectionModel は変換されませんでした。

javax.swing.DefaultListSelectionModel.addListSelectionListener は変換されませんでした。

javax.swing.DefaultListSelectionModel.addSelectionInterval は変換されませんでした。

javax.swing.DefaultListSelectionModel.clone は変換されませんでした。

javax.swing.DefaultListSelectionModel.DefaultListSelectionModel は変換されませんでした。

javax.swing.DefaultListSelectionModel.fireValueChanged(boolean) は変換されませんでした。

javax.swing.DefaultListSelectionModel.fireValueChanged(int, int) は変換されませんでした。

javax.swing.DefaultListSelectionModel.fireValueChanged(int, int, boolean) は変換されませんでした。

javax.swing.DefaultListSelectionModel.getAnchorSelectionIndex は変換されませんでした。

javax.swing.DefaultListSelectionModel.getLeadSelectionIndex は変換されませんでした。

javax.swing.DefaultListSelectionModel.getListeners は変換されませんでした。

javax.swing.DefaultListSelectionModel.getMaxSelectionIndex は変換されませんでした。

javax.swing.DefaultListSelectionModel.getMinSelectionIndex は変換されませんでした。

javax.swing.DefaultListSelectionModel.getSelectionMode は変換されませんでした。

javax.swing.DefaultListSelectionModel.getValueIsAdjusting は変換されませんでした。

javax.swing.DefaultListSelectionModel.insertIndexInterval は変換されませんでした。

javax.swing.DefaultListSelectionModel.isLeadAnchorNotificationEnabled は変換されませんでした。

javax.swing.DefaultListSelectionModel.leadAnchorNotificationEnabled は変換されませんでした。

javax.swing.DefaultListSelectionModel.listenerList は変換されませんでした。

javax.swing.DefaultListSelectionModel.removeIndexInterval は変換されませんでした。

javax.swing.DefaultListSelectionModel.removeListSelectionListener は変換されませんでした。

javax.swing.DefaultListSelectionModel.removeSelectionInterval は変換されませんでした。

javax.swing.DefaultListSelectionModel.setAnchorSelectionIndex は変換されませんでした。

javax.swing.DefaultListSelectionModel.setLeadAnchorNotificationEnabled は変換されませんでした。

javax.swing.DefaultListSelectionModel.setLeadSelectionIndex は変換されませんでした。

javax.swing.DefaultListSelectionModel.setSelectionInterval は変換されませんでした。

javax.swing.DefaultListSelectionModel.setSelectionMode は変換されませんでした。

javax.swing.DefaultListSelectionModel.setValueIsAdjusting は変換されませんでした。

javax.swing.DefaultSingleSelectionModel は変換されませんでした。

javax.swing.DefaultSingleSelectionModel.addChangeListener は変換されませんでした。

javax.swing.DefaultSingleSelectionModel.changeEvent は変換されませんでした。

javax.swing.DefaultSingleSelectionModel.DefaultSingleSelectionModel は変換されませんでした。

javax.swing.DefaultSingleSelectionModel.fireStateChanged は変換されませんでした。

javax.swing.DefaultSingleSelectionModel.getListeners は変換されませんでした。

javax.swing.DefaultSingleSelectionModel.listenerList は変換されませんでした。

javax.swing.DefaultSingleSelectionModel.removeChangeListener は変換されませんでした。

javax.swing.DesktopManager は変換されませんでした。

javax.swing.DesktopManager.deiconifyFrame は変換されませんでした。

javax.swing.DesktopManager.iconifyFrame は変換されませんでした。

javax.swing.DesktopManager.maximizeFrame は変換されませんでした。

javax.swing.DesktopManager.minimizeFrame は変換されませんでした。

javax.swing.FocusManager は変換されませんでした。

javax.swing.GrayFilter は変換されませんでした。

javax.swing.GrayFilter.createDisabledImage は変換されませんでした。

javax.swing.Icon.paintIcon は変換されませんでした。

javax.swing.ImageIcon は変換されませんでした。

javax.swing.ImageIcon.component は変換されませんでした。

javax.swing.ImageIcon.getAccessibleContext は変換されませんでした。

javax.swing.ImageIcon.getDescription は変換されませんでした。

javax.swing.ImageIcon.getImageLoadStatus は変換されませんでした。

javax.swing.ImageIcon.getImageObserver は変換されませんでした。

javax.swing.ImageIcon.ImageIcon は変換されませんでした。

javax.swing.ImageIcon.ImageIcon(byte[]) は変換されませんでした。

javax.swing.ImageIcon.ImageIcon(byte[], String) は変換されませんでした。

javax.swing.ImageIcon.ImageIcon(Image) は変換されませんでした。

javax.swing.ImageIcon.ImageIcon(Image, String) は変換されませんでした。

javax.swing.ImageIcon.ImageIcon(String) は変換されませんでした。

javax.swing.ImageIcon.ImageIcon(String, String) は変換されませんでした。

javax.swing.ImageIcon.ImageIcon(URL) は変換されませんでした。

javax.swing.ImageIcon.ImageIcon(URL, String) は変換されませんでした。

javax.swing.ImageIcon.loadImage は変換されませんでした。

javax.swing.ImageIcon.paintIcon は変換されませんでした。

javax.swing.ImageIcon.setDescription は変換されませんでした。

javax.swing.ImageIcon.setImageObserver は変換されませんでした。

javax.swing.ImageIcon.tracker は変換されませんでした。

javax.swing.InputMap は変換されませんでした。

javax.swing.InputMap.allKeys は変換されませんでした。

javax.swing.InputMap.get は変換されませんでした。

javax.swing.InputMap.getParent は変換されませんでした。

javax.swing.InputMap.keys は変換されませんでした。

javax.swing.InputMap.setParent は変換されませんでした。

javax.swing.InputVerifier は変換されませんでした。

javax.swing.JApplet は変換されませんでした。

javax.swing.JApplet.accessibleContext は変換されませんでした。

javax.swing.JApplet.addImpl は変換されませんでした。

javax.swing.JApplet.createRootPane は変換されませんでした。

javax.swing.JApplet.getGlassPane は変換されませんでした。

javax.swing.JApplet.getJMenuBar は変換されませんでした。

javax.swing.JApplet.getLayeredPane は変換されませんでした。

javax.swing.JApplet.getRootPane は変換されませんでした。

javax.swing.JApplet.isRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JApplet.paramString は変換されませんでした。

javax.swing.JApplet.rootPane は変換されませんでした。

javax.swing.JApplet.rootPaneCheckingEnabled は変換されませんでした。

javax.swing.JApplet.setContentPane は変換されませんでした。

javax.swing.JApplet.setGlassPane は変換されませんでした。

javax.swing.JApplet.setJMenuBar は変換されませんでした。

javax.swing.JApplet.setLayeredPane は変換されませんでした。

javax.swing.JApplet.setRootPane は変換されませんでした。

javax.swing.JApplet.setRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JButton.configurePropertiesFromAction は変換されませんでした。

javax.swing.JButton.isDefaultButton は変換されませんでした。

javax.swing.JButton.isDefaultCapable は変換されませんでした。

javax.swing.JButton.JButton は変換されませんでした。

javax.swing.JButton.paramString は変換されませんでした。

javax.swing.JButton.setDefaultCapable は変換されませんでした。

javax.swing.JButton.updateUI は変換されませんでした。

javax.swing.JCheckBox.configurePropertiesFromAction は変換されませんでした。

javax.swing.JCheckBox.createActionPropertyChangeListener は変換されませんでした。

javax.swing.JCheckBox.JCheckBox は変換されませんでした。

javax.swing.JCheckBox.updateUI は変換されませんでした。

javax.swing.JCheckBoxMenuItem.getAccessibleContext は変換されませんでした。

javax.swing.JCheckBoxMenuItem.getUIClassID は変換されませんでした。

javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(Action) は変換されませんでした。

javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(Icon) は変換されませんでした。

javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(String, boolean) は変換されませんでした。

javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(String, Icon) は変換されませんでした。

javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(String, Icon, boolean) は変換されませんでした。

javax.swing.JCheckBoxMenuItem.requestFocus は変換されませんでした。

javax.swing.JColorChooser.accessibleContext は変換されませんでした。

javax.swing.JColorChooser.addChooserPanel は変換されませんでした。

javax.swing.JColorChooser.CHOOSER_PANELS_PROPERTY は変換されませんでした。

javax.swing.JColorChooser.createDialog は変換されませんでした。

javax.swing.JColorChooser.getAccessibleContext は変換されませんでした。

javax.swing.JColorChooser.getChooserPanels は変換されませんでした。

javax.swing.JColorChooser.getPreviewPanel は変換されませんでした。

javax.swing.JColorChooser.getSelectionModel は変換されませんでした。

javax.swing.JColorChooser.getUI は変換されませんでした。

javax.swing.JColorChooser.getUIClassID は変換されませんでした。

javax.swing.JColorChooser.JColorChooser は変換されませんでした。

javax.swing.JColorChooser.PREVIEW_PANEL_PROPERTY は変換されませんでした。

javax.swing.JColorChooser.removeChooserPanel は変換されませんでした。

javax.swing.JColorChooser.SELECTION_MODEL_PROPERTY は変換されませんでした。

javax.swing.JColorChooser.setChooserPanels は変換されませんでした。

javax.swing.JColorChooser.setPreviewPanel は変換されませんでした。

javax.swing.JColorChooser.setSelectionModel は変換されませんでした。

javax.swing.JColorChooser.setUI は変換されませんでした。

javax.swing.JColorChooser.showDialog は変換されませんでした。

javax.swing.JColorChooser.updateUI は変換されませんでした。

javax.swing.JComboBox.actionCommand は変換されませんでした。

javax.swing.JComboBox.actionPerformed は変換されませんでした。

javax.swing.JComboBox.configureEditor は変換されませんでした。

javax.swing.JComboBox.configurePropertiesFromAction は変換されませんでした。

javax.swing.JComboBox.contentsChanged は変換されませんでした。

javax.swing.JComboBox.createActionPropertyChangeListener は変換されませんでした。

javax.swing.JComboBox.createDefaultKeySelectionManager は変換されませんでした。

javax.swing.JComboBox.editor は変換されませんでした。

javax.swing.JComboBox.getAction は変換されませんでした。

javax.swing.JComboBox.getActionCommand は変換されませんでした。

javax.swing.JComboBox.getEditor は変換されませんでした。

javax.swing.JComboBox.getKeySelectionManager は変換されませんでした。

javax.swing.JComboBox.getRenderer は変換されませんでした。

javax.swing.JComboBox.getUI は変換されませんでした。

javax.swing.JComboBox.getUIClassID は変換されませんでした。

javax.swing.JComboBox.installAncestorListener は変換されませんでした。

javax.swing.JComboBox.intervalAdded は変換されませんでした。

javax.swing.JComboBox.intervalRemoved は変換されませんでした。

javax.swing.JComboBox.isLightWeightPopupEnabled は変換されませんでした。

javax.swing.JComboBox.JComboBox は変換されませんでした。

javax.swing.JComboBox.keySelectionManager は変換されませんでした。

javax.swing.JComboBox.KeySelectionManager は変換されませんでした。

javax.swing.JComboBox.lightWeightPopupEnabled は変換されませんでした。

javax.swing.JComboBox.renderer は変換されませんでした。

javax.swing.JComboBox.selectedItemReminder は変換されませんでした。

javax.swing.JComboBox.selectWithKeyChar は変換されませんでした。

javax.swing.JComboBox.setActionCommand は変換されませんでした。

javax.swing.JComboBox.setEditor は変換されませんでした。

javax.swing.JComboBox.setKeySelectionManager は変換されませんでした。

javax.swing.JComboBox.setLightWeightPopupEnabled は変換されませんでした。

javax.swing.JComboBox.setModel は変換されませんでした。

javax.swing.JComboBox.setRenderer は変換されませんでした。

javax.swing.JComboBox.setUI は変換されませんでした。

javax.swing.JComboBox.updateUI は変換されませんでした。

javax.swing.JComponent.accessibleContext は変換されませんでした。

javax.swing.JComponent.AccessibleJComponent は変換されませんでした。

javax.swing.JComponent.addAncestorListener は変換されませんでした。

javax.swing.JComponent.addNotify は変換されませんでした。

javax.swing.JComponent.addPropertyChangeListener は変換されませんでした。

javax.swing.JComponent.addVetoableChangeListener は変換されませんでした。

javax.swing.JComponent.computeVisibleRect は変換されませんでした。

javax.swing.JComponent.createToolTip は変換されませんでした。

javax.swing.JComponent.disable は変換されませんでした。

javax.swing.JComponent.firePropertyChange は変換されませんでした。

javax.swing.JComponent.fireVetoableChange は変換されませんでした。

javax.swing.JComponent.getActionForKeyStroke は変換されませんでした。

javax.swing.JComponent.getActionMap は変換されませんでした。

javax.swing.JComponent.getAlignmentX は変換されませんでした。

javax.swing.JComponent.getAlignmentY は変換されませんでした。

javax.swing.JComponent.getAutoscrolls は変換されませんでした。

javax.swing.JComponent.getBorder は変換されませんでした。

javax.swing.JComponent.getClientProperty は変換されませんでした。

javax.swing.JComponent.getConditionForKeyStroke は変換されませんでした。

javax.swing.JComponent.getDebugGraphicsOptions は変換されませんでした。

javax.swing.JComponent.getInputMap は変換されませんでした。

javax.swing.JComponent.getInputVerifier は変換されませんでした。

javax.swing.JComponent.getListeners は変換されませんでした。

javax.swing.JComponent.getLocation は変換されませんでした。

javax.swing.JComponent.getMaximumSize は変換されませんでした。

javax.swing.JComponent.getMinimumSize は変換されませんでした。

javax.swing.JComponent.getNextFocusableComponent は変換されませんでした。

javax.swing.JComponent.getPreferredSize は変換されませんでした。

javax.swing.JComponent.getRegisteredKeyStrokes は変換されませんでした。

javax.swing.JComponent.getRootPane は変換されませんでした。

javax.swing.JComponent.getSize は変換されませんでした。

javax.swing.JComponent.getToolTipLocation は変換されませんでした。

javax.swing.JComponent.getToolTipText は変換されませんでした。

javax.swing.JComponent.getTopLevelAncestor は変換されませんでした。

javax.swing.JComponent.getVerifyInputWhenFocusTarget は変換されませんでした。

javax.swing.JComponent.getVisibleRect は変換されませんでした。

javax.swing.JComponent.isDoubleBuffered は変換されませんでした。

javax.swing.JComponent.isFocusCycleRoot は変換されませんでした。

javax.swing.JComponent.isLightweightComponent は変換されませんでした。

javax.swing.JComponent.isManagingFocus は変換されませんでした。

javax.swing.JComponent.isMaximumSizeSet は変換されませんでした。

javax.swing.JComponent.isMinimumSizeSet は変換されませんでした。

javax.swing.JComponent.isOpaque は変換されませんでした。

javax.swing.JComponent.isOptimizedDrawingEnabled は変換されませんでした。

javax.swing.JComponent.isPaintingTile は変換されませんでした。

javax.swing.JComponent.isPreferredSizeSet は変換されませんでした。

javax.swing.JComponent.isValidateRoot は変換されませんでした。

javax.swing.JComponent.listenerList は変換されませんでした。

javax.swing.JComponent.paint は変換されませんでした。

javax.swing.JComponent.paintBorder は変換されませんでした。

javax.swing.JComponent.paintChildren は変換されませんでした。

javax.swing.JComponent.paintComponent は変換されませんでした。

javax.swing.JComponent.paintImmediately は変換されませんでした。

javax.swing.JComponent.print は変換されませんでした。

javax.swing.JComponent.printAll は変換されませんでした。

javax.swing.JComponent.printBorder は変換されませんでした。

javax.swing.JComponent.printChildren は変換されませんでした。

javax.swing.JComponent.printComponent は変換されませんでした。

javax.swing.JComponent.processKeyBinding は変換されませんでした。

javax.swing.JComponent.putClientProperty は変換されませんでした。

javax.swing.JComponent.registerKeyboardAction は変換されませんでした。

javax.swing.JComponent.removeAncestorListener は変換されませんでした。

javax.swing.JComponent.removePropertyChangeListener は変換されませんでした。

javax.swing.JComponent.removeVetoableChangeListener は変換されませんでした。

javax.swing.JComponent.repaint は変換されませんでした。

javax.swing.JComponent.requestDefaultFocus は変換されませんでした。

javax.swing.JComponent.requestFocus は変換されませんでした。

javax.swing.JComponent.resetKeyboardActions は変換されませんでした。

javax.swing.JComponent.scrollRectToVisible は変換されませんでした。

javax.swing.JComponent.setActionMap は変換されませんでした。

javax.swing.JComponent.setAlignmentX は変換されませんでした。

javax.swing.JComponent.setAlignmentY は変換されませんでした。

javax.swing.JComponent.setAutoscrolls は変換されませんでした。

javax.swing.JComponent.setBorder は変換されませんでした。

javax.swing.JComponent.setDebugGraphicsOptions は変換されませんでした。

javax.swing.JComponent.setDoubleBuffered は変換されませんでした。

javax.swing.JComponent.setEnabled は変換されませんでした。

javax.swing.JComponent.setInputMap は変換されませんでした。

javax.swing.JComponent.setInputVerifier は変換されませんでした。

javax.swing.JComponent.setMaximumSize は変換されませんでした。

javax.swing.JComponent.setMinimumSize は変換されませんでした。

javax.swing.JComponent.setNextFocusableComponent は変換されませんでした。

javax.swing.JComponent.setOpaque は変換されませんでした。

javax.swing.JComponent.setPreferredSize は変換されませんでした。

javax.swing.JComponent.setRequestFocusEnabled は変換されませんでした。

javax.swing.JComponent.setToolTipText は変換されませんでした。

javax.swing.JComponent.setUI は変換されませんでした。

javax.swing.JComponent.setVerifyInputWhenFocusTarget は変換されませんでした。

javax.swing.JComponent.setVisible は変換されませんでした。

javax.swing.JComponent.TOOL_TIP_TEXT_KEY は変換されませんでした。

javax.swing.JComponent.ui は変換されませんでした。

javax.swing.JComponent.unregisterKeyboardAction は変換されませんでした。

javax.swing.JComponent.updateUI は変換されませんでした。

javax.swing.JDesktopPane は変換されませんでした。

javax.swing.JDesktopPane.<DragMode> は変換されませんでした。

javax.swing.JDesktopPane.getDesktopManager は変換されませんでした。

javax.swing.JDesktopPane.getDragMode は変換されませんでした。

javax.swing.JDesktopPane.getUI は変換されませんでした。

javax.swing.JDesktopPane.getUIClassID は変換されませんでした。

javax.swing.JDesktopPane.isOpaque は変換されませんでした。

javax.swing.JDesktopPane.JDesktopPane は変換されませんでした。

javax.swing.JDesktopPane.setDesktopManager は変換されませんでした。

javax.swing.JDesktopPane.setDragMode は変換されませんでした。

javax.swing.JDesktopPane.setSelectedFrame は変換されませんでした。

javax.swing.JDesktopPane.setUI は変換されませんでした。

javax.swing.JDesktopPane.updateUI は変換されませんでした。

javax.swing.JDialog.accessibleContext は変換されませんでした。

javax.swing.JDialog.addImpl は変換されませんでした。

javax.swing.JDialog.dialogInit は変換されませんでした。

javax.swing.JDialog.getDefaultCloseOperation は変換されませんでした。

javax.swing.JDialog.getGlassPane は変換されませんでした。

javax.swing.JDialog.getLayeredPane は変換されませんでした。

javax.swing.JDialog.getRootPane は変換されませんでした。

javax.swing.JDialog.isRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JDialog.JDialog は変換されませんでした。

javax.swing.JDialog.JDialog(Dialog, boolean) は変換されませんでした。

javax.swing.JDialog.JDialog(Dialog, String, boolean) は変換されませんでした。

javax.swing.JDialog.JDialog(Frame, boolean) は変換されませんでした。

javax.swing.JDialog.JDialog(Frame, String, boolean) は変換されませんでした。

javax.swing.JDialog.rootPane は変換されませんでした。

javax.swing.JDialog.rootPaneCheckingEnabled は変換されませんでした。

javax.swing.JDialog.setContentPane は変換されませんでした。

javax.swing.JDialog.setDefaultCloseOperation は変換されませんでした。

javax.swing.JDialog.setGlassPane は変換されませんでした。

javax.swing.JDialog.setLayeredPane は変換されませんでした。

javax.swing.JDialog.setLocationRelativeTo は変換されませんでした。

javax.swing.JDialog.setRootPane は変換されませんでした。

javax.swing.JDialog.setRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JEditorPane は変換されませんでした。

javax.swing.JEditorPane.createDefaultEditorKit は変換されませんでした。

javax.swing.JEditorPane.createEditorKitForContentType は変換されませんでした。

javax.swing.JEditorPane.fireHyperlinkUpdate は変換されませんでした。

javax.swing.JEditorPane.getContentType は変換されませんでした。

javax.swing.JEditorPane.getEditorKit は変換されませんでした。

javax.swing.JEditorPane.getEditorKitClassNameForContentType は変換されませんでした。

javax.swing.JEditorPane.getEditorKitForContentType は変換されませんでした。

javax.swing.JEditorPane.getPage は変換されませんでした。

javax.swing.JEditorPane.getPreferredSize は変換されませんでした。

javax.swing.JEditorPane.getScrollableTracksViewportHeight は変換されませんでした。

javax.swing.JEditorPane.getScrollableTracksViewportWidth は変換されませんでした。

javax.swing.JEditorPane.getStream は変換されませんでした。

javax.swing.JEditorPane.getUIClassID は変換されませんでした。

javax.swing.JEditorPane.isFocusCycleRoot は変換されませんでした。

javax.swing.JEditorPane.JEditorPane(String) は変換されませんでした。

javax.swing.JEditorPane.JEditorPane(String, String) は変換されませんでした。

javax.swing.JEditorPane.JEditorPane(URL) は変換されませんでした。

javax.swing.JEditorPane.read は変換されませんでした。

javax.swing.JEditorPane.registerEditorKitForContentType(String, String) は変換されませんでした。

javax.swing.JEditorPane.registerEditorKitForContentType(String, String, ClassLoader) は変換されませんでした。

javax.swing.JEditorPane.scrollToReference は変換されませんでした。

javax.swing.JEditorPane.setContentType は変換されませんでした。

javax.swing.JEditorPane.setEditorKit は変換されませんでした。

javax.swing.JEditorPane.setEditorKitForContentType は変換されませんでした。

javax.swing.JEditorPane.setPage は変換されませんでした。

javax.swing.JFileChooser.accept は変換されませんでした。

javax.swing.JFileChooser.accessibleContext は変換されませんでした。

javax.swing.JFileChooser.addChoosableFileFilter は変換されませんでした。

javax.swing.JFileChooser.approveSelection は変換されませんでした。

javax.swing.JFileChooser.cancelSelection は変換されませんでした。

javax.swing.JFileChooser.changeToParentDirectory は変換されませんでした。

javax.swing.JFileChooser.ensureFileIsVisible は変換されませんでした。

javax.swing.JFileChooser.fireActionPerformed は変換されませんでした。

javax.swing.JFileChooser.getAcceptAllFileFilter は変換されませんでした。

javax.swing.JFileChooser.getAccessibleContext は変換されませんでした。

javax.swing.JFileChooser.getAccessory は変換されませんでした。

javax.swing.JFileChooser.getApproveButtonMnemonic は変換されませんでした。

javax.swing.JFileChooser.getApproveButtonText は変換されませんでした。

javax.swing.JFileChooser.getApproveButtonToolTipText は変換されませんでした。

javax.swing.JFileChooser.getChoosableFileFilters は変換されませんでした。

javax.swing.JFileChooser.getCurrentDirectory は変換されませんでした。

javax.swing.JFileChooser.getDescription は変換されませんでした。

javax.swing.JFileChooser.getFileFilter は変換されませんでした。

javax.swing.JFileChooser.getFileSelectionMode は変換されませんでした。

javax.swing.JFileChooser.getFileSystemView は変換されませんでした。

javax.swing.JFileChooser.getFileView は変換されませんでした。

javax.swing.JFileChooser.getIcon は変換されませんでした。

javax.swing.JFileChooser.getSelectedFiles は変換されませんでした。

javax.swing.JFileChooser.getTypeDescription は変換されませんでした。

javax.swing.JFileChooser.getUI は変換されませんでした。

javax.swing.JFileChooser.isAcceptAllFileFilterUsed は変換されませんでした。

javax.swing.JFileChooser.isFileHidingEnabled は変換されませんでした。

javax.swing.JFileChooser.isFileSelectionEnabled は変換されませんでした。

javax.swing.JFileChooser.isTraversable は変換されませんでした。

javax.swing.JFileChooser.JFileChooser(File, FileSystemView) は変換できませんでした。

javax.swing.JFileChooser.JFileChooser(FileSystemView) は変換できませんでした。

javax.swing.JFileChooser.JFileChooser(String, FileSystemView) は変換できませんでした。

javax.swing.JFileChooser.removeChoosableFileFilter は変換されませんでした。

javax.swing.JFileChooser.rescanCurrentDirectory は変換されませんでした。

javax.swing.JFileChooser.resetChoosableFileFilters は変換されませんでした。

javax.swing.JFileChooser.setAcceptAllFileFilterUsed は変換されませんでした。

javax.swing.JFileChooser.setAccessory は変換されませんでした。

javax.swing.JFileChooser.setApproveButtonMnemonic は変換されませんでした。

javax.swing.JFileChooser.setApproveButtonText は変換されませんでした。

javax.swing.JFileChooser.setApproveButtonToolTipText は変換されませんでした。

javax.swing.JFileChooser.setControlButtonsAreShown は変換されませんでした。

javax.swing.JFileChooser.setCurrentDirectory は変換されませんでした。

javax.swing.JFileChooser.setDialogType は変換されませんでした。

javax.swing.JFileChooser.setFileFilter は変換されませんでした。

javax.swing.JFileChooser.setFileHidingEnabled は変換されませんでした。

javax.swing.JFileChooser.setFileSelectionMode は変換されませんでした。

javax.swing.JFileChooser.setFileSystemView は変換されませんでした。

javax.swing.JFileChooser.setFileView は変換されませんでした。

javax.swing.JFileChooser.setSelectedFiles は変換されませんでした。

javax.swing.JFileChooser.setup は変換されませんでした。

javax.swing.JFileChooser.showDialog は変換されませんでした。

javax.swing.JFileChooser.showSaveDialog は変換されませんでした。

javax.swing.JFileChooser.updateUI は変換されませんでした。

javax.swing.JFrame.EXIT_ON_CLOSE は変換されませんでした。

javax.swing.JFrame.frameInit は変換されませんでした。

javax.swing.JFrame.getContentPane は変換されませんでした。

javax.swing.JFrame.getDefaultCloseOperation は変換されませんでした。

javax.swing.JFrame.getGlassPane は変換されませんでした。

javax.swing.JFrame.getLayeredPane は変換されませんでした。

javax.swing.JFrame.isRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JFrame.rootPane は変換されませんでした。

javax.swing.JFrame.rootPaneCheckingEnabled は変換されませんでした。

javax.swing.JFrame.setContentPane は変換されませんでした。

javax.swing.JFrame.setDefaultCloseOperation は変換されませんでした。

javax.swing.JFrame.setGlassPane は変換されませんでした。

javax.swing.JFrame.setLayeredPane は変換されませんでした。

javax.swing.JFrame.setRootPane は変換されませんでした。

javax.swing.JFrame.setRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JInternalFrame は変換されませんでした。

javax.swing.JInternalFrame.<PropertyName> は変換されませんでした。

javax.swing.JInternalFrame.closable は変換されませんでした。

javax.swing.JInternalFrame.desktopIcon は変換されませんでした。

javax.swing.JInternalFrame.doDefaultCloseAction は変換されませんでした。

javax.swing.JInternalFrame.fireInternalFrameEvent は変換されませんでした。

javax.swing.JInternalFrame.getDefaultCloseOperation は変換されませんでした。

javax.swing.JInternalFrame.getDesktopIcon は変換されませんでした。

javax.swing.JInternalFrame.getGlassPane は変換されませんでした。

javax.swing.JInternalFrame.getLayer は変換されませんでした。

javax.swing.JInternalFrame.getLayeredPane は変換されませんでした。

javax.swing.JInternalFrame.getNormalBounds は変換されませんでした。

javax.swing.JInternalFrame.getUI は変換されませんでした。

javax.swing.JInternalFrame.getUIClassID は変換されませんでした。

javax.swing.JInternalFrame.iconable は変換されませんでした。

javax.swing.JInternalFrame.isClosable は変換されませんでした。

javax.swing.JInternalFrame.isClosed は変換されませんでした。

javax.swing.JInternalFrame.isIcon は変換されませんでした。

javax.swing.JInternalFrame.isMaximum は変換されませんでした。

javax.swing.JInternalFrame.isRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JInternalFrame.isSelected は変換されませんでした。

javax.swing.JInternalFrame.JDesktopIcon は変換されませんでした。

javax.swing.JInternalFrame.maximizable は変換されませんでした。

javax.swing.JInternalFrame.moveToBack は変換されませんでした。

javax.swing.JInternalFrame.moveToFront は変換されませんでした。

javax.swing.JInternalFrame.pack は変換されませんでした。

javax.swing.JInternalFrame.paintComponent は変換されませんでした。

javax.swing.JInternalFrame.paramString は変換されませんでした。

javax.swing.JInternalFrame.resizable は変換されませんでした。

javax.swing.JInternalFrame.restoreSubcomponentFocus は変換されませんでした。

javax.swing.JInternalFrame.rootPane は変換されませんでした。

javax.swing.JInternalFrame.rootPaneCheckingEnabled は変換されませんでした。

javax.swing.JInternalFrame.setClosable は変換されませんでした。

javax.swing.JInternalFrame.setClosed は変換されませんでした。

javax.swing.JInternalFrame.setContentPane は変換されませんでした。

javax.swing.JInternalFrame.setDefaultCloseOperation は変換されませんでした。

javax.swing.JInternalFrame.setDesktopIcon は変換されませんでした。

javax.swing.JInternalFrame.setFrameIcon は変換されませんでした。

javax.swing.JInternalFrame.setGlassPane は変換されませんでした。

javax.swing.JInternalFrame.setLayer は変換されませんでした。

javax.swing.JInternalFrame.setLayeredPane は変換されませんでした。

javax.swing.JInternalFrame.setNormalBounds は変換されませんでした。

javax.swing.JInternalFrame.setRootPane は変換されませんでした。

javax.swing.JInternalFrame.setRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JInternalFrame.setSelected は変換されませんでした。

javax.swing.JInternalFrame.setUI は変換されませんでした。

javax.swing.JInternalFrame.updateUI は変換されませんでした。

javax.swing.JLabel.checkHorizontalKey は変換されませんでした。

javax.swing.JLabel.checkVerticalKey は変換されませんでした。

javax.swing.JLabel.getDisabledIcon は変換されませんでした。

javax.swing.JLabel.getDisplayedMnemonic は変換されませんでした。

javax.swing.JLabel.getIconTextGap は変換されませんでした。

javax.swing.JLabel.getLabelFor は変換されませんでした。

javax.swing.JLabel.getUI は変換されませんでした。

javax.swing.JLabel.getUIClassID は変換されませんでした。

javax.swing.JLabel.imageUpdate は変換されませんでした。

javax.swing.JLabel.labelFor は変換されませんでした。

javax.swing.JLabel.setDisabledIcon は変換されませんでした。

javax.swing.JLabel.setHorizontalAlignment は変換されませんでした。

javax.swing.JLabel.setHorizontalTextPosition は変換されませんでした。

javax.swing.JLabel.setIconTextGap は変換されませんでした。

javax.swing.JLabel.setLabelFor は変換されませんでした。

javax.swing.JLabel.setUI は変換されませんでした。

javax.swing.JLabel.setVerticalAlignment は変換されませんでした。

javax.swing.JLabel.setVerticalTextPosition は変換されませんでした。

javax.swing.JLabel.updateUI は変換されませんでした。

javax.swing.JLayeredPane は変換されませんでした。

javax.swing.JLayeredPane.addImpl は変換されませんでした。

javax.swing.JLayeredPane.getComponentCountInLayer は変換されませんでした。

javax.swing.JLayeredPane.getComponentsInLayer は変換されませんでした。

javax.swing.JLayeredPane.getComponentToLayer は変換されませんでした。

javax.swing.JLayeredPane.getIndexOf は変換されませんでした。

javax.swing.JLayeredPane.getLayer は変換されませんでした。

javax.swing.JLayeredPane.getLayeredPaneAbove は変換されませんでした。

javax.swing.JLayeredPane.getObjectForLayer は変換されませんでした。

javax.swing.JLayeredPane.getPosition は変換されませんでした。

javax.swing.JLayeredPane.highestLayer は変換されませんでした。

javax.swing.JLayeredPane.insertIndexForLayer は変換されませんでした。

javax.swing.JLayeredPane.isOptimizedDrawingEnabled は変換されませんでした。

javax.swing.JLayeredPane.JLayeredPane は変換されませんでした。

javax.swing.JLayeredPane.LAYER_PROPERTY は変換されませんでした。

javax.swing.JLayeredPane.lowestLayer は変換されませんでした。

javax.swing.JLayeredPane.moveToBack は変換されませんでした。

javax.swing.JLayeredPane.moveToFront は変換されませんでした。

javax.swing.JLayeredPane.paramString は変換されませんでした。

javax.swing.JLayeredPane.putLayer は変換されませんでした。

javax.swing.JLayeredPane.remove は変換されませんでした。

javax.swing.JLayeredPane.setLayer は変換されませんでした。

javax.swing.JLayeredPane.setPosition は変換されませんでした。

javax.swing.JList.addListSelectionListener は変換されませんでした。

javax.swing.JList.addSelectionInterval は変換されませんでした。

javax.swing.JList.createSelectionModel は変換されませんでした。

javax.swing.JList.ensureIndexIsVisible は変換されませんでした。

javax.swing.JList.fireSelectionValueChanged は変換されませんでした。

javax.swing.JList.getAnchorSelectionIndex は変換されませんでした。

javax.swing.JList.getCellBounds は変換されませんでした。

javax.swing.JList.getCellRenderer は変換されませんでした。

javax.swing.JList.getFixedCellHeight は変換されませんでした。

javax.swing.JList.getFixedCellWidth は変換されませんでした。

javax.swing.JList.getLastVisibleIndex は変換されませんでした。

javax.swing.JList.getLeadSelectionIndex は変換されませんでした。

javax.swing.JList.getMaxSelectionIndex は変換されませんでした。

javax.swing.JList.getMinSelectionIndex は変換されませんでした。

javax.swing.JList.getModel は変換されませんでした。

javax.swing.JList.getPreferredScrollableViewportSize は変換されませんでした。

javax.swing.JList.getPrototypeCellValue は変換されませんでした。

javax.swing.JList.getScrollableBlockIncrement は変換されませんでした。

javax.swing.JList.getScrollableTracksViewportHeight は変換されませんでした。

javax.swing.JList.getScrollableTracksViewportWidth は変換されませんでした。

javax.swing.JList.getScrollableUnitIncrement は変換されませんでした。

javax.swing.JList.setSelectedIndex は変換されませんでした。

javax.swing.JList.getSelectedIndices は変換されませんでした。

javax.swing.JList.getSelectedValues は変換されませんでした。

javax.swing.JList.getSelectionBackground は変換されませんでした。

javax.swing.JList.getSelectionForeground は変換されませんでした。

javax.swing.JList.getSelectionModel は変換されませんでした。

javax.swing.JList.getUI は変換されませんでした。

javax.swing.JList.getUIClassID は変換されませんでした。

javax.swing.JList.getValueIsAdjusting は変換されませんでした。

javax.swing.JList.getVisibleRowCount は変換されませんでした。

javax.swing.JList.indexToLocation は変換されませんでした。

javax.swing.JList.isSelectedIndex は変換されませんでした。

javax.swing.JList.JList(ListModel) は変換されませんでした。

javax.swing.JList.JList(Object[]) は変換できませんでした。

javax.swing.JList.paramString は変換されませんでした。

javax.swing.JList.removeListSelectionListener は変換されませんでした。

javax.swing.JList.removeSelectionInterval は変換されませんでした。

javax.swing.JList.setCellRenderer は変換されませんでした。

javax.swing.JList.setFixedCellHeight は変換されませんでした。

javax.swing.JList.setFixedCellWidth は変換されませんでした。

javax.swing.JList.setListData は変換されませんでした。

javax.swing.JList.setModel は変換されませんでした。

javax.swing.JList.setPrototypeCellValue は変換されませんでした。

javax.swing.JList.setSelectedIndices は変換されませんでした。

javax.swing.JList.setSelectionBackground は変換されませんでした。

javax.swing.JList.setSelectionForeground は変換されませんでした。

javax.swing.JList.setSelectionInterval は変換されませんでした。

javax.swing.JList.setSelectionModel は変換されませんでした。

javax.swing.JList.setUI は変換されませんでした。

javax.swing.JList.setValueIsAdjusting は変換されませんでした。

javax.swing.JList.setVisibleRowCount は変換されませんでした。

javax.swing.JList.updateUI は変換されませんでした。

javax.swing.JMenu.add(Component) は変換されませんでした。

javax.swing.JMenu.add(Component, int) は変換されませんでした。

javax.swing.JMenu.add(MenuItem) は変換されませんでした。

javax.swing.JMenu.createActionChangeListener は変換されませんでした。

javax.swing.JMenu.createWinListener は変換されませんでした。

javax.swing.JMenu.doClick は変換されませんでした。

javax.swing.JMenu.fireMenuCanceled は変換されませんでした。

javax.swing.JMenu.fireMenuDeselected は変換されませんでした。

javax.swing.JMenu.fireMenuSelected は変換されませんでした。

javax.swing.JMenu.getAccessibleContext は変換されませんでした。

javax.swing.JMenu.getComponent は変換されませんでした。

javax.swing.JMenu.getDelay は変換されませんでした。

javax.swing.JMenu.getMenuComponent は変換されませんでした。

javax.swing.JMenu.getMenuComponents は変換されませんでした。

javax.swing.JMenu.getPopupMenu は変換されませんでした。

javax.swing.JMenu.getPopupMenuOrigin は変換されませんでした。

javax.swing.JMenu.getUIClassID は変換されませんでした。

javax.swing.JMenu.isMenuComponent は変換されませんでした。

javax.swing.JMenu.isPopupMenuVisible は変換されませんでした。

javax.swing.JMenu.isTearOff は変換されませんでした。

javax.swing.JMenu.isTopLevelMenu は変換されませんでした。

javax.swing.JMenu.menuSelectionChanged は変換されませんでした。

javax.swing.JMenu.popupListener は変換されませんでした。

javax.swing.JMenu.setAccelerator は変換されませんでした。

javax.swing.JMenu.setDelay は変換されませんでした。

javax.swing.JMenu.setMenuLocation は変換されませんでした。

javax.swing.JMenu.setModel は変換されませんでした。

javax.swing.JMenu.setPopupMenuVisible は変換されませんでした。

javax.swing.JMenu.updateUI は変換されませんでした。

javax.swing.JMenuBar.getAccessibleContext は変換されませんでした。

javax.swing.JMenuBar.getComponent は変換されませんでした。

javax.swing.JMenuBar.getComponentAtIndex は変換されませんでした。

javax.swing.JMenuBar.getComponentIndex は変換されませんでした。

javax.swing.JMenuBar.getHelpMenu は変換されませんでした。

javax.swing.JMenuBar.getMargin は変換されませんでした。

javax.swing.JMenuBar.getSelectionModel は変換されませんでした。

javax.swing.JMenuBar.getUI は変換されませんでした。

javax.swing.JMenuBar.getUIClassID は変換されませんでした。

javax.swing.JMenuBar.isBorderPainted は変換されませんでした。

javax.swing.JMenuBar.isManagingFocus は変換されませんでした。

javax.swing.JMenuBar.isSelected は変換されませんでした。

javax.swing.JMenuBar.menuSelectionChanged は変換されませんでした。

javax.swing.JMenuBar.paintBorder は変換されませんでした。

javax.swing.JMenuBar.processKeyBinding は変換されませんでした。

javax.swing.JMenuBar.setBorderPainted は変換されませんでした。

javax.swing.JMenuBar.setHelpMenu は変換されませんでした。

javax.swing.JMenuBar.setMargin は変換されませんでした。

javax.swing.JMenuBar.setSelected は変換されませんでした。

javax.swing.JMenuBar.setSelectionModel は変換されませんでした。

javax.swing.JMenuBar.setUI は変換されませんでした。

javax.swing.JMenuBar.updateUI は変換されませんでした。

javax.swing.JMenuItem.addMenuDragMouseListener は変換されませんでした。

javax.swing.JMenuItem.addMenuKeyListener は変換されませんでした。

javax.swing.JMenuItem.configurePropertiesFromAction は変換されませんでした。

javax.swing.JMenuItem.createActionPropertyChangeListener は変換されませんでした。

javax.swing.JMenuItem.fireMenuDragMouseDragged は変換されませんでした。

javax.swing.JMenuItem.fireMenuDragMouseEntered は変換されませんでした。

javax.swing.JMenuItem.fireMenuDragMouseExited は変換されませんでした。

javax.swing.JMenuItem.fireMenuDragMouseReleased は変換されませんでした。

javax.swing.JMenuItem.fireMenuKeyPressed は変換されませんでした。

javax.swing.JMenuItem.fireMenuKeyReleased は変換されませんでした。

javax.swing.JMenuItem.fireMenuKeyTyped は変換されませんでした。

javax.swing.JMenuItem.getAccelerator は変換されませんでした。

javax.swing.JMenuItem.getAccessibleContext は変換されませんでした。

javax.swing.JMenuItem.getComponent は変換されませんでした。

javax.swing.JMenuItem.getUIClassID は変換されませんでした。

javax.swing.JMenuItem.init は変換されませんでした。

javax.swing.JMenuItem.isArmed は変換されませんでした。

javax.swing.JMenuItem.JMenuItem は変換されませんでした。

javax.swing.JMenuItem.menuSelectionChanged は変換されませんでした。

javax.swing.JMenuItem.removeMenuDragMouseListener は変換されませんでした。

javax.swing.JMenuItem.removeMenuKeyListener は変換されませんでした。

javax.swing.JMenuItem.setAccelerator は変換されませんでした。

javax.swing.JMenuItem.setArmed は変換されませんでした。

javax.swing.JMenuItem.setUI は変換されませんでした。

javax.swing.JMenuItem.updateUI は変換されませんでした。

javax.swing.JOptionPane は変換されませんでした。

javax.swing.JOptionPane.showConfirmDialog は変換されませんでした。

javax.swing.JOptionPane.showInternalConfirmDialog は変換されませんでした。

javax.swing.JOptionPane.showInternalMessageDialog は変換されませんでした。

javax.swing.JOptionPane.showMessageDialog は変換されませんでした。

javax.swing.JPanel.JPanel は変換されませんでした。

javax.swing.JPasswordField.getUIClassID は変換されませんでした。

javax.swing.JPasswordField.JPasswordField は変換されませんでした。

javax.swing.JPasswordField.JPasswordField(Document, String, int) は変換されませんでした。

javax.swing.JPasswordField.JPasswordField(int) は変換されませんでした。

javax.swing.JPasswordField.JPasswordField(String, int) は変換されませんでした。

javax.swing.JPopupMenu.add は変換されませんでした。

javax.swing.JPopupMenu.createActionChangeListener は変換されませんでした。

javax.swing.JPopupMenu.createActionComponent は変換されませんでした。

javax.swing.JPopupMenu.firePopupMenuCanceled は変換されませんでした。

javax.swing.JPopupMenu.firePopupMenuWillBecomeInvisible は変換されませんでした。

javax.swing.JPopupMenu.firePopupMenuWillBecomeVisible は変換されませんでした。

javax.swing.JPopupMenu.getAccessibleContext は変換されませんでした。

javax.swing.JPopupMenu.getComponent は変換されませんでした。

javax.swing.JPopupMenu.getComponentAtIndex は変換されませんでした。

javax.swing.JPopupMenu.getComponentIndex は変換されませんでした。

javax.swing.JPopupMenu.getDefaultLightWeightPopupEnabled は変換されませんでした。

javax.swing.JPopupMenu.getInvoker は変換されませんでした。

javax.swing.JPopupMenu.getLabel は変換されませんでした。

javax.swing.JPopupMenu.getSelectionModel は変換されませんでした。

javax.swing.JPopupMenu.getUI は変換されませんでした。

javax.swing.JPopupMenu.getUIClassID は変換されませんでした。

javax.swing.JPopupMenu.insert は変換されませんでした。

javax.swing.JPopupMenu.isBorderPainted は変換されませんでした。

javax.swing.JPopupMenu.isLightWeightPopupEnabled は変換されませんでした。

javax.swing.JPopupMenu.isPopupTrigger は変換されませんでした。

javax.swing.JPopupMenu.isVisible は変換されませんでした。

javax.swing.JPopupMenu.menuSelectionChanged は変換されませんでした。

javax.swing.JPopupMenu.pack は変換されませんでした。

javax.swing.JPopupMenu.paintBorder は変換されませんでした。

javax.swing.JPopupMenu.Separator は変換されませんでした。

javax.swing.JPopupMenu.setBorderPainted は変換されませんでした。

javax.swing.JPopupMenu.setDefaultLightWeightPopupEnabled は変換されませんでした。

javax.swing.JPopupMenu.setLabel は変換されませんでした。

javax.swing.JPopupMenu.setLightWeightPopupEnabled は変換されませんでした。

javax.swing.JPopupMenu.setLocation は変換されませんでした。

javax.swing.JPopupMenu.setPopupSize(Dimension) は変換されませんでした。

javax.swing.JPopupMenu.setPopupSize(int, int) は変換されませんでした。

javax.swing.JPopupMenu.setSelected は変換されませんでした。

javax.swing.JPopupMenu.setSelectionModel は変換されませんでした。

javax.swing.JPopupMenu.setUI は変換されませんでした。

javax.swing.JPopupMenu.setVisible は変換されませんでした。

javax.swing.JPopupMenu.show は変換されませんでした。

javax.swing.JPopupMenu.updateUI は変換されませんでした。

javax.swing.JProgressBar.addChangeListener は変換されませんでした。

javax.swing.JProgressBar.changeEvent は変換されませんでした。

javax.swing.JProgressBar.changeListener は変換されませんでした。

javax.swing.JProgressBar.createChangeListener は変換されませんでした。

javax.swing.JProgressBar.fireStateChanged は変換されませんでした。

javax.swing.JProgressBar.getModel は変換されませんでした。

javax.swing.JProgressBar.getOrientation は変換されませんでした。

javax.swing.JProgressBar.getString は変換されませんでした。

javax.swing.JProgressBar.getUI は変換されませんでした。

javax.swing.JProgressBar.getUIClassID は変換されませんでした。

javax.swing.JProgressBar.isBorderPainted は変換されませんでした。

javax.swing.JProgressBar.isStringPainted は変換されませんでした。

javax.swing.JProgressBar.JProgressBar(Bounded RangeModel) は変換されませんでした。

javax.swing.JProgressBar.JProgressBar(int) は変換されませんでした。

javax.swing.JProgressBar.JProgressBar(int, int, int) は変換されませんでした。

javax.swing.JProgressBar.model は変換されませんでした。

javax.swing.JProgressBar.orientation は変換されませんでした。

javax.swing.JProgressBar.paintBorder は変換されませんでした。

javax.swing.JProgressBar.paintString は変換されませんでした。

javax.swing.JProgressBar.progressString は変換されませんでした。

javax.swing.JProgressBar.removeChangeListener は変換されませんでした。

javax.swing.JProgressBar.setBorderPainted は変換されませんでした。

javax.swing.JProgressBar.setModel は変換されませんでした。

javax.swing.JProgressBar.setOrientation は変換されませんでした。

javax.swing.JProgressBar.setString は変換されませんでした。

javax.swing.JProgressBar.setStringPainted は変換されませんでした。

javax.swing.JProgressBar.setUI は変換されませんでした。

javax.swing.JRadioButton.configurePropertiesFromAction は変換されませんでした。

javax.swing.JRadioButton.createActionPropertyChangeListener は変換されませんでした。

javax.swing.JRadioButton.getUIClassID は変換されませんでした。

javax.swing.JRadioButton.JRadioButton は変換されませんでした。

javax.swing.JRadioButton.updateUI は変換されませんでした。

javax.swing.JRadioButtonMenuItem.getAccessibleContext は変換されませんでした。

javax.swing.JRadioButtonMenuItem.getUIClassID は変換されませんでした。

javax.swing.JRadioButtonMenuItem.JRadioButtonMenuItem は変換されませんでした。

javax.swing.JRadioButtonMenuItem.JRadioButtonMenuItem(Action) は変換されませんでした。

javax.swing.JRadioButtonMenuItem.JRadioButtonMenuItem(Icon) は変換されませんでした。

javax.swing.JRadioButtonMenuItem.JRadioButtonMenuItem(Icon, boolean) は変換されませんでした。

javax.swing.JRadioButtonMenuItem.JRadioButtonMenuItem(String) は変換されませんでした。

javax.swing.JRadioButtonMenuItem.JRadioButtonMenuItem(String, boolean) は変換されませんでした。

javax.swing.JRadioButtonMenuItem.JRadioButtonMenuItem(String, Icon) は変換されませんでした。

javax.swing.JRadioButtonMenuItem.JRadioButtonMenuItem(String, Icon, boolean) は変換されませんでした。

javax.swing.JRadioButtonMenuItem.requestFocus は変換されませんでした。

javax.swing.JRootPane.contentPane は変換されませんでした。

javax.swing.JRootPane.createGlassPane は変換されませんでした。

javax.swing.JRootPane.createLayeredPane は変換されませんでした。

javax.swing.JRootPane.createRootLayout は変換されませんでした。

javax.swing.JRootPane.defaultButton は変換されませんでした。

javax.swing.JRootPane.defaultPressAction は変換されませんでした。

javax.swing.JRootPane.defaultReleaseAction は変換されませんでした。

javax.swing.JRootPane.getDefaultButton は変換されませんでした。

javax.swing.JRootPane.getGlassPane は変換されませんでした。

javax.swing.JRootPane.getLayeredPane は変換されませんでした。

javax.swing.JRootPane.getUI は変換されませんでした。

javax.swing.JRootPane.getUIClassID は変換されませんでした。

javax.swing.JRootPane.glassPane は変換されませんでした。

javax.swing.JRootPane.isFocusCycleRoot は変換されませんでした。

javax.swing.JRootPane.isOptimizedDrawingEnabled は変換されませんでした。

javax.swing.JRootPane.isValidateRoot は変換されませんでした。

javax.swing.JRootPane.layeredPane は変換されませんでした。

javax.swing.JRootPane.menuBar は変換されませんでした。

javax.swing.JRootPane.setDefaultButton は変換されませんでした。

javax.swing.JRootPane.setGlassPane は変換されませんでした。

javax.swing.JRootPane.setLayeredPane は変換されませんでした。

javax.swing.JRootPane.setUI は変換されませんでした。

javax.swing.JRootPane.updateUI は変換されませんでした。

javax.swing.JScrollBar は変換されませんでした。

javax.swing.JScrollBar.fireAdjustmentValueChanged は変換されませんでした。

javax.swing.JScrollBar.getBlockIncrement は変換されませんでした。

javax.swing.JScrollBar.getMaximumSize は変換されませんでした。

javax.swing.JScrollBar.getModel は変換されませんでした。

javax.swing.JScrollBar.getUI は変換されませんでした。

javax.swing.JScrollBar.getUIClassID は変換されませんでした。

javax.swing.JScrollBar.getUnitIncrement は変換されませんでした。

javax.swing.JScrollBar.getValueIsAdjusting は変換されませんでした。

javax.swing.JScrollBar.JScrollBar は変換されませんでした。

javax.swing.JScrollBar.model は変換されませんでした。

javax.swing.JScrollBar.orientation は変換されませんでした。

javax.swing.JScrollBar.setBlockIncrement は変換されませんでした。

javax.swing.JScrollBar.setModel は変換されませんでした。

javax.swing.JScrollBar.setOrientation は変換されませんでした。

javax.swing.JScrollBar.setUnitIncrement は変換されませんでした。

javax.swing.JScrollBar.setValueIsAdjusting は変換されませんでした。

javax.swing.JScrollBar.setValues は変換されませんでした。

javax.swing.JScrollBar.setVisibleAmount は変換されませんでした。

javax.swing.JScrollPane は変換されませんでした。

javax.swing.JScrollPane.columnHeader は変換されませんでした。

javax.swing.JScrollPane.createHorizontalScrollBar は変換されませんでした。

javax.swing.JScrollPane.createVerticalScrollBar は変換されませんでした。

javax.swing.JScrollPane.createViewport は変換されませんでした。

javax.swing.JScrollPane.getColumnHeader は変換されませんでした。

javax.swing.JScrollPane.getCorner は変換されませんでした。

javax.swing.JScrollPane.getHorizontalScrollBar は変換されませんでした。

javax.swing.JScrollPane.getHorizontalScrollBarPolicy は変換されませんでした。

javax.swing.JScrollPane.getRowHeader は変換されませんでした。

javax.swing.JScrollPane.getUI は変換されませんでした。

javax.swing.JScrollPane.getUIClassID は変換されませんでした。

javax.swing.JScrollPane.getVerticalScrollBar は変換されませんでした。

javax.swing.JScrollPane.getVerticalScrollBarPolicy は変換されませんでした。

javax.swing.JScrollPane.getViewport は変換されませんでした。

javax.swing.JScrollPane.getViewportBorder は変換されませんでした。

javax.swing.JScrollPane.horizontalScrollBar は変換されませんでした。

javax.swing.JScrollPane.horizontalScrollBarPolicy は変換されませんでした。

javax.swing.JScrollPane.isValidateRoot は変換されませんでした。

javax.swing.JScrollPane.JScrollPane は変換されませんでした。

javax.swing.JScrollPane.JScrollPane(Component) は変換されませんでした。

javax.swing.JScrollPane.JScrollPane(Component, int, int) は変換されませんでした。

javax.swing.JScrollPane.JScrollPane(int, int) は変換されませんでした。

javax.swing.JScrollPane.lowerLeft は変換されませんでした。

javax.swing.JScrollPane.lowerRight は変換されませんでした。

javax.swing.JScrollPane.rowHeader は変換されませんでした。

javax.swing.JScrollPane.setColumnHeader は変換されませんでした。

javax.swing.JScrollPane.setCorner は変換されませんでした。

javax.swing.JScrollPane.setHorizontalScrollBar は変換されませんでした。

javax.swing.JScrollPane.setHorizontalScrollBarPolicy は変換されませんでした。

javax.swing.JScrollPane.setRowHeader は変換されませんでした。

javax.swing.JScrollPane.setUI は変換されませんでした。

javax.swing.JScrollPane.setVerticalScrollBar は変換されませんでした。

javax.swing.JScrollPane.setVerticalScrollBarPolicy は変換されませんでした。

javax.swing.JScrollPane.setViewport は変換されませんでした。

javax.swing.JScrollPane.setViewportBorder は変換されませんでした。

javax.swing.JScrollPane.updateUI は変換されませんでした。

javax.swing.JScrollPane.upperLeft は変換されませんでした。

javax.swing.JScrollPane.upperRight は変換されませんでした。

javax.swing.JScrollPane.verticalScrollBar は変換されませんでした。

javax.swing.JScrollPane.verticalScrollBarPolicy は変換されませんでした。

javax.swing.JScrollPane.viewport は変換されませんでした。

javax.swing.JSeparator は変換されませんでした。

javax.swing.JSlider.changeEvent は変換されませんでした。

javax.swing.JSlider.changeListener は変換されませんでした。

javax.swing.JSlider.createChangeListener は変換されませんでした。

javax.swing.JSlider.createStandardLabels(int) は変換されませんでした。

javax.swing.JSlider.createStandardLabels(int, int) は変換されませんでした。

javax.swing.JSlider.fireStateChanged は変換されませんでした。

javax.swing.JSlider.getExtent は変換されませんでした。

javax.swing.JSlider.getLabelTable は変換されませんでした。

javax.swing.JSlider.getMajorTickSpacing は変換されませんでした。

javax.swing.JSlider.getModel は変換されませんでした。

javax.swing.JSlider.getUI は変換されませんでした。

javax.swing.JSlider.getUIClassID は変換されませんでした。

javax.swing.JSlider.getValueIsAdjusting は変換されませんでした。

javax.swing.JSlider.JSlider は変換されませんでした。

javax.swing.JSlider.majorTickSpacing は変換されませんでした。

javax.swing.JSlider.setExtent は変換されませんでした。

javax.swing.JSlider.setInverted は変換されませんでした。

javax.swing.JSlider.setLabelTable は変換されませんでした。

javax.swing.JSlider.setMajorTickSpacing は変換されませんでした。

javax.swing.JSlider.setModel は変換されませんでした。

javax.swing.JSlider.setPaintLabels は変換されませんでした。

javax.swing.JSlider.setPaintTrack は変換されませんでした。

javax.swing.JSlider.setSnapToTicks は変換されませんでした。

javax.swing.JSlider.setUI は変換されませんでした。

javax.swing.JSlider.setValueIsAdjusting は変換されませんでした。

javax.swing.JSlider.sliderModel は変換されませんでした。

javax.swing.JSlider.snapToTicks は変換されませんでした。

javax.swing.JSlider.updateLabelUIs は変換されませんでした。

javax.swing.JSlider.updateUI は変換されませんでした。

javax.swing.JSplitPane は変換されませんでした。

javax.swing.JSplitPane.continuousLayout は変換されませんでした。

javax.swing.JSplitPane.getResizeWeight は変換されませんでした。

javax.swing.JSplitPane.getUI は変換されませんでした。

javax.swing.JSplitPane.getUIClassID は変換されませんでした。

javax.swing.JSplitPane.isContinuousLayout は変換されませんでした。

javax.swing.JSplitPane.isOneTouchExpandable は変換されませんでした。

javax.swing.JSplitPane.isValidateRoot は変換されませんでした。

javax.swing.JSplitPane.JSplitPane(int, boolean) は変換されませんでした。

javax.swing.JSplitPane.JSplitPane(int, boolean, Component, Component) は変換されませんでした。

javax.swing.JSplitPane.oneTouchExpandable は変換されませんでした。

javax.swing.JSplitPane.paintChildren は変換されませんでした。

javax.swing.JSplitPane.resetToPreferredSizes は変換されませんでした。

javax.swing.JSplitPane.setContinuousLayout は変換されませんでした。

javax.swing.JSplitPane.setOneTouchExpandable は変換されませんでした。

javax.swing.JSplitPane.setResizeWeight は変換されませんでした。

javax.swing.JSplitPane.setUI は変換されませんでした。

javax.swing.JSplitPane.updateUI は変換されませんでした。

javax.swing.JTabbedPane.add(Component) は変換されませんでした。

javax.swing.JTabbedPane.add(Component, int) は変換されませんでした。

javax.swing.JTabbedPane.add(Component, Object) は変換されませんでした。

javax.swing.JTabbedPane.add(Component, Object, int) は変換されませんでした。

javax.swing.JTabbedPane.add(String, Component) は変換されませんでした。

javax.swing.JTabbedPane.addTab(String, Component) は変換されませんでした。

javax.swing.JTabbedPane.addTab(String, Icon, Component, String) は変換されませんでした。

javax.swing.JTabbedPane.changeEvent は変換されませんでした。

javax.swing.JTabbedPane.changeListener は変換されませんでした。

javax.swing.JTabbedPane.createChangeListener は変換されませんでした。

javax.swing.JTabbedPane.fireStateChanged は変換されませんでした。

javax.swing.JTabbedPane.getBackgroundAt は変換されませんでした。

javax.swing.JTabbedPane.getBoundsAt は変換されませんでした。

javax.swing.JTabbedPane.getDisabledIconAt は変換されませんでした。

javax.swing.JTabbedPane.getForegroundAt は変換されませんでした。

javax.swing.JTabbedPane.getIconAt は変換されませんでした。

javax.swing.JTabbedPane.getModel は変換されませんでした。

javax.swing.JTabbedPane.getTabPlacement は変換されませんでした。

javax.swing.JTabbedPane.getTabRunCount は変換されませんでした。

javax.swing.JTabbedPane.getToolTipText は変換されませんでした。

javax.swing.JTabbedPane.getUI は変換されませんでした。

javax.swing.JTabbedPane.getUIClassID は変換されませんでした。

javax.swing.JTabbedPane.indexOfComponent は変換されませんでした。

javax.swing.JTabbedPane.indexOfTab(Icon) は変換されませんでした。

javax.swing.JTabbedPane.indexOfTab(String) は変換されませんでした。

javax.swing.JTabbedPane.insertTab は変換されませんでした。

javax.swing.JTabbedPane.JTabbedPane は変換されませんでした。

javax.swing.JTabbedPane.model は変換されませんでした。

javax.swing.JTabbedPane.paramString は変換されませんでした。

javax.swing.JTabbedPane.remove は変換されませんでした。

javax.swing.JTabbedPane.setBackgroundAt は変換されませんでした。

javax.swing.JTabbedPane.setComponentAt は変換されませんでした。

javax.swing.JTabbedPane.setDisabledIconAt は変換されませんでした。

javax.swing.JTabbedPane.setForegroundAt は変換されませんでした。

javax.swing.JTabbedPane.setIconAt は変換されませんでした。

javax.swing.JTabbedPane.setModel は変換されませんでした。

javax.swing.JTabbedPane.setSelectedComponent は変換されませんでした。

javax.swing.JTabbedPane.setSelectedIndex は変換されませんでした。

javax.swing.JTabbedPane.setTabPlacement は変換されませんでした。

javax.swing.JTabbedPane.setToolTipTextAt は変換されませんでした。

javax.swing.JTabbedPane.setUI は変換されませんでした。

javax.swing.JTabbedPane.tabPlacement は変換されませんでした。

javax.swing.JTabbedPane.updateUI は変換されませんでした。

javax.swing.JTable は変換されませんでした。

javax.swing.JTable.<ResizeType> は変換されませんでした。

javax.swing.JTable.addColumnSelectionInterval は変換されませんでした。

javax.swing.JTable.addRowSelectionInterval は変換されませんでした。

javax.swing.JTable.autoCreateColumnsFromModel は変換されませんでした。

javax.swing.JTable.autoResizeMode は変換されませんでした。

javax.swing.JTable.cellEditor は変換されませんでした。

javax.swing.JTable.cellSelectionEnabled は変換されませんでした。

javax.swing.JTable.changeSelection は変換されませんでした。

javax.swing.JTable.clearSelection は変換されませんでした。

javax.swing.JTable.columnAdded は変換されませんでした。

javax.swing.JTable.columnAtPoint は変換されませんでした。

javax.swing.JTable.columnMarginChanged は変換されませんでした。

javax.swing.JTable.columnMoved は変換されませんでした。

javax.swing.JTable.columnRemoved は変換されませんでした。

javax.swing.JTable.columnSelectionChanged は変換されませんでした。

javax.swing.JTable.configureEnclosingScrollPane は変換されませんでした。

javax.swing.JTable.convertColumnIndexToModel は変換されませんでした。

javax.swing.JTable.convertColumnIndexToView は変換されませんでした。

javax.swing.JTable.createDefaultColumnModel は変換されませんでした。

javax.swing.JTable.createDefaultColumnsFromModel は変換されませんでした。

javax.swing.JTable.createDefaultEditors は変換されませんでした。

javax.swing.JTable.createDefaultRenderers は変換されませんでした。

javax.swing.JTable.createDefaultSelectionModel は変換されませんでした。

javax.swing.JTable.createDefaultTableHeader は変換されませんでした。

javax.swing.JTable.createScrollPaneForTable は変換されませんでした。

javax.swing.JTable.defaultEditorsByColumnClass は変換されませんでした。

javax.swing.JTable.defaultRenderersByColumnClass は変換されませんでした。

javax.swing.JTable.doLayout は変換されませんでした。

javax.swing.JTable.editCellAt は変換されませんでした。

javax.swing.JTable.editingCanceled は変換されませんでした。

javax.swing.JTable.editingColumn は変換されませんでした。

javax.swing.JTable.editingRow は変換されませんでした。

javax.swing.JTable.editingStopped は変換されませんでした。

javax.swing.JTable.editorComp は変換されませんでした。

javax.swing.JTable.getAutoCreateColumnsFromModel は変換されませんでした。

javax.swing.JTable.getAutoResizeMode は変換されませんでした。

javax.swing.JTable.getCellEditor は変換されませんでした。

javax.swing.JTable.getCellRect は変換されませんでした。

javax.swing.JTable.getCellRenderer は変換されませんでした。

javax.swing.JTable.getCellSelectionEnabled は変換されませんでした。

javax.swing.JTable.getColumn は変換されませんでした。

javax.swing.JTable.getColumnSelectionAllowed は変換されませんでした。

javax.swing.JTable.getDefaultEditor は変換されませんでした。

javax.swing.JTable.getDefaultRenderer は変換されませんでした。

javax.swing.JTable.getEditingColumn は変換されませんでした。

javax.swing.JTable.getEditingRow は変換されませんでした。

javax.swing.JTable.getEditorComponent は変換されませんでした。

javax.swing.JTable.getIntercellSpacing は変換されませんでした。

javax.swing.JTable.getPreferredScrollableViewportSize は変換されませんでした。

javax.swing.JTable.getRowHeight は変換されませんでした。

javax.swing.JTable.getRowHeight(int) は変換されませんでした。

javax.swing.JTable.getRowMargin は変換されませんでした。

javax.swing.JTable.getRowSelectionAllowed は変換されませんでした。

javax.swing.JTable.getScrollableBlockIncrement は変換されませんでした。

javax.swing.JTable.getScrollableTracksViewportHeight は変換されませんでした。

javax.swing.JTable.getScrollableTracksViewportWidth は変換されませんでした。

javax.swing.JTable.getScrollableUnitIncrement は変換されませんでした。

javax.swing.JTable.getSelectedColumn は変換されませんでした。

javax.swing.JTable.getSelectedColumnCount は変換されませんでした。

javax.swing.JTable.getSelectedColumns は変換されませんでした。

javax.swing.JTable.getSelectedRowCount は変換されませんでした。

javax.swing.JTable.getSelectedRows は変換されませんでした。

javax.swing.JTable.getSelectionBackground は変換されませんでした。

javax.swing.JTable.getSelectionForeground は変換されませんでした。

javax.swing.JTable.getSelectionModel は変換されませんでした。

javax.swing.JTable.getShowHorizontalLines は変換されませんでした。

javax.swing.JTable.getShowVerticalLines は変換されませんでした。

javax.swing.JTable.getTableHeader は変換されませんでした。

javax.swing.JTable.getToolTipText は変換されませんでした。

javax.swing.JTable.getUI は変換されませんでした。

javax.swing.JTable.getUIClassID は変換されませんでした。

javax.swing.JTable.gridColor は変換されませんでした。

javax.swing.JTable.initializeLocalVars は変換されませんでした。

javax.swing.JTable.isCellEditable は変換されませんでした。

javax.swing.JTable.isCellSelected は変換されませんでした。

javax.swing.JTable.isColumnSelected は変換されませんでした。

javax.swing.JTable.isEditing は変換されませんでした。

javax.swing.JTable.isFocusTraversable は変換されませんでした。

javax.swing.JTable.isManagingFocus は変換されませんでした。

javax.swing.JTable.isRowSelected は変換されませんでした。

javax.swing.JTable.JTable は変換されませんでした。

javax.swing.JTable.JTable(int, int) は変換されませんでした。

javax.swing.JTable.JTable(Object[][], Object[]) は変換されませんでした。

javax.swing.JTable.JTable(TableModel) は変換されませんでした。

javax.swing.JTable.JTable(TableModel, TableColumnMode) は変換されませんでした。

javax.swing.JTable.JTable(TableModel, TableColumnModel, ListSelectionModel) は変換されませんでした。

javax.swing.JTable.JTable(Vector, Vector) は変換されませんでした。

javax.swing.JTable.moveColumn は変換されませんでした。

javax.swing.JTable.paramString は変換されませんでした。

javax.swing.JTable.preferredViewportSize は変換されませんでした。

javax.swing.JTable.prepareEditor は変換されませんでした。

javax.swing.JTable.prepareRenderer は変換されませんでした。

javax.swing.JTable.processKeyBinding は変換されませんでした。

javax.swing.JTable.removeColumnSelectionInterval は変換されませんでした。

javax.swing.JTable.removeEditor は変換されませんでした。

javax.swing.JTable.removeRowSelectionInterval は変換されませんでした。

javax.swing.JTable.rowAtPoint は変換されませんでした。

javax.swing.JTable.rowHeight は変換されませんでした。

javax.swing.JTable.rowMargin は変換されませんでした。

javax.swing.JTable.rowSelectionAllowed は変換されませんでした。

javax.swing.JTable.selectAll は変換されませんでした。

javax.swing.JTable.selectionBackground は変換されませんでした。

javax.swing.JTable.selectionForeground は変換されませんでした。

javax.swing.JTable.selectionModel は変換されませんでした。

javax.swing.JTable.setAutoCreateColumnsFromModel は変換されませんでした。

javax.swing.JTable.setAutoResizeMode は変換されませんでした。

javax.swing.JTable.setCellEditor は変換されませんでした。

javax.swing.JTable.setCellSelectionEnabled は変換されませんでした。

javax.swing.JTable.setColumnModel は変換されませんでした。

javax.swing.JTable.setColumnSelectionAllowed は変換されませんでした。

javax.swing.JTable.setColumnSelectionInterval は変換されませんでした。

javax.swing.JTable.setComponentOrientation は変換されませんでした。

javax.swing.JTable.setDefaultEditor は変換されませんでした。

javax.swing.JTable.setDefaultRenderer は変換されませんでした。

javax.swing.JTable.setEditingColumn は変換されませんでした。

javax.swing.JTable.setEditingRow は変換されませんでした。

javax.swing.JTable.setIntercellSpacing は変換されませんでした。

javax.swing.JTable.setPreferredScrollableViewportSize は変換されませんでした。

javax.swing.JTable.setRowHeight は変換されませんでした。

javax.swing.JTable.setRowMargin は変換されませんでした。

javax.swing.JTable.setRowSelectionAllowed は変換されませんでした。

javax.swing.JTable.setRowSelectionInterval は変換されませんでした。

javax.swing.JTable.setSelectionBackground は変換されませんでした。

javax.swing.JTable.setSelectionForeground は変換されませんでした。

javax.swing.JTable.setSelectionMode は変換されませんでした。

javax.swing.JTable.setSelectionModel は変換されませんでした。

javax.swing.JTable.setShowGrid は変換されませんでした。

javax.swing.JTable.setShowHorizontalLines は変換されませんでした。

javax.swing.JTable.setShowVerticalLines は変換されませんでした。

javax.swing.JTable.setTableHeader は変換されませんでした。

javax.swing.JTable.setUI は変換されませんでした。

javax.swing.JTable.showHorizontalLines は変換されませんでした。

javax.swing.JTable.showVerticalLines は変換されませんでした。

javax.swing.JTable.sizeColumnsToFit は変換されませんでした。

javax.swing.JTable.tableChanged は変換されませんでした。

javax.swing.JTable.tableHeader は変換されませんでした。

javax.swing.JTable.unconfigureEnclosingScrollPane は変換されませんでした。

javax.swing.JTable.updateUI は変換されませんでした。

javax.swing.JTable.valueChanged は変換されませんでした。

javax.swing.JTextArea.createDefaultModel は変換されませんでした。

javax.swing.JTextArea.getColumns は変換されませんでした。

javax.swing.JTextArea.getColumnWidth は変換されませんでした。

javax.swing.JTextArea.getLineEndOffset は変換されませんでした。

javax.swing.JTextArea.getLineOfOffset は変換されませんでした。

javax.swing.JTextArea.getLineStartOffset は変換されませんでした。

javax.swing.JTextArea.getPreferredScrollableViewportSize は変換されませんでした。

javax.swing.JTextArea.getRowHeight は変換されませんでした。

javax.swing.JTextArea.getRows は変換されませんでした。

javax.swing.JTextArea.getScrollableTracksViewportWidth は変換されませんでした。

javax.swing.JTextArea.getScrollableUnitIncrement は変換されませんでした。

javax.swing.JTextArea.getTabSize は変換されませんでした。

javax.swing.JTextArea.getUIClassID は変換されませんでした。

javax.swing.JTextArea.getWrapStyleWord は変換されませんでした。

javax.swing.JTextArea.isManagingFocus は変換されませんでした。

javax.swing.JTextArea.JTextArea(Document) は変換されませんでした。

javax.swing.JTextArea.JTextArea(Document, String, int, int) は変換されませんでした。

javax.swing.JTextArea.setColumns は変換されませんでした。

javax.swing.JTextArea.setRows は変換されませんでした。

javax.swing.JTextArea.setTabSize は変換されませんでした。

javax.swing.JTextField.configurePropertiesFromAction は変換されませんでした。

javax.swing.JTextField.createActionPropertyChangeListener は変換されませんでした。

javax.swing.JTextField.createDefaultModel は変換されませんでした。

javax.swing.JTextField.fireActionPerformed は変換されませんでした。

javax.swing.JTextField.getAction は変換されませんでした。

javax.swing.JTextField.getActions は変換されませんでした。

javax.swing.JTextField.getColumns は変換されませんでした。

javax.swing.JTextField.getColumnWidth は変換されませんでした。

javax.swing.JTextField.getHorizontalVisibility は変換されませんでした。

javax.swing.JTextField.getScrollOffset は変換されませんでした。

javax.swing.JTextField.getUIClassID は変換されませんでした。

javax.swing.JTextField.isValidateRoot は変換されませんでした。

javax.swing.JTextField.JTextField(Document, String, int) は変換されませんでした。

javax.swing.JTextField.JTextField(int) は変換されませんでした。

javax.swing.JTextField.JTextField(String, int) は変換されませんでした。

javax.swing.JTextField.notifyAction は変換されませんでした。

javax.swing.JTextField.postActionEvent は変換されませんでした。

javax.swing.JTextField.scrollRectToVisible は変換されませんでした。

javax.swing.JTextField.setActionCommand は変換されませんでした。

javax.swing.JTextField.setColumns は変換されませんでした。

javax.swing.JTextField.setScrollOffset は変換されませんでした。

javax.swing.JTextPane.addStyle は変換されませんでした。

javax.swing.JTextPane.createDefaultEditorKit は変換されませんでした。

javax.swing.JTextPane.getCharacterAttributes は変換されませんでした。

javax.swing.JTextPane.getInputAttributes は変換されませんでした。

javax.swing.JTextPane.getLogicalStyle は変換されませんでした。

javax.swing.JTextPane.getParagraphAttributes は変換されませんでした。

javax.swing.JTextPane.getStyle は変換されませんでした。

javax.swing.JTextPane.getStyledDocument は変換されませんでした。

javax.swing.JTextPane.getStyledEditorKit は変換されませんでした。

javax.swing.JTextPane.getUIClassID は変換されませんでした。

javax.swing.JTextPane.insertComponent は変換されませんでした。

javax.swing.JTextPane.insertIcon は変換されませんでした。

javax.swing.JTextPane.removeStylev は変換されませんでした。

javax.swing.JTextPane.setCharacterAttributes は変換されませんでした。

javax.swing.JTextPane.setDocument は変換されませんでした。

javax.swing.JTextPane.setEditorKit は変換されませんでした。

javax.swing.JTextPane.setLogicalStyle は変換されませんでした。

javax.swing.JTextPane.setParagraphAttributes は変換されませんでした。

javax.swing.JTextPane.setStyledDocument は変換されませんでした。

javax.swing.JToggleButton.getUIClassID は変換されませんでした。

javax.swing.JToggleButton.JToggleButton は変換されませんでした。

javax.swing.JToggleButton.ToggleButtonModel は変換されませんでした。

javax.swing.JToggleButton.updateUI は変換されませんでした。

javax.swing.JToolBar.addSeparator は変換されませんでした。

javax.swing.JToolBar.addSeparator(Dimension) は変換されませんでした。

javax.swing.JToolBar.createActionChangeListener は変換されませんでした。

javax.swing.JToolBar.getComponentAtIndex は変換されませんでした。

javax.swing.JToolBar.getComponentIndex は変換されませんでした。

javax.swing.JToolBar.getMargin は変換されませんでした。

javax.swing.JToolBar.getOrientation は変換されませんでした。

javax.swing.JToolBar.getUI は変換されませんでした。

javax.swing.JToolBar.getUIClassID は変換されませんでした。

javax.swing.JToolBar.isBorderPainted は変換されませんでした。

javax.swing.JToolBar.isFloatable は変換されませんでした。

javax.swing.JToolBar.paintBorder は変換されませんでした。

javax.swing.JToolBar.Separator は変換されませんでした。

javax.swing.JToolBar.setBorderPainted は変換されませんでした。

javax.swing.JToolBar.setFloatable は変換されませんでした。

javax.swing.JToolBar.setMargin は変換されませんでした。

javax.swing.JToolBar.setOrientation は変換されませんでした。

javax.swing.JToolBar.setUI は変換されませんでした。

javax.swing.JToolBar.updateUI は変換されませんでした。

javax.swing.JToolTip.getAccessibleContext は変換されませんでした。

javax.swing.JToolTip.getComponent は変換されませんでした。

javax.swing.JToolTip.getTipText は変換されませんでした。

javax.swing.JToolTip.getUI は変換されませんでした。

javax.swing.JToolTip.getUIClassID は変換されませんでした。

javax.swing.JToolTip.paramString は変換されませんでした。

javax.swing.JToolTip.setComponent は変換されませんでした。

javax.swing.JToolTip.setTipText は変換されませんでした。

javax.swing.JToolTip.updateUI は変換されませんでした。

javax.swing.JTree は変換されませんでした。

javax.swing.JTree.<PropertyName> は変換されませんでした。

javax.swing.JTree.addSelectionInterval は変換されませんでした。

javax.swing.JTree.addSelectionPath は変換されませんでした。

javax.swing.JTree.addSelectionPaths は変換されませんでした。

javax.swing.JTree.addSelectionRow は変換されませんでした。

javax.swing.JTree.addSelectionRows は変換されませんでした。

javax.swing.JTree.cancelEditing は変換されませんでした。

javax.swing.JTree.cellEditor は変換されませんでした。

javax.swing.JTree.cellRenderer は変換されませんでした。

javax.swing.JTree.clearToggledPaths は変換されませんでした。

javax.swing.JTree.collapsePath は変換されませんでした。

javax.swing.JTree.collapseRow は変換されませんでした。

javax.swing.JTree.convertValueToText は変換されませんでした。

javax.swing.JTree.createTreeModel は変換されませんでした。

javax.swing.JTree.createTreeModelListener は変換されませんでした。

javax.swing.JTree.DynamicUtilTreeNode は変換されませんでした。

javax.swing.JTree.expandPath は変換されませんでした。

javax.swing.JTree.expandRow は変換されませんでした。

javax.swing.JTree.fireTreeCollapsed は変換されませんでした。

javax.swing.JTree.fireTreeExpanded は変換されませんでした。

javax.swing.JTree.fireTreeWillCollapse は変換されませんでした。

javax.swing.JTree.fireTreeWillExpand は変換されませんでした。

javax.swing.JTree.fireValueChanged は変換されませんでした。

javax.swing.JTree.getAnchorSelectionPath は変換されませんでした。

javax.swing.JTree.getCellEditor は変換されませんでした。

javax.swing.JTree.getCellRenderer は変換されませんでした。

javax.swing.JTree.getClosestPathForLocation は変換されませんでした。

javax.swing.JTree.getClosestRowForLocation は変換されませんでした。

javax.swing.JTree.getDefaultTreeModel は変換されませんでした。

javax.swing.JTree.getDescendantToggledPaths は変換されませんでした。

javax.swing.JTree.getEditingPath は変換されませんでした。

javax.swing.JTree.getExpandedDescendants は変換されませんでした。

javax.swing.JTree.getExpandsSelectedPaths は変換されませんでした。

javax.swing.JTree.getInvokesStopCellEditing は変換されませんでした。

javax.swing.JTree.getLeadSelectionPath は変換されませんでした。

javax.swing.JTree.getLeadSelectionRow は変換されませんでした。

javax.swing.JTree.getMaxSelectionRow は変換されませんでした。

javax.swing.JTree.getMinSelectionRow は変換されませんでした。

javax.swing.JTree.getModel は変換されませんでした。

javax.swing.JTree.getPathBetweenRows は変換されませんでした。

javax.swing.JTree.getPathBounds は変換されませんでした。

javax.swing.JTree.getPathForLocation は変換されませんでした。

javax.swing.JTree.getPathForRow は変換されませんでした。

javax.swing.JTree.getPreferredScrollableViewportSize は変換されませんでした。

javax.swing.JTree.getRowBounds は変換されませんでした。

javax.swing.JTree.getRowCount は変換されませんでした。

javax.swing.JTree.getRowForLocation は変換されませんでした。

javax.swing.JTree.getRowForPath は変換されませんでした。

javax.swing.JTree.getRowHeight は変換されませんでした。

javax.swing.JTree.getScrollableBlockIncrement は変換されませんでした。

javax.swing.JTree.getScrollableTracksViewportHeight は変換されませんでした。

javax.swing.JTree.getScrollableTracksViewportWidth は変換されませんでした。

javax.swing.JTree.getScrollableUnitIncrement は変換されませんでした。

javax.swing.JTree.getScrollsOnExpand は変換されませんでした。

javax.swing.JTree.getSelectionCount は変換されませんでした。

javax.swing.JTree.getSelectionModel は変換されませんでした。

javax.swing.JTree.getSelectionPath は変換されませんでした。

javax.swing.JTree.getSelectionPaths は変換されませんでした。

javax.swing.JTree.getSelectionRows は変換されませんでした。

javax.swing.JTree.getToggleClickCount は変換されませんでした。

javax.swing.JTree.getToolTipText は変換されませんでした。

javax.swing.JTree.getUI は変換されませんでした。

javax.swing.JTree.getUIClassID は変換されませんでした。

javax.swing.JTree.hasBeenExpanded は変換されませんでした。

javax.swing.JTree.invokesStopCellEditing は変換されませんでした。

javax.swing.JTree.isCollapsed(int) は変換されませんでした。

javax.swing.JTree.isCollapsed(TreePath) は変換されませんでした。

javax.swing.JTree.isEditing は変換されませんでした。

javax.swing.JTree.isExpanded(int) は変換されませんでした。

javax.swing.JTree.isExpanded(TreePath) は変換されませんでした。

javax.swing.JTree.isFixedRowHeight は変換されませんでした。

javax.swing.JTree.isLargeModel は変換されませんでした。

javax.swing.JTree.isPathEditable は変換されませんでした。

javax.swing.JTree.isPathSelected は変換されませんでした。

javax.swing.JTree.isRootVisible は変換されませんでした。

javax.swing.JTree.isRowSelected は変換されませんでした。

javax.swing.JTree.isVisible は変換されませんでした。

javax.swing.JTree.JTree(Hashtable) は変換されませんでした。

javax.swing.JTree.JTree(TreeNode, boolean) は変換されませんでした。

javax.swing.JTree.largeModel は変換されませんでした。

javax.swing.JTree.makeVisible は変換されませんでした。

javax.swing.JTree.removeDescendantSelectedPaths は変換されませんでした。

javax.swing.JTree.removeDescendantToggledPaths は変換されませんでした。

javax.swing.JTree.removeSelectionInterval は変換されませんでした。

javax.swing.JTree.removeSelectionPath は変換されませんでした。

javax.swing.JTree.removeSelectionPaths は変換されませんでした。

javax.swing.JTree.removeSelectionRow は変換されませんでした。

javax.swing.JTree.removeSelectionRows は変換されませんでした。

javax.swing.JTree.rootVisible は変換されませんでした。

javax.swing.JTree.scrollPathToVisible は変換されませんでした。

javax.swing.JTree.scrollRowToVisible は変換されませんでした。

javax.swing.JTree.scrollsOnExpand は変換されませんでした。

javax.swing.JTree.selectionModel は変換されませんでした。

javax.swing.JTree.selectionRedirector は変換されませんでした。

javax.swing.JTree.setAnchorSelectionPath は変換されませんでした。

javax.swing.JTree.setCellEditor は変換されませんでした。

javax.swing.JTree.setCellRenderer は変換されませんでした。

javax.swing.JTree.setExpandedState は変換されませんでした。

javax.swing.JTree.setExpandsSelectedPaths は変換されませんでした。

javax.swing.JTree.setInvokesStopCellEditing は変換されませんでした。

javax.swing.JTree.setLargeModel は変換されませんでした。

javax.swing.JTree.setLeadSelectionPath は変換されませんでした。

javax.swing.JTree.setRootVisible は変換されませんでした。

javax.swing.JTree.setRowHeight は変換されませんでした。

javax.swing.JTree.setScrollsOnExpand は変換されませんでした。

javax.swing.JTree.setSelectionInterval は変換されませんでした。

javax.swing.JTree.setSelectionModel は変換されませんでした。

javax.swing.JTree.setSelectionPath は変換されませんでした。

javax.swing.JTree.setSelectionPaths は変換されませんでした。

javax.swing.JTree.setSelectionRow は変換されませんでした。

javax.swing.JTree.setSelectionRows は変換されませんでした。

javax.swing.JTree.setToggleClickCount は変換されませんでした。

javax.swing.JTree.setUI は変換されませんでした。

javax.swing.JTree.setVisibleRowCount は変換されませんでした。

javax.swing.JTree.startEditingAtPath は変換されませんでした。

javax.swing.JTree.stopEditing は変換されませんでした。

javax.swing.JTree.toggleClickCount は変換されませんでした。

javax.swing.JTree.treeDidChange は変換されませんでした。

javax.swing.JTree.treeModel は変換されませんでした。

javax.swing.JTree.treeModelListener は変換されませんでした。

javax.swing.JTree.updateUI は変換されませんでした。

javax.swing.JTree.visibleRowCount は変換されませんでした。

javax.swing.JViewport は変換されませんでした。

javax.swing.JWindow.getGlassPane は変換されませんでした。

javax.swing.JWindow.getLayeredPane は変換されませんでした。

javax.swing.JWindow.isRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JWindow.JWindow(GraphicsConfiguration) は変換されませんでした。

javax.swing.JWindow.JWindow(GraphicsConfiguration, Window) は変換されませんでした。

javax.swing.JWindow.rootPane は変換されませんでした。

javax.swing.JWindow.rootPaneCheckingEnabled は変換されませんでした。

javax.swing.JWindow.setContentPane は変換されませんでした。

javax.swing.JWindow.setGlassPane は変換されませんでした。

javax.swing.JWindow.setLayeredPane は変換されませんでした。

javax.swing.JWindow.setRootPane は変換されませんでした。

javax.swing.JWindow.setRootPaneCheckingEnabled は変換されませんでした。

javax.swing.JWindow.windowInit は変換されませんでした。

javax.swing.KeyStroke.getKeyStroke は変換されませんでした。

javax.swing.KeyStroke.getKeyStrokeForEvent は変換されませんでした。

javax.swing.KeyStroke.isOnKeyRelease は変換されませんでした。

javax.swing.KeyStroke.KeyStroke は変換されませんでした。

javax.swing.ListCellRenderer は変換されませんでした。

javax.swing.ListModel は変換されませんでした。

javax.swing.ListModel.addListDataListener は変換されませんでした。

javax.swing.ListModel.removeListDataListener は変換されませんでした。

javax.swing.ListSelectionModel は変換されませんでした。

javax.swing.ListSelectionModel.addListSelectionListener は変換されませんでした。

javax.swing.ListSelectionModel.getValueIsAdjusting は変換されませんでした。

javax.swing.ListSelectionModel.insertIndexInterval は変換されませんでした。

javax.swing.ListSelectionModel.removeListSelectionListener は変換されませんでした。

javax.swing.ListSelectionModel.setValueIsAdjusting は変換されませんでした。

javax.swing.LookAndFeel は変換されませんでした。

javax.swing.MenuElement.menuSelectionChanged は変換されませんでした。

javax.swing.MenuSelectionManager は変換されませんでした。

javax.swing.OverlayLayout は変換されませんでした。

javax.swing.ProgressMonitor は変換されませんでした。

javax.swing.ProgressMonitorInputStream は変換されませんでした。

javax.swing.Renderer は変換されませんでした。

javax.swing.RepaintManager は変換されませんでした。

javax.swing.RootPaneContainer.getGlassPane は変換されませんでした。

javax.swing.RootPaneContainer.getLayeredPane は変換されませんでした。

javax.swing.RootPaneContainer.setContentPane は変換されませんでした。

javax.swing.RootPaneContainer.setGlassPane は変換されませんでした。

javax.swing.RootPaneContainer.setLayeredPane は変換されませんでした。

javax.swing.Scrollable は変換されませんでした。

javax.swing.ScrollPaneConstants は変換されませんでした。

javax.swing.ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED は変換されませんでした。

javax.swing.ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER は変換されませんでした。

javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED は変換されませんでした。

javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER は変換されませんでした。

javax.swing.ScrollPaneLayout は変換されませんでした。

javax.swing.ScrollPaneLayout.UIResource は変換されませんでした。

javax.swing.SingleSelectionModel は変換されませんでした。

javax.swing.SingleSelectionModel.addChangeListener は変換されませんでした。

javax.swing.SingleSelectionModel.removeChangeListener は変換されませんでした。

javax.swing.SizeRequirements は変換されませんでした。

javax.swing.SizeSequence は変換されませんでした。

javax.swing.SizeSequence.getIndex は変換されませんでした。

javax.swing.SizeSequence.getPosition は変換されませんでした。

javax.swing.SizeSequence.getSize は変換されませんでした。

javax.swing.SizeSequence.getSizes は変換されませんでした。

javax.swing.SizeSequence.insertEntries は変換されませんでした。

javax.swing.SizeSequence.removeEntries は変換されませんでした。

javax.swing.SizeSequence.setSize は変換されませんでした。

javax.swing.SizeSequence.setSizes は変換されませんでした。

javax.swing.SizeSequence.SizeSequence は変換されませんでした。

javax.swing.SwingConstants は変換されませんでした。

javax.swing.SwingUtilities は変換されませんでした。

javax.swing.SwingUtilities.getAccessibleAt は変換されませんでした。

javax.swing.SwingUtilities.getAncestorNamed は変換されませんでした。

javax.swing.SwingUtilities.getDeepestComponentAt は変換されませんでした。

javax.swing.SwingUtilities.getLocalBounds は変換されませんでした。

javax.swing.SwingUtilities.getRoot は変換されませんでした。

javax.swing.SwingUtilities.getRootPane は変換されませんでした。

javax.swing.SwingUtilities.getWindowAncestor は変換されませんでした。

javax.swing.Timer.fireActionPerformed は変換されませんでした。

javax.swing.Timer.getInitialDelay は変換されませんでした。

javax.swing.Timer.getListeners は変換されませんでした。

javax.swing.Timer.getLogTimers は変換されませんでした。

javax.swing.Timer.isCoalesce は変換されませんでした。

javax.swing.Timer.restart は変換されませんでした。

javax.swing.Timer.setCoalesce は変換されませんでした。

javax.swing.Timer.setInitialDelay は変換されませんでした。

javax.swing.Timer.setLogTimers は変換されませんでした。

javax.swing.ToolTipManager.heavyWeightPopupEnabled は変換されませんでした。

javax.swing.ToolTipManager.isLightWeightPopupEnabled は変換されませんでした。

javax.swing.ToolTipManager.lightWeightPopupEnabled は変換されませんでした。

javax.swing.ToolTipManager.mouseDragged は変換されませんでした。

javax.swing.ToolTipManager.mouseEntered は変換されませんでした。

javax.swing.ToolTipManager.mouseExited は変換されませんでした。

javax.swing.ToolTipManager.mouseMoved は変換されませんでした。

javax.swing.ToolTipManager.mousePressed は変換されませんでした。

javax.swing.ToolTipManager.registerComponent は変換されませんでした。

javax.swing.ToolTipManager.setLightWeightPopupEnabled は変換されませんでした。

javax.swing.ToolTipManager.ToolTipManager は変換されませんでした。

javax.swing.ToolTipManager.unregisterComponent は変換されませんでした。

javax.swing.UIDefaults は変換されませんでした。

javax.swing.UIDefaults.ActiveValue は変換されませんでした。

javax.swing.UIDefaults.LazyInputMap は変換されませんでした。

javax.swing.UIDefaults.LazyValue は変換されませんでした。

javax.swing.UIDefaults.ProxyLazyValue は変換されませんでした。

javax.swing.UIManager は変換されませんでした。

javax.swing.UIManager.LookAndFeelInfo は変換されませんでした。

javax.swing.UnsupportedLookAndFeelException は変換されませんでした。

javax.swing.ViewportLayout は変換されませんでした。

javax.swing.WindowConstants は変換されませんでした。

Javax.swing.beaninfo のエラー メッセージ

javax.swing.beaninfo.SwingBeanInfo は変換されませんでした。

Javax.swing.border のエラー メッセージ

javax.swing.border.AbstractBorder は変換されませんでした。

javax.swing.border.BevelBorder は変換されませんでした。

javax.swing.border.BevelBorder.<Type> は変換されませんでした。

javax.swing.border.BevelBorder.BevelBorder は変換されませんでした。

javax.swing.border.BevelBorder.bevelType は変換されませんでした。

javax.swing.border.BevelBorder.getBevelType は変換されませんでした。

javax.swing.border.BevelBorder.getBorderInsets は変換されませんでした。

javax.swing.border.BevelBorder.getHighlightInnerColor は変換されませんでした。

javax.swing.border.BevelBorder.getHighlightOuterColor は変換されませんでした。

javax.swing.border.BevelBorder.getShadowInnerColor は変換されませんでした。

javax.swing.border.BevelBorder.getShadowOuterColor は変換されませんでした。

javax.swing.border.BevelBorder.highlightInner は変換されませんでした。

javax.swing.border.BevelBorder.highlightOuter は変換されませんでした。

javax.swing.border.BevelBorder.isBorderOpaque は変換されませんでした。

javax.swing.border.BevelBorder.paintBorder は変換されませんでした。

javax.swing.border.BevelBorder.paintLoweredBevel は変換されませんでした。

javax.swing.border.BevelBorder.paintRaisedBevel は変換されませんでした。

javax.swing.border.BevelBorder.shadowInner は変換されませんでした。

javax.swing.border.BevelBorder.shadowOuter は変換されませんでした。

javax.swing.border.Border は変換されませんでした。

javax.swing.border.Border.getBorderInsets は変換されませんでした。

javax.swing.border.Border.isBorderOpaque は変換されませんでした。

javax.swing.border.Border.paintBorder は変換されませんでした。

javax.swing.border.CompoundBorder は変換されませんでした。

javax.swing.border.EmptyBorder は変換されませんでした。

javax.swing.border.EmptyBorder.bottom は変換されませんでした。

javax.swing.border.EmptyBorder.EmptyBorder は変換されませんでした。

javax.swing.border.EmptyBorder.getBorderInsets は変換されませんでした。

javax.swing.border.EmptyBorder.isBorderOpaque は変換されませんでした。

javax.swing.border.EmptyBorder.left は変換されませんでした。

javax.swing.border.EmptyBorder.paintBorder は変換されませんでした。

javax.swing.border.EmptyBorder.right は変換されませんでした。

javax.swing.border.EmptyBorder.top は変換されませんでした。

javax.swing.border.EtchedBorder は変換されませんでした。

javax.swing.border.EtchedBorder.<Type> は変換されませんでした。

javax.swing.border.EtchedBorder.EtchedBorder は変換されませんでした。

javax.swing.border.EtchedBorder.etchType は変換されませんでした。

javax.swing.border.EtchedBorder.getBorderInsets は変換されませんでした。

javax.swing.border.EtchedBorder.getEtchType は変換されませんでした。

javax.swing.border.EtchedBorder.getHighlightColor は変換されませんでした。

javax.swing.border.EtchedBorder.getShadowColor は変換されませんでした。

javax.swing.border.EtchedBorder.highlight は変換されませんでした。

javax.swing.border.EtchedBorder.isBorderOpaque は変換されませんでした。

javax.swing.border.EtchedBorder.paintBorder は変換されませんでした。

javax.swing.border.EtchedBorder.shadow は変換されませんでした。

javax.swing.border.LineBorder は変換されませんでした。

javax.swing.border.LineBorder.createBlackLineBorder は変換されませんでした。

javax.swing.border.LineBorder.createGrayLineBorder は変換されませんでした。

javax.swing.border.LineBorder.getBorderInsets は変換されませんでした。

javax.swing.border.LineBorder.getLineColor は変換されませんでした。

javax.swing.border.LineBorder.getRoundedCorners は変換されませんでした。

javax.swing.border.LineBorder.getThickness は変換されませんでした。

javax.swing.border.LineBorder.isBorderOpaque は変換されませんでした。

javax.swing.border.LineBorder.LineBorder は変換されませんでした。

javax.swing.border.LineBorder.lineColor は変換されませんでした。

javax.swing.border.LineBorder.paintBorder は変換されませんでした。

javax.swing.border.LineBorder.roundedCorners は変換されませんでした。

javax.swing.border.LineBorder.thickness は変換されませんでした。

javax.swing.border.MatteBorder は変換されませんでした。

javax.swing.border.SoftBevelBorder は変換されませんでした。

javax.swing.border.SoftBevelBorder.getBorderInsets は変換されませんでした。

javax.swing.border.SoftBevelBorder.isBorderOpaque は変換されませんでした。

javax.swing.border.SoftBevelBorder.paintBorder は変換されませんでした。

javax.swing.border.SoftBevelBorder.SoftBevelBorder は変換されませんでした。

javax.swing.border.TitledBorder は変換されませんでした。

Javax.swing.colorchooser のエラー メッセージ

javax.swing.colorchooser.AbstractColorChooserPanel は変換されませんでした。

javax.swing.colorchooser.ColorChooserComponentFactory は変換されませんでした。

javax.swing.colorchooser.ColorSelectionModel は変換されませんでした。

javax.swing.colorchooser.DefaultColorSelectionModel は変換されませんでした。

Javax.swing.event のエラー メッセージ

javax.swing.event.AncestorEvent を変換できませんでした。

javax.swing.event.CaretEvent を変換できませんでした。

javax.swing.event.DocumentEvent を変換できませんでした。

javax.swing.event.DocumentEvent.ElementChange を変換できませんでした。

javax.swing.event.DocumentEvent.EventType を変換できませんでした。

javax.swing.event.EventListenerList.getListenerCount を変換できませんでした。

javax.swing.event.EventListenerList.getListeners を変換できませんでした。

javax.swing.event.HyperlinkEvent.EventType を変換できませんでした。

javax.swing.event.HyperlinkEvent.getDescription を変換できませんでした。

javax.swing.event.HyperlinkEvent.getEventType を変換できませんでした。

javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_ACTIVATED を変換できませんでした。

javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_CLOSED を変換できませんでした。

javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_CLOSING を変換できませんでした。

javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_DEACTIVATED を変換できませんでした。

javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_DEICONIFIED を変換できませんでした。

javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_FIRST を変換できませんでした。

javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_ICONIFIED を変換できませんでした。

javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_LAST を変換できませんでした。

javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_OPENED を変換できませんでした。

javax.swing.event.ListDataEvent を変換できませんでした。

javax.swing.event.ListSelectionEvent.getFirstIndex を変換できませんでした。

javax.swing.event.ListSelectionEvent.getLastIndex を変換できませんでした。

javax.swing.event.ListSelectionEvent.getValueIsAdjusting を変換できませんでした。

javax.swing.event.MenuDragMouseEvent を変換できませんでした。

javax.swing.event.MenuKeyEvent を変換できませんでした。

javax.swing.event.SwingPropertyChangeSupport を変換できませんでした。

javax.swing.event.TableColumnModelEvent.fromIndex を変換できませんでした。

javax.swing.event.TableColumnModelEvent.getFromIndex を変換できませんでした。

javax.swing.event.TableColumnModelEvent.getToIndex を変換できませんでした。

javax.swing.event.TableColumnModelEvent.TableColumnModelEvent を変換できませんでした。

javax.swing.event.TableColumnModelEvent.toIndex を変換できませんでした。

javax.swing.event.TableModelEvent.ALL_COLUMNS を変換できませんでした。

javax.swing.event.TableModelEvent.column を変換できませんでした。

javax.swing.event.TableModelEvent.firstRow を変換できませんでした。

javax.swing.event.TableModelEvent.getColumn を変換できませんでした。

javax.swing.event.TableModelEvent.getFirstRow を変換できませんでした。

javax.swing.event.TableModelEvent.getLastRow を変換できませんでした。

javax.swing.event.TableModelEvent.HEADER_ROW を変換できませんでした。

javax.swing.event.TableModelEvent.lastRow を変換できませんでした。

javax.swing.event.TableModelEvent.TableModelEvent を変換できませんでした。

javax.swing.event.TreeExpansionEvent.getPath を変換できませんでした。

javax.swing.event.TreeExpansionEvent.path を変換できませんでした。

javax.swing.event.TreeExpansionEvent.TreeExpansionEvent を変換できませんでした。

javax.swing.event.TreeModelEvent を変換できませんでした。

javax.swing.event.TreeSelectionEvent.areNew を変換できませんでした。

javax.swing.event.TreeSelectionEvent.cloneWithSource を変換できませんでした。

javax.swing.event.TreeSelectionEvent.getNewLeadSelectionPath を変換できませんでした。

javax.swing.event.TreeSelectionEvent.getOldLeadSelectionPath を変換できませんでした。

javax.swing.event.TreeSelectionEvent.getPath を変換できませんでした。

javax.swing.event.TreeSelectionEvent.getPaths を変換できませんでした。

javax.swing.event.TreeSelectionEvent.isAddedPath を変換できませんでした。

javax.swing.event.TreeSelectionEvent.newLeadSelectionPath を変換できませんでした。

javax.swing.event.TreeSelectionEvent.oldLeadSelectionPath を変換できませんでした。

javax.swing.event.TreeSelectionEvent.paths を変換できませんでした。

javax.swing.event.TreeSelectionEvent.TreeSelectionEvent を変換できませんでした。

javax.swing.event.UndoableEditEvent を変換できませんでした。

Javax.swing.filechooser のエラー メッセージ

javax.swing.filechooser.FileFilter は変換されませんでした。

javax.swing.filechooser.FileSystemView.createNewFolder は変換されませんでした。

javax.swing.filechooser.FileSystemView.FileSystemView は変換されませんでした。

javax.swing.filechooser.FileSystemView.getFileSystemView は変換されませんでした。

javax.swing.filechooser.FileSystemView.getHomeDirectory は変換されませんでした。

javax.swing.filechooser.FileSystemView.getRoots は変換されませんでした。

javax.swing.filechooser.FileSystemView.isRoot は変換されませんでした。

javax.swing.filechooser.FileView は変換されませんでした。

Javax.swing.table のエラー メッセージ

javax.swing.table.AbstractTableModel は変換されませんでした。

javax.swing.table.AbstractTableModel.addTableModelListener は変換されませんでした。

javax.swing.table.AbstractTableModel.fireTableCellUpdated は変換されませんでした。

javax.swing.table.AbstractTableModel.fireTableChanged は変換されませんでした。

javax.swing.table.AbstractTableModel.fireTableDataChanged は変換されませんでした。

javax.swing.table.AbstractTableModel.fireTableRowsDeleted は変換されませんでした。

javax.swing.table.AbstractTableModel.fireTableRowsInserted は変換されませんでした。

javax.swing.table.AbstractTableModel.fireTableRowsUpdated は変換されませんでした。

javax.swing.table.AbstractTableModel.fireTableStructureChanged は変換されませんでした。

javax.swing.table.AbstractTableModel.getListeners は変換されませんでした。

javax.swing.table.AbstractTableModel.listenerList は変換されませんでした。

javax.swing.table.AbstractTableModel.removeTableModelListener は変換されませんでした。

javax.swing.table.DefaultTableCellRenderer は変換されませんでした。

javax.swing.table.DefaultTableCellRenderer.UIResource は変換されませんでした。

javax.swing.table.DefaultTableColumnModel は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.addColumnModelListener は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.changeEvent は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.columnMargin は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.columnSelectionAllowed は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.createSelectionModel は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.DefaultTableColumnModel は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.fireColumnAdded は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.fireColumnMarginChanged は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.fireColumnMoved は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.fireColumnRemoved は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.fireColumnSelectionChanged は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.getColumnIndexAtX は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.getColumnMargin は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.getColumnSelectionAllowed は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.getListeners は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.getSelectedColumnCount は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.getSelectedColumns は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.getSelectionModel は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.getTotalColumnWidth は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.listenerList は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.moveColumn は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.propertyChange は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.recalcWidthCache は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.removeColumnModelListener は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.selectionModel は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.setColumnMargin は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.setColumnSelectionAllowed は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.setSelectionModel は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.tableColumns は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.totalColumnWidth は変換されませんでした。

javax.swing.table.DefaultTableColumnModel.valueChanged は変換されませんでした。

javax.swing.table.DefaultTableModel は変換されませんでした。

javax.swing.table.DefaultTableModel.addColumn(Object, Object[]) は変換されませんでした。

javax.swing.table.DefaultTableModel.addColumn(Object, Vector) は変換されませんでした。

javax.swing.table.DefaultTableModel.columnIdentifiers は変換されませんでした。

javax.swing.table.DefaultTableModel.dataVector は変換されませんでした。

javax.swing.table.DefaultTableModel.DefaultTableModel(int, int) は変換されませんでした。

javax.swing.table.DefaultTableModel.DefaultTableModel(Object[], int) は変換されませんでした。

javax.swing.table.DefaultTableModel.DefaultTableModel(Object[][], Object[]) は変換されませんでした。

javax.swing.table.DefaultTableModel.DefaultTableModel(Vector, int) は変換されませんでした。

javax.swing.table.DefaultTableModel.DefaultTableModel(Vector, Vector) は変換されませんでした。

javax.swing.table.DefaultTableModel.getDataVector は変換されませんでした。

javax.swing.table.DefaultTableModel.insertRow は変換されませんでした。

javax.swing.table.DefaultTableModel.isCellEditable は変換されませんでした。

javax.swing.table.DefaultTableModel.moveRow は変換されませんでした。

javax.swing.table.DefaultTableModel.newDataAvailable は変換されませんでした。

javax.swing.table.DefaultTableModel.newRowsAdded は変換されませんでした。

javax.swing.table.DefaultTableModel.rowsRemoved は変換されませんでした。

javax.swing.table.DefaultTableModel.setColumnCount は変換されませんでした。

javax.swing.table.DefaultTableModel.setColumnIdentifiers は変換されませんでした。

javax.swing.table.DefaultTableModel.setDataVector は変換されませんでした。

javax.swing.table.DefaultTableModel.setNumRows は変換されませんでした。

javax.swing.table.DefaultTableModel.setRowCount は変換されませんでした。

javax.swing.table.JTableHeader は変換されませんでした。

javax.swing.table.TableCellEditor は変換されませんでした。

javax.swing.table.TableCellRenderer は変換されませんでした。

javax.swing.table.TableColumn は変換されませんでした。

javax.swing.table.TableColumn.<PropertyName> は変換されませんでした。

javax.swing.table.TableColumn.addPropertyChangeListener は変換されませんでした。

javax.swing.table.TableColumn.disableResizedPosting は変換されませんでした。

javax.swing.table.TableColumn.enableResizedPosting は変換されませんでした。

javax.swing.table.TableColumn.getHeaderValue は変換されませんでした。

javax.swing.table.TableColumn.getIdentifier は変換されませんでした。

javax.swing.table.TableColumn.getMaxWidth は変換されませんでした。

javax.swing.table.TableColumn.getMinWidth は変換されませんでした。

javax.swing.table.TableColumn.getModelIndex は変換されませんでした。

javax.swing.table.TableColumn.getPreferredWidth は変換されませんでした。

javax.swing.table.TableColumn.getResizable は変換されませんでした。

javax.swing.table.TableColumn.getWidth は変換されませんでした。

javax.swing.table.TableColumn.headerValue は変換されませんでした。

javax.swing.table.TableColumn.identifier は変換されませんでした。

javax.swing.table.TableColumn.isResizable は変換されませんでした。

javax.swing.table.TableColumn.maxWidth は変換されませんでした。

javax.swing.table.TableColumn.minWidth は変換されませんでした。

javax.swing.table.TableColumn.modelIndex は変換されませんでした。

javax.swing.table.TableColumn.removePropertyChangeListener は変換されませんでした。

javax.swing.table.TableColumn.resizedPostingDisableCount は変換されませんでした。

javax.swing.table.TableColumn.setHeaderValue は変換されませんでした。

javax.swing.table.TableColumn.setIdentifier は変換されませんでした。

javax.swing.table.TableColumn.setMaxWidth は変換されませんでした。

javax.swing.table.TableColumn.setMinWidth は変換されませんでした。

javax.swing.table.TableColumn.setModelIndex は変換されませんでした。

javax.swing.table.TableColumn.setPreferredWidth は変換されませんでした。

javax.swing.table.TableColumn.setResizable は変換されませんでした。

javax.swing.table.TableColumn.setWidth は変換されませんでした。

javax.swing.table.TableColumn.sizeWidthToFit は変換されませんでした。

javax.swing.table.TableColumn.TableColumn は変換されませんでした。

javax.swing.table.TableColumn.width は変換されませんでした。

javax.swing.table.TableColumnModel は変換されませんでした。

javax.swing.table.TableColumnModel.getColumnIndexAtX は変換されませんでした。

javax.swing.table.TableColumnModel.getColumnMargin は変換されませんでした。

javax.swing.table.TableColumnModel.addColumnModelListener は変換されませんでした。

javax.swing.table.TableColumnModel.getColumnSelectionAllowed は変換されませんでした。

javax.swing.table.TableColumnModel.getSelectedColumnCount は変換されませんでした。

javax.swing.table.TableColumnModel.getSelectedColumns は変換されませんでした。

javax.swing.table.TableColumnModel.getSelectionModel は変換されませんでした。

javax.swing.table.TableColumnModel.getTotalColumnWidth は変換されませんでした。

javax.swing.table.TableColumnModel.moveColumn は変換されませんでした。

javax.swing.table.TableColumnModel.removeColumnModelListener は変換されませんでした。

javax.swing.table.TableColumnModel.setColumnMargin は変換されませんでした。

javax.swing.table.TableColumnModel.setColumnSelectionAllowed は変換されませんでした。

javax.swing.table.TableColumnModel.setSelectionModel は変換されませんでした。

javax.swing.table.TableModel は変換されませんでした。

javax.swing.table.TableModel.addTableModelListener は変換されませんでした。

javax.swing.table.TableModel.removeTableModelListener は変換されませんでした。

Javax.swing.text のエラー メッセージ

javax.swing.text.AbstractDocument は変換されませんでした。

javax.swing.text.AbstractDocument.AbstractElement は変換されませんでした。

javax.swing.text.AbstractDocument.AttributeContext は変換されませんでした。

javax.swing.text.AbstractDocument.BranchElement は変換されませんでした。

javax.swing.text.AbstractDocument.Content は変換されませんでした。

javax.swing.text.AbstractDocument.DefaultDocumentEvent は変換されませんでした。

javax.swing.text.AbstractDocument.ElementEdit は変換されませんでした。

javax.swing.text.AbstractDocument.LeafElement は変換されませんでした。

javax.swing.text.AbstractWriter.AbstractWriter は変換されませんでした。

javax.swing.text.AbstractWriter.decrIndent は変換されませんでした。

javax.swing.text.AbstractWriter.getCanWrapLines は変換されませんでした。

javax.swing.text.AbstractWriter.getCurrentLineLength は変換されませんでした。

javax.swing.text.AbstractWriter.getDocument は変換されませんでした。

javax.swing.text.AbstractWriter.getElementIterator は変換されませんでした。

javax.swing.text.AbstractWriter.getEndOffset は変換されませんでした。

javax.swing.text.AbstractWriter.getIndentLevel は変換されませんでした。

javax.swing.text.AbstractWriter.getIndentSpace は変換されませんでした。

javax.swing.text.AbstractWriter.getLineLength は変換されませんでした。

javax.swing.text.AbstractWriter.getStartOffset は変換されませんでした。

javax.swing.text.AbstractWriter.getText は変換されませんでした。

javax.swing.text.AbstractWriter.getWriter は変換されませんでした。

javax.swing.text.AbstractWriter.incrIndent は変換されませんでした。

javax.swing.text.AbstractWriter.indent は変換されませんでした。

javax.swing.text.AbstractWriter.inRange は変換されませんでした。

javax.swing.text.AbstractWriter.isLineEmpty は変換されませんでした。

javax.swing.text.AbstractWriter.NEWLINE は変換されませんでした。

javax.swing.text.AbstractWriter.setCanWrapLines は変換されませんでした。

javax.swing.text.AbstractWriter.setCurrentLineLength は変換されませんでした。

javax.swing.text.AbstractWriter.setIndentSpace は変換されませんでした。

javax.swing.text.AbstractWriter.setLineLength は変換されませんでした。

javax.swing.text.AbstractWriter.text は変換されませんでした。

javax.swing.text.AbstractWriter.writeAttributes は変換されませんでした。

javax.swing.text.AsyncBoxView は変換されませんでした。

javax.swing.text.AsyncBoxView.ChildLocator は変換されませんでした。

javax.swing.text.AsyncBoxView.ChildState は変換されませんでした。

javax.swing.text.AttributeSet.CharacterAttribute は変換されませんでした。

javax.swing.text.AttributeSet.ColorAttribute は変換されませんでした。

javax.swing.text.AttributeSet.copyAttributes は変換されませんでした。

javax.swing.text.AttributeSet.FontAttribute は変換されませんでした。

javax.swing.text.AttributeSet.getResolveParent は変換されませんでした。

javax.swing.text.AttributeSet.isDefined は変換されませんでした。

javax.swing.text.AttributeSet.isEqual は変換されませんでした。

javax.swing.text.AttributeSet.NameAttribute は変換されませんでした。

javax.swing.text.AttributeSet.ParagraphAttribute は変換されませんでした。

javax.swing.text.AttributeSet.ResolveAttribute は変換されませんでした。

javax.swing.text.BadLocationException は変換されませんでした。

javax.swing.text.BadLocationException.offsetRequested は変換されませんでした。

javax.swing.text.BoxView は変換されませんでした。

javax.swing.text.Caret は変換されませんでした。

javax.swing.text.ChangedCharSetException は変換されませんでした。

javax.swing.text.ChangedCharSetException.keyEqualsCharSet は変換されませんでした。

javax.swing.text.ComponentView は変換されませんでした。

javax.swing.text.CompositeView は変換されませんでした。

javax.swing.text.DefaultCaret は変換されませんでした。

javax.swing.text.DefaultEditorKit は変換されませんでした。

javax.swing.text.DefaultEditorKit.BeepAction は変換されませんでした。

javax.swing.text.DefaultEditorKit.CopyAction は変換されませんでした。

javax.swing.text.DefaultEditorKit.CutAction は変換されませんでした。

javax.swing.text.DefaultEditorKit.DefaultKeyTypedAction は変換されませんでした。

javax.swing.text.DefaultEditorKit.InsertBreakAction は変換されませんでした。

javax.swing.text.DefaultEditorKit.InsertContentAction は変換されませんでした。

javax.swing.text.DefaultEditorKit.InsertTabAction は変換されませんでした。

javax.swing.text.DefaultEditorKit.PasteAction は変換されませんでした。

javax.swing.text.DefaultHighlighter は変換されませんでした。

javax.swing.text.DefaultHighlighter.DefaultHighlightPainter は変換されませんでした。

javax.swing.text.DefaultStyledDocument は変換されませんでした。

javax.swing.text.DefaultStyledDocument.AttributeUndoableEdit は変換されませんでした。

javax.swing.text.DefaultStyledDocument.ElementBuffer は変換されませんでした。

javax.swing.text.DefaultStyledDocument.ElementSpec は変換されませんでした。

javax.swing.text.DefaultTextUI は変換されませんでした。

javax.swing.text.Document は変換されませんでした。

javax.swing.text.EditorKit は変換されませんでした。

javax.swing.text.Element は変換されませんでした。

javax.swing.text.ElementIterator は変換されませんでした。

javax.swing.text.FieldView は変換されませんでした。

javax.swing.text.FlowView は変換されませんでした。

javax.swing.text.FlowView.FlowStrategy は変換されませんでした。

javax.swing.text.GapContent は変換されませんでした。

javax.swing.text.GlyphView は変換されませんでした。

javax.swing.text.GlyphView.GlyphPainter は変換されませんでした。

javax.swing.text.Highlighter は変換されませんでした。

javax.swing.text.Highlighter.Highlight は変換されませんでした。

javax.swing.text.Highlighter.HighlightPainter は変換されませんでした。

javax.swing.text.html.BlockView は変換されませんでした。

javax.swing.text.html.CSS は変換されませんでした。

javax.swing.text.html.CSS.Attribute.<AttributeName> は変換されませんでした。

javax.swing.text.html.CSS.Attribute.getDefaultValue は変換されませんでした。

javax.swing.text.html.CSS.Attribute.isInherited は変換されませんでした。

javax.swing.text.html.FormView は変換されませんでした。

javax.swing.text.html.HTML.Attribute.<AttributeName> は変換されませんでした。

javax.swing.text.html.HTML.Tag.<ElementName> は変換されませんでした。

javax.swing.text.html.HTML.Tag.breaksFlow は変換されませんでした。

javax.swing.text.html.HTML.Tag.isBlock は変換されませんでした。

javax.swing.text.html.HTML.Tag.isPreformatted は変換されませんでした。

javax.swing.text.html.HTML.Tag.Tag は変換されませんでした。

javax.swing.text.html.HTML.UnknownTag は変換されませんでした。

javax.swing.text.html.HTMLDocument.AdditionalComments は変換されませんでした。

javax.swing.text.html.HTMLDocument.BlockElement.BlockElement は変換されませんでした。

javax.swing.text.html.HTMLDocument.BlockElement.getResolveParent は変換されませんでした。

javax.swing.text.html.HTMLDocument.create は変換されませんでした。

javax.swing.text.html.HTMLDocument.createBranchElement は変換されませんでした。

javax.swing.text.html.HTMLDocument.createDefaultRoot は変換されませんでした。

javax.swing.text.html.HTMLDocument.createLeafElement は変換されませんでした。

javax.swing.text.html.HTMLDocument.fireChangedUpdate は変換されませんでした。

javax.swing.text.html.HTMLDocument.fireUndoableEditUpdate は変換されませんでした。

javax.swing.text.html.HTMLDocument.getElement は変換されませんでした。

javax.swing.text.html.HTMLDocument.getParser は変換されませんでした。

javax.swing.text.html.HTMLDocument.getPreservesUnknownTags は変換されませんでした。

javax.swing.text.html.HTMLDocument.getReader は変換されませんでした。

javax.swing.text.html.HTMLDocument.getTokenThreshold は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLDocument は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLDocument(AbstractDocumentContent, StyleSheet) は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLDocument(StyleSheet) は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader.BlockAction は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader.CharacterAction は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader.FormAction は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader.HiddenAction は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader.IsindexAction は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader.ParagraphAction は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader.PreAction は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader.SpecialAction は変換されませんでした。

javax.swing.text.html.HTMLDocument.HTMLReader.TagAction は変換されませんでした。

javax.swing.text.html.HTMLDocument.insert は変換されませんでした。

javax.swing.text.html.HTMLDocument.insertAfterEnd は変換されませんでした。

javax.swing.text.html.HTMLDocument.insertAfterStart は変換されませんでした。

javax.swing.text.html.HTMLDocument.insertBeforeEnd は変換されませんでした。

javax.swing.text.html.HTMLDocument.insertBeforeStart は変換されませんでした。

javax.swing.text.html.HTMLDocument.insertUpdate は変換されませんでした。

javax.swing.text.html.HTMLDocument.Iterator.getAttributes は変換されませんでした。

javax.swing.text.html.HTMLDocument.Iterator.getEndOffset は変換されませんでした。

javax.swing.text.html.HTMLDocument.Iterator.getStartOffset は変換されませんでした。

javax.swing.text.html.HTMLDocument.Iterator.isValid は変換されませんでした。

javax.swing.text.html.HTMLDocument.Iterator.Iterator は変換されませんでした。

javax.swing.text.html.HTMLDocument.processHTMLFrameHyperlinkEvent は変換されませんでした。

javax.swing.text.html.HTMLDocument.RunElement は変換されませんでした。

javax.swing.text.html.HTMLDocument.setInnerHTML は変換されませんでした。

javax.swing.text.html.HTMLDocument.setOuterHTML は変換されませんでした。

javax.swing.text.html.HTMLDocument.setParagraphAttributes は変換されませんでした。

javax.swing.text.html.HTMLDocument.setParser は変換されませんでした。

javax.swing.text.html.HTMLDocument.setPreservesUnknownTags は変換されませんでした。

javax.swing.text.html.HTMLDocument.setTokenThreshold は変換されませんでした。

javax.swing.text.html.HTMLEditorKit は変換されませんでした。

javax.swing.text.html.HTMLEditorKit.HTMLFactory は変換されませんでした。

javax.swing.text.html.HTMLEditorKit.HTMLTextAction は変換されませんでした。

javax.swing.text.html.HTMLEditorKit.InsertHTMLTextAction は変換されませんでした。

javax.swing.text.html.HTMLEditorKit.InsertHTMLTextAction.html は変換されませんでした。

javax.swing.text.html.HTMLEditorKit.LinkController は変換されませんでした。

javax.swing.text.html.HTMLEditorKit.Parser は変換されませんでした。

javax.swing.text.html.HTMLEditorKit.ParserCallback は変換されませんでした。

javax.swing.text.html.HTMLFrameHyperlinkEvent は変換されませんでした。

javax.swing.text.html.HTMLWriter は変換されませんでした。

javax.swing.text.html.HTMLWriter.closeOutUnwantedEmbeddedTags は変換されませんでした。

javax.swing.text.html.HTMLWriter.comment は変換されませんでした。

javax.swing.text.html.HTMLWriter.emptyTag は変換されませんでした。

javax.swing.text.html.HTMLWriter.endTag は変換されませんでした。

javax.swing.text.html.HTMLWriter.HTMLWriter は変換されませんでした。

javax.swing.text.html.HTMLWriter.isBlockTag は変換されませんでした。

javax.swing.text.html.HTMLWriter.matchNameAttribute は変換されませんでした。

javax.swing.text.html.HTMLWriter.selectContent は変換されませんでした。

javax.swing.text.html.HTMLWriter.startTag は変換されませんでした。

javax.swing.text.html.HTMLWriter.synthesizedElement は変換されませんでした。

javax.swing.text.html.HTMLWriter.text は変換されませんでした。

javax.swing.text.html.HTMLWriter.textAreaContent は変換されませんでした。

javax.swing.text.html.HTMLWriter.writeEmbeddedTags は変換されませんでした。

javax.swing.text.html.InlineView は変換されませんでした。

javax.swing.text.html.ListView は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.endFontTag は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.inFontTag は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.isText は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.MinimalHTMLWriter は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.startFontTag は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.text は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.write は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.writeComponent は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.writeContent は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.writeHeader は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.writeHTMLTags は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.writeImage は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.writeLeaf は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.writeNonHTMLAttributes は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.writeStartParagraph は変換されませんでした。

javax.swing.text.html.MinimalHTMLWriter.writeStyles は変換されませんでした。

javax.swing.text.html.ObjectView は変換されませんでした。

javax.swing.text.html.Option.getAttributes は変換されませんでした。

javax.swing.text.html.Option.Option は変換されませんでした。

javax.swing.text.html.ParagraphView は変換されませんでした。

javax.swing.text.html.parser.AttributeList は変換されませんでした。

javax.swing.text.html.parser.ContentModel は変換されませんでした。

javax.swing.text.html.parser.DocumentParser は変換されませんでした。

javax.swing.text.html.parser.DTD は変換されませんでした。

javax.swing.text.html.parser.DTDConstants は変換されませんでした。

javax.swing.text.html.parser.Element は変換されませんでした。

javax.swing.text.html.parser.Entity は変換されませんでした。

javax.swing.text.html.parser.Parser は変換されませんでした。

javax.swing.text.html.parser.ParserDelegator は変換されませんでした。

javax.swing.text.html.parser.TagElement は変換されませんでした。

javax.swing.text.html.StyleSheet.addAttribute は変換されませんでした。

javax.swing.text.html.StyleSheet.addAttributes は変換されませんでした。

javax.swing.text.html.StyleSheet.addCSSAttribute は変換されませんでした。

javax.swing.text.html.StyleSheet.addCSSAttributeFromHTML は変換されませんでした。

javax.swing.text.html.StyleSheet.addStyleSheet は変換されませんでした。

javax.swing.text.html.StyleSheet.BoxPainter は変換されませんでした。

javax.swing.text.html.StyleSheet.createLargeAttributeSet は変換されませんでした。

javax.swing.text.html.StyleSheet.createSmallAttributeSet は変換されませんでした。

javax.swing.text.html.StyleSheet.getBackground は変換されませんでした。

javax.swing.text.html.StyleSheet.getBase は変換されませんでした。

javax.swing.text.html.StyleSheet.getBoxPainter は変換されませんでした。

javax.swing.text.html.StyleSheet.getDeclaration は変換されませんでした。

javax.swing.text.html.StyleSheet.getFont は変換されませんでした。

javax.swing.text.html.StyleSheet.getForeground は変換されませんでした。

javax.swing.text.html.StyleSheet.getIndexOfSize は変換されませんでした。

javax.swing.text.html.StyleSheet.getListPainter は変換されませんでした。

javax.swing.text.html.StyleSheet.getPointSize(int) は変換されませんでした。

javax.swing.text.html.StyleSheet.getPointSize(String) は変換されませんでした。

javax.swing.text.html.StyleSheet.getRule は変換されませんでした。

javax.swing.text.html.StyleSheet.getStyleSheets は変換されませんでした。

javax.swing.text.html.StyleSheet.getViewAttributes は変換されませんでした。

javax.swing.text.html.StyleSheet.importStyleSheet は変換されませんでした。

javax.swing.text.html.StyleSheet.ListPainter は変換されませんでした。

javax.swing.text.html.StyleSheet.loadRules は変換されませんでした。

javax.swing.text.html.StyleSheet.removeAttribute は変換されませんでした。

javax.swing.text.html.StyleSheet.removeAttributes は変換されませんでした。

javax.swing.text.html.StyleSheet.removeStyle は変換されませんでした。

javax.swing.text.html.StyleSheet.removeStyleSheet は変換されませんでした。

javax.swing.text.html.StyleSheet.setBase は変換されませんでした。

javax.swing.text.html.StyleSheet.setBaseFontSize(int) は変換されませんでした。

javax.swing.text.html.StyleSheet.setBaseFontSize(String) は変換されませんでした。

javax.swing.text.html.StyleSheet.stringToColor は変換されませんでした。

javax.swing.text.html.StyleSheet.StyleSheet は変換されませんでした。

javax.swing.text.html.StyleSheet.translateHTMLToCSS は変換されませんでした。

javax.swing.text.IconView は変換されませんでした。

javax.swing.text.MutableAttributeSet.addAttributes は変換されませんでした。

javax.swing.text.MutableAttributeSet.removeAttributes(AttributeSet) は変換されませんでした。

javax.swing.text.MutableAttributeSet.removeAttributes(Enumeration) は変換されませんでした。

javax.swing.text.MutableAttributeSet.setResolveParent は変換されませんでした。

javax.swing.text.JTextComponent.AccessibleJTextComponent は変換されませんでした。

javax.swing.text.JTextComponent.addCaretListener は変換されませんでした。

javax.swing.text.JTextComponent.addInputMethodListener は変換されませんでした。

javax.swing.text.JTextComponent.addKeymap は変換されませんでした。

javax.swing.text.JTextComponent.DEFAULT_KEYMAP は変換されませんでした。

javax.swing.text.JTextComponent.fireCaretUpdate は変換されませんでした。

javax.swing.text.JTextComponent.FOCUS_ACCELERATOR_KEY は変換されませんでした。

javax.swing.text.JTextComponent.getActions は変換されませんでした。

javax.swing.text.JTextComponent.getCaret は変換されませんでした。

javax.swing.text.JTextComponent.getCaretColor は変換されませんでした。

javax.swing.text.JTextComponent.getDisabledTextColor は変換されませんでした。

javax.swing.text.JTextComponent.getDocument は変換されませんでした。

javax.swing.text.JTextComponent.getFocusAccelerator は変換されませんでした。

javax.swing.text.JTextComponent.getHighlighter は変換されませんでした。

javax.swing.text.JTextComponent.getInputMethodRequests は変換されませんでした。

javax.swing.text.JTextComponent.getKeymap は変換されませんでした。

javax.swing.text.JTextComponent.getKeymap(String) は変換されませんでした。

javax.swing.text.JTextComponent.getMargin は変換されませんでした。

javax.swing.text.JTextComponent.getPreferredScrollableViewportSize は変換されませんでした。

javax.swing.text.JTextComponent.getScrollableBlockIncrement は変換されませんでした。

javax.swing.text.JTextComponent.getScrollableTracksViewportHeight は変換されませんでした。

javax.swing.text.JTextComponent.getScrollableTracksViewportWidth は変換されませんでした。

javax.swing.text.JTextComponent.getScrollableUnitIncrement は変換されませんでした。

javax.swing.text.JTextComponent.getSelectedTextColor は変換されませんでした。

javax.swing.text.JTextComponent.getSelectionColor は変換されませんでした。

javax.swing.text.JTextComponent.getUI は変換されませんでした。

javax.swing.text.JTextComponent.loadKeymap は変換されませんでした。

javax.swing.text.JTextComponent.modelToView は変換されませんでした。

javax.swing.text.JTextComponent.processInputMethodEvent は変換されませんでした。

javax.swing.text.JTextComponent.processKeyEvent は変換されませんでした。

javax.swing.text.JTextComponent.read は変換されませんでした。

javax.swing.text.JTextComponent.removeCaretListener は変換されませんでした。

javax.swing.text.JTextComponent.removeKeymap は変換されませんでした。

javax.swing.text.JTextComponent.setCaret は変換されませんでした。

javax.swing.text.JTextComponent.setCaretColor は変換されませんでした。

javax.swing.text.JTextComponent.setDisabledTextColor は変換されませんでした。

javax.swing.text.JTextComponent.setDocument は変換されませんでした。

javax.swing.text.JTextComponent.setFocusAccelerator は変換されませんでした。

javax.swing.text.JTextComponent.setHighlighter は変換されませんでした。

javax.swing.text.JTextComponent.setKeymap は変換されませんでした。

javax.swing.text.JTextComponent.setMargin は変換されませんでした。

javax.swing.text.JTextComponent.setSelectedTextColor は変換されませんでした。

javax.swing.text.JTextComponent.setSelectionColor は変換されませんでした。

javax.swing.text.JTextComponent.setText は変換されませんでした。

javax.swing.text.JTextComponent.setUI は変換されませんでした。

javax.swing.text.JTextComponent.updateUI は変換されませんでした。

javax.swing.text.JTextComponent.viewToModel は変換されませんでした。

javax.swing.text.JTextComponent.write は変換されませんでした。

javax.swing.text.JTextComponentBeanInfo は変換されませんでした。

javax.swing.text.Keymap は変換されませんでした。

javax.swing.text.LabelView は変換されませんでした。

javax.swing.text.LayeredHighlighter は変換されませんでした。

javax.swing.text.LayeredHighlighter.LayerPainter は変換されませんでした。

javax.swing.text.LayoutQueue は変換されませんでした。

javax.swing.text.MutableAttributeSet.addAttributes は変換されませんでした。

javax.swing.text.MutableAttributeSet.removeAttributes(AttributeSet) は変換されませんでした。

javax.swing.text.MutableAttributeSet.removeAttributes(Enumeration) は変換されませんでした。

javax.swing.text.MutableAttributeSet.setResolveParent は変換されませんでした。

javax.swing.text.ParagraphView は変換されませんでした。

javax.swing.text.PasswordView は変換されませんでした。

javax.swing.text.PlainDocument は変換されませんでした。

javax.swing.text.PlainView は変換されませんでした。

javax.swing.text.Position は変換されませんでした。

javax.swing.text.Position.Bias は変換されませんでした。

javax.swing.text.rtf.RTFEditorKit は変換されませんでした。

javax.swing.text.SimpleAttributeSet.addAttributes は変換されませんでした。

javax.swing.text.SimpleAttributeSet.containsAttributes は変換されませんでした。

javax.swing.text.SimpleAttributeSet.copyAttributes は変換されませんでした。

javax.swing.text.SimpleAttributeSet.EMPTY は変換されませんでした。

javax.swing.text.SimpleAttributeSet.equals は変換されませんでした。

javax.swing.text.SimpleAttributeSet.getAttributeNames は変換されませんでした。

javax.swing.text.SimpleAttributeSet.getResolveParent は変換されませんでした。

javax.swing.text.SimpleAttributeSet.isEqual は変換されませんでした。

javax.swing.text.SimpleAttributeSet.removeAttributes(AttributeSet) は変換されませんでした。

javax.swing.text.SimpleAttributeSet.removeAttributes(Enumeration) は変換されませんでした。

javax.swing.text.SimpleAttributeSet.setResolveParent は変換されませんでした。

javax.swing.text.SimpleAttributeSet.toString は変換されませんでした。

javax.swing.text.StringContent.createPosition は変換されませんでした。

javax.swing.text.StringContent.getPositionsInRange は変換されませんでした。

javax.swing.text.StringContent.updateUndoPositions は変換されませんでした。

javax.swing.text.Style は変換されませんでした。

javax.swing.text.StyleConstants は変換されませんでした。

javax.swing.text.StyleConstants.CharacterConstants は変換されませんでした。

javax.swing.text.StyleConstants.ColorConstants は変換されませんでした。

javax.swing.text.StyleConstants.FontConstants は変換されませんでした。

javax.swing.text.StyleConstants.ParagraphConstants は変換されませんでした。

javax.swing.text.StyleContext は変換されませんでした。

javax.swing.text.StyleContext.NamedStyle は変換されませんでした。

javax.swing.text.StyleContext.SmallAttributeSet は変換されませんでした。

javax.swing.text.StyledDocument は変換されませんでした。

javax.swing.text.StyledEditorKit は変換されませんでした。

javax.swing.text.StyledEditorKit.AlignmentAction は変換されませんでした。

javax.swing.text.StyledEditorKit.BoldAction は変換されませんでした。

javax.swing.text.StyledEditorKit.FontFamilyAction は変換されませんでした。

javax.swing.text.StyledEditorKit.FontSizeAction は変換されませんでした。

javax.swing.text.StyledEditorKit.ForegroundAction は変換されませんでした。

javax.swing.text.StyledEditorKit.ItalicAction は変換されませんでした。

javax.swing.text.StyledEditorKit.StyledTextAction は変換されませんでした。

javax.swing.text.StyledEditorKit.UnderlineAction は変換されませんでした。

javax.swing.text.TabableView は変換されませんでした。

javax.swing.text.TabExpander は変換されませんでした。

javax.swing.text.TableView は変換されませんでした。

javax.swing.text.TableView.TableCell は変換されませんでした。

javax.swing.text.TableView.TableRow は変換されませんでした。

javax.swing.text.TabSet は変換されませんでした。

javax.swing.text.TabStop は変換されませんでした。

javax.swing.text.TextAction は変換されませんでした。

javax.swing.text.Utilities は変換されませんでした。

javax.swing.text.View は変換されませんでした。

javax.swing.text.ViewFactory は変換されませんでした。

javax.swing.text.WrappedPlainView は変換されませんでした。

javax.swing.text.ZoneView は変換されませんでした。

Javax.swing.tree のエラー メッセージ

javax.swing.tree.AbstractLayoutCache は変換されませんでした。

javax.swing.tree.AbstractLayoutCache.NodeDimensions は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.allowsChildren は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.breadthFirstEnumeration は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.children は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.depthFirstEnumeration は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.getAllowsChildren は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.getDepth は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.getLeafCount は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.getPath は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.getUserObject は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.getUserObjectPath は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.isNodeDescendant は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.setAllowsChildren は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.setUserObject は変換されませんでした。

javax.swing.tree.DefaultMutableTreeNode.userObject は変換されませんでした。

javax.swing.tree.DefaultTreeCellEditor は変換されませんでした。

javax.swing.tree.DefaultTreeCellEditor.DefaultTextField は変換されませんでした。

javax.swing.tree.DefaultTreeCellEditor.EditorContainer は変換されませんでした。

javax.swing.tree.DefaultTreeCellRenderer は変換されませんでした。

javax.swing.tree.DefaultTreeModel は変換されませんでした。

javax.swing.tree.DefaultTreeModel.addTreeModelListener は変換されませんでした。

javax.swing.tree.DefaultTreeModel.asksAllowsChildren は変換されませんでした。

javax.swing.tree.DefaultTreeModel.DefaultTreeModel(TreeNode) は変換されませんでした。

javax.swing.tree.DefaultTreeModel.DefaultTreeModel(TreeNode, boolean) は変換されませんでした。

javax.swing.tree.DefaultTreeModel.fireTreeNodesChanged は変換されませんでした。

javax.swing.tree.DefaultTreeModel.fireTreeNodesInserted は変換されませんでした。

javax.swing.tree.DefaultTreeModel.fireTreeNodesRemoved は変換されませんでした。

javax.swing.tree.DefaultTreeModel.fireTreeStructureChanged は変換されませんでした。

javax.swing.tree.DefaultTreeModel.getChild は変換されませんでした。

javax.swing.tree.DefaultTreeModel.getChildCount は変換されませんでした。

javax.swing.tree.DefaultTreeModel.getIndexOfChild は変換されませんでした。

javax.swing.tree.DefaultTreeModel.getListeners は変換されませんでした。

javax.swing.tree.DefaultTreeModel.getRoot は変換されませんでした。

javax.swing.tree.DefaultTreeModel.insertNodeInto は変換されませんでした。

javax.swing.tree.DefaultTreeModel.isLeaf は変換されませんでした。

javax.swing.tree.DefaultTreeModel.listenerList は変換されませんでした。

javax.swing.tree.DefaultTreeModel.nodeChanged は変換されませんでした。

javax.swing.tree.DefaultTreeModel.nodesChanged は変換されませんでした。

javax.swing.tree.DefaultTreeModel.nodeStructureChanged は変換されませんでした。

javax.swing.tree.DefaultTreeModel.nodesWereInserted は変換されませんでした。

javax.swing.tree.DefaultTreeModel.nodesWereRemoved は変換されませんでした。

javax.swing.tree.DefaultTreeModel.reload は変換されませんでした。

javax.swing.tree.DefaultTreeModel.removeNodeFromParent は変換されませんでした。

javax.swing.tree.DefaultTreeModel.removeTreeModelListener は変換されませんでした。

javax.swing.tree.DefaultTreeModel.root は変換されませんでした。

javax.swing.tree.DefaultTreeModel.setAsksAllowsChildren は変換されませんでした。

javax.swing.tree.DefaultTreeModel.setRoot は変換されませんでした。

javax.swing.tree.DefaultTreeModel.valueForPathChanged は変換されませんでした。

javax.swing.tree.DefaultTreeSelectionModel は変換されませんでした。

javax.swing.tree.ExpandVetoException.event は変換されませんでした。

javax.swing.tree.ExpandVetoException.ExpandVetoException(TreeExpansionEvent) は変換されませんでした。

javax.swing.tree.ExpandVetoException.ExpandVetoException(TreeExpansionEvent, String) は変換されませんでした。

javax.swing.tree.FixedHeightLayoutCache は変換されませんでした。

javax.swing.tree.MutableTreeNode.setUserObject は変換されませんでした。

javax.swing.tree.RowMapper は変換されませんでした。

javax.swing.tree.TreeCellEditor は変換されませんでした。

javax.swing.tree.TreeCellRenderer は変換されませんでした。

javax.swing.tree.TreeModel は変換されませんでした。

javax.swing.tree.TreeModel.addTreeModelListener は変換されませんでした。

javax.swing.tree.TreeModel.getChild は変換されませんでした。

javax.swing.tree.TreeModel.getChildCount は変換されませんでした。

javax.swing.tree.TreeModel.getIndexOfChild は変換されませんでした。

javax.swing.tree.TreeModel.getRoot は変換されませんでした。

javax.swing.tree.TreeModel.isLeaf は変換されませんでした。

javax.swing.tree.TreeModel.removeTreeModelListener は変換されませんでした。

javax.swing.tree.TreeModel.valueForPathChanged は変換されませんでした。

javax.swing.tree.TreeNode.getAllowsChildren は変換されませんでした。

javax.swing.tree.TreePath は変換されませんでした。

javax.swing.tree.TreeSelectionModel は変換されませんでした。

javax.swing.tree.VariableHeightLayoutCache は変換されませんでした。

Javax.swing.undo のエラー メッセージ

javax.swing.undo.AbstractUndoableEdit は変換されませんでした。

javax.swing.undo.CompoundEdit は変換されませんでした。

javax.swing.undo.StateEdit は変換されませんでした。

javax.swing.undo.StateEditable は変換されませんでした。

javax.swing.undo.UndoableEdit は変換されませんでした。

javax.swing.undo.UndoableEditSupport は変換されませんでした。

javax.swing.undo.UndoManager は変換されませんでした。

Javax.transaction のエラー メッセージ

javax.transaction.HeuristicCommitException を変換できませんでした。

javax.transaction.HeuristicMixedException を変換できませんでした。

javax.transaction.HeuristicRollbackException を変換できませんでした。

javax.transaction.InvalidTransactionException を変換できませんでした。

javax.transaction.RollbackException を変換できませんでした。

javax.transaction.Status.STATUS_ACTIVE を変換できませんでした。

javax.transaction.Status.STATUS_COMMITTING を変換できませんでした。

javax.transaction.Status.STATUS_MARKED_ROLLBACK を変換できませんでした。

javax.transaction.Status.STATUS_PREPARED を変換できませんでした。

javax.transaction.Status.STATUS_PREPARING を変換できませんでした。

javax.transaction.Status.STATUS_UNKNOWN を変換できませんでした。

javax.transaction.SystemException.errorCode を変換できませんでした。

javax.transaction.SystemException.SystemException を変換できませんでした。

javax.transaction.Synchronization を変換できませんでした。

javax.transaction.Transaction を変換できませんでした。

javax.transaction.Transaction.commit を変換できませんでした。

javax.transaction.Transaction.delistResource を変換できませんでした。

javax.transaction.Transaction.enlistResource を変換できませんでした。

javax.transaction.Transaction.getStatus を変換できませんでした。

javax.transaction.Transaction.registerSynchronization を変換できませんでした。

javax.transaction.Transaction.rollback を変換できませんでした。

javax.transaction.Transaction.setRollbackOnly を変換できませんでした。

javax.transaction.TransactionManager を変換できませんでした。

javax.transaction.TransactionManager.setRollbackOnly を変換できませんでした。

javax.transaction.TransactionRequiredException を変換できませんでした。

javax.transaction.TransactionRolledbackException を変換できませんでした。

javax.transaction.UserTransaction を変換できませんでした。

javax.transaction.UserTransaction.begin を変換できませんでした。

javax.transaction.UserTransaction.setRollbackOnly を変換できませんでした。

javax.transaction.xa.XAException を変換できませんでした。

javax.transaction.xa.XAResource を変換できませんでした。

javax.transaction.xa.Xid を変換できませんでした。

Javax.xml のエラー メッセージ

javax.xml.parsers.DocumentBuilder は変換されませんでした。

javax.xml.parsers.DocumentBuilder.isValidating は変換されませんでした。

javax.xml.parsers.DocumentBuilder.parse は変換されませんでした。

javax.xml.parsers.DocumentBuilder.setEntityResolver は変換されませんでした。

javax.xml.parsers.DocumentBuilder.setErrorHandler は変換されませんでした。

javax.xml.parsers.DocumentBuilderFactory は変換されませんでした。

javax.xml.parsers.FactoryConfigurationError は変換されませんでした。

javax.xml.parsers.ParserConfigurationException は変換されませんでした。

javax.xml.parsers.SAXParser.getProperty は変換されませんでした。

javax.xml.parsers.SAXParser.setProperty は変換されませんでした。

javax.xml.parsers.SAXParserFactory.getFeature は変換されませんでした。

javax.xml.parsers.SAXParserFactory.setFeature は変換されませんでした。

javax.xml.transform.dom.DOMLocator は変換されませんでした。

javax.xml.transform.dom.DOMResult は変換されませんでした。

javax.xml.transform.dom.DOMResult.DOMResult は変換されませんでした。

javax.xml.transform.dom.DOMSource は変換されませんでした。

javax.xml.transform.ErrorListener は変換されませんでした。

javax.xml.transform.OutputKeys は変換されませんでした。

javax.xml.transform.Result.PI_DISABLE_OUTPUT_ESCAPING は変換されませんでした。

javax.xml.transform.Result.PI_ENABLE_OUTPUT_ESCAPING は変換されませんでした。

javax.xml.transform.sax.SAXResult は変換されませんでした。

javax.xml.transform.sax.SAXResult.SAXResult は変換されませんでした。

javax.xml.transform.sax.SAXTransformerFactory は変換されませんでした。

javax.xml.transform.sax.TemplatesHandler は変換されませんでした。

javax.xml.transform.sax.TransformerHandler は変換されませんでした。

javax.xml.transform.stream.StreamResult.setSystemId は変換されませんでした。

javax.xml.transform.stream.StreamSource.getPublicId は変換されませんでした。

javax.xml.transform.stream.StreamSource.setPublicId は変換されませんでした。

javax.xml.transform.stream.StreamSource.setSystemId は変換されませんでした。

javax.xml.transform.Templates.getOutputProperties は変換されませんでした。

javax.xml.transform.Transformer.getOutputProperties は変換されませんでした。

javax.xml.transform.Transformer.getOutputProperty は変換されませんでした。

javax.xml.transform.Transformer.getURIResolver は変換されませんでした。

javax.xml.transform.Transformer.setErrorListener は変換されませんでした。

javax.xml.transform.Transformer.setOutputProperties は変換されませんでした。

javax.xml.transform.Transformer.setOutputProperty は変換されませんでした。

javax.xml.transform.Transformer.transform は変換されませんでした。

javax.xml.transform.TransformerConfigurationException.TransformerConfigurationException は変換されませんでした。

javax.xml.transform.TransformerException.initCause は変換されませんでした。

javax.xml.transform.TransformerException.setLocator は変換されませんでした。

javax.xml.transform.TransformerException.TransformerException は変換されませんでした。

javax.xml.transform.TransformerFactory.getAssociatedStylesheet は変換されませんでした。

javax.xml.transform.TransformerFactory.getAttribute は変換されませんでした。

javax.xml.transform.TransformerFactory.getURIResolver は変換されませんでした。

javax.xml.transform.TransformerFactory.newTransformer は変換されませんでした。

javax.xml.transform.TransformerFactory.setAttribute は変換されませんでした。

javax.xml.transform.TransformerFactory.setErrorListener は変換されませんでした。

javax.xml.transform.TransformerFactoryConfigurationError は変換されませんでした。

javax.xml.transform.URIResolver は変換されませんでした。

javax.xml.transform.URIResolver.resolve は変換されませんでした。

Org.omg のエラー メッセージ

org.omg.CORBA._IDLTypeStub は変換されませんでした。

org.omg.CORBA._PolicyStub は変換されませんでした。

org.omg.CORBA.Any.create_input_stream は変換されませんでした。

org.omg.CORBA.Any.create_output_stream は変換されませんでした。

org.omg.CORBA.Any.extract_Principal は変換されませんでした。

org.omg.CORBA.Any.extract_Value は変換されませんでした。

org.omg.CORBA.Any.insert_fixed は変換されませんでした。

org.omg.CORBA.Any.insert_Object は変換されませんでした。

org.omg.CORBA.Any.insert_Principal は変換されませんでした。

org.omg.CORBA.Any.insert_Streamable は変換されませんでした。

org.omg.CORBA.Any.insert_Value(Serializable) は変換されませんでした。

org.omg.CORBA.Any.insert_Value(Serializable, TypeCode) は変換されませんでした。

org.omg.CORBA.Any.read_value は変換されませんでした。

org.omg.CORBA.Any.type は変換されませんでした。

org.omg.CORBA.Any.write_value は変換されませんでした。

org.omg.CORBA.AnyHolder は変換されませんでした。

org.omg.CORBA.AnyHolder._read は変換されませんでした。

org.omg.CORBA.AnyHolder._write は変換されませんでした。

org.omg.CORBA.AnySeqHelper は変換されませんでした。

org.omg.CORBA.AnySeqHolder は変換されませんでした。

org.omg.CORBA.ARG_IN は変換されませんでした。

org.omg.CORBA.ARG_INOUT は変換されませんでした。

org.omg.CORBA.ARG_OUT は変換されませんでした。

org.omg.CORBA.BAD_CONTEXT.BAD_CONTEXT(int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.BAD_CONTEXT.BAD_CONTEXT(String, int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.BAD_INV_ORDER.BAD_INV_ORDER(int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.BAD_INV_ORDER.BAD_INV_ORDER(String, int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.BAD_OPERATION.BAD_OPERATION(int, ComletionStatus) は変換されませんでした。

org.omg.CORBA.BAD_OPERATION.BAD_OPERATION(String, int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.BAD_PARAM.BAD_PARAM(int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.BAD_PARAM.BAD_PARAM(String, int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.BAD_POLICY は変換されませんでした。

org.omg.CORBA.BAD_POLICY_TYPE は変換されませんでした。

org.omg.CORBA.BAD_POLICY_VALUE は変換されませんでした。

org.omg.CORBA.BooleanHolder._read は変換されませんでした。

org.omg.CORBA.BooleanHolder._write は変換されませんでした。

org.omg.CORBA.BooleanSeqHelper は変換されませんでした。

org.omg.CORBA.BooleanSeqHolder は変換されませんでした。

org.omg.CORBA.ByteHolder._read は変換されませんでした。

org.omg.CORBA.ByteHolder._write は変換されませんでした。

org.omg.CORBA.CharHolder._read は変換されませんでした。

org.omg.CORBA.CharHolder._write は変換されませんでした。

org.omg.CORBA.CharSeqHelper は変換されませんでした。

org.omg.CORBA.CharSeqHolder は変換されませんでした。

org.omg.CORBA.CompletionStatus は変換されませんでした。

org.omg.CORBA.CompletionStatusHelper は変換されませんでした。

org.omg.CORBA.Context は変換されませんでした。

org.omg.CORBA.ContextList は変換されませんでした。

org.omg.CORBA.CTX_RESTRICT_SCOPE は変換されませんでした。

org.omg.CORBA.Current は変換されませんでした。

org.omg.CORBA.CurrentHelper は変換されませんでした。

org.omg.CORBA.CurrentHolder は変換されませんでした。

org.omg.CORBA.CurrentOperations は変換されませんでした。

org.omg.CORBA.CustomMarshal は変換されませんでした。

org.omg.CORBA.DataInputStream は変換されませんでした。

org.omg.CORBA.DataOutputStream は変換されませんでした。

org.omg.CORBA.DefinitionKind は変換されませんでした。

org.omg.CORBA.DefinitionKindHelper は変換されませんでした。

org.omg.CORBA.DomainManager は変換されませんでした。

org.omg.CORBA.DomainManagerOperations は変換されませんでした。

org.omg.CORBA.DoubleHolder._read は変換されませんでした。

org.omg.CORBA.DoubleHolder._write は変換されませんでした。

org.omg.CORBA.DoubleSeqHelper は変換されませんでした。

org.omg.CORBA.DoubleSeqHolder は変換されませんでした。

org.omg.CORBA.DynamicImplementation は変換されませんでした。

org.omg.CORBA.DynAny は変換されませんでした。

org.omg.CORBA.DynArray は変換されませんでした。

org.omg.CORBA.DynEnum は変換されませんでした。

org.omg.CORBA.DynFixed は変換されませんでした。

org.omg.CORBA.DynSequence は変換されませんでした。

org.omg.CORBA.DynStruct は変換されませんでした。

org.omg.CORBA.DynUnion は変換されませんでした。

org.omg.CORBA.DynValue は変換されませんでした。

org.omg.CORBA.Environment は変換されませんでした。

org.omg.CORBA.ExceptionList は変換されませんでした。

org.omg.CORBA.FieldNameHelper は変換されませんでした。

org.omg.CORBA.FixedHolder は変換されませんでした。

org.omg.CORBA.FloatHolder._read は変換されませんでした。

org.omg.CORBA.FloatHolder._write は変換されませんでした。

org.omg.CORBA.FloatSeqHelper は変換されませんでした。

org.omg.CORBA.FloatSeqHolder は変換されませんでした。

org.omg.CORBA.IdentifierHelper は変換されませんでした。

org.omg.CORBA.IDLType は変換されませんでした。

org.omg.CORBA.IDLTypeHelper は変換されませんでした。

org.omg.CORBA.IDLTypeOperations は変換されませんでした。

org.omg.CORBA.IntHolder._read は変換されませんでした。

org.omg.CORBA.IntHolder._write は変換されませんでした。

org.omg.CORBA.INV_POLICY.INV_POLICY(int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.INV_POLICY.INV_POLICY(String, int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.IRObject は変換されませんでした。

org.omg.CORBA.IRObjectOperations は変換されませんでした。

org.omg.CORBA.LocalObject は変換されませんでした。

org.omg.CORBA.LongHolder._read は変換されませんでした。

org.omg.CORBA.LongHolder._write は変換されませんでした。

org.omg.CORBA.LongLongSeqHelper は変換されませんでした。

org.omg.CORBA.LongLongSeqHolder は変換されませんでした。

org.omg.CORBA.LongSeqHelper は変換されませんでした。

org.omg.CORBA.LongSeqHolder は変換されませんでした。

org.omg.CORBA.MARSHAL.MARSHAL(int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.MARSHAL.MARSHAL(String, int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.NamedValue は変換されませんでした。

org.omg.CORBA.NameValuePair は変換されませんでした。

org.omg.CORBA.NameValuePairHelper は変換されませんでした。

org.omg.CORBA.NO_PERMISSION.NO_PERMISSION(int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.NO_PERMISSION.NO_PERMISSION(String, int, CompletionStatus) は変換されませんでした。

org.omg.CORBA.NVList は変換されませんでした。

org.omg.CORBA.Object._create_request(Context, String, NVList, NamedValue) は変換されませんでした。

org.omg.CORBA.Object._create_request(Context, String, NVList, NamedValue, ExceptionList, ContextList) は変換されませんでした。

org.omg.CORBA.Object._duplicate は変換されませんでした。

org.omg.CORBA.Object._get_domain_managers は変換されませんでした。

org.omg.CORBA.Object._get_interface_def は変換されませんでした。

org.omg.CORBA.Object._get_policy は変換されませんでした。

org.omg.CORBA.Object._hash は変換されませんでした。

org.omg.CORBA.Object._is_a は変換されませんでした。

org.omg.CORBA.Object._is_equivalent は変換されませんでした。

org.omg.CORBA.Object._non_existent は変換されませんでした。

org.omg.CORBA.Object._release は変換されませんでした。

org.omg.CORBA.Object._request は変換されませんでした。

org.omg.CORBA.Object._set_policy_override は変換されませんでした。

org.omg.CORBA.ObjectHelper は変換されませんでした。

org.omg.CORBA.ObjectHolder._read は変換されませんでした。

org.omg.CORBA.ObjectHolder._write は変換されませんでした。

org.omg.CORBA.OctetSeqHelper は変換されませんでした。

org.omg.CORBA.OctetSeqHolder は変換されませんでした。

org.omg.CORBA.OMGVMCID は変換されませんでした。

org.omg.CORBA.ORB は変換されませんでした。

org.omg.CORBA.Policy は変換されませんでした。

org.omg.CORBA.PolicyError.reason は変換されませんでした。

org.omg.CORBA.PolicyHelper は変換されませんでした。

org.omg.CORBA.PolicyHolder は変換されませんでした。

org.omg.CORBA.PolicyListHelper は変換されませんでした。

org.omg.CORBA.PolicyListHolder は変換されませんでした。

org.omg.CORBA.PolicyOperations は変換されませんでした。

org.omg.CORBA.PolicyTypeHelper は変換されませんでした。

org.omg.CORBA.portable.ApplicationException.getId は変換されませんでした。

org.omg.CORBA.portable.ApplicationException.getInputStream は変換されませんでした。

org.omg.CORBA.portable.BoxedValueHelper は変換されませんでした。

org.omg.CORBA.portable.CustomValue は変換されませんでした。

org.omg.CORBA.portable.Delegate は変換されませんでした。

org.omg.CORBA.portable.IDLEntity は変換されませんでした。

org.omg.CORBA.portable.IndirectionException.IndirectionException は変換されませんでした。

org.omg.CORBA.portable.IndirectionException.offset は変換されませんでした。

org.omg.CORBA.portable.InputStream は変換されませんでした。

org.omg.CORBA.portable.InvokeHandler は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._create_request(Context, String, NVList, NamedValue) は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._create_request(Context, String, NVList, NamedValue, ExceptionList, ContextList) は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._duplicate は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._get_delegate は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._get_domain_managers は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._get_interface_def は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._get_policy は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._hash は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._ids は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._invoke は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._is_a は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._is_equivalent は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._is_local は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._non_existent は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._orb は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._release は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._releaseReply は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._request(String) は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._request(String, boolean) は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._servant_postinvoke は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._servant_preinvoke は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._set_delegate は変換されませんでした。

org.omg.CORBA.portable.ObjectImpl._set_policy_override は変換されませんでした。

org.omg.CORBA.portable.OutputStream は変換されませんでした。

org.omg.CORBA.portable.ResponseHandler は変換されませんでした。

org.omg.CORBA.portable.ServantObject は変換されませんでした。

org.omg.CORBA.portable.Streamable は変換されませんでした。

org.omg.CORBA.portable.StreamableValue は変換されませんでした。

org.omg.CORBA.portable.ValueBase は変換されませんでした。

org.omg.CORBA.portable.ValueFactory は変換されませんでした。

org.omg.CORBA.Principal は変換されませんでした。

org.omg.CORBA.PrincipalHolder は変換されませんでした。

org.omg.CORBA.PRIVATE_MEMBER は変換されませんでした。

org.omg.CORBA.PUBLIC_MEMBER は変換されませんでした。

org.omg.CORBA.RepositoryIdHelper は変換されませんでした。

org.omg.CORBA.Request は変換されませんでした。

org.omg.CORBA.ServerRequest は変換されませんでした。

org.omg.CORBA.ServiceDetail は変換されませんでした。

org.omg.CORBA.ServiceDetailHelper は変換されませんでした。

org.omg.CORBA.ServiceInformation は変換されませんでした。

org.omg.CORBA.ServiceInformationHelper は変換されませんでした。

org.omg.CORBA.ServiceInformationHolder は変換されませんでした。

org.omg.CORBA.SetOverrideType は変換されませんでした。

org.omg.CORBA.SetOverrideTypeHelper は変換されませんでした。

org.omg.CORBA.ShortHolder._read は変換されませんでした。

org.omg.CORBA.ShortHolder._write は変換されませんでした。

org.omg.CORBA.ShortSeqHelper は変換されませんでした。

org.omg.CORBA.ShortSeqHolder は変換されませんでした。

org.omg.CORBA.StringHolder._read は変換されませんでした。

org.omg.CORBA.StringHolder._write は変換されませんでした。

org.omg.CORBA.StringHolder.StringHolder は変換されませんでした。

org.omg.CORBA.StringHolder.StringHolder(String) は変換されませんでした。

org.omg.CORBA.StringValueHelper は変換されませんでした。

org.omg.CORBA.StructMember は変換されませんでした。

org.omg.CORBA.StructMemberHelper は変換されませんでした。

org.omg.CORBA.SystemException.completed は変換されませんでした。

org.omg.CORBA.SystemException.minor は変換されませんでした。

org.omg.CORBA.TCKind.<DataType> は変換されませんでした。

org.omg.CORBA.TCKind.TCKind は変換されませんでした。

org.omg.CORBA.TypeCode は変換されませんでした。

org.omg.CORBA.TypeCode.concrete_base_type は変換されませんでした。

org.omg.CORBA.TypeCode.content_type は変換されませんでした。

org.omg.CORBA.TypeCode.default_index は変換されませんでした。

org.omg.CORBA.TypeCode.discriminator_type は変換されませんでした。

org.omg.CORBA.TypeCode.equivalent は変換されませんでした。

org.omg.CORBA.TypeCode.fixed_digits は変換されませんでした。

org.omg.CORBA.TypeCode.fixed_scale は変換されませんでした。

org.omg.CORBA.TypeCode.get_compact_typecode は変換されませんでした。

org.omg.CORBA.TypeCode.id は変換されませんでした。

org.omg.CORBA.TypeCode.length は変換されませんでした。

org.omg.CORBA.TypeCode.member_count は変換されませんでした。

org.omg.CORBA.TypeCode.member_label は変換されませんでした。

org.omg.CORBA.TypeCode.member_name は変換されませんでした。

org.omg.CORBA.TypeCode.member_type は変換されませんでした。

org.omg.CORBA.TypeCode.member_visibility は変換されませんでした。

org.omg.CORBA.TypeCode.name は変換されませんでした。

org.omg.CORBA.TypeCode.type_modifier は変換されませんでした。

org.omg.CORBA.TypeCode.TypeCode は変換されませんでした。

org.omg.CORBA.TypeCodeHolder は変換されませんでした。

org.omg.CORBA.ULongLongSeqHelper は変換されませんでした。

org.omg.CORBA.ULongLongSeqHolder は変換されませんでした。

org.omg.CORBA.ULongSeqHelper は変換されませんでした。

org.omg.CORBA.ULongSeqHolder は変換されませんでした。

org.omg.CORBA.UnionMember は変換されませんでした。

org.omg.CORBA.UnionMemberHelper は変換されませんでした。

org.omg.CORBA.UnknownUserException.except は変換されませんでした。

org.omg.CORBA.UNSUPPORTED_POLICY は変換されませんでした。

org.omg.CORBA.UNSUPPORTED_POLICY_VALUE は変換されませんでした。

org.omg.CORBA.UShortSeqHelper は変換されませんでした。

org.omg.CORBA.UShortSeqHolder は変換されませんでした。

org.omg.CORBA.ValueBaseHelper は変換されませんでした。

org.omg.CORBA.ValueBaseHolder は変換されませんでした。

org.omg.CORBA.ValueBaseHolder._read は変換されませんでした。

org.omg.CORBA.ValueBaseHolder._write は変換されませんでした。

org.omg.CORBA.ValueBaseHolder.ValueBaseHolder は変換されませんでした。

org.omg.CORBA.ValueMember は変換されませんでした。

org.omg.CORBA.ValueMemberHelper は変換されませんでした。

org.omg.CORBA.VersionSpecHelper は変換されませんでした。

org.omg.CORBA.VisibilityHelper は変換されませんでした。

org.omg.CORBA.VM_ABSTRACT は変換されませんでした。

org.omg.CORBA.VM_CUSTOM は変換されませんでした。

org.omg.CORBA.VM_NONE は変換されませんでした。

org.omg.CORBA.VM_TRUNCATABLE は変換されませんでした。

org.omg.CORBA.WCharSeqHelper は変換されませんでした。

org.omg.CORBA.WCharSeqHolder は変換されませんでした。

org.omg.CORBA.WStringValueHelper は変換されませんでした。

org.omg.CORBA_2_3.ORB は変換されませんでした。

org.omg.CORBA_2_3.portable.Delegate は変換されませんでした。

org.omg.CORBA_2_3.portable.InputStream は変換されませんでした。

org.omg.CORBA_2_3.portable.ObjectImpl は変換されませんでした。

org.omg.CORBA_2_3.portable.OutputStream は変換されませんでした。

org.omg.CosNaming._BindingIteratorImplBase は変換されませんでした。

org.omg.CosNaming._BindingIteratorStub は変換されませんでした。

org.omg.CosNaming._NamingContextImplBase は変換されませんでした。

org.omg.CosNaming._NamingContextStub は変換されませんでした。

org.omg.CosNaming.Binding は変換されませんでした。

org.omg.CosNaming.BindingHelper は変換されませんでした。

org.omg.CosNaming.BindingHolder は変換されませんでした。

org.omg.CosNaming.BindingIterator は変換されませんでした。

org.omg.CosNaming.BindingIteratorHelper は変換されませんでした。

org.omg.CosNaming.BindingIteratorHolder は変換されませんでした。

org.omg.CosNaming.BindingIteratorOperations は変換されませんでした。

org.omg.CosNaming.BindingListHelper は変換されませんでした。

org.omg.CosNaming.BindingListHolder は変換されませんでした。

org.omg.CosNaming.BindingType は変換されませんでした。

org.omg.CosNaming.BindingTypeHelper は変換されませんでした。

org.omg.CosNaming.BindingTypeHolder は変換されませんでした。

org.omg.CosNaming.IstringHelper は変換されませんでした。

org.omg.CosNaming.NameComponent は変換されませんでした。

org.omg.CosNaming.NameComponentHelper は変換されませんでした。

org.omg.CosNaming.NameComponentHolder は変換されませんでした。

org.omg.CosNaming.NameHelper は変換されませんでした。

org.omg.CosNaming.NameHolder は変換されませんでした。

org.omg.CosNaming.NamingContext は変換されませんでした。

org.omg.CosNaming.NamingContextHelper は変換されませんでした。

org.omg.CosNaming.NamingContextHolder は変換されませんでした。

org.omg.CosNaming.NamingContextOperations は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.AlreadyBoundHelper は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.AlreadyBoundHolder は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.CannotProceed.CannotProceed は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.CannotProceed.cxt は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.CannotProceed.rest_of_name は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.CannotProceedHelper は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.CannotProceedHolder は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.InvalidNameHelper は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.InvalidNameHolder は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotEmptyHelper は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotEmptyHolder は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotFound.NotFound は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotFound.rest_of_name は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotFound.why は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotFoundHelper は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotFoundHolder は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotFoundReason は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotFoundReasonHelper は変換されませんでした。

org.omg.CosNaming.NamingContextPackage.NotFoundReasonHolder は変換されませんでした。

org.omg.stub.java.rmi._Remote_Stub は変換されませんでした。

Org.w3c のエラー メッセージ

org.w3c.dom.DOMException を変換できませんでした。

org.w3c.dom.DOMException.<ExceptionType> を変換できませんでした。

org.w3c.dom.DOMException.code を変換できませんでした。

org.w3c.dom.DOMImplementation.createDocumentType を変換できませんでした。

org.w3c.dom.Element.setAttribute を変換できませんでした。

org.w3c.dom.Node.getChildNodes を変換できませんでした。

org.w3c.dom.Node.setPrefix を変換できませんでした。

org.w3c.dom.ranges.DocumentRange を変換できませんでした。

org.w3c.dom.ranges.Range を変換できませんでした。

org.w3c.dom.ranges.RangeException を変換できませんでした。

org.w3c.dom.traversal.DocumentTraversal.createNodeIterator を変換できませんでした。

org.w3c.dom.traversal.DocumentTraversal.createTreeWalker を変換できませんでした。

org.w3c.dom.traversal.NodeFilter.SHOW_ALL を変換できませんでした。

org.w3c.dom.traversal.NodeIterator.getExpandEntityReferences を変換できませんでした。

org.w3c.dom.traversal.TreeWalker.getExpandEntityReferences を変換できませんでした。

Org.xml のエラー メッセージ

org.xml.sax.AttributeList.getLength は変換されませんでした。

org.xml.sax.AttributeList.getName は変換されませんでした。

org.xml.sax.AttributeList.getType は変換されませんでした。

org.xml.sax.AttributeList.getValue は変換されませんでした。

org.xml.sax.Attributes.getIndex は変換されませんでした。

org.xml.sax.Attributes.getLength は変換されませんでした。

org.xml.sax.Attributes.getLocalName は変換されませんでした。

org.xml.sax.Attributes.getQName は変換されませんでした。

org.xml.sax.Attributes.getType は変換されませんでした。

org.xml.sax.Attributes.getURI は変換されませんでした。

org.xml.sax.Attributes.getValue は変換されませんでした。

org.xml.sax.DocumentHandler は変換されませんでした。

org.xml.sax.DTDHandler は変換されませんでした。

org.xml.sax.EntityResolver は変換されませんでした。

org.xml.sax.ErrorHandler.error は変換されませんでした。

org.xml.sax.ErrorHandler.fatalError は変換されませんでした。

org.xml.sax.ErrorHandler.warning は変換されませんでした。

org.xml.sax.ext.DeclHandler は変換されませんでした。

org.xml.sax.ext.LexicalHandler は変換されませんでした。

org.xml.sax.HandlerBase は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.addAttribute は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.AttributeListImpl は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.AttributeListImpl(AttributeList) は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.clear は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.getLength は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.getName は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.getType は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.getValue は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.removeAttribute は変換されませんでした。

org.xml.sax.helpers.AttributeListImpl.setAttributeList は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.addAttribute は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.AttributesImpl は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.AttributesImpl(Attributes) は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.clear は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.getIndex(String) は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.getIndex(String, String) は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.getLength は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.getLocalName は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.getQName は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.getType は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.getURI は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.getValue は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.removeAttribute は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.setAttribute は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.setAttributes は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.setLocalName は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.setQName は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.setType は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.setURI は変換されませんでした。

org.xml.sax.helpers.AttributesImpl.setValue は変換されませんでした。

org.xml.sax.helpers.NamespaceSupport.getDeclaredPrefixes は変換されませんでした。

org.xml.sax.helpers.NamespaceSupport.getPrefixes は変換されませんでした。

org.xml.sax.helpers.NamespaceSupport.getPrefixes(String) は変換されませんでした。

org.xml.sax.helpers.NamespaceSupport.processName は変換されませんでした。

org.xml.sax.helpers.NamespaceSupport.reset は変換されませんでした。

org.xml.sax.helpers.NamespaceSupport.XMLNS は変換されませんでした。

org.xml.sax.helpers.ParserAdapter は変換されませんでした。

org.xml.sax.helpers.ParserAdapter.getDTDHandler は変換されませんでした。

org.xml.sax.helpers.ParserAdapter.getEntityResolver は変換されませんでした。

org.xml.sax.helpers.ParserAdapter.getFeature は変換されませんでした。

org.xml.sax.helpers.ParserAdapter.getProperty は変換されませんでした。

org.xml.sax.helpers.ParserAdapter.setDTDHandler は変換されませんでした。

org.xml.sax.helpers.ParserAdapter.setErrorHandler は変換されませんでした。

org.xml.sax.helpers.ParserAdapter.setFeature は変換されませんでした。

org.xml.sax.helpers.ParserAdapter.setProperty は変換されませんでした。

org.xml.sax.helpers.XMLFilterImpl は変換されませんでした。

org.xml.sax.helpers.XMLReaderAdapter は変換されませんでした。

org.xml.sax.helpers.XMLReaderAdapter.setDTDHandler は変換されませんでした。

org.xml.sax.helpers.XMLReaderAdapter.setErrorHandler は変換されませんでした。

org.xml.sax.helpers.XMLReaderAdapter.setLocale は変換されませんでした。

org.xml.sax.helpers.XMLReaderFactory は変換されませんでした。

org.xml.sax.InputSource.getEncoding は変換されませんでした。

org.xml.sax.InputSource.getPublicId は変換されませんでした。

org.xml.sax.InputSource.setEncoding は変換されませんでした。

org.xml.sax.InputSource.setPublicId は変換されませんでした。

org.xml.sax.Parser.parse は変換されませんでした。

org.xml.sax.Parser.setDocumentHandler は変換されませんでした。

org.xml.sax.Parser.setDTDHandler は変換されませんでした。

org.xml.sax.Parser.setEntityResolver は変換されませんでした。

org.xml.sax.Parser.setErrorHandler は変換されませんでした。

org.xml.sax.Parser.setLocale は変換されませんでした。

org.xml.sax.SAXException は変換されませんでした。

org.xml.sax.SAXNotRecognizedException は変換されませんでした。

org.xml.sax.SAXNotSupportedException は変換されませんでした。

org.xml.sax.SAXParseException は変換されませんでした。

org.xml.sax.SAXParseException.SAXParseException(String, Locator) は変換されませんでした。

org.xml.sax.SAXParseException.SAXParseException(String, Locator, Exception) は変換されませんでした。

org.xml.sax.SAXParseException.SAXParseException(String, String, int, int) は変換されませんでした。

org.xml.sax.XMLFilter.getParent は変換されませんでした。

org.xml.sax.XMLFilter.setParent は変換されませんでした。

org.xml.sax.XMLReader.getContentHandler は変換されませんでした。

org.xml.sax.XMLReader.getDTDHandler は変換されませんでした。

org.xml.sax.XMLReader.getEntityResolver は変換されませんでした。

org.xml.sax.XMLReader.getErrorHandler は変換されませんでした。

org.xml.sax.XMLReader.getFeature は変換されませんでした。

org.xml.sax.XMLReader.getProperty は変換されませんでした。

org.xml.sax.XMLReader.parse は変換されませんでした。

org.xml.sax.XMLReader.setContentHandler は変換されませんでした。

org.xml.sax.XMLReader.setDTDHandler は変換されませんでした。

org.xml.sax.XMLReader.setEntityResolver は変換されませんでした。

org.xml.sax.XMLReader.setErrorHandler は変換されませんでした。

org.xml.sax.XMLReader.setFeature は変換されませんでした。

org.xml.sax.XMLReader.setProperty は変換されませんでした。

C++ 経験者が C# で開発する場合

次の表には、C# とネイティブ C++ の重要な比較が記載されています。ここでは **/clr** を使用しません。C++ のプログラマならば、この表を見ると 2 つの言語間の最も重要な相違点がすぐにわかります。

メモ :
C++ と C# のプロジェクトは、異なるプロジェクト モデルが基になっています。C++ プロジェクトと C# プロジェクトの相違の詳細については、「プロジェクトにおける項目の管理」および「ソリューション エクスプローラの使用」を参照してください。

機能	参照トピック
継承 : C++ ではクラスと構造体はほとんど同じですが、C# の場合はまったく異なります。C# のクラスは任意の数のインターフェイスを実装できますが、1 つの基本クラスからしか継承できません。さらに、C# の構造体では、継承と明示的な既定コンストラクタ (既定で 1 つ提供されます) がサポートされません。	class interface struct (C# リファレンス)
配列 : C++ の配列は単にポインタです。C# の配列は、メソッドとプロパティを含むオブジェクトです。たとえば、配列のサイズは、Length プロパティ経由で問い合わせることができます。また、C# の配列ではインデクサが採用され、配列のアクセスに使用する各インデックスを検証できます。C# の配列を宣言する構文は、C++ の場合と異なります。(変数ではなく) C# の配列型に続いて、トークン "[]" が表示されます。	配列 (C# プログラミング ガイド) インデクサ (C# プログラミング ガイド)
ブール型 : C++ では、**bool** 型は実質的に整数です。C# では、**bool** 型とその他の型は変換できません。	bool
long 型 : C# では **long** 型は 64 ビットですが、C++ では 32 ビットです。	long
パラメータを渡す方法 : C++ の場合、明示的にポインタまたは参照で渡される場合を除き、すべての変数は値によって渡されます。C# の場合、クラスは参照によって渡されます。構造体は、**ref** または **out** のパラメータ修飾子が指定された参照によって明示的に渡される場合を除き、値によって渡されます。	struct class ref (C# リファレンス) out (C# リファレンス)
switch ステートメント : C++ の **switch** ステートメントとは異なり、C# は、ある case ラベルから下にある case ラベルへの移動をサポートしていません。	switch
デリゲート : C# のデリゲートは、C++ の関数ポインタとほぼ同じですが、タイプ セーフであり、安全です。	delegate
基本クラスのメソッド : C# では、派生クラスからオーバーライドされた基本クラスのメンバを呼び出す、**base** キーワードをサポートしています。また、C# では、仮想メソッドまたは抽象メソッドのオーバーライドは明示的です。**override** キーワードを使用します。	base 「override」の例も参照してください。
メソッドの隠ぺい : C++ では、継承によってメソッドが暗黙的に "隠ぺい" されます。C# では、**new** 修飾子を使用して、継承メンバを明示的に隠ぺいする必要があります。	new
条件付きコンパイルには、プリプロセッサ ディレクティブを使用します。C# ではヘッダー ファイルを使用しません。	C# プリプロセッサ ディレクティブ
例外処理 : C# では、例外がスローされたかどうかにかかわらず実行するコードに、**finally** キーワードを指定します。	try-finally try-catch-finally

C# の演算子 : C# では、**is** や **typeof** などの演算子のサポートが追加されています。また、一部の論理演算子については異なる機能が導入されています。	& 演算子 | 演算子 (C# リファレンス) ^ 演算子 is typeof
extern キーワード : C++ では、型をインポートするときに **extern** が使用されます。C# で **extern** を使用するのは、同じアセンブリの異なるバージョンを使用するために、エイリアスを作成するときです。	extern
static キーワード : C++ では、クラス レベルの要素を宣言するときと、モジュール独自の型を宣言するときに、**static** キーワードを使用できます。C# では、クラス レベルの要素を宣言するときにのみ、**static** を使用できます。	static
C# の **Main** メソッドの宣言方法は、C++ の **main** 関数の場合と異なります。C# では大文字で記載され、常に **static** です。また、コマンド ライン引数を処理する場合のサポートは、C# の方が強力です。	Main() とコマンド ライン引数 (C# プログラミング ガイド)
C# でもポインタを使用できますが、使用できるのは **unsafe** モードの場合のみです。	unsafe
C# では、演算子をオーバーロードする方法が異なります。	C# の演算子
文字列 : C++ の文字列は、単に一連の文字です。C# の文字列は、強力な検索メソッドをサポートするオブジェクトです。	string String
foreach キーワードを使用すると、配列およびコレクションの反復処理を実行できます。	foreach、in
グローバル : C# では、グローバル メソッドとグローバル変数はサポートされません。メソッドと変数は、**class** または **struct** に含める必要があります。	C# プログラムの一般的な構造
型のインポート : C++ では、複数のモジュールに共通した型は、ヘッダー ファイルに配置されます。C# では、この情報はメタデータ経由で使用できます。	using メタデータの概要
C# のローカル変数は、使用する前に初期化する必要があります。	メソッド (C# プログラミング ガイド)
メモリ管理 : C++ は、ガベージ コレクションが行われる言語ではありません。明示的に解放されないメモリは、プロセスが終了するまで割り当てられたままです。C# は、ガベージ コレクションが行われる言語です。	ガベージ コレクション
デストラクタ : C# は、アンマネージ リソースを確定的に解放する場合の構文が異なります。	デストラクタ using ステートメント (C# リファレンス)
コンストラクタ : C++ と同様に、C# でクラス コンストラクタを指定しない場合は、既定のコンストラクタが自動的に生成されます。既定のコンストラクタは、すべてのフィールドを既定値に初期化します。	インスタンス コンストラクタ 既定値の一覧表
C# では、ビット フィールドはサポートされていません。	C++ Bit Fields
C# の入出力サービスおよび書式指定は、.NET Framework のランタイム ライブラリに依存します。	C# 言語ツアー 数値結果の書式指定の一覧表

C# では、メソッドのパラメータに既定値を設定できません。同様の機能が必要な場合は、メソッドのオーバーロードを使用します。	コンパイラ エラー CS0241
C# では、ジェネリック型とジェネリック メソッドで、型のパラメータ化が規定されます。その方法はある程度 C++ のテンプレートに似ていますが、大きな違いがあります。	C# のジェネリック
as キーワードは、標準のキャストに似ていますが、変換が失敗した場合に例外がスローされるのではなく、戻り値が null になります。これは、C++ で **static_cast** を使用する場合と似ています。**dynamic_cast** とは異なり、実行時のチェックが実行されないため、エラー時に例外がスローされません。	as (C# リファレンス)

C# と他のプログラミング言語とで異なるキーワードの比較の詳細については、「各言語の比較」を参照してください。C# アプリケーションの一般的な構造については、「C# プログラムの一般構造 (C# プログラミング ガイド)」を参照してください。

参照
概念
C# プログラミング ガイド
プロジェクトにおける項目の管理
ソリューション エクスプローラの使用

Visual C# によるアプリケーションの作成

C# は、簡素でありながらも高機能を備えた、タイプセーフなオブジェクト指向言語で、広範なアプリケーションを構築できます。Visual C# は、.NET Framework と組み合わせて使用することで、Windows アプリケーション、Web サービス、データベース ツール、コンポーネント、コントロールなどを作成できます。

ここでは、C# アプリケーションの基盤となるさまざまな Microsoft プラットフォーム テクノロジについて説明します。

このセクションの内容

.NET Framework クラス ライブラリの使用 (Visual C#)

.NET Framework クラス ライブラリの型を Visual C# プロジェクトで使用する方法について説明します。

ASP.NET Web アプリケーションの作成 (Visual C#)

Visual Web Developer 内で C# コード エディタを使用して、分離コード ページを含む Web アプリケーションを C# で作成する方法について説明します。

Windows フォーム アプリケーションの作成 (Visual C#)

Windows フォームを使用して Windows アプリケーションを作成する方法について説明します。

コンソール アプリケーションの作成 (Visual C#)

グラフィカル ユーザー インターフェイスを必要としないアプリケーションの作成について説明します。

データのアクセスと表示 (Visual C#)

データベースとの対話について説明します。

モバイル アプリケーションと組み込みアプリケーションの作成 (Visual C#)

スマート デバイス、埋め込みデバイス、およびシン モバイル クライアント用のアプリケーションの作成について説明します。

Web サービスの作成とアクセス (Visual C#)

XML Web サービスとの対話について説明します。

コンポーネントの作成 (Visual C#)

.NET Framework 用のユーザー コントロールおよびその他のコンポーネントの作成について説明します。

Office プラットフォーム上での開発 (Visual C#)

Visual Studio Tools for Office を使用してスマート ドキュメントを作成する方法について説明します。

エンタープライズ向けの開発 (Visual C#)

SQL Server、Microsoft Exchange Server などを対象にしたアプリケーションの開発について説明します。

Tablet PC のプログラミング (Visual C#)

Tablet PC 用のインク ベースのアプリケーションの開発について説明します。

オーディオ、ビデオ、ゲーム、およびグラフィックス (Visual C#)

Windows Media および DirectX for Managed Code の使用について説明します。

スタート キットの作成 (Visual C#)

他の開発者がサンプル コードをすばやく起動して実行するのに役立つスタート キットの作成について説明します。

参照

その他の技術情報

Visual C#
C# リファレンス
Visual C# について
Visual C# IDE の使用

.NET Framework クラス ライブラリの使用 (Visual C#)

Visual C# 開発プロジェクトのほとんどは、ファイル システムのアクセスや文字列の操作から、Windows フォームや ASP.NET のユーザー インターフェイス コントロールまで、広範囲に .NET Framework クラス ライブラリを使用してします。

クラス ライブラリは、名前空間に構成され、それぞれに関連するクラスと構造体が含まれています。たとえば、System.Drawing 名前空間には、フォント、ペン、線、図形、色などを表すさまざまな種類があります。

ディレクティブと参照の使用

C# プログラムで特定の名前空間のクラスを使用するには、あらかじめその名前空間のusing ディレクティブ (C# リファレンス)を C# ソース ファイルに追加しておく必要があります。場合によっては、名前空間を含む DLL への参照を追加する必要もあります。Visual C# では、よく使用されるクラス ライブラリの DLL に対し、自動的に参照が追加されます。ソリューション エクスプローラの [参照設定] ノードの下位に、追加した参照が表示されます。詳細については、「プロジェクトの作成 (Visual C#)」を参照してください。

名前空間に **using** ディレクティブを追加すると、ソース コードで宣言したときと同様に、型のインスタンスを作成し、メソッドを呼び出して、イベントに応答できます。また、Visual C# コード エディタでは、型またはメンバの名前にカーソルを移動し、F1 キーを押すと、ヘルプが表示されます。また、.NET Framework クラスと構造体に関する型情報を確認するには、オブジェクト ブラウザ ツールとメタデータをソース形式で表示する機能も使用できます。詳細については、「コードのモデリングと解析 (Visual C#)」を参照してください。

参照項目

- .NET Framework クラス ライブラリの詳細については、「.NET Framework クラス ライブラリの概要」および「.NET Framework のプログラミング」を参照してください。
- .NET Framework アーキテクチャの詳細については、「.NET Framework の概要」を参照してください。
- インターネットでは、.NET Framework Developer Centerにクラス ライブラリのドキュメントとコード例が数多くあります。
- クラス ライブラリを使用した具体的なタスクの実行方法の詳細については、「C# での操作方法」を参照してください。または、Visual C# の [ヘルプ] メニューにある [カテゴリから検索] をクリックしてください。

参照

その他の技術情報

Visual C# によるアプリケーションの作成
Visual C# IDE の使用

ASP.NET Web アプリケーションの作成 (Visual C#)

ASP.NET は、エンタープライズ クラスの Web アプリケーションを作成する場合に開発者が必要とするサービスなどを含む、統一化された Web 開発モデルを提供します。ASP.NET は、.NET Framework の一部です。ASP.NET を使用すると、タイプ セーフ、継承、言語の相互運用性、バージョン管理など、共通言語ランタイム (CLR) のすべての機能を利用できます。

Visual Studio を使用して ASP.NET Web サイトを作成するときは、Visual Web Developer という統合開発環境 (IDE) の一部を使用します。Visual Web Developer は、Visual C# とは異なり、Web ページでユーザー インターフェイスを作成するための独自のデザイナと、Web 開発および Web サイト管理に適したその他のツールがあります。ただし、Web コントロール用に C# で分離コード ページを作成するときは、C# のコード エディタを使用します。また、エディタの機能はすべて、Visual C# の場合と同様に Visual Web Developer でも使用できます。

ASP.NET Web サイトと Web ページの作成方法の詳細については、「Visual Web Developer」を参照してください。

MSDN オンラインの「ASP.NET Developer Center」には、製品ドキュメントに含まれていない有益なドキュメントが数多くあります。

参照

その他の技術情報

Visual C# によるアプリケーションの作成

Windows フォーム アプリケーションの作成 (Visual C#)

Windows フォームは、.NET Framework で実行される Windows ベースのスマート クライアント アプリケーションを作成するときに、Visual C# で使用するテクノロジです。Windows アプリケーション プロジェクトを作成すると、Windows フォームに基づいてアプリケーションを作成することになります。ユーザー インターフェイスを作成するときには Windows フォーム デザイナを使用します。また、次に示す他のデザイン機能や実行時の機能へもアクセスできます。

- ClickOnce の配置
- DataGridView コントロールを使用した豊富なデータベースのサポート
- Microsoft® Windows® XP、Microsoft Office、または Microsoft Internet Explorer の外観や動作を備えることができる、ツール バーなどのユーザー インターフェイス要素

詳細については、「ユーザー インターフェイスのデザイン (Visual C#)」および「Windows フォーム」を参照してください。

参照

その他の技術情報

Visual C# によるアプリケーションの作成

コンソール アプリケーションの作成 (Visual C#)

C# を使用して、コマンド ライン コンソールで入力し、出力を表示するアプリケーションを作成できます。このようなアプリケーションは、ユーザー インターフェイスがシンプルであるため、C# 開発を学習するときに最適です。また、コンソール アプリケーションは、ユーザー入力が少ない、またはまったく必要ないユーティリティ プログラムにも適しています。

コンソール アプリケーションを Visual C# で作成するには、[ファイル] メニューの [新規作成] をクリックし、[プロジェクト] を選択します。[Visual C#] フォルダの [コンソール アプリケーション] プロジェクト テンプレートをクリックし、ファイル名を入力して、[OK] をクリックします。

System.Console クラスの使用

コンソールの各文字または 1 行全体の読み取りと書き込みには、Console クラスを使用します。さまざまな形式で出力できます。詳細については、「書式設定の概要」を参照してください。

コマンド ライン引数にアクセスするには、`Main` メソッドに関連付けられているオプションの文字列配列を使用します。詳細については、「コマンド ライン引数 (C# プログラミング ガイド)」を参照してください。

サンプル

- 方法 : コンソール アプリケーション クライアントを作成する
- コマンド ライン パラメータのサンプル

参照

その他の技術情報

Visual C# によるアプリケーションの作成

データのアクセスと表示 (Visual C#)

一般に、Visual C# アプリケーションでは、.NET Framework クラス ライブラリの System.Data および関連する名前空間で公開されている ADO.NET テクノロジを使用してデータベースに接続します。データの操作方法の詳細については、「データへのアクセス (Visual Studio)」を参照してください。

Windows フォーム アプリケーションでは、データベースから取得したデータを表示するときに使用される主なユーザー インターフェイス コントロールは、DataGridView です。詳細については、「DataGridView コントロール (Windows フォーム)」を参照してください。テキスト ボックスやリスト ボックスなどのユーザー インターフェイス コントロールにデータ ソースを接続する方法は、データ バインディングという機能を使用することで格段に単純化されます。コントロールをデータ ソースのフィールドにバインドした場合、一方を変更すると、もう一方に変更内容が自動的に反映されます。詳細については、「Windows フォームでのデータ バインディング」を参照してください。

データベースの作成方法と管理方法、ストアド プロシージャの記述方法、およびその他の関連情報の詳細については、「SQL Server プロジェクト」と「SQL Server チュートリアル」を参照してください。

次のリンクには、Visual Studio を使用してデータにアクセスする方法に関する情報が含まれます。

- Visual Studio でのデータ アプリケーションの作成
- Visual Database Tools

インターネット上の「Data Access and Storage Developer Center」は、新しいドキュメントと例で継続的に更新されています。

参照

その他の技術情報

Visual C# によるアプリケーションの作成

モバイル アプリケーションと組み込みアプリケーションの作成 (Visual C#)

Microsoft Visual Studio 2005 では、Pocket PC や Smartphone など、Windows CE ベースのスマート デバイスを実行するソフトウェアを開発するために、統合的で充実したサポートを実現しています。Visual C# を使用すると、.NET Compact Framework で実行されるマネージ アプリケーションを記述できます。デスクトップ コンピュータの開発と同じコード エディタ、デザイナ、デバッガのインターフェイスを使用できます。選択した言語で使用できるスマート デバイス プロジェクトのいずれかを選択し、コーディングを始めるだけです。

Visual Studio には、スマート デバイスのエミュレータがあります。エミュレータを使用すると、コードを開発コンピュータで実行してデバッグできます。また、エンド ユーザーに配布するための、アプリケーションとリソースを CAB ファイルにパッケージ化するプロセスを容易にするツールもあります。詳細については、「スマート デバイス」を参照してください。

Visual Web Developer を使用すると、ASP.NET に基づいてモバイル Web アプリケーションを開発することもできます。詳細については、「ASP.NET モバイル Web ページの概要」を参照してください。

Windows モバイル テクノロジの最新情報については、Mobile Developer Centerを参照してください。

参照

その他の技術情報

Visual C# によるアプリケーションの作成

Web サービスの作成とアクセス (Visual C#)

XML Web サービスは、疎結合された環境内で、HTTP、XML、XSD、SOAP、WSDL などの標準プロトコルを中心に構築された事前定義のメッセージ交換を使用して、通信するアプリケーション機能を提供します。プロトコルと仕様は公開され、プラットフォーム固有ではないため、XML Web サービスを使用すると、各サービスが同じコンピュータまたはデバイス上には存在しない場合でも、同じコンピュータに存在するかどうかを通信できます。

Web サービスを構築または使用するときに、各仕様を詳しく理解する必要はありません。.NET Framework クラスと Visual Studio ウィザードでは、なじみのあるオブジェクト指向のプログラミング モデルを使用して、XML Web サービスを構築したり、サービスとやり取りしたりできます。

XML Web サービスの詳細については、「XML Web サービスの概要」を参照してください。

ASP.NET を使用した XML Web サービスの作成方法とアクセス方法の詳細については、「ASP.NET を使用した XML Web サービス」を参照してください。

XML Web サービスを簡単に構築し、使用するために Visual Studio に用意されているツールの詳細については、「XML Web サービス (Visual Studio)」を参照してください。

XML Web サービスの作成方法とアクセス方法の詳細については、「マネージ コードを使用した XML Web サービス」を参照してください。

MSDN オンラインの「Web Services Developer Center」にも、その他のドキュメントとリソースがあります。

参照

その他の技術情報

Visual C# によるアプリケーションの作成

コンポーネントの作成 (Visual C#)

ソフトウェア業界でコンポーネントという用語は、1 つ以上のインターフェイスを標準の方法でクライアントに対して公開されている、再利用できるオブジェクトを指すときに使われることがよくあります。コンポーネントは、単一のクラス、または複数クラスとして実装されることがあります。主な要件は、基本のパブリック インターフェイスが定義済みである、ということです。たとえば、ネイティブ Windows プログラミングのコンテキストにあるコンポーネント オブジェクト モデル (COM) では、すべてのコンポーネントに IUnknown インターフェイスとその他の特殊なインターフェイスが実装されている必要があります。

.NET Framework のコンテキストでは、コンポーネントは、IComponent インターフェイスを実装する単一のクラスまたは複数クラス、あるいはそのインターフェイスを実装したクラスから直接的または間接的に派生したクラスです。**IComponent** インターフェイスの既定の基本クラス実装は Component です。

.NET Framework プログラミングで最もよく使用されるコンポーネントとして、Button コントロール (Windows フォーム)、ComboBox コントロール (Windows フォーム) など、Windows フォームに追加する視覚的なコントロールがあります。表示できる形式を持たないコンポーネントには、Timer Control、SerialPort、および ServiceController などがあります。

C# でコンポーネントを作成すると、共通言語仕様 に準拠する多言語で記述されたクライアントからも使用できます。

Visual C# で独自にコンポーネントを作成するには、コンポーネント デザイナ を使用すると、Windows フォームをアセンブルするときと同様に、表示できる形式を持たないコンポーネント クラスをアセンブルできます。詳細については、「チュートリアル : コンポーネント デザイナによる Windows サービス アプリケーションの作成」を参照してください。

Visual Studio を使用したコンポーネントのプログラミングの詳細については、「Visual Studio のコンポーネント」を参照してください。

参照

その他の技術情報

Visual C# によるアプリケーションの作成

Office プラットフォーム上での開発 (Visual C#)

Microsoft Visual Studio 2005 Tools for the Microsoft Office System を使用すると、Microsoft Office ドキュメントおよび Microsoft Office Outlook をマネージ コードでカスタマイズできます。Visual Studio Tools for Office は、Microsoft Office Word および Microsoft Office Excel を Visual Studio 開発環境内のデザイナとしてホストする機能や、データおよびプレゼンテーション オブジェクトに対して直接プログラミングを行う機能、ドキュメント上および [ドキュメント アクション] 作業ウィンドウ内で Windows フォーム コントロールを使用する機能など、新しい機能を Visual Studio に追加します。

Office プラットフォーム上での開発の詳細については、次のトピックを参照してください。

Visual Studio Tools for Office の新機能

Excel のドキュメント レベルのカスタマイズのプログラミングの概要

Word 用のドキュメント レベルのカスタマイズのプログラミングについて

アプリケーション レベルのアドインのプログラミングについて

Office プログラミングの共通タスク

参照

その他の技術情報

Visual C# によるアプリケーションの作成

エンタープライズ向けの開発 (Visual C#)

C# でプログラミングすると、サーバー ベースのアーキテクチャおよびサーバー製品上で実行される、さまざまなエンタープライズ アプリケーションを開発できます。MSDN ライブラリの「Servers and Enterprise Development」には、アーキテクチャに関するガイダンス、パターン、および実践的な情報の他に、次のような Microsoft のサーバー製品を使用するためのドキュメントと技術文書が含まれます。

- Microsoft SQL Server
- Microsoft BizTalk Server
- Microsoft Commerce Server
- Microsoft Content Management Server
- Microsoft Exchange Server
- Microsoft Host Integration Server
- Microsoft Internet Security and Acceleration Server 2000
- Microsoft Business Solutions
- Microsoft MapPoint
- Microsoft Speech Server

参照

その他の技術情報

Visual C# によるアプリケーションの作成

Tablet PC のプログラミング (Visual C#)

Tablet PC は、インク対応、ペン対応、音声対応の各アプリケーション向けに合わせて、Microsoft® Windows® XP で強化されたパーソナル コンピュータです。Tablet PC のソフトウェアとハードウェアを組み合わせることでユーザーとやり取りでき、対話形式で使いやすい操作性を実現します。

Tablet PC プラットフォームには、Windows XP と、Tablet PC で手書きデータと音声データの入出力を可能にする拡張が含まれています。また、このデータを他のコンピュータとやり取りすることもできます。

詳細については、MSDN オンラインの「Windows XP Tablet PC Edition」および「Tablet and Mobile PC Developer Center」を参照してください。

参照

その他の技術情報

Visual C# によるアプリケーションの作成
Tablet and Mobile PC Developer Center

オーディオ、ビデオ、ゲーム、およびグラフィックス (Visual C#)

Visual C# では、DirectX のマネージ コードおよびWindows Media テクノロジに基づいてマルチメディア アプリケーションを作成できます。

Managed DirectX

Microsoft® DirectX® は、Microsoft Windows® オペレーティング システムに組み込まれた、高度なマルチメディア アプリケーション プログラミング インターフェイス (API) のスイートです。DirectX には、ソフトウェアの開発時にハードウェア固有のコードを記述しなくてもハードウェア固有の機能にアクセスできる、Windows ベースのコンピュータ向けに標準的な開発プラットフォームがあります。このテクノロジは 1995 年に初めて導入され、現在では、Windows プラットフォームでマルチメディア アプリケーションを開発するときの標準になっています。

DirectX とは、簡単に説明すると、コンピュータでゲームをプレイしたりビデオを鑑賞するときに、グラフィックス (フルカラーのグラフィックス、ビデオ、3-D アニメーションなど) とサウンド (サラウンド サウンドなど) で高パフォーマンスを実現する Windows テクノロジです。

C# アプリケーションで DirectX を使用する方法の詳細については、MSDN オンラインの「DirectX 9.0 for Managed Code」および Microsoft DirectX Developer Center を参照してください。

Windows メディア プレーヤー

Windows メディア プレーヤーの ActiveX コントロールを C# アプリケーションに使用すると、オーディオとビデオの再生機能を追加できます。Microsoft Windows メディア プレーヤー 10 Software Development Kit (SDK) には、Windows メディア プレーヤーをカスタマイズし、Windows メディア プレーヤー ActiveX コントロールを使用するための情報とツールが用意されています。この SDK には、C# アプリケーションからメディア プレーヤーの ActiveX コントロールを使用する方法を示すドキュメントとコード例が含まれます。

詳細については、MSDN オンラインの「Windows Media Player 10 SDK」を参照してください。

Windows Media Encoder

Windows Media Encoder 9 シリーズ SDK で、C# を使用して次のようなタスクを実行できます。

- **生放送のコンテンツ**。報道機関では、オートメーション APIを使用して、生放送のコンテンツの自動的なキャプチャと放送の計画を立てることができます。たとえば、現地の運輸部門で、問題が発生している複数の路面状況について映像をストリーム放送し、ドライバに道路の混雑状況を警告して、別の経路についてアドバイスできます。
- **バッチ処理のコンテンツ**。サイズの大きなファイルを大量に処理する必要があるメディア制作機関であれば、バッチ処理が役に立ちます。オートメーション APIを使用すると、連続して各ファイルのストリームのキャプチャとエンコーディングを行うことができます。また、オートメーション APIを使用すると、好適なスクリプティング言語と Windows スクリプト ホストでストリーム メディア サービスを管理できます。Windows スクリプト ホストは言語に依存しないホストです。Microsoft Windows® 95 以降、Windows NT、または Windows 2000 の各オペレーティング システムでどのようなスクリプト エンジンを実行している場合でも使用できます。
- **カスタム ユーザー インターフェイスの作成**。インターネット サービス プロバイダ (ISP) の場合、メディア ストリームをキャプチャし、エンコードして、放送するオートメーション APIの機能を使用するインターフェイスを構築できます。または、同じ用途でオートメーション APIに事前定義されているユーザー インターフェイスを使用することもできます。
- **Windows Media Encoder アプリケーションのリモート管理**。オートメーション APIを使用して、リモート コンピュータから Windows Media Encoder アプリケーションの実行、トラブルシューティング、および管理を行うことができます。

詳細については、MSDN オンラインの「Windows Media Encoder 9 Series SDK」を参照してください。また「Programming C#」には、C# で作業するとき含める参照について説明されています。

Windows Media Server

Microsoft® Windows Media® Services 9 シリーズ Software Development Kit (SDK) は、強力なオートメーション ベースのアプリケーション プログラミング インターフェイス (API) です。この API は、Windows Media Services 9 シリーズ アプリケーションの開発者向けにデザインされています。C# でこの SDK を使用すると、Windows Media Server をプログラムで管理し、ユニキャストまたはマルチキャスト対応のネットワークで、デジタル メディア コンテンツをクライアントに送信できます。詳細については、「Windows Media Services 9 Series SDK」を参照してください。

参照

その他の技術情報

Visual C# によるアプリケーションの作成

スタート キットの作成 (Visual C#)

スタート キットには、完全なアプリケーションのコードと、そのアプリケーションを変更または拡張する方法に関するドキュメントが含まれます。Visual C# 2005 には、2 つのスタート キットがあり、[新しいプロジェクト] ダイアログ ボックスからアクセスできます。また、独自のスタート キットを作成して、任意の種類のアプリケーションをすばやく起動および実行できるようにすることも可能です。スタート キットを作成する方法は、通常のプロジェクト テンプレートを作成する場合と基本的には同じです。唯一の違いは、スタート キットには、ドキュメント ファイルが含まれ、スタート キットに基づいてプロジェクトを作成したときに表示されることです。

詳細については、「スタート キットの概要」を参照してください。

参照

その他の技術情報

Visual C# によるアプリケーションの作成

C# プログラミング ガイド

ここでは、C# 言語の重要な機能に関する詳細と、.NET Framework 経由でアクセスできる C# の機能について説明します。

言語セクション

インサイド C# プログラム

Main() とコマンド ライン引数 (C# プログラミング ガイド)

データ型 (C# プログラミング ガイド)

配列 (C# プログラミング ガイド)

文字列 (C# プログラミング ガイド)

ステートメント、式、および演算子 (C# プログラミング ガイド)

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

プロパティ (C# プログラミング ガイド)

インデクサ (C# プログラミング ガイド)

デリゲート (C# プログラミング ガイド)

イベント (C# プログラミング ガイド)

ジェネリック (C# プログラミング ガイド)

反復子 (C# プログラミング ガイド)

名前空間 (C# プログラミング ガイド)

null 許容型 (C# プログラミング ガイド)

unsafe コードとポインタ (C# プログラミング ガイド)

XML ドキュメント コメント (C# プログラミング ガイド)

プラットフォーム セクション

アプリケーション ドメイン (C# プログラミング ガイド)

アセンブリとグローバル アセンブリ キャッシュ (C# プログラミング ガイド)

属性 (C# プログラミング ガイド)

コレクション クラス (C# プログラミング ガイド)

例外と例外処理 (C# プログラミング ガイド)

相互運用性 (C# プログラミング ガイド)

スレッド処理 (C# プログラミング ガイド)

パフォーマンス (C# プログラミング ガイド)

リフレクション (C# プログラミング ガイド)

C# の DLL (C# プログラミング ガイド)

セキュリティ (C# プログラミング ガイド)

参照

その他の技術情報

C# リファレンス

Visual C#

インサイド C# プログラム

ここでは、C# プログラムの全般的な構造について標準的な "Hello, World!" 例を挙げて説明します。

このセクションの内容

- Hello World -- 最初のプログラム (C# プログラミング ガイド)
- C# プログラムの一般構造 (C# プログラミング ガイド)

関連項目

- Visual C# について
- Visual C# への移行
- C# プログラミング ガイド
- C# リファレンス
- Visual C# のサンプル

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1 概要

参照
概念
C# プログラミング ガイド

Hello World -- 最初のプログラム (C# プログラミング ガイド)

次のコンソール プログラムは、従来の "Hello World!" プログラムの C# バージョンで、「`Hello World!`」という文字列を表示します。

C#

```
using System;
// A "Hello World!" program in C#
namespace HelloWorld
{
    class Hello
    {
        static void Main()
        {
            System.Console.WriteLine("Hello World!");
        }
    }
}
```

次に、このプログラムの重要な部分を検討してみることにします。

説明

最初の行はコメントになっています。

C#

```
// A "Hello World!" program in C#
```

「`//`」という文字があると、これ以降その行はコメントになります。次の例に示すように、テキスト ブロックの前後を「`/*`」と「`*/`」で囲んでコメントにすることもできます。

C#

```
/* A "Hello World!" program in C#.
This program displays the string "Hello World!" on the screen. */
```

Main メソッド

C# プログラムには、`Main` メソッドが必要です。このメソッドの中で制御を開始して終了します。`Main` メソッドでは、オブジェクトを作成し、ほかのメソッドを実行します。

`Main` メソッドは静的メソッドで、クラスまたは構造体の中に存在します。前の "Hello World!" の例では、`Hello` というクラスの中に存在しています。次の方法のいずれかで `Main` メソッドを宣言してください。

- **void** 型を返すことができます。

  C#

  ```
  static void Main()
  {
      //...
  }
  ```

- **int** 型を返すこともできます。

  C#

  ```
  static int Main()
  {
  ```

```
    //...
    return 0;
}
```

- どちらの戻り値の型でも、次のように引数を受け取ることができます。

  C#

```
static void Main(string[] args)
{
    //...
}
```

  または

  C#

```
static int Main(string[] args)
{
    //...
    return 0;
}
```

`Main` メソッドのパラメータは `string` の配列で、プログラムの実行時に使用したコマンド ライン引数を表しています。C++ とは異なり、この配列には実行可能 (exe) ファイルの名前は含まれていないことに注意してください。

コマンド ライン引数の使い方の詳細については、「Main() とコマンド ライン引数 (C# プログラミング ガイド)」および「方法 : C# DLL を作成して使用する (C# プログラミング ガイド)」の例を参照してください。

入出力

C# プログラムは、普通、.NET Framework のランタイム ライブラリが提供する入出力サービスを使用します。`System.Console.WriteLine("Hello World!");` ステートメントでは、ランタイム ライブラリの `Console` クラスに含まれる出力メソッドの 1 つである WriteLine メソッドが使用されます。**WriteLine** メソッドは、文字列パラメータを標準出力ストリームに出力し、最後に改行を付け加えます。別の入出力操作には、他の Console メソッドを使用します。`using System;` ディレクティブをプログラムの開始時にインクルードした場合は、完全に修飾せずに System クラスおよびメソッドを直接使用できます。たとえば、`System.Console.Writeline` を指定する代わりに、`Console.WriteLine` を呼び出すことができます。

C#

```
using System;
```

C#

```
Console.WriteLine("Hello World!");
```

入出力メソッドの詳細については、「System.IO」を参照してください。

コンパイルと実行

"Hello World!" プログラムをコンパイルするには、Visual Studio IDE でプロジェクトを作成するか、またはコマンド ラインを使用してください。Visual Studio コマンド プロンプトを使用するか、vsvars32.bat を実行して、Visual C# ツール セットのパスをコマンド プロンプトに設定します。

コマンド ラインからプログラムをコンパイルするには :

- テキスト エディタを使ってソース ファイルを作成し、ファイル名を `Hello.cs` として保存します。C# のソース コード ファイルでは、`.cs` という拡張子を使います。
- コンパイラを起動するには、次のコマンドを入力します。

```
csc Hello.cs
```

プログラムにコンパイル エラーがない場合は、`Hello.exe` というファイルが作成されます。

- プログラムを実行するには、次のコマンドを入力します。

```
Hello
```

C# のコンパイラとオプションの詳細については、「C# コンパイラ オプション」を参照してください。

参照

関連項目

インサイド C# プログラム

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

C# リファレンス

C# プログラムの一般構造 (C# プログラミング ガイド)

C# プログラムは、1 つ以上のファイルで構成できます。各ファイルには、0 以上の名前空間を含めることができます。名前空間には、他の名前空間に加えて、クラス、構造体、インターフェイス、列挙、デリゲートなどの型を含めることができます。次に示すのは、これらの要素をすべて備えた C# プログラムのスケルトンです。

C#

```
// A skeleton of a C# program
using System;
namespace YourNamespace
{
    class YourClass
    {
    }

    struct YourStruct
    {
    }

    interface IYourInterface
    {
    }

    delegate int YourDelegate();

    enum YourEnum
    {
    }

    namespace YourNestedNamespace
    {
        struct YourStruct
        {
        }
    }

    class YourMainClass
    {
        static void Main(string[] args)
        {
            //Your program starts here...
        }
    }
}
```

関連項目

詳細情報

- クラス (C# プログラミング ガイド)
- 構造体 (C# プログラミング ガイド)
- 名前空間 (C# プログラミング ガイド)
- インターフェイス (C# プログラミング ガイド)
- デリゲート (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.2 プログラム構造

- 9.1 コンパイル単位 (名前空間)

参照
関連項目
インサイド C# プログラム
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス
Visual C# のサンプル

Main() とコマンド ライン引数 (C# プログラミング ガイド)

Main メソッドは、オブジェクトを作成し、他のメソッドを呼び出す、プログラムのエントリ ポイントです。C# プログラムでは、エントリ ポイントは 1 つだけに限られます。

C#

```
class TestClass
{
    static void Main(string[] args)
    {
        // Display the number of command line arguments:
        System.Console.WriteLine(args.Length);
    }
}
```

概要

- **Main** メソッドはプログラムのエントリ ポイントで、プログラムの制御を開始および終了する場所です。
- `Main` メソッドは、クラスまたは構造体の内部で宣言されます。これは静的である必要があります。パブリックにはしないでください (上の例では、既定アクセスであるプライベートを受け取ります)。
- 戻り値の型は、void か int のどちらかです。
- **Main** メソッドは、パラメータなしでも宣言できます。
- パラメータは、ゼロから始まるインデックス付きのコマンド ライン引数として読み取ることができます。
- C や C++ と違って、プログラムの名前は、最初のコマンド ライン引数として扱われません。

このセクションの内容

- コマンド ライン引数 (C# プログラミング ガイド)
- 方法 : コマンド ライン引数を表示する (C# プログラミング ガイド)
- 方法 : foreach を使用してコマンド ライン引数にアクセスする (C# プログラミング ガイド)
- Main() の戻り値 (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.1 Hello world

参照

関連項目

インサイド C# プログラム

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

コマンド ライン引数 (C# プログラミング ガイド)

Main メソッドでは、引数を使うことができます。その場合、次のいずれかの形式を使用します。

C#

```
static int Main(string[] args)
```

C#

```
static void Main(string[] args)
```

Main メソッドのパラメータは String の配列で、コマンド ライン引数を表しています。通常は、Length プロパティを調べて引数があるかどうかを確認します。次はその例です。

C#

```
if (args.Length == 0)
{
    System.Console.WriteLine("Please enter a numeric argument.");
    return 1;
}
```

また、Convert クラスまたは Parse メソッドを使って、文字列型の引数を数値型に変換できます。たとえば、次のステートメントでは、Int64 クラスの Parse メソッドを使用して文字列を long 値に変換します。

```
long num = Int64.Parse(args[0]);
```

C# の long 型を使うこともできます。これは Int64 のエイリアスです。

```
long num = long.Parse(args[0]);
```

また、同じ変換に Convert クラスの ToInt64 メソッドを使うこともできます。

```
long num = Convert.ToInt64(s);
```

詳細については、「Parse」および「Convert」を参照してください。

使用例

次の例のプログラムは、実行時に引数を 1 つ受け取り、整数に変換し、その値の階乗を計算しています。引数がない場合は、プログラムの正しい使用方法を説明するメッセージを表示します。

C#

```
// arguments: 3
```

C#

```
public class Functions
{
    public static long Factorial(int n)
    {
        if (n < 0) { return -1; }    //error result - undefined
        if (n > 256) { return -2; }  //error result - input is too big

        if (n == 0) { return 1; }
```

```
        // Calculate the factorial iteratively rather than recursively:

        long tempResult = 1;
        for (int i = 1; i <= n; i++)
        {
            tempResult *= i;
        }
        return tempResult;
    }
}
```

C#

```
class MainClass
{
    static int Main(string[] args)
    {
        // Test if input arguments were supplied:
        if (args.Length == 0)
        {
            System.Console.WriteLine("Please enter a numeric argument.");
            System.Console.WriteLine("Usage: Factorial <num>");
            return 1;
        }

        try
        {
            // Convert the input arguments to numbers:
            int num = int.Parse(args[0]);

            System.Console.WriteLine("The Factorial of {0} is {1}.", num, Functions.Factori
al(num));
            return 0;
        }
        catch (System.FormatException)
        {
            System.Console.WriteLine("Please enter a numeric argument.");
            System.Console.WriteLine("Usage: Factorial <num>");
            return 1;
        }
    }
}
```

出力

```
The Factorial of 3 is 6.
```

説明

`Factorial.exe` という名前のプログラムの実行例を次に 2 つ示します。

実行サンプル 1 :

次のコマンド ラインを入力します。

```
Factorial 10
```

次の結果が得られます。

```
The Factorial of 10 is 3628800.
```

実行サンプル 2 :

次のコマンド ラインを入力します。

```
Factorial
```

次の結果が得られます。

```
Please enter a numeric argument.
```

```
Usage: Factorial <num>
```

コマンド ライン引数の使用方法の例については、「方法 : C# DLL を作成して使用する (C# プログラミング ガイド)」の例を参照してください。

参照

処理手順

方法 : コマンド ライン引数を表示する (C# プログラミング ガイド)
方法 : foreach を使用してコマンド ライン引数にアクセスする (C# プログラミング ガイド)

関連項目

Main() とコマンド ライン引数 (C# プログラミング ガイド)
Main() の戻り値 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

方法 : コマンド ライン引数を表示する (C# プログラミング ガイド)

実行可能ファイルに対してコマンド ラインで指定した引数には、省略可能なパラメータを介して `Main` からアクセスできます。引数は、文字列の配列の形式で指定します。配列の各要素には、1 つの引数が格納されます。引数間の空白は削除されます。架空の実行可能ファイルを呼び出すためのコマンド ラインの例を次に示します。

コマンド ラインでの入力	Main に渡される文字列の配列
executable.exe a b c	"a" "b" "c"
executable.exe one two	"one" "two"
executable.exe "one two" three	"one two" "three"

使用例

この例は、コマンド ライン アプリケーションに渡されるコマンド ライン引数を示しています。次の出力は、上の表の最初のエントリに対するものです。

C#

```
class CommandLine
{
    static void Main(string[] args)
    {
        // The Length property provides the number of array elements
        System.Console.WriteLine("parameter count = {0}", args.Length);

        for (int i = 0; i < args.Length; i++)
        {
            System.Console.WriteLine("Arg[{0}] = [{1}]", i, args[i]);
        }
    }
}
```

出力

```
parameter count = 3
Arg[0] = [a]
Arg[1] = [b]
Arg[2] = [c]
```

参照

処理手順

方法 : foreach を使用してコマンド ライン引数にアクセスする (C# プログラミング ガイド)

関連項目

Main() とコマンド ライン引数 (C# プログラミング ガイド)

Main() の戻り値 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

方法 : foreach を使用してコマンド ライン引数にアクセスする (C# プログラミング ガイド)

配列の反復処理には、この例で示すように foreach ステートメントを使用する方法もあります。**foreach** ステートメントは、配列、.NET Framework コレクション クラス、または IEnumerable インターフェイスを実装する任意のクラスや構造体の反復処理に使用できます。

使用例

この例では、**foreach** を使用したコマンド ライン引数の出力方法を示します。

C#

```
// arguments: John Paul Mary
```

C#

```
class CommandLine2
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Number of command line parameters = {0}", args.Length);

        foreach (string s in args)
        {
            System.Console.WriteLine(s);
        }
    }
}
```

出力

```
Number of command line parameters = 3
John
Paul
Mary
```

参照

処理手順

方法 : コマンド ライン引数を表示する (C# プログラミング ガイド)

関連項目

foreach、in (C# リファレンス)
Main() とコマンド ライン引数 (C# プログラミング ガイド)
Main() の戻り値 (C# プログラミング ガイド)
Array
System.Collections

概念

C# プログラミング ガイド

Main() の戻り値 (C# プログラミング ガイド)

Main メソッドは、**void** 型として宣言できます。

C#

```
static void Main()
{
    //...
}
```

int 型を返すこともできます。

C#

```
static int Main()
{
    //...
    return 0;
}
```

Main の戻り値を使用しない場合は、**void** を返します。そうすると、コードをわずかに簡素化できますが、整数を返すと、他のプログラムや、実行可能ファイルを呼び出すスクリプトにステータス情報を関連付けることができます。Main の戻り値の使用例を次に示します。

使用例

次の例では、バッチ ファイルを使用してプログラムを実行し、Main 関数の戻り値をテストします。プログラムを Windows で実行すると、Main 関数から返された値が、ERRORLEVEL という環境変数に格納されます。ERRORLEVEL 変数を検査することで、バッチ ファイルが実行結果を判断できます。従来より、ゼロの戻り値は、実行が成功したことを示します。次の例は、Main 関数からゼロを返す簡単なプログラムです。

C#

```
class MainReturnValTest
{
    static int Main()
    {
        //...
        return 0;
    }
}
```

この例ではバッチ ファイルを使用するため、このコードは、「方法 : 環境変数を設定する」で示すように、コマンド ラインからコンパイルするのが最適です。

次に、バッチ ファイルを使用して、上のコード例から生成された実行可能ファイルを呼び出します。このコードはゼロを返すため、バッチ ファイルは、成功を報告しますが、ゼロ以外の値を返すように上のコードを変更し、再コンパイルした後にバッチ ファイルを実行すると、失敗が示されます。

```
rem test.bat
@echo off
MainReturnValueTest
@if "%ERRORLEVEL%" == "0" goto good

:fail
    echo Execution Failed
    echo return value = %ERRORLEVEL%
    goto end

:good
    echo Execution Succeded
    echo return value = %ERRORLEVEL%
    goto end
```

```
:end
```

出力例

```
Execution Succeded

return value = 0
```

参照

処理手順

方法 : コマンド ライン引数を表示する (C# プログラミング ガイド)

方法 : foreach を使用してコマンド ライン引数にアクセスする (C# プログラミング ガイド)

関連項目

Main() とコマンド ライン引数 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

データ型 (C# プログラミング ガイド)

C# は厳密に型指定された言語であるため、すべての変数とオブジェクトには、宣言された型が必要です。

データ型の概要

データ型は、次のいずれかとして表現できます。

- 組み込みのデータ型 (**int** や **char** など)、または
- ユーザー定義のデータ型 (**class** または **interface**)
- データ型は、次のいずれかとして定義することもできます。
- 値を格納する 値型 (C# リファレンス)、または
- 実際のデータへの参照を格納する 参照型 (C# リファレンス)

関連項目

詳細情報

- キャスト (C# プログラミング ガイド)
- ボックス化とボックス化解除 (C# プログラミング ガイド)
- 型 (C# リファレンス)
- オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

C# 言語仕様

型の詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 3.8 名前空間と型の名前
- 4.1 値型
- 4.2 参照型
- 4.3 ボックス化とボックス化解除

参照
関連項目
各言語のデータ型の比較
整数型の一覧表 (C# リファレンス)
概念
C# プログラミング ガイド
XML データ型の変換
その他の技術情報
C# リファレンス

キャスト (C# プログラミング ガイド)

データ型間の変換は、キャストを明示的に使用して実行できますが、場合によっては、暗黙の変換を実行できることもあります。次に例を示します。

C#

```
static void TestCasting()
{
    int i = 10;
    float f = 0;
    f = i;  // An implicit conversion, no data will be lost.
    f = 0.5F;
    i = (int)f;  // An explicit conversion. Information will be lost.
}
```

キャストは、1 つの型から別の型への変換演算子を明示的に呼び出します。このような変換演算子が定義されていない場合、キャストは失敗します。カスタム変換演算子を記述すると、ユーザー定義の型の間で変換できます。変換演算子を定義する方法の詳細については、「explicit (C# リファレンス)」および「implicit (C# リファレンス)」を参照してください。

使用例

double を int にキャストするプログラムは、次のとおりです。このプログラムは、キャストしないとコンパイルできません。

C#

```
class Test
{
    static void Main()
    {
        double x = 1234.7;
        int a;
        a = (int)x;  // cast double to int
        System.Console.WriteLine(a);
    }
}
```

出力

```
1234
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 7.6.6 キャスト式
- 6.1 暗黙の変換
- 6.2 明示的な変換

参照

関連項目

データ型 (C# プログラミング ガイド)
() 演算子 (C# リファレンス)
explicit (C# リファレンス)
implicit (C# リファレンス)

概念

C# プログラミング ガイド
変換演算子 (C# プログラミング ガイド)
一般的な型変換
明示的な変換
エクスポート時の型の変換

ボックス化とボックス化解除 (C# プログラミング ガイド)

ボックス化およびボックス化解除を使用して、値型をオブジェクトとして扱うことができます。値型をボックス化すると、Object 参照型のインスタンスの内部に値型がパッケージ化されます。これにより、値型をガベージ コレクション ヒープに格納できます。ボックス化解除すると、値型がオブジェクトから抽出されます。次の例では、整数の変数 `i` をボックス化し、オブジェクト `o` に代入しています。

C#

```
int i = 123;
object o = (object)i;  // boxing
```

次に、オブジェクト `o` は、次のようにボックス化解除し、整数の変数 `i` に代入できます。

C#

```
o = 123;
i = (int)o;  // unboxing
```

パフォーマンス

簡単な代入と比べて、ボックス化およびボックス化解除は負荷の大きいプロセスです。値型をボックス化するときは、完全に新しいオブジェクトを割り当てて構築する必要があります。ボックス化ほどではありませんが、ボックス化解除に必要なキャストも大きな負荷がかかります。詳細については、「パフォーマンス (C# プログラミング ガイド)」を参照してください。

関連項目

詳細情報

- ボックス化
- ボックス化解除
- 参照型
- 値型

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 4.3 ボックス化とボックス化解除

参照

概念

C# プログラミング ガイド
ボックス化変換された型

ボックス化変換 (C# プログラミング ガイド)

ボックス化は、値型をガベージ コレクション ヒープに格納するために使用します。ボックス化とは、値型 (C# リファレンス) から **object** 型、またはその値型によって実装されている任意のインターフェイス型への暗黙の変換のことです。値型をボックス化すると、オブジェクト インスタンスがヒープに割り当てられ、値が新しいオブジェクトにコピーされます。

値型の変数の宣言例を次に示します。

C#

```
int i = 123;
```

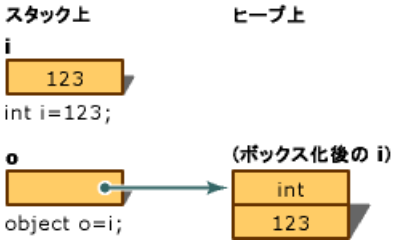
次のステートメントは、変数 `i` にボックス化を暗黙的に適用します。

C#

```
object o = i;  // implicit boxing
```

このステートメントによって、ヒープ上にある **int** 型の値を参照するオブジェクト参照 `o` がスタック上に作成されます。この値は、変数 `i` に割り当てられた値型の値のコピーです。2 つの変数 `i` と `o` の違いを次の図に示します。

ボックス化



次の例に示すように、明示的にボックス化を実行することもできますが、明示的なボックス化は不要です。

C#

```
int i = 123;
object o = (object)i;  // explicit boxing
```

例

ここでは、ボックス化によって整数の変数 `i` をオブジェクト `o` に変換する例を示します。変換後に、変数 `i` の値を `123` から `456` に変更します。この例は、元の値型とボックス化されたオブジェクトが別個のメモリ位置を使用するため、それぞれ別々の値を格納できることを示しています。

C#

```
class TestBoxing
{
    static void Main()
    {
        int i = 123;
        object o = i;  // implicit boxing

        i = 456;  // change the contents of i

        System.Console.WriteLine("The value-type value = {0}", i);
        System.Console.WriteLine("The object-type value = {0}", o);
    }
}
```

出力

```
The value-type value = 456

The object-type value = 123
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 4.3.1 ボックス化変換

参照

概念

C# プログラミング ガイド
ボックス化とボックス化解除 (C# プログラミング ガイド)
ボックス化解除変換 (C# プログラミング ガイド)
ボックス化変換された型

ボックス化解除変換 (C# プログラミング ガイド)

ボックス化解除とは、**object** 型から値型へ、またはインターフェイス型からそのインターフェイスを実装している値型への明示的な変換のことです。ボックス化解除では、次の処理が行われます。

- オブジェクト インスタンスが、指定された値型のボックス化された値かどうかを確認します。
- インスタンスの値を値型の変数にコピーします。

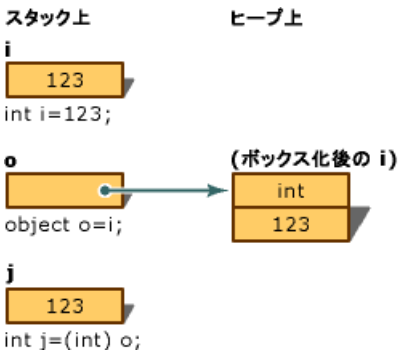
次のステートメントに、ボックス化およびボックス化解除の両方を示します。

C#

```
int i = 123;      // a value type
object o = i;     // boxing
int j = (int)o;  // unboxing
```

上のステートメントの結果は、次の図に示すとおりです。

ボックス化解除



実行時に値型のボックス化解除を成功させるには、ボックス化解除の対象項目が、同じ値型のインスタンスのボックス化によって既に作成済みのオブジェクトへの参照である必要があります。**null**、または互換性のない値型への参照をボックス化解除しようとすると、InvalidCastException が発生します。

例

次の例は、無効なボックス化解除の結果、**InvalidCastException** が発生する場合を示しています。**try** と **catch** を使用すると、エラーの発生時にエラー メッセージが表示されます。

C#

```
class TestUnboxing
{
    static void Main()
    {
        int i = 123;
        object o = i;  // implicit boxing

        try
        {
            int j = (short)o;  // attempt to unbox

            System.Console.WriteLine("Unboxing OK.");
        }
        catch (System.InvalidCastException e)
        {
            System.Console.WriteLine("{0} Error: Incorrect unboxing.", e.Message);
        }
    }
}
```

出力

```
Specified cast is not valid. Error: Incorrect unboxing.
```

エラーを修正するには、次のステートメントを変更します。

```
int j = (short) o;
```

この行を次のように変更します。

```
int j = (int) o;
```

ステートメントを変更すると、変換が実行されて次の出力が得られます。

```
Unboxing OK.
```

参照

概念

C# プログラミング ガイド
ボックス化とボックス化解除 (C# プログラミング ガイド)
ボックス化変換 (C# プログラミング ガイド)

配列 (C# プログラミング ガイド)

配列とは、同じ型の複数の変数を含むデータ構造です。配列は、次のように型を使用して宣言します。

```
type[] arrayName;
```

次の例では、1 次元配列、多次元配列、およびジャグ配列を作成しています。

C#

```
class TestArraysClass
{
    static void Main()
    {
        // Declare a single-dimensional array
        int[] array1 = new int[5];

        // Declare and set array element values
        int[] array2 = new int[] { 1, 3, 5, 7, 9 };

        // Alternative syntax
        int[] array3 = { 1, 2, 3, 4, 5, 6 };

        // Declare a two dimensional array
        int[,] multiDimensionalArray1 = new int[2, 3];

        // Declare and set array element values
        int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

        // Declare a jagged array
        int[][] jaggedArray = new int[6][];

        // Set the values of the first array in the jagged array structure
        jaggedArray[0] = new int[4] { 1, 2, 3, 4 };
    }
}
```

配列の概要

配列には、次の特徴があります。

- 配列は、1 次元配列、多次元配列、またはジャグ配列のいずれかになります。
- 数値配列要素の既定値はゼロに設定され、参照要素は null に設定されます。
- ジャグ配列は配列の配列です。そのため、配列要素は参照型で、**null** に初期化されます。
- 配列には、ゼロから始まるインデックスが付けられます。`n` 個の要素を含む配列には、`0` から `n-1` までのインデックスが付けられます。
- 配列の要素および配列型は、どのような型でもかまいません。
- 配列型は、抽象基本型 Array から派生した参照型です。この型は IEnumerable と IEnumerable を実装するので、C# のすべての配列で foreach 反復処理を使用できます。

関連項目

- オブジェクトとしての配列 (C# プログラミング ガイド)
- 配列での foreach の使用 (C# プログラミング ガイド)
- パラメータとしての配列の受け渡し (C# プログラミング ガイド)
- ref と out を使用した配列の引き渡し (C# プログラミング ガイド)
- 配列のサンプル

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.8 配列
- 12 配列

参照

関連項目

配列の使用方法のガイドライン

概念

C# プログラミング ガイド
コレクション クラス (C# プログラミング ガイド)
Array コレクション型
共通型システムの配列

オブジェクトとしての配列 (C# プログラミング ガイド)

C# の配列はオブジェクトそのものであり、C や C++ の場合のように、単なるアドレス指定可能な連続メモリ領域ではありません。Array は、すべての配列型の抽象基本型です。**Array** のプロパティとその他のクラス メンバを使用できます。この例としては、Length プロパティを使用して配列の長さを取得する場合などがあります。`numbers` 配列の長さ `5` を `lengthOfNumbers` という変数に代入するコードは、次のようになります。

C#

```
int[] numbers = { 1, 2, 3, 4, 5 };
int lengthOfNumbers = numbers.Length;
```

System.Array クラスには、配列の並べ替え、検索、コピーを行うための、多くの便利なメソッドやプロパティが他にも用意されています。

使用例

次の例では、Rank プロパティを使用して、配列の次元数を表示します。

C#

```
class TestArraysClass
{
    static void Main()
    {
        // Declare and initialize an array:
        int[,] theArray = new int[5, 10];
        System.Console.WriteLine("The array has {0} dimensions.", theArray.Rank);
    }
}
```

出力

```
The array has 2 dimensions.
```

参照

関連項目

1 次元配列 (C# プログラミング ガイド)
多次元配列 (C# プログラミング ガイド)
ジャグ配列 (C# プログラミング ガイド)

概念

C# プログラミング ガイド
配列 (C# プログラミング ガイド)

1 次元配列 (C# プログラミング ガイド)

5 個の整数値を含む配列は、次の例のように宣言できます。

C#

```
int[] array = new int[5];
```

この配列には、`array[0]` から `array[4]` までの要素が含まれています。new 演算子を使って配列を作成すると、配列の要素は、既定値で初期化されます。この例では、配列要素はすべて 0 に初期化されます。

文字列を格納する配列も、同様の方法で宣言できます。次に例を示します。

C#

```
string[] stringArray = new string[6];
```

配列の初期化

宣言時に配列を初期化できます。その場合、ランク指定子は、初期化リスト内の要素数で既に与えられているので必要ありません。次に例を示します。

C#

```
int[] array1 = new int[5] { 1, 3, 5, 7, 9 };
```

文字列配列も、同じ方法で初期化できます。次に示す文字列配列の宣言では、各配列要素を曜日の名前で初期化しています。

C#

```
string[] weekDays = new string[] { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```

宣言時に配列を初期化するときは、次の短縮形を使用できます。

C#

```
int[] array2 = { 1, 3, 5, 7, 9 };
string[] weekDays2 = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```

初期化せずに配列変数を宣言することはできますが、そのような変数に配列を代入するときは、**new** 演算子を使用する必要があります。次に例を示します。

C#

```
int[] array3;
array3 = new int[] { 1, 3, 5, 7, 9 };   // OK
//array3 = {1, 3, 5, 7, 9};   // Error
```

値型と参照型の配列

配列宣言の例を次に示します。

C#

```
SomeType[] array4 = new SomeType[10];
```

このステートメントの結果は、`SomeType` が値型か参照型かによって異なります。値型の場合は、`SomeType` 型のインスタンス 10 個の配列が作成されます。`SomeType` が参照型の場合は、10 個の要素から成る配列が作成されて、各要素は null 参照で初期化されます。

値型と参照型の詳細については、「型 (C# リファレンス)」を参照してください。

参照

関連項目

多次元配列 (C# プログラミング ガイド)

ジャグ配列 (C# プログラミング ガイド)

Array

概念

C# プログラミング ガイド

配列 (C# プログラミング ガイド)

多次元配列 (C# プログラミング ガイド)

複数の次元で構成される配列を作成できます。たとえば、次の宣言では、4 つの行と 2 つの列から成る 2 次元配列が作成されます。

C#

```
int[,] array = new int[4, 2];
```

また、次の宣言では、大きさが 4、2、3 の 3 つの次元を持つ配列が作成されます。

C#

```
int[, ,] array1 = new int[4, 2, 3];
```

配列の初期化

次の例に示すように、宣言時に配列を初期化できます。

C#

```
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
int[, ,] array3D = new int[,,] { { { 1, 2, 3 } }, { { 4, 5, 6 } } };
```

ランクを指定せずに配列を初期化することもできます。

C#

```
int[,] array4 = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

初期化せずに配列変数を宣言する場合は、**new** 演算子を使用して配列を変数に割り当てます。次に例を示します。

C#

```
int[,] array5;
array5 = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };   // OK
//array5 = {{1,2}, {3,4}, {5,6}, {7,8}};   // Error
```

配列の要素に変数を代入することもできます。次に例を示します。

C#

```
array5[2, 1] = 25;
```

配列変数を既定値に初期化するコード例を次に示します (ジャグ配列の場合を除きます)。

C#

```
int[,] array6 = new int[10, 10];
```

参照

関連項目

1 次元配列 (C# プログラミング ガイド)

ジャグ配列 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

配列 (C# プログラミング ガイド)

ジャグ配列 (C# プログラミング ガイド)

ジャグ配列とは、その要素も配列である配列です。ジャグ配列の要素は、次元やサイズが異なっていてもかまいません。ジャグ配列は、"配列の配列" と呼ばれることもあります。次の例で、ジャグ配列の宣言、初期化、アクセスの各方法について説明します。

次に示すのは、3 つの要素を持つ 1 次元配列の宣言で、各要素は整数の 1 次元配列です。

C#

```
int[][] jaggedArray = new int[3][];
```

`jaggedArray` を使用するには、先に要素を初期化する必要があります。要素の初期化は、次の方法で行うことができます。

C#

```
jaggedArray[0] = new int[5];
jaggedArray[1] = new int[4];
jaggedArray[2] = new int[2];
```

各要素は、整数の 1 次元配列です。1 番目の要素は 5 個の整数値の配列、2 番目の要素は 4 個の整数値の配列、そして 3 番目の要素は 2 個の整数値の配列です。

初期化子を使って配列要素に値を格納することもできます。この場合、配列のサイズは必要ありません。たとえば、次のようにします。

C#

```
jaggedArray[0] = new int[] { 1, 3, 5, 7, 9 };
jaggedArray[1] = new int[] { 0, 2, 4, 6 };
jaggedArray[2] = new int[] { 11, 22 };
```

次に示すように、宣言時に配列を初期化することもできます。

C#

```
    int[][] jaggedArray2 = new int[][]
{
    new int[] {1,3,5,7,9},
    new int[] {0,2,4,6},
    new int[] {11,22}
};
```

次の簡単な形式を使用できます。要素の初期化では **new** 演算子を省略できないことに注意してください。これは、要素に既定の初期化はないためです。

C#

```
    int[][] jaggedArray3 =
{
    new int[] {1,3,5,7,9},
    new int[] {0,2,4,6},
    new int[] {11,22}
};
```

ジャグ配列は配列の配列です。そのため、配列要素は参照型で、**null** に初期化されます。

各配列要素は、次の例に示す方法で参照できます。

C#

```
// Assign 77 to the second element ([1]) of the first array ([0]):
jaggedArray3[0][1] = 77;
```

```
// Assign 88 to the second element ([1]) of the third array ([2]):
jaggedArray3[2][1] = 88;
```

ジャグ配列と多次元配列を併用できます。次に示すのは、サイズが異なる 2 次元配列要素を含む 1 次元のジャグ配列を宣言および初期化する例です。

C#

```
int[][,] jaggedArray4 = new int[3][,]
{
    new int[,] { {1,3}, {5,7} },
    new int[,] { {0,2}, {4,6}, {8,10} },
    new int[,] { {11,22}, {99,88}, {0,9} }
};
```

各要素には、次の例に示す方法でアクセスできます。この例では、最初の配列の要素 `[1,0]` の値 (5) を表示しています。

C#

```
System.Console.Write("{0}", jaggedArray4[0][1, 0]);
```

メソッド `Length` は、ジャグ配列に含まれる配列数を返します。たとえば、前の配列を宣言した状態で、次の行を指定したとします。

C#

```
System.Console.WriteLine(jaggedArray4.Length);
```

この行は、値 3 を返します。

使用例

次の例では、配列を要素とする配列を作成します。各配列要素のサイズは同じではありません。

C#

```
class ArrayTest
{
    static void Main()
    {
        // Declare the array of two elements:
        int[][] arr = new int[2][];

        // Initialize the elements:
        arr[0] = new int[5] { 1, 3, 5, 7, 9 };
        arr[1] = new int[4] { 2, 4, 6, 8 };

        // Display the array elements:
        for (int i = 0; i < arr.Length; i++)
        {
            System.Console.Write("Element({0}): ", i);

            for (int j = 0; j < arr[i].Length; j++)
            {
                System.Console.Write("{0}{1}", arr[i][j], j == (arr[i].Length - 1) ? "" : "
");
            }
            System.Console.WriteLine();
        }
    }
}
```

出力

```
Element(0): 1 3 5 7 9

Element(1): 2 4 6 8
```

参照

関連項目

1 次元配列 (C# プログラミング ガイド)

多次元配列 (C# プログラミング ガイド)

Array

概念

C# プログラミング ガイド

配列 (C# プログラミング ガイド)

配列での foreach の使用 (C# プログラミング ガイド)

C# には、foreach ステートメントも用意されています。このステートメントでは、配列要素の反復処理を簡単に、また安全に行うことができます。たとえば、次のコードは `numbers` という配列を作成し、**foreach** ステートメントで配列内の反復処理を行います。

C#

```
int[] numbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };
foreach (int i in numbers)
{
    System.Console.WriteLine(i);
}
```

多次元配列を使用する場合は、同じメソッドを使用して配列要素を反復処理できます。次に例を示します。

C#

```
int[,] numbers2D = new int[3, 2] { { 9, 99 }, { 3, 33 }, { 5, 55 } };
foreach (int i in numbers2D)
{
    System.Console.Write("{0} ", i);
}
```

この例の出力は次のとおりです。

```
9 99 3 33 5 55
```

ただし、多次元配列では、入れ子になった for ループを使用した方が配列要素をより厳密に制御できます。

参照

関連項目

1 次元配列 (C# プログラミング ガイド)
多次元配列 (C# プログラミング ガイド)
ジャグ配列 (C# プログラミング ガイド)
Array

概念

C# プログラミング ガイド
配列 (C# プログラミング ガイド)

パラメータとしての配列の受け渡し (C# プログラミング ガイド)

配列は、パラメータとしてメソッドに渡すことができます。配列は参照型であるため、メソッドで配列要素の値を変更できます。

パラメータとしての 1 次元配列の受け渡し

初期化された 1 次元配列をメソッドに渡すことができます。次に例を示します。

C#

```
PrintArray(theArray);
```

上の行で呼び出されるメソッドは、次のように定義できます。

C#

```
void PrintArray(int[] arr)
{
    // method code
}
```

配列の初期化と受け渡しを 1 ステップで行うこともできます。次に例を示します。

C#

```
PrintArray(new int[] { 1, 3, 5, 7, 9 });
```

例 1

次の例では、文字列配列を初期化し、パラメータとして `PrintArray` メソッドに渡しています。このメソッドでは、渡された配列の要素を表示します。

C#

```
class ArrayClass
{
    static void PrintArray(string[] arr)
    {
        for (int i = 0; i < arr.Length; i++)
        {
            System.Console.Write(arr[i] + "{0}", i < arr.Length - 1 ? " " : "");
        }
        System.Console.WriteLine();
    }

    static void Main()
    {
        // Declare and initialize an array:
        string[] weekDays = new string[] { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
};

        // Pass the array as a parameter:
        PrintArray(weekDays);
    }
}
```

出力 1

```
Sun Mon Tue Wed Thu Fri Sat
```

パラメータとしての多次元配列の受け渡し

初期化された多次元配列をメソッドに渡すことができます。たとえば、次の `theArray` が 2 次元配列だとします。

C#

```
PrintArray(theArray);
```

上の行で呼び出されるメソッドは、次のように定義できます。

C#

```
void PrintArray(int[,] arr)
{
    // method code
}
```

配列の初期化と受け渡しを 1 ステップで行うこともできます。次に例を示します。

C#

```
PrintArray(new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } });
```

例 2

次の例では、2 次元配列を初期化し、`PrintArray` メソッドに渡しています。このメソッドでは、渡された配列の要素を表示します。

C#

```
class ArrayClass2D
{
    static void PrintArray(int[,] arr)
    {
        // Display the array elements:
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                System.Console.WriteLine("Element({0},{1})={2}", i, j, arr[i, j]);
            }
        }
    }
    static void Main()
    {
        // Pass the array as a parameter:
        PrintArray(new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } });
    }
}
```

出力 2

```
Element(0,0)=1
Element(0,1)=2
Element(1,0)=3
Element(1,1)=4
Element(2,0)=5
Element(2,1)=6
Element(3,0)=7
Element(3,1)=8
```

参照

関連項目

1 次元配列 (C# プログラミング ガイド)
多次元配列 (C# プログラミング ガイド)
ジャグ配列 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

配列 (C# プログラミング ガイド)

ref と out を使用した配列の引き渡し (C# プログラミング ガイド)

すべての out パラメータと同じように、配列型の **out** パラメータも使用する前に代入される必要があります。つまり、呼び出される側で代入する必要があります。次に例を示します。

C#

```
static void TestMethod1(out int[] arr)
{
    arr = new int[10];   // definite assignment of arr
}
```

すべての ref パラメータと同じように、配列型の **ref** パラメータも、呼び出し側で明示的に代入する必要があります。したがって、呼び出される側で明示的に代入する必要はありません。配列型の **ref** パラメータは、呼び出しの結果として変更される場合があります。たとえば、配列に null 値が代入されたり、別の配列で初期化されたりする場合があります。次に例を示します。

C#

```
static void TestMethod2(ref int[] arr)
{
    arr = new int[10];   // arr initialized to a different array
}
```

次の 2 つの例は、メソッドに配列を渡すときに使われる **out** と **ref** の違いを示しています。

例 1

この例では、配列 `theArray` は呼び出し元 (`Main` メソッド) で宣言されて、`FillArray` メソッドで初期化されます。その後、配列要素は呼び出し元に返されて表示されます。

C#

```
class TestOut
{
    static void FillArray(out int[] arr)
    {
        // Initialize the array:
        arr = new int[5] { 1, 2, 3, 4, 5 };
    }

    static void Main()
    {
        int[] theArray; // Initialization is not required

        // Pass the array to the callee using out:
        FillArray(out theArray);

        // Display the array elements:
        System.Console.WriteLine("Array elements are:");
        for (int i = 0; i < theArray.Length; i++)
        {
            System.Console.Write(theArray[i] + " ");
        }
    }
}
```

出力 1

```
Array elements are:

1 2 3 4 5
```

例 2

この例では、配列 `theArray` は呼び出し元 (`Main` メソッド) で初期化された後、**ref** パラメータを使って `FillArray` メソッドに渡されます。一部の配列要素は、`FillArray` メソッドの中で変更されます。その後、配列要素は呼び出し元に返されて表示されます。

C#

```
class TestRef
{
    static void FillArray(ref int[] arr)
    {
        // Create the array on demand:
        if (arr == null)
        {
            arr = new int[10];
        }
        // Fill the array:
        arr[0] = 1111;
        arr[4] = 5555;
    }

    static void Main()
    {
        // Initialize the array:
        int[] theArray = { 1, 2, 3, 4, 5 };

        // Pass the array using ref:
        FillArray(ref theArray);

        // Display the updated array:
        System.Console.WriteLine("Array elements are:");
        for (int i = 0; i < theArray.Length; i++)
        {
            System.Console.Write(theArray[i] + " ");
        }
    }
}
```

出力 2

```
Array elements are:

1111 2 3 4 5555
```

参照

関連項目

1 次元配列 (C# プログラミング ガイド)
多次元配列 (C# プログラミング ガイド)
ジャグ配列 (C# プログラミング ガイド)

概念

C# プログラミング ガイド
配列 (C# プログラミング ガイド)

文字列 (C# プログラミング ガイド)

次のセクションでは、String クラスのエイリアスである **string** データ型について説明します。

このセクションの内容

文字列の使用 (C# プログラミング ガイド)

方法 : Split メソッドを使用して文字列を解析する (C# プログラミング ガイド)

方法 : String のメソッドを使用して文字列を検索する (C# プログラミング ガイド)

方法 : 正規表現を使用して文字列を検索する (C# プログラミング ガイド)

方法 : 複数の文字列を連結する (C# プログラミング ガイド)

方法 : 文字列の内容を変更する (C# プログラミング ガイド)

関連するセクション

基本的な文字列操作

文字列の比較

参照

概念

C# プログラミング ガイド

文字列の使用 (C# プログラミング ガイド)

C# の文字列は、**string** キーワードを使用して宣言された文字配列です。リテラル文字列を宣言するときには、次の例に示すように引用符を使用します。

C#

```
string s = "Hello, World!";
```

次のように部分文字列を抽出して文字列を連結できます。

C#

```
string s1 = "orange";
string s2 = "red";

s1 += s2;
System.Console.WriteLine(s1);  // outputs "orangered"

s1 = s1.Substring(2, 5);
System.Console.WriteLine(s1);  // outputs "anger"
```

文字列オブジェクトは変更不可です。つまり、作成した文字列オブジェクトは変更できません。文字列を操作するメソッドは、実際には新しい文字列オブジェクトを返します。上記の例では、`s1` と `s2` の内容は連結され、1 つの文字列が形成されます。`"orange"` と `"red"` を含む 2 つの文字列は、どちらも変更されません。**+=** 演算子で、連結した内容を含む新しい文字列が作成されます。この結果、`s1` から異なる文字列全体が参照されるようになりました。`"orange"` のみを含む文字列は存在しますが、`s1` が連結されると参照されなくなります。

メモ :

文字列の参照を作成するときは注意が必要です。文字列の参照を作成し、文字列を "変更" する場合、参照は文字列が変更されたときに作成された新しいオブジェクトではなく、元のオブジェクトを指したままになります。次にこの問題の例を示します。

```
string s1 = "Hello";
string s2 = s1;
s1 += " and goodbye.";
Console.WriteLine(s2); //outputs "Hello"
```

文字列に変更を加えると、新しい文字列オブジェクトが作成されます。パフォーマンスの低下を防ぐため、大規模な連結などの文字列操作は StringBuilder クラスを使用して行います。次に例を示します。

C#

```
System.Text.StringBuilder sb = new System.Text.StringBuilder();
sb.Append("one ");
sb.Append("two ");
sb.Append("three");
string str = sb.ToString();
```

StringBuilder クラスの詳細については、「StringBuilder の使用」を参照してください。

文字列操作

エスケープ文字

文字列には "\n" (改行) や "\t" (タブ) などのエスケープ文字を使用できます。次の行があるとします。

C#

```
string hello = "Hello\nWorld!";
```

上の例と同じものを次に示します。

```
Hello

World!
```

円記号を使用する場合は、その前に円記号をもう 1 つ挿入してください。次の文字列があるとします。

C#

```
string filePath = "\\\\My Documents\\";
```

上の例と実際には同じものを次に示します。

```
\\My Documents\
```

@ 記号

@ 記号は、文字列コンストラクタに対し、エスケープ文字と改行を無視するように指示します。したがって次の 2 つの文字列は同一です。

C#

```
string p1 = "\\\\My Documents\\My Files\\";
string p2 = @"\\My Documents\My Files\";
```

ToString()

Object から派生するすべてのオブジェクトと同様に、文字列には、値を文字列に変換する ToString メソッドがあります。このメソッドでは、次のように数値を文字列に変換できます。

C#

```
int year = 1999;
string msg = "Eve was born in " + year.ToString();
System.Console.WriteLine(msg);  // outputs "Eve was born in 1999"
```

各文字へのアクセス

文字列のそれぞれの文字にアクセスするには、**SubString()**、**Replace()**、**Split()** および **Trim()** などのメソッドを使用します。

C#

```
string s3 = "Visual C# Express";

System.Console.WriteLine(s3.Substring(7, 2));          // outputs "C#"
System.Console.WriteLine(s3.Replace("C#", "Basic"));  // outputs "Visual Basic Express"
```

また、次に示すように文字を文字配列にコピーすることもできます。

C#

```
string s4 = "Hello, World";
char[] arr = s4.ToCharArray(0, s4.Length);

foreach (char c in arr)
{
    System.Console.Write(c);  // outputs "Hello, World"
}
```

文字列のそれぞれの文字にアクセスするには、インデックスを次のように使用します。

C#

```
string s5 = "Printing backwards";

for (int i = 0; i < s5.Length; i++)
```

```
{
    System.Console.Write(s5[s5.Length - i - 1]);  // outputs "sdrawkcab gnitnirP"
}
```

大文字と小文字の変更

文字列の文字の大文字/小文字を変更するには、**ToUpper()** または **ToLower()** を次のように使用します。

C#

```
string s6 = "Battle of Hastings, 1066";

System.Console.WriteLine(s6.ToUpper());  // outputs "BATTLE OF HASTINGS 1066"
System.Console.WriteLine(s6.ToLower());  // outputs "battle of hastings 1066"
```

比較

2 つの文字列を比較する最も簡単な方法は、**==** 記号と **!=** 記号を使用する方法です。これらの記号は、大文字と小文字を区別して比較を実行します。

C#

```
string color1 = "red";
string color2 = "green";
string color3 = "red";

if (color1 == color3)
{
    System.Console.WriteLine("Equal");
}
if (color1 != color2)
{
    System.Console.WriteLine("Not equal");
}
```

文字列オブジェクトの **CompareTo()** メソッドは、ある文字列が別の文字列より小さいか (<) または大きいか (>) に基づいて整数値を返します。文字列比較では Unicode 値が使用されます。また、小文字の値は対応する大文字の値よりも小さくなります。

C#

```
string s7 = "ABC";
string s8 = "abc";

if (s7.CompareTo(s8) > 0)
{
    System.Console.WriteLine("Greater-than");
}
else
{
    System.Console.WriteLine("Less-than");
}
```

文字列の中にある文字列を検索するには、**IndexOf()** を使用します。**IndexOf()** は、検索文字列が見つからない場合は -1 を返し、見つかった場合は該当文字列の最初の位置を示す 0 から始まるインデックスを返します。

C#

```
string s9 = "Battle of Hastings, 1066";

System.Console.WriteLine(s9.IndexOf("Hastings"));  // outputs 10
System.Console.WriteLine(s9.IndexOf("1967"));      // outputs -1
```

部分文字列への文字列の分割

文字列を部分文字列に分割する操作 (文を単語に分割する操作など) は、一般的なプログラミング タスクです。**Split()** メソッドは、区切り記号 (空白文字など) から成る **char** 配列を受け取り、部分文字列の配列を返します。この配列にアクセスするには、**foreach** を次のように使用します。

C#

```
char[] delimit = new char[] { ' ' };
string s10 = "The cat sat on the mat.";
foreach (string substr in s10.Split(delimit))
{
    System.Console.WriteLine(substr);
}
```

上記のコードにより、次に示すように 1 行に 1 つの単語が出力されます。

```
The
cat
sat
on
the
mat.
```

null 文字列と空の文字列

空の文字列は、文字数ゼロの System.String オブジェクトのインスタンスです。空の文字列は、空のテキスト フィールドを表すため、さまざまなプログラミング シナリオでよく使用されます。空の文字列は有効な **System.String** オブジェクトなので、メソッドを呼び出すことができます。空の文字列は、次のように初期化されます。

```
string s = "";
```

一方、null 文字列は **System.String** オブジェクトのインスタンスではないので、null 文字列でメソッドを呼び出そうとすると NullReferenceException が発生します。しかし、null 文字列を他の文字列に連結したり、他の文字列と比較することは可能です。次に、null 文字列の参照によって例外がスローされる場合とされない場合の例を示します。

```
string str = "hello";
string nullStr = null;
string emptyStr = "";

string tempStr  = str + nullStr; // tempStr = "hello"
bool b = (emptyStr == nullStr);// b = false;
emptyStr + nullStr = ""; // creates a new empty string
int I  = nullStr.Length; // throws NullReferenceException
```

StringBuilder の使用

StringBuilder クラスが作成する文字列バッファにより、プログラムで大量の文字列操作を実行する場合のパフォーマンスが向上します。**StringBuilder** 文字列を使用して、組み込み文字列データ型ではサポートされていない個別の文字を再割り当てできます。たとえば、このコードでは、新しい文字列を作成せずに、文字列の内容を変更します。

C#

```
System.Text.StringBuilder sb = new System.Text.StringBuilder("Rat: the ideal pet");
sb[0] = 'C';
System.Console.WriteLine(sb.ToString());
System.Console.ReadLine();
```

この例では、**StringBuilder** オブジェクトを使用して、複数の数値型から 1 つの文字列を作成します。

C#

```
class TestStringBuilder
{
    static void Main()
    {
        System.Text.StringBuilder sb = new System.Text.StringBuilder();

        // Create a string composed of numbers 0 - 9
        for (int i = 0; i < 10; i++)
        {
            sb.Append(i.ToString());
        }
        System.Console.WriteLine(sb);  // displays 0123456789

        // Copy one character of the string (not possible with a System.String)
        sb[0] = sb[9];

        System.Console.WriteLine(sb);  // displays 9123456789
    }
}
```

参照項目

C# プログラマーズ リファレンス:

- string (C# リファレンス)

参照

処理手順

方法 : 正規表現を使用して文字列を検索する (C# プログラミング ガイド)
方法 : String のメソッドを使用して文字列を検索する (C# プログラミング ガイド)
方法 : Split メソッドを使用して文字列を解析する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

文字列の解析

方法：Split メソッドを使用して文字列を解析する (C# プログラミング ガイド)

System.String.Split メソッドを使用して、文字列を解析するコード例を紹介します。このメソッドは、各要素が WORD である文字列の配列を返すことで機能します。Split では、区切り記号として使用する文字を示す、char の配列を入力に使用します。この例では、空白、コンマ、ピリオド、コロン、およびタブが使用されています。このような区切り記号を含む配列が **Split** に渡され、文の各単語は、結果の文字列配列を使用して、個別に表示されます。

使用例

C#

```
class TestStringSplit
{
    static void Main()
    {
        char[] delimiterChars = { ' ', ',', '.', ':', '\t' };

        string text = "one\ttwo three:four,five six seven";
        System.Console.WriteLine("Original text: '{0}'", text);

        string[] words = text.Split(delimiterChars);
        System.Console.WriteLine("{0} words in text:", words.Length);

        foreach (string s in words)
        {
            System.Console.WriteLine(s);
        }
    }
}
```

出力

```
Original text: 'one     two three:four,five six seven'
7 words in text:
one
two
three
four
five
six
seven
```

参照

概念

C# プログラミング ガイド

その他の技術情報

文字列 (C# プログラミング ガイド)

.NET Framework の正規表現

C++ での .NET プログラミング

方法：String のメソッドを使用して文字列を検索する (C# プログラミング ガイド)

string 型は、System.String クラスのエイリアスであり、文字列の内容を検索するための多数の便利なメソッドを提供します。次の例では、IndexOf、LastIndexOf、StartsWith、および EndsWith の各メソッドが使用されています。

使用例

C#

```
class StringSearch
{
    static void Main()
    {
        string str = "A silly sentence used for silly purposes.";
        System.Console.WriteLine("'{0}'",str);

        bool test1 = str.StartsWith("a silly");
        System.Console.WriteLine("starts with 'a silly'? {0}", test1);

        bool test2 = str.StartsWith("a silly", System.StringComparison.OrdinalIgnoreCase);
        System.Console.WriteLine("starts with 'a silly'? {0} (ignoring case)", test2);

        bool test3 = str.EndsWith(".");
        System.Console.WriteLine("ends with '.'? {0}", test3);

        int first = str.IndexOf("silly");
        int last = str.LastIndexOf("silly");
        string str2 = str.Substring(first, last - first);
        System.Console.WriteLine("between two 'silly' words: '{0}'", str2);
    }
}
```

出力

```
'A silly sentence used for silly purposes.'
starts with 'a silly'? False
starts with 'a silly'? True (ignore case)
ends with '.'? True
between two 'silly' words: 'silly sentence used for '
```

参照

概念

C# プログラミング ガイド

その他の技術情報

文字列 (C# プログラミング ガイド)

方法：正規表現を使用して文字列を検索する (C# プログラミング ガイド)

文字列の検索には、System.Text.RegularExpressions.Regex クラスを使用できます。単純な検索から、正規表現を活用した複雑な検索まで、さまざまな検索があります。**Regex** クラスを使用して文字列を検索する 2 つの例を次に示します。詳細については、「.NET Framework の正規表現」を参照してください。

使用例

次のコードは、大文字と小文字を区別せずに配列の文字列を単純に検索するコンソール アプリケーションです。静的メソッド System.Text.RegularExpressions.Regex.IsMatch(System.String,System.String,System.Text.RegularExpressions.RegexOptions) では、検索する文字列と、検索パターンを含む文字列を前提として、検索を実行します。この場合、3 つ目の引数は、大文字と小文字の区別を無視することを示します。詳細については、「System.Text.RegularExpressions.RegexOptions」を参照してください。

C#

```
class TestRegularExpressions
{
    static void Main()
    {
        string[] sentences =
        {
            "cow over the moon",
            "Betsy the Cow",
            "cowering in the corner",
            "no match here"
        };

        string sPattern = "cow";

        foreach (string s in sentences)
        {
            System.Console.Write("{0,24}", s);

            if (System.Text.RegularExpressions.Regex.IsMatch(s, sPattern, System.Text.Regul
arExpressions.RegexOptions.IgnoreCase))
            {
                System.Console.WriteLine("  (match for '{0}' found)", sPattern);
            }
            else
            {
                System.Console.WriteLine();
            }
        }
    }
}
```

出力

```
      cow over the moon  (match for 'cow' found)
          Betsy the Cow  (match for 'cow' found)
cowering in the corner  (match for 'cow' found)
          no match here
```

次のコードは、正規表現を使用して、配列の各文字列の形式を検証するコンソール アプリケーションです。各文字列が電話番号の形式であることが検証されます。つまり、3 グループの数値がダッシュで区切られ、最初の 2 グループには 3 桁の数値が含まれ、3 つ目のグループには 4 桁の数値が含まれることが検証されます。これは、正規表現 `^\\d{3}-\\d{3}-\\d{4}$` で表すことができます。詳細については、「正規表現言語要素」を参照してください。

C#

```
class TestRegularExpressionValidation
{
    static void Main()
    {
        string[] numbers =
```

```
        {
            "123-456-7890",
            "444-234-22450",
            "690-203-6578",
            "146-893-232",
            "146-839-2322",
            "4007-295-1111",
            "407-295-1111",
            "407-2-5555",
        };

        string sPattern = "^\\d{3}-\\d{3}-\\d{4}$";

        foreach (string s in numbers)
        {
            System.Console.Write("{0,14}", s);

            if (System.Text.RegularExpressions.Regex.IsMatch(s, sPattern))
            {
                System.Console.WriteLine(" - valid");
            }
            else
            {
                System.Console.WriteLine(" - invalid");
            }
        }
    }
}
```

出力

```
    123-456-7890 - valid
    444-234-22450 - invalid
    690-203-6578 - valid
    146-893-232 - invalid
    146-839-2322 - valid
    4007-295-1111 - invalid
    407-295-1111 - valid
    407-2-5555 - invalid
```

参照

概念

C# プログラミング ガイド

その他の技術情報

文字列 (C# プログラミング ガイド)

方法：複数の文字列を連結する (C# プログラミング ガイド)

複数の文字列を連結する方法は 2 つあります。1 つは、String クラスがオーバーロードする + 演算子を使用する方法で、もう 1 つは、StringBuilder クラスを使用する方法です。+ 演算子は使いやすく、直観的なコードの作成に役立ちますが、連続して動作するため、使用するたびに新しい文字列が作成されます。そのため、複数の演算子を連結するのは非効率です。次に例を示します。

C#

```
string two = "two";
string str = "one " + two + " three";
System.Console.WriteLine(str);
```

このコードには、連結される 3 つの文字列と、これら 3 つの文字列をすべて含む最終文字列の合わせて 4 つの文字列がありますが、1 番目と 2 番目の 2 つの文字列が最初に連結されて "one two" を含む文字列が作成されるため、合計で 5 つの文字列が作成されます。3 番目の文字列は別個に追加され、`str` に格納される最終文字列を形成します。

代わりに、**StringBuilder** クラスを使用すると、各文字列をオブジェクトに追加して、最終文字列を 1 ステップで作成できます。この方法を次の例に示します。

使用例

次のコード例では、**StringBuilder** クラスの Append メソッドを使用して、+ 演算子の連結効果を使用せずに、3 つの文字列を連結します。

C#

```
class StringBuilderTest
{
    static void Main()
    {
        string two = "two";

        System.Text.StringBuilder sb = new System.Text.StringBuilder();
        sb.Append("one ");
        sb.Append(two);
        sb.Append(" three");
        System.Console.WriteLine(sb.ToString());

        string str = sb.ToString();
        System.Console.WriteLine(str);
    }
}
```

参照

概念

C# プログラミング ガイド

その他の技術情報

文字列 (C# プログラミング ガイド)

方法：文字列の内容を変更する (C# プログラミング ガイド)

文字列は変更不可であるため、文字列の内容は変更できません。ただし、文字列の内容を変更不可でない形式に抽出して変更し、新しい文字列インスタンスを作成することは可能です。

使用例

次の例では、ToCharArray メソッドを使用して、文字列の内容を char 型の配列に抽出します。次に、この配列の要素の一部を変更します。その後、**char** 配列を使用して新しい文字列インスタンスを作成します。

C#

```
class ModifyStrings
{
    static void Main()
    {
        string str = "The quick brown fox jumped over the fence";
        System.Console.WriteLine(str);

        char[] chars = str.ToCharArray();
        int animalIndex = str.IndexOf("fox");
        if (animalIndex != -1)
        {
            chars[animalIndex++] = 'c';
            chars[animalIndex++] = 'a';
            chars[animalIndex] = 't';
        }

        string str2 = new string(chars);
        System.Console.WriteLine(str2);
    }
}
```

出力

```
The quick brown fox jumped over the fence
The quick brown cat jumped over the fence
```

参照

概念

C# プログラミング ガイド

その他の技術情報

文字列 (C# プログラミング ガイド)

ステートメント、式、および演算子 (C# プログラミング ガイド)

ここでは、C# プログラムを構成する基本要素について説明します。アプリケーションを構成する C# コードは、C# のキーワード、式、および演算子から成るステートメントで構成されます。

詳細については、次のトピックを参照してください。

- ステートメント (C# プログラミング ガイド)
- 式 (C# プログラミング ガイド)
- 演算子 (C# プログラミング ガイド)
- オーバーロードされた演算子 (C# プログラミング ガイド)
- 変換演算子 (C# プログラミング ガイド)
    - 変換演算子の使用 (C# プログラミング ガイド)
    - 方法 : 構造体間にユーザー定義の変換を実装する (C# プログラミング ガイド)
- 方法 : 演算子のオーバーロードを使用して複素数クラスを作成する (C# プログラミング ガイド)
- Equals() と演算子 == のオーバーロードに関するガイドライン (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の以下のセクションを参照してください。

- 1.4 式
- 1.5 ステートメント
- 1.6.6.5 演算子
- 5.3.3 確実な代入を判断するための正確な規則
- 7 式
- 7.2 演算子
- 8 ステートメント

参照
関連項目
キャスト (C# プログラミング ガイド)
概念
C# プログラミング ガイド

ステートメント (C# プログラミング ガイド)

ステートメントは、すべての C# プログラムを構築する土台となる手順型のビルド ブロックです。ステートメントでは、ローカル変数やローカル定数の宣言、メソッドの呼び出し、オブジェクトの作成、あるいは変数、プロパティ、またはフィールドへの値の代入などを実行できます。制御ステートメントでは、for ループなどのループの作成や、if ステートメントまたは switch ステートメントなどの決定および新しいコード ブロックへの分岐などを行うことができます。ステートメントは、通常、セミコロン (;) で終了します。詳細については、「ステートメントの種類 (C# リファレンス)」を参照してください。

中かっこ ({}) で囲まれた一連のステートメントは、コード ブロックを形成します。メソッド本体は、コード ブロックの一例です。コード ブロックは、多くの場合、制御ステートメントの後に続きます。コード ブロック内で宣言された変数や定数は、同じコード ブロック内のステートメントだけが使用できます。たとえば、次のコードは、メソッド ブロックと、制御ステートメントに続くコード ブロックを示しています。

C#

```
bool IsPositive(int number)
{
    if (number > 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

C# のステートメントには、式がよく含まれます。C# の式は、リテラル値、簡易名、または演算子とそのオペランドを含むコードです。多くの一般式が、評価されたときにリテラル値、変数、オブジェクト プロパティ、またはオブジェクトのインデクサ アクセスを生成します。変数、オブジェクト プロパティ、またはオブジェクトのインデクサ アクセスが式によって識別されると、その項目の値が式の値として使用されます。C# の式は、式が最終的に必要な型に評価される限り、値やオブジェクトが必要とされる任意の位置に配置できます。

式には、名前空間、型、メソッド グループ、またはイベント アクセスに評価されるものもあります。このような特殊目的の式は、通常、より大きな式の一部として、特定の時点でのみ有効であり、適切に使用しないとコンパイラ エラーになります。

関連項目

- ステートメントの種類 (C# リファレンス)
- 式 (C# プログラミング ガイド)
- 演算子 (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.5 ステートメント
- 7 式
- 8 ステートメント

参照
概念
C# プログラミング ガイド

式 (C# プログラミング ガイド)

式は、1 つの値、オブジェクト、メソッド、または名前空間に評価できるコード片です。式には、リテラル値、メソッドの呼び出し、演算子とそのオペランド、または簡易名を含めることができます。簡易名には、変数、型のメンバ、メソッドのパラメータ、名前空間、または型の名前を指定できます。

式では、他の式をパラメータとして使用する演算子や、他のメソッド呼び出しとなるパラメータを持つメソッド呼び出しを使用できます。そのため、式には単純なものもあれば、非常に複雑なものもあります。

リテラルと簡易名

式の中で最も単純なのがリテラルと簡易名の 2 つのタイプです。リテラルは、名前を持たない定数値です。たとえば、次のコード例の `5` と `"Hello World"` は共にリテラル値です。

C#

```
int i = 5;
string s = "Hello World";
```

リテラルの詳細については、「型 (C# リファレンス)」を参照してください。

上の例の `i` と `s` は、共にローカル変数を識別する簡易名です。これらの変数を式で使用すると、変数の値が取得され、式で使用されます。たとえば、次のコード例で `DoWork` を呼び出すと、このメソッドは既定で値 5 を受け取り、変数 `var` にアクセスできません。

C#

```
int var = 5;
DoWork(var);
```

呼び出し式

次のコード例に示されている `DoWork` への呼び出しも、呼び出し式と呼ばれる式の一種です。

C#

```
DoWork(var);
```

具体的に言うと、メソッドを呼び出すのがメソッド呼び出し式です。メソッドの呼び出しでは、メソッドの名前 (上の例のような名前、または別の式の結果としての名前) を使用し、その後にかっことメソッド パラメータを記述する必要があります。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。デリゲートの呼び出しでは、デリゲートの名前とメソッド パラメータをかっこで囲んで使用します。詳細については、「デリゲート (C# プログラミング ガイド)」を参照してください。メソッドの呼び出しとデリゲートの呼び出しは、メソッドが値を返す場合、メソッドの戻り値に評価されます。void を返すメソッドは、式の値の代わりに使用できません。

解説

変数、オブジェクト プロパティ、またはオブジェクトのインデクサ アクセスが式によって識別されると、その項目の値が式の値として使用されます。C# の式は、式が最終的に必要な型に評価される限り、値やオブジェクトが必要とされる任意の位置に配置できます。

参照

関連項目

メソッド (C# プログラミング ガイド)
演算子 (C# プログラミング ガイド)
データ型 (C# プログラミング ガイド)

概念

C# プログラミング ガイド
デリゲート (C# プログラミング ガイド)

演算子 (C# プログラミング ガイド)

C# では演算子とは、オペランドと呼ばれる 1 つ以上の式を入力として受け取り、値を返す用語または記号のことです。インクリメント演算子 (**++**) や **new** など、1 つのオペランドを受け取る演算子を単項演算子といいます。また、算術演算子 (**+**、**-**、*****、**/**) など、2 つのオペランドを受け取る演算子を二項演算子といいます。条件演算子 (**?:**) は、3 つのオペランドを受け取る、C# でただ 1 つの三項演算子です。

次の C# ステートメントには、1 つの単項演算子と 1 つのオペランドがあります。インクリメント演算子 **++** は、オペランド `y` の値を変更します。

C#

```
y++;
```

次の C# ステートメントには、それぞれ 2 つのオペランドを持つ 2 つの二項演算子があります。代入演算子 **=** には、オペランドとして整数 `y` と式 `2 + 3` があります。式 `2 + 3` 自体には、加算演算子があり、`2` と `3` の 2 つの整数をオペランドとして使用します。

C#

```
y = 2 + 3;
```

オペランドには、他の任意の数の演算で構成される、任意のサイズの有効な式を使用できます。

式の演算子は、演算子の優先順位と呼ばれる特定の順序で評価されます。以下の表では、実行する演算の種類を基にして演算子を分類しています。各カテゴリは、優先順位に従って配列されています。

1 次式	x.y、f(x)、a[x]、x**++**、x**--**、**new**、**typeof**、**checked**、**unchecked**
単項式	**+**、**-**、**!**、**~**、**++x**、**--x**、(T)x
算術 — 乗法	*****、**/**、**%**
算術 — 加法	**+**、**-**
シフト	**<<**、**>>**
関係式と型検査	**<**、**>**、**<=**、**>=**、**is**、**as**
等値式	**==**、**!=**
論理 (優先順)	**&**、**^**、**|**
条件 (優先順)	**&&**、**||**、**?:**
代入	**=**、**+=**、**-=**、***=**、**/=**、**%=**、**&=**、**|=**、**^=**、**<<=**、**>>=**

優先順位が同じ演算子が式に 2 つ含まれている場合、それらの演算子は、結合規則に基づいて評価されます。結合規則が左から右の演算子は、左から右に評価されます。たとえば、`x * y / z` は `(x * y) / z` と評価されます。結合規則が右から左の演算子は、右から左に評価されます。代入演算子と三項演算子 (**?:**) は、結合規則が右から左です。他の二項演算子はすべて結合規則が左から右です。ただし、C# の標準では、式の中でインクリメント命令の "設定" 部分をいつ実行するかを指定していません。たとえば、次のコード例の出力は 6 になります。

C#

```
int num1 = 5;
num1++;
System.Console.WriteLine(num1);
```

ただし、次のコード例の出力は未定義です。

C#

```
int num2 = 5;
num2 = num2++;  //not recommended
System.Console.WriteLine(num2);
```

そのため、後者はお勧めできません。かっこを使って式を囲むと、その式は他の式よりも先に評価されます。たとえば、`2 + 3 * 2` は通常、8 になります。その理由は、乗算演算子の方が加算演算子よりも優先順位が高いからです。しかし、この式を `(2 + 3 ) * 2` と記述すると、結果は 10 になります。この場合、加算演算子 (+) を乗算演算子 (*) より先に評価するように C# コンパイラに指示するからです。

カスタム クラスやカスタム構造体では、演算子の動作を変更できます。このプロセスを演算子のオーバーロードと言います。詳細については、「オーバーロードされた演算子 (C# プログラミング ガイド)」を参照してください。

関連項目

詳細については、「演算子キーワード (C# リファレンス)」および「C# の演算子」を参照してください。

参照
関連項目
ステートメント、式、および演算子 (C# プログラミング ガイド)
概念
C# プログラミング ガイド

オーバーロードされた演算子 (C# プログラミング ガイド)

C# では、operator キーワードを使用して静的メンバ関数を定義することにより、ユーザー定義型が演算子をオーバーロードできます。ただし、すべての演算子をオーバーロードできるわけではありません。また、制約がある場合もあります。詳細については、次の表を参照してください。

演算子	オーバーロードできるかどうか
+、-、!、~、++、--、true、false	これらの単項演算子は、オーバーロードできます。
+、-、*、/、%、&、|、^、<<、>>	これらの二項演算子は、オーバーロードできます。
==、!=、<、>、<=、>=	比較演算子はオーバーロードできます。後述のメモを参照してください。
&&、||	条件論理演算子はオーバーロードできませんが、オーバーロードできる **&** および **|** を使用して評価されます。
[]	配列の添字演算子はオーバーロードできませんが、インデクサは定義できます。
()	キャスト演算子はオーバーロードできませんが、新しい変換演算子は定義できます。詳細については、「explicit」および「implicit」を参照してください。
+=、-=、*=、/=、%=、&=、|=、^=、<<=、>>=	代入演算子はオーバーロードできませんが、**+=** などは、オーバーロードできる **+** を使用して評価されます。
=、.、?:、->、new、is、sizeof、typeof	これらの演算子は、オーバーロードできません。

メモ :

比較演算子をオーバーロードする場合は、1 組でオーバーロードする必要があります。つまり、**==** をオーバーロードする場合は、**!=** もオーバーロードする必要があります。これは、逆の場合でも同じです。また、**<** と **>** および **<=** と **>=** の場合も 1 組でオーバーロードします。

カスタム クラスの演算子をオーバーロードするには、正しいシグネチャを使用してクラスでメソッドを作成する必要があります。メソッドには、"operator X" という名前を付ける必要があります。この X は、オーバーロードされる演算子の名前または記号です。単項演算子のパラメータは 1 つ、二項演算子のパラメータは 2 つです。いずれの場合も、次の例に示すように、1 つのパラメータは、その宣言元のクラスまたは構造体と同じ型である必要があります。

C#

```
public static Complex operator +(Complex c1, Complex c2)
```

詳細については、「方法 : 演算子のオーバーロードを使用して複素数クラスを作成する (C# プログラミング ガイド)」を参照してください。

参照

関連項目

ステートメント、式、および演算子 (C# プログラミング ガイド)

演算子 (C# プログラミング ガイド)

C# の演算子

概念

C# プログラミング ガイド

変換演算子 (C# プログラミング ガイド)

C# では、クラスや構造体で変換を宣言して、他のクラスや構造体と基本型との相互変換を行うことができます。変換は演算子のように定義でき、変換先の型に応じた名前が付けられます。変換対象の型引数または変換結果の型のうち両方ではなく一方は、包含する型である必要があります。

C#

```
class SampleClass
{
    public static explicit operator SampleClass(int i)
    {
        SampleClass temp = new SampleClass();
        // code to convert from int to SampleClass...

        return temp;
    }
}
```

変換演算子の概要

変換演算子には、次の特徴があります。

- **implicit** として宣言された変換は、必要に応じて自動的に行われます。
- **explicit** として宣言された変換では、キャストを呼び出す必要があります。
- 変換はすべて **static** にする必要があります。

関連項目

詳細情報

- 変換演算子の使用 (C# プログラミング ガイド)
- 変換演算子の設計ガイドライン
- 方法 : 構造体間にユーザー定義の変換を実装する (C# プログラミング ガイド)
- explicit (C# リファレンス)
- implicit (C# リファレンス)
- static (C# リファレンス)

参照

概念

C# プログラミング ガイド

変換演算子の使用 (C# プログラミング ガイド)

変換演算子は、**explicit** の形式にも **implicit** の形式にもできます。暗黙の変換演算子の方が簡単に使用できますが、変換中であることを演算子のユーザーが認識できるようにするには、明示的な演算子が便利です。このトピックでは、両方の型について説明します。

例 1

ここでは、明示的な変換演算子の例を示します。この演算子は、Byte 型を `Digit` という値型に変換します。すべての byte 型を Digit 型に変換できるとは限らないため、変換は明示的に行うように指定されています。つまり、`Main` メソッドに示すように、キャストを使用する必要があります。

C#

```
struct Digit
{
    byte value;

    public Digit(byte value)  //constructor
    {
        if (value > 9)
        {
            throw new System.ArgumentException();
        }
        this.value = value;
    }

    public static explicit operator Digit(byte b)  // explicit byte to digit conversion ope
rator
    {
        Digit d = new Digit(b);  // explicit conversion

        System.Console.WriteLine("conversion occurred");
        return d;
    }
}

class TestExplicitConversion
{
    static void Main()
    {
        try
        {
            byte b = 3;
            Digit d = (Digit)b;  // explicit conversion
        }
        catch (System.Exception e)
        {
            System.Console.WriteLine("{0} Exception caught.", e);
        }
    }
}
```

出力 1

```
conversion occurred
```

例 2

この例は、上の例で行った処理を元に戻す変換演算子を定義することにより、暗黙の変換演算子を示しています。この例では、`Digit` という値クラスを整数 **Byte** 型に変換します。すべての Digit 型を **Byte** に変換できるため、変換をユーザーに明示する必要はありません。

C#

```
struct Digit
{
    byte value;
```

```
    public Digit(byte value)  //constructor
    {
        if (value > 9)
        {
            throw new System.ArgumentException();
        }
        this.value = value;
    }

    public static implicit operator byte(Digit d)  // implicit digit to byte conversion ope
rator
    {
        System.Console.WriteLine("conversion occurred");
        return d.value;  // implicit conversion
    }
}

class TestImplicitConversion
{
    static void Main()
    {
        Digit d = new Digit(3);
        byte b = d;  // implicit conversion -- no cast needed
    }
}
```

出力 2

```
conversion occurred
```

参照

概念

C# プログラミング ガイド

変換演算子 (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

方法：構造体間にユーザー定義の変換を実装する (C# プログラミング ガイド)

この例では、RomanNumeral と BinaryNumeral の 2 つの構造体が定義されていて、これらの構造体間の変換を示しています。

使用例

C#

```
struct RomanNumeral
{
    private int value;

    public RomanNumeral(int value)  //constructor
    {
        this.value = value;
    }

    static public implicit operator RomanNumeral(int value)
    {
        return new RomanNumeral(value);
    }

    static public implicit operator RomanNumeral(BinaryNumeral binary)
    {
        return new RomanNumeral((int)binary);
    }

    static public explicit operator int(RomanNumeral roman)
    {
        return roman.value;
    }

    static public implicit operator string(RomanNumeral roman)
    {
        return ("Conversion not yet implemented");
    }
}

struct BinaryNumeral
{
    private int value;

    public BinaryNumeral(int value)  //constructor
    {
        this.value = value;
    }

    static public implicit operator BinaryNumeral(int value)
    {
        return new BinaryNumeral(value);
    }

    static public explicit operator int(BinaryNumeral binary)
    {
        return (binary.value);
    }

    static public implicit operator string(BinaryNumeral binary)
    {
        return ("Conversion not yet implemented");
    }
}

class TestConversions
{
```

```
    static void Main()
    {
        RomanNumeral roman;
        BinaryNumeral binary;

        roman = 10;

        // Perform a conversion from a RomanNumeral to a BinaryNumeral:
        binary = (BinaryNumeral)(int)roman;

        // Perform a conversion from a BinaryNumeral to a RomanNumeral:
        // No cast is required:
        roman = binary;

        System.Console.WriteLine((int)binary);
        System.Console.WriteLine(binary);
    }
}
```

出力

```
10
Conversion not yet implemented
```

堅牢性の高いプログラム

- 前述の例の次のステートメントは、

  C#

  ```
  binary = (BinaryNumeral)(int)roman;
  ```

  このステートメントは、`RomanNumeral` から `BinaryNumeral` への変換を実行します。`RomanNumeral` から `BinaryNumeral` への直接変換はないため、`RomanNumeral` から **int** へ変換するためのキャストと、**int** から `BinaryNumeral` へ変換するためのキャストが使用されています。

- また、次のステートメントは、

  C#

  ```
  roman = binary;
  ```

  `BinaryNumeral` から `RomanNumeral` への変換を実行します。`RomanNumeral` は `BinaryNumeral` からの暗黙の型変換を定義しているため、キャストは不要です。

参照

概念

C# プログラミング ガイド

変換演算子 (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

方法：演算子のオーバーロードを使用して複素数クラスを作成する (C# プログラミング ガイド)

この例では、演算子のオーバーロードを使用して、複素数の加算を定義する複素数クラス `Complex` を作成する方法を示します。プログラムは、**ToString** メソッドのオーバーライドを使用して、数値の虚数部と実数部、および加算結果を表示します。

使用例

C#

```
public struct Complex
{
    public int real;
    public int imaginary;

    public Complex(int real, int imaginary)  //constructor
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    // Declare which operator to overload (+),
    // the types that can be added (two Complex objects),
    // and the return type (Complex):
    public static Complex operator +(Complex c1, Complex c2)
    {
        return new Complex(c1.real + c2.real, c1.imaginary + c2.imaginary);
    }

    // Override the ToString() method to display a complex number in the traditional format
:
    public override string ToString()
    {
        return (System.String.Format("{0} + {1}i", real, imaginary));
    }
}

class TestComplex
{
    static void Main()
    {
        Complex num1 = new Complex(2, 3);
        Complex num2 = new Complex(3, 4);

        // Add two Complex objects through the overloaded plus operator:
        Complex sum = num1 + num2;

        // Print the numbers and the sum using the overriden ToString method:
        System.Console.WriteLine("First complex number:  {0}", num1);
        System.Console.WriteLine("Second complex number: {0}", num2);
        System.Console.WriteLine("The sum of the two numbers: {0}", sum);
    }
}
```

出力

```
First complex number:  2 + 3i
Second complex number: 3 + 4i
The sum of the two numbers: 5 + 7i
```

参照

関連項目

C# の演算子

operator (C# リファレンス)

概念

C# プログラミング ガイド

Equals() と演算子 == のオーバーロードに関するガイドライン (C# プログラミング ガイド)

C# には、2 種類の等価があります。1 つは参照の等価で、もう 1 つは値の等価です。値が等価であるとは、一般的に理解されている意味での等価で、2 つのオブジェクトが同じ値を含んでいることを意味します。たとえば、2 という値を持つ 2 つの整数は値が等価です。一方、参照が等価であるとは、比較する 2 つのオブジェクトが存在しないことを意味します。この場合は、オブジェクトの代わりに、共に同じオブジェクトを参照する 2 つのオブジェクト参照が存在します。次の例に示すように、これは簡単な代入によって生じます。

C#

```
System.Object a = new System.Object();
System.Object b = a;
System.Object.ReferenceEquals(a, b);  //returns true
```

このコードでは、オブジェクトは 1 つだけですが、そのオブジェクトへの参照は、a、b と複数あります。これらはいずれも同じオブジェクトを参照するので、参照が等価です。2 つのオブジェクトの参照が等価である場合、これらは値が等価でもありますが、値が等価であるからといって必ずしも参照が等価であるとは限りません。

参照が等価であることを確認するには、ReferenceEquals を使用します。値が等価であることを確認するには、Equals または Equals を使用します。

Equals のオーバーライド

Equals は仮想メソッドで、これを使用すると、任意のクラスで実装をオーバーライドできます。値を表すクラス、実質的にあらゆる値型を表すクラス、または値のセットをグループとして表すクラス (複素数クラスなど) では、**Equals** をオーバーライドする必要があります。型が IComparable を実装する場合は、**Equals** をオーバーライドする必要があります。

Equals の新しい実装は、Equals が保証する以下の事項をすべて踏襲する必要があります。

- x.**Equals**(x) は true を返します。
- x.**Equals**(y) は、y.**Equals**(x) と同じ値を返します。
- (x.**Equals**(y) && y.**Equals**(z)) が true を返す場合、x.**Equals**(z) も true を返します。
- x.**Equals**(y) が連続して呼び出された場合は、x および y が参照するオブジェクトが変更されていない限り、同じ値を返します。
- x.**Equals**(null) は false を返します。

Equals の新しい実装は、例外をスローできません。**Equals** をオーバーライドするクラスでは、System.Object.GetHashCode もオーバーライドすることをお勧めします。また、**Equals**(オブジェクト) を実装する他に、パフォーマンスを向上するために、それぞれ固有の型の **Equals**(型) をすべてのクラスで実装することもお勧めします。次に例を示します。

C#

```
class TwoDPoint : System.Object
{
    public readonly int x, y;

    public TwoDPoint(int x, int y)  //constructor
    {
        this.x = x;
        this.y = y;
    }

    public override bool Equals(System.Object obj)
    {
        // If parameter is null return false.
        if (obj == null)
        {
            return false;
        }

        // If parameter cannot be cast to Point return false.
        TwoDPoint p = obj as TwoDPoint;
```

```
        if ((System.Object)p == null)
        {
            return false;
        }

        // Return true if the fields match:
        return (x == p.x) && (y == p.y);
    }

    public bool Equals(TwoDPoint p)
    {
        // If parameter is null return false:
        if ((object)p == null)
        {
            return false;
        }

        // Return true if the fields match:
        return (x == p.x) && (y == p.y);
    }

    public override int GetHashCode()
    {
        return x ^ y;
    }
}
```

基本クラスの **Equals** を呼び出すことができる派生クラスでは、比較を完了する前に呼び出す必要があります。次の例では、**Equals** で基本クラスの **Equals** を呼び出し、null パラメータを確認して、パラメータの型と派生クラスの型を比較します。これにより、派生クラスの **Equals** の実装に、派生クラスで宣言された新しいデータ フィールドをチェックするタスクが委ねられます。

C#

```
class ThreeDPoint : TwoDPoint
{
    public readonly int z;

    public ThreeDPoint(int x, int y, int z)
        : base(x, y)
    {
        this.z = z;
    }

    public override bool Equals(System.Object obj)
    {
        // If parameter cannot be cast to ThreeDPoint return false:
        ThreeDPoint p = obj as ThreeDPoint;
        if ((object)p == null)
        {
            return false;
        }

        // Return true if the fields match:
        return base.Equals(obj) && z == p.z;
    }

    public bool Equals(ThreeDPoint p)
    {
        // Return true if the fields match:
        return base.Equals((TwoDPoint)p) && z == p.z;
    }

    public override int GetHashCode()
    {
        return base.GetHashCode() ^ z;
    }
}
```

演算子 == のオーバーライド

既定では、演算子 == は、2 つの参照が同じオブジェクトを示すかどうかを確認して参照の等価をテストするため、この機能を取得するために参照型で演算子 == を実装する必要はありません。型が変更できない場合、つまりインスタンスに含まれているデータを変更できない場合、その型は、変更不可能なオブジェクトとして、同じ値を持つ限り同一と見なされるので、参照の等価の代わりに値の等価を比較するように演算子 == をオーバーロードするのが有効です。変更不可能な型以外で演算子 == をオーバーライドすることはお勧めしません。

オーバーロードされた演算子 == の実装は、例外をスローできません。演算子 == をオーバーロードする型では、演算子 **!=** もオーバーロードする必要があります。次に例を示します。

C#

```
//add this code to class ThreeDPoint as defined previously
//
public static bool operator ==(ThreeDPoint a, ThreeDPoint b)
{
    // If both are null, or both are same instance, return true.
    if (System.Object.ReferenceEquals(a, b))
    {
        return true;
    }

    // If one is null, but not both, return false.
    if (((object)a == null) || ((object)b == null))
    {
        return false;
    }

    // Return true if the fields match:
    return a.x == b.x && a.y == b.y && a.z == b.z;
}

public static bool operator !=(ThreeDPoint a, ThreeDPoint b)
{
    return !(a == b);
}
```

メモ :

演算子 == のオーバーロードでは、`(a == b)`、`(a == null)`、または `(b == null)` を使用して参照が等価であることを確認するというエラーがよくあります。このような場合は、オーバーロードされた演算子 == が呼び出され、無限ループが生じます。無限ループを避けるには、**ReferenceEquals** を使用するか、型を Object にキャストしてください。

参照

関連項目

ステートメント、式、および演算子 (C# プログラミング ガイド)
演算子 (C# プログラミング ガイド)
オーバーロードされた演算子 (C# プログラミング ガイド)
== 演算子 (C# リファレンス)
() 演算子 (C# リファレンス)

概念

C# プログラミング ガイド

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

C# は、オブジェクト指向のプログラミング言語であり、クラスと構造体を使用して、Windows フォーム、ユーザー インターフェイス コントロール、データ構造体などの型を実装します。一般的な C# アプリケーションは、プログラマ定義のクラスと .NET Framework のクラスの組み合わせで構成されます。

C# では、さまざまな方法でクラスを定義できます。たとえば、異なるアクセス レベルを指定したり、他のクラスから機能を継承したりできます。また、型をインスタンス化または破棄したときに実行される処理を指定することもできます。

クラスは、型パラメータを使用してジェネリックとして定義することもできます。型パラメータを使用すると、クライアント コードでタイプ セーフかつ効率的にクラスをカスタマイズできます。.NET Framework クラス ライブラリの System.Collections.Generic.List など、単一のジェネリック クラスをクライアント コードで使用すると、整数、文字列、または他の型のオブジェクトを格納できます。

概要

オブジェクト、クラス、および構造体には、次の特徴があります。

- オブジェクトは、特定のデータ型のインスタンスです。データ型は、アプリケーションの実行時に作成 (インスタンス化) されるオブジェクトの設計図になります。
- 新しいデータ型は、クラスおよび構造体を使用して定義します。
- クラスと構造体は、コードとデータを含む、C# アプリケーションのビルド ブロックを形成します。C# アプリケーションには、必ず少なくとも 1 つのクラスが存在します。
- 構造体は、少量のデータを格納するデータ型を作成するのに適した軽量のクラスと考えることができます。構造体は、継承によって後で拡張される可能性がある型を表しません。
- C# クラスは継承をサポートするため、定義済みのクラスから派生できます。

関連項目

- 型のデザインのガイドライン
- オブジェクト (C# プログラミング ガイド)
- クラス
- 構造体
- 継承
- メンバ (C# プログラミング ガイド)
- メソッド
- コンストラクタ (C# プログラミング ガイド)
- デストラクタ
- コンストラクタとデストラクタの使用方法
- フィールド (C# プログラミング ガイド)
- 定数
- 入れ子にされた型
- アクセス修飾子
- 部分クラス定義 (C# プログラミング ガイド)
- 静的クラスと静的クラス メンバ (C# プログラミング ガイド)
- 方法 : メソッドに構造体を渡すこととクラス参照を渡すことの違いを理解する (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6 クラスとオブジェクト

- 1.7 構造体
- 3.4.4 クラスのメンバ
- 4.2.1 クラス型
- 10 クラス
- 11 構造体

参照
概念
C# プログラミング ガイド

オブジェクト (C# プログラミング ガイド)

オブジェクトは、データ、動作、および ID が割り当てられたプログラミング構成要素です。オブジェクトのデータは、オブジェクトのフィールド、プロパティ、およびイベントに格納され、オブジェクトの動作は、オブジェクトのメソッドとインターフェイスによって定義されます。

オブジェクトには ID があります。2 つのオブジェクトが同じデータ セットを持っていても、それらが同じオブジェクトであるとは限りません。

C# のオブジェクトは、**classes** および **structs** によって定義されます。これらは、同じ型のすべてのオブジェクトの動作の基礎となる単一の設計図となります。

オブジェクトの概要

オブジェクトには、次の特徴があります。

- C# で使用するものは、Windows フォームやコントロールを含め、すべてオブジェクトです。
- オブジェクトはインスタンス化されます。つまり、クラスと構造体で定義されたテンプレートから作成されます。
- オブジェクトでは プロパティ (C# プログラミング ガイド) を使用して、格納する情報を取得および変更します。
- オブジェクトには、多くの場合、メソッドやイベントがあり、それらによって特定のアクションを実行できます。
- Visual Studio には、オブジェクトを操作するためのツールがあります。そのうちの [プロパティ] ウィンドウ では、Windows フォームなどのオブジェクトの属性を変更でき、オブジェクト ブラウザ では、オブジェクトの内容をチェックできます。
- C# オブジェクトはすべて Object から継承されます。

関連項目

詳細情報

- クラス (C# プログラミング ガイド)
- 構造体 (C# プログラミング ガイド)
- コンストラクタ (C# プログラミング ガイド)
- デストラクタ (C# プログラミング ガイド)
- イベント (C# プログラミング ガイド)

参照
関連項目
object (C# リファレンス)
継承 (C# プログラミング ガイド)
class (C# リファレンス)
struct (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
リモート オブジェクト

クラス (C# プログラミング ガイド)

クラスは、C# で最も強力なデータ型です。クラスは、構造体と同様にデータ型のデータと動作を定義します。クラスを定義すると、そのクラスのインスタンスであるオブジェクトを作成できます。構造体と違って、クラスは、オブジェクト指向プログラミングの基本部分である継承をサポートします。詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

クラスの宣言

クラスは、次の例に示すように、class キーワードを使用して定義します。

C#

```
public class Customer
{
    //Fields, properties, methods and events go here...
}
```

class キーワードは、アクセス レベルの後に配置します。この例では、public が使用されているため、だれもがこのクラスからオブジェクトを作成できます。**class** キーワードの後にクラス名を記述します。定義の残りの部分がクラス本体で、そこで動作とデータを定義します。クラスのフィールド、プロパティ、メソッド、およびイベントはクラス メンバと総称されます。

オブジェクトの作成

クラスとオブジェクトは、同義的に使用されることがありますが、これらは異なるものです。クラスはオブジェクトの型を定義しますが、オブジェクト自体ではありません。オブジェクトは、クラスに基づく具体的なエンティティであり、クラスのインスタンスと呼ばれることもあります。

オブジェクトを作成するには、次のように、new キーワードの後にオブジェクトの基になるクラスの名前を指定します。

C#

```
Customer object1 = new Customer();
```

クラスのインスタンスを作成すると、そのオブジェクトへの参照が返されます。上の例の `object1` は、`Customer` に基づくオブジェクトへの参照です。この参照は、新しいオブジェクトを参照しますが、オブジェクト データ自体を含みません。実際、オブジェクト参照は、オブジェクトを作成しなくても作成できます。この例を次に示します。

C#

```
Customer object2;
```

上のような、オブジェクトを参照しないオブジェクト参照を作成するのはお勧めできません。実行時にこのような参照を通じてオブジェクトへのアクセスを試みると失敗するからです。ただし、新しいオブジェクトを作成するか、既存のオブジェクトに割り当てると、このような参照でオブジェクトを参照できるようになります。この例を次に示します。

C#

```
Customer object3 = new Customer();
Customer object4 = object3;
```

上のコードでは、同じオブジェクトを共に参照する 2 つのオブジェクトが作成されます。そのため、`object3` を通じて行われたオブジェクトの変更は、その後、`object4` を使用する際にも反映されます。これは、クラスに基づくオブジェクトが参照によって参照されるからです。このためクラスは参照型と呼ばれています。

クラスの継承

継承は、派生を使用して行われます。派生とは、データと動作の継承元である基本クラスを使用してクラスを宣言することを意味します。基本クラスは、派生クラス名の後に、コロンと基本クラス名を追加して指定します。この例を次に示します。

C#

```
public class Manager : Employee
```

```
{
    // Employee fields, properties, methods and events are inherited
    // New Manager fields, properties, methods and events go here...
}
```

クラスで基本クラスを宣言している場合、基本クラスで定義されているクラス メンバもすべて新しいクラスの一部になります。基本クラス自体が別のクラスを継承したものであり、その別のクラスもさらに別のクラスを継承している可能性があるため、逆に言えば 1 つのクラスから多くの基本クラスが派生されることがあります。

例

次の例では、フィールド、メソッド、およびコンストラクタという特殊なメソッドをそれぞれ 1 つずつ含むパブリック クラスを定義しています。詳細については、「コンストラクタ (C# プログラミング ガイド)」を参照してください。このクラスは、**new** キーワードによってインスタンス化されています。

C#

```
public class Person
{
    // Field
    public string name;

    // Constructor
    public Person()
    {
        name = "unknown";
    }

    // Method
    public void SetName(string newName)
    {
        name = newName;
    }
}
class TestPerson
{
    static void Main()
    {
        Person person1 = new Person();
        System.Console.WriteLine(person1.name);

        person1.SetName("John Smith");
        System.Console.WriteLine(person1.name);
    }
}
```

出力

```
unknown
John Smith
```

クラスの概要

クラスには、次の特徴があります。

- C++ と異なり、単一継承のみをサポートします。クラスは、1 つの基本クラスからのみ実装を継承できます。
- 1 つのクラスで複数のインターフェイスを実装できます。詳細については、「インターフェイス (C# プログラミング ガイド)」を参照してください。
- クラス定義は、別々のソース ファイルに分割できます。詳細については、「部分クラス定義 (C# プログラミング ガイド)」を参照してください。
- 静的クラスは、静的メソッドだけを含むシール クラスです。詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」を参照してください。
-

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6 クラスとオブジェクト
- 10 クラス

参照

関連項目

メンバ (C# プログラミング ガイド)
メソッド (C# プログラミング ガイド)
デストラクタ (C# プログラミング ガイド)
オブジェクト (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

コンストラクタ (C# プログラミング ガイド)

構造体 (C# プログラミング ガイド)

構造体は、次のように struct キーワードで定義します。

C#

```
public struct PostalAddress
{
    // Fields, properties, methods and events go here...
}
```

構造体は、構文上ではクラスとほとんど変わりませんが、次のようにクラスよりも制限されます。

- 構造体宣言内では、const または static と宣言されているフィールド以外は初期化できません。
- 構造体では、既定のコンストラクタ (パラメータなしのコンストラクタ) やデストラクタを宣言できません。

構造体のコピーは、コンパイラによって自動的に作成され、破棄されるので、既定のコンストラクタやデストラクタは不要です。コンパイラは、すべて既定値のフィールドを代入することにより事実上既定のコンストラクタを実装します (「既定値の一覧表 (C# リファレンス)」を参照)。構造体は、クラスや他の構造体を継承できません。

構造体は値型です。オブジェクトを構造体から作成し、変数に代入すると、その変数には、構造体の値全体が含まれます。構造体を含む変数をコピーすると、すべてのデータがコピーされ、新しいコピーを変更しても、以前のコピーのデータは変更されません。構造体は参照を使用しないので、構造体には ID がありません。そのため、同じデータを含む、値型の 2 つのインスタンスは区別できません。C# のすべての値型は本質的に、Object を継承する ValueType から派生します。

値型は、ボックス化と呼ばれる、コンパイラの処理によって参照型に変換できます。詳細については、「ボックス化とボックス化解除」を参照してください。

構造体の概要

構造体には、次の特徴があります。

- 構造体は値型ですが、クラスは参照型です。
- クラスとは異なり、構造体は new 演算子を使用せずにインスタンス化できます。
- 構造体はコンストラクタを宣言できますが、パラメータを受け取る必要があります。
- 構造体は、他の構造体やクラスから継承できず、基本クラスになれません。すべての構造体が **System.ValueType** を直接継承し、System.ValueType は **System.Object** を継承します。
- 構造体では、インターフェイスを実装できます。

関連項目

詳細情報

- 構造体の使用 (C# プログラミング ガイド)
- コンストラクタとデストラクタの使用方法
- 方法 : メソッドに構造体を渡すこととクラス参照を渡すことの違いを理解する (C# プログラミング ガイド)
- 方法 : 構造体間にユーザー定義の変換を実装する (C# プログラミング ガイド)

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

構造体のデザイン

クラス (C# プログラミング ガイド)

構造体の使用 (C# プログラミング ガイド)

struct 型は、**Point**、**Rectangle**、**Color** などの軽量のオブジェクトを表すのに適しています。点はクラスで表現できますが、一部の事例では構造体の方がより効果的です。たとえば、1,000 個の **Point** オブジェクトから成る配列を宣言する場合は、各オブジェクトの参照用に新たにメモリが割り当てられます。この場合、構造体であれば処理上の負荷を抑えることができます。.NET Framework には Point というオブジェクトが含まれているため、ここで使用する構造体は "CoOrds" と呼ぶことにします。

C#

```
public struct CoOrds
{
    public int x, y;

    public CoOrds(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
```

構造体に対して既定の (パラメータなしの) コンストラクタを宣言するとエラーになります。構造体メンバを既定値に初期化する既定のコンストラクタが常備されています。また、構造体のインスタンス フィールドを初期化するとエラーになります。

new 演算子を使用して struct オブジェクトを作成すると、オブジェクトが作成されて適切なコンストラクタが呼び出されます。クラスとは異なり、構造体は **new** 演算子を使用せずにインスタンス化できます。**new** を使用しなかった場合、各フィールドは未割り当てのままになり、すべてのフィールドが初期化されるまでオブジェクトを使用できません。

クラスには継承がありますが、構造体には継承がありません。構造体は、他の構造体やクラスから継承できず、基本クラスになれません。ただし、構造体は、基本クラス Object から継承します。構造体は、クラスの場合とまったく同じ方法でインターフェイスを実装できます。

C++ とは異なり、キーワード **struct** を使用してクラスを宣言できません。C# では、クラスと構造体は、意味が異なります。構造体は値型ですが、クラスは参照型です。詳細については、「値型 (C# リファレンス)」を参照してください。

参照型の機能が必要な場合以外は、小さいクラスの方が、構造体としてより効果的に処理されることがあります。

例 1

既定のコンストラクタとパラメータ化されたコンストラクタの両方を使用した **struct** の初期化の例を次に示します。

C#

```
public struct CoOrds
{
    public int x, y;

    public CoOrds(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
```

C#

```
// Declare and initialize struct objects.
class TestCoOrds
{
    static void Main()
    {
        // Initialize:
        CoOrds coords1 = new CoOrds();
        CoOrds coords2 = new CoOrds(10, 10);

        // Display results:
```

```
            System.Console.Write("CoOrds 1: ");
            System.Console.WriteLine("x = {0}, y = {1}", coords1.x, coords1.y);

            System.Console.Write("CoOrds 2: ");
            System.Console.WriteLine("x = {0}, y = {1}", coords2.x, coords2.y);
        }
    }
```

出力

```
CoOrds 1: x = 0, y = 0
CoOrds 2: x = 10, y = 10
```

例 2

struct 固有の機能を次の例に示します。**new** 演算子を使用せずに CoOrds オブジェクトが作成されます。**struct** を **class** という単語に置き換えた場合、プログラムはコンパイルされません。

C#

```
    public struct CoOrds
    {
        public int x, y;

        public CoOrds(int p1, int p2)
        {
            x = p1;
            y = p2;
        }
    }
```

C#

```
    // Declare a struct object without "new."
    class TestCoOrdsNoNew
    {
        static void Main()
        {
            // Declare an object:
            CoOrds coords1;

            // Initialize:
            coords1.x = 10;
            coords1.y = 20;

            // Display results:
            System.Console.Write("CoOrds 1: ");
            System.Console.WriteLine("x = {0}, y = {1}", coords1.x, coords1.y);
        }
    }
```

出力

```
CoOrds 1: x = 10, y = 20
```

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

構造体 (C# プログラミング ガイド)

継承 (C# プログラミング ガイド)

クラスは、別のクラスを継承できます。この処理を行うには、次のようにクラスの宣言時にクラス名の後にコロンを配置し、コロンの後に継承元のクラス (基本クラス) の名前を指定します。

C#

```
public class A
{
    public A() { }
}

public class B : A
{
    public B() { }
}
```

これで新しいクラス (派生クラス) は、基本クラスの非プライベート データと動作をすべて取得し、さらに独自に定義した他のデータと動作も取得します。新しいクラスの有効な型は 2 つになります。1 つは新しいクラスの型で、もう 1 つは継承元のクラスの型です。

上の例では、クラス B は、B でも A でも有効です。B オブジェクトにアクセスするときは、キャスト操作を使用して、B オブジェクトを A オブジェクトに変換できます。B オブジェクトはキャストによって変更されませんが、B オブジェクトのビューは、A のデータと動作に制限されるようになります。B を A にキャストした後、この A は B にキャスト バックできます。その際、B にキャストできるのは、A のすべてのインスタンスではなく、B の実際のインスタンスだけに限定されます。クラス B に B 型としてアクセスすると、クラス A とクラス B のデータと動作を取得します。オブジェクトが複数の型を表す機能をポリモーフィズムと言います。詳細については、「ポリモーフィズム (C# プログラミング ガイド)」を参照してください。キャストの詳細については、「キャスト (C# プログラミング ガイド)」を参照してください。

構造体は、他の構造体やクラスを継承できません。クラスと構造体は共に 1 つ以上のインターフェイスを継承できます。詳細については、「インターフェイス (C# プログラミング ガイド)」を参照してください。

このセクションの内容

- 抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)
- ポリモーフィズム (C# プログラミング ガイド)
- インターフェイス (C# プログラミング ガイド)

参照
関連項目
オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
class (C# リファレンス)
struct (C# リファレンス)
概念
C# プログラミング ガイド

抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)

abstract キーワードを使用すると、継承専用のクラスとクラス メンバを作成し、派生した非抽象クラスの機能を定義できます。また、sealed キーワードを使用すると、既に virtual とマークされているクラスや特定のクラス メンバを継承しないようにできます。詳細については、「方法 : 抽象プロパティを定義する (C# プログラミング ガイド)」を参照してください。

抽象クラスと抽象クラス メンバ

クラスは抽象として宣言できます。このように宣言するには、クラス定義で **class** キーワードの前に **abstract** キーワードを配置します。次に例を示します。

C#

```
public abstract class A
{
    // Class members here.
}
```

抽象クラスはインスタンス化できません。抽象クラスの目的は、複数の派生クラスで共有できる基本クラスの共通の定義を提供することです。たとえば、クラス ライブラリでは、その多くの関数のパラメータとして使用される抽象クラスを定義できます。このライブラリを使用する場合は、派生クラスを作成してクラスの独自の実装を提供する必要があります。

抽象クラスでは、抽象メソッドも定義できます。抽象メソッドを定義するには、メソッドの戻り値の型の前に **abstract** キーワードを記述します。次に例を示します。

C#

```
public abstract class A
{
    public abstract void DoWork(int i);
}
```

抽象メソッドには実装がないので、メソッド定義の後に、通常のメソッド ブロックの代わりにセミコロン (;) を配置します。抽象クラスの派生クラスでは、すべての抽象メソッドを実装する必要があります。抽象クラスが基本クラスから仮想メソッドを継承した場合は、この抽象クラスでは抽象メソッドで仮想メソッドをオーバーライドできます。次に例を示します。

C#

```
// compile with: /target:library
public class D
{
    public virtual void DoWork(int i)
    {
        // Original implementation.
    }
}

public abstract class E : D
{
    public abstract override void DoWork(int i);
}

public class F : E
{
    public override void DoWork(int i)
    {
        // New implementation.
    }
}
```

仮想メソッドが抽象として宣言されている場合は、抽象クラスを継承するすべてのクラスに対しても仮想です。抽象メソッドを継承するクラスでは、そのメソッドの元の実装にアクセスできません。上の例では、クラス F の `DoWork` は、クラス D の `DoWork` を呼び出すことができません。このよ

うにして抽象クラスは、派生クラスに対し、仮想メソッドの新しいメソッド実装を強制的に提供させることができます。

シール クラスとシール クラス メンバ

クラスは、シールとして宣言できます。このように宣言するには、クラス定義で **class** キーワードの前に **sealed** キーワードを配置します。次に例を示します。

C#

```
public sealed class D
{
    // Class members here.
}
```

シール クラスは、基本クラスとして使用できません。このため、シール クラスは抽象クラスになることもできません。シール クラスは、主に派生を防ぐために使用します。シール クラスは基本クラスとして使用できないので、実行時の最適化で、シール クラス メンバを多少高速に呼び出すことができる場合があります。

基本クラスの仮想メンバをオーバーライドしている派生クラスのクラス メンバ、メソッド、フィールド、プロパティ、またはイベントでは、そのメンバをシールとして宣言できます。これにより、その後の派生クラスでは、メンバの仮想性が無効になります。このように宣言するには、クラス メンバ宣言で override キーワードの前に **sealed** キーワードを配置します。次に例を示します。

C#

```
public class D : C
{
    public sealed override void DoWork() { }
}
```

参照

処理手順

方法 : 抽象プロパティを定義する (C# プログラミング ガイド)

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

継承 (C# プログラミング ガイド)

メソッド (C# プログラミング ガイド)

フィールド (C# プログラミング ガイド)

概念

C# プログラミング ガイド

方法：抽象プロパティを定義する (C# プログラミング ガイド)

次の例では、抽象プロパティの定義方法を示します。抽象プロパティの宣言では、プロパティ アクセサは実装されません。クラスがプロパティをサポートしていることは宣言しますが、アクセサの実装は派生クラスに委ねます。基本クラスから継承された抽象プロパティを実装する方法を次の例に示します。

この例は、次の 3 つのファイルで構成されています。各ファイルは個別にコンパイルされ、生成されたアセンブリが次のコンパイルによって参照されます。

- abstractshape.cs: `Area` 抽象プロパティを含む `Shape` クラス。
- shapes.cs: `Shape` クラスのサブクラス。
- shapetest.cs: `Shape` から派生したオブジェクトの面積を表示するテスト プログラム。

この例をコンパイルするには、次のコマンドを入力します。

```
csc abstractshape.cs shapes.cs shapetest.cs
```

これで、実行可能ファイル shapetest.exe が作成されます。

使用例

このファイルは、**double** 型の `Area` プロパティを持つ `Shape` クラスを宣言します。

C#

```
// compile with: csc /target:library abstractshape.cs
public abstract class Shape
{
    private string m_id;

    public Shape(string s)
    {
        // calling the set accessor of the Id property.
        Id = s;
    }

    public string Id
    {
        get
        {
            return m_id;
        }

        set
        {
            m_id = value;
        }
    }

    // Area is a read-only property - only a get accessor is needed:
    public abstract double Area
    {
        get;
    }

    public override string ToString()
    {
        return Id + " Area = " + string.Format("{0:F2}", Area);
    }
}
```

- プロパティの修飾子は、プロパティ宣言自体に設定されます。この例を次に示します。

```
public abstract double Area
```

- 抽象プロパティ (例では `Area`) を宣言するときは、使用できるプロパティ アクセサを指示するだけで、実装はしません。この例では、get アクセサだけが有効なため、プロパティは読み取り専用です。

次のコードは、`Shape` の 3 種類のサブクラスと、それらがどのように `Area` プロパティをオーバーライドして独自の実装を提供するかを示しています。

C#

```
// compile with: csc /target:library /reference:abstractshape.dll shapes.cs
public class Square : Shape
{
    private int m_side;

    public Square(int side, string id)
        : base(id)
    {
        m_side = side;
    }

    public override double Area
    {
        get
        {
            // Given the side, return the area of a square:
            return m_side * m_side;
        }
    }
}

public class Circle : Shape
{
    private int m_radius;

    public Circle(int radius, string id)
        : base(id)
    {
        m_radius = radius;
    }

    public override double Area
    {
        get
        {
            // Given the radius, return the area of a circle:
            return m_radius * m_radius * System.Math.PI;
        }
    }
}

public class Rectangle : Shape
{
    private int m_width;
    private int m_height;

    public Rectangle(int width, int height, string id)
        : base(id)
    {
        m_width = width;
        m_height = height;
    }

    public override double Area
    {
        get
        {
```

```
            // Given the width and height, return the area of a rectangle:
            return m_width * m_height;
        }
    }
}
```

次のコードは、Shape から派生するオブジェクトを作成し、それらの面積を出力するテスト プログラムを示しています。

C#

```
// compile with: csc /reference:abstractshape.dll;shapes.dll shapetest.cs
class TestClass
{
    static void Main()
    {
        Shape[] shapes =
        {
            new Square(5, "Square #1"),
            new Circle(3, "Circle #1"),
            new Rectangle( 4, 5, "Rectangle #1")
        };

        System.Console.WriteLine("Shapes Collection");
        foreach (Shape s in shapes)
        {
            System.Console.WriteLine(s);
        }
    }
}
```

出力

```
Shapes Collection
Square #1 Area = 25.00
Circle #1 Area = 28.27
Rectangle #1 Area = 20.00
```

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)
プロパティ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

ポリモーフィズム (C# プログラミング ガイド)

継承を通じて、クラスは、固有の型、基本型、またはインターフェイスを実装する場合はインターフェイス型など、複数の型として使用できます。これをポリモーフィズムと言います。C# では、すべての型がポリモーフィックです。型は固有の型として使用することも、Object インスタンスとして使用することもできますが、これは、どの型も **Object** を基本型として自動的に取り扱うからです。

ポリモーフィズムは、派生クラスだけでなく、基本クラスに対しても重要です。base クラスを使用しているときは、実際には、基本クラス型にキャストされた派生クラスのオブジェクトを使用していることがあります。基本クラスを設計するときは、派生型で変更される可能性がある部分を予測できます。たとえば、自動車の基本クラスの場合は、問題の自動車がミニバンまたはコンバーチブルのときに変更する動作を含めることができます。その基本クラスでクラス メンバを virtual とマークすると、コンバーチブルやミニバンを表す派生クラスがその動作をオーバーライドできるようになります。

詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

ポリモーフィズムの概要

基本クラスから派生クラスを継承すると、派生クラスは、基本クラスのすべてのメソッド、フィールド、プロパティ、およびイベントを継承します。基本クラスのデータと動作を変更するには、2 つの方法があります。1 つは、基本メンバを新しい派生メンバに置き換える方法で、もう 1 つは、仮想基本メンバをオーバーライドする方法です。

基本クラスのメンバを新しい派生メンバに置き換えるには、new キーワードを使用する必要があります。基本クラスでメソッド、フィールド、またはプロパティを定義している場合、**new** キーワードを使用して、該当するメソッド、フィールド、またはプロパティの新しい定義を派生クラスで作成します。**new** キーワードは、置き換えられるクラス メンバの戻り値の型の前に配置します。次に例を示します。

C#

```
public class BaseClass
{
    public void DoWork() { }
    public int WorkField;
    public int WorkProperty
    {
        get { return 0; }
    }
}

public class DerivedClass : BaseClass
{
    public new void DoWork() { }
    public new int WorkField;
    public new int WorkProperty
    {
        get { return 0; }
    }
}
```

new キーワードを使用すると、置き換えられた基本クラス メンバの代わりに新しいクラス メンバが呼び出されます。置き換えられた基本クラス メンバは、隠しメンバと呼ばれます。隠しクラス メンバは、派生クラスのインスタンスが基本クラスのインスタンスにキャストされた場合に呼び出すことができます。次に例を示します。

C#

```
DerivedClass B = new DerivedClass();
B.DoWork();  // Calls the new method.

BaseClass A = (BaseClass)B;
A.DoWork();  // Calls the old method.
```

派生クラスのインスタンスが基本クラスのクラス メンバを完全に継承できるようにするには、基本クラスでそのメンバを virtual として宣言する必要があります。そのように宣言するには、そのメンバの戻り値の型の前に virtual キーワードを追加します。これにより、派生クラスでは、**new** の代わりに override キーワードを使用して、基本クラスの実装を派生クラス固有の実装に置き換えることができます。次に例を示します。

C#

```
public class BaseClass
{
    public virtual void DoWork() { }
    public virtual int WorkProperty
    {
        get { return 0; }
    }
}
public class DerivedClass : BaseClass
{
    public override void DoWork() { }
    public override int WorkProperty
    {
        get { return 0; }
    }
}
```

フィールドは仮想メンバにできません。仮想メンバにできるのは、メソッド、プロパティ、イベント、およびインデクサだけに限られます。派生クラスが仮想メンバをオーバーライドすると、派生クラスのメンバは、そのクラスのインスタンスが基本クラスのインスタンスとしてアクセスされるときでも呼び出されます。次に例を示します。

C#

```
DerivedClass B = new DerivedClass();
B.DoWork();  // Calls the new method.

BaseClass A = (BaseClass)B;
A.DoWork();  // Also calls the new method.
```

仮想メソッドや仮想プロパティにより、今後の拡張を前もって計画することできます。仮想メンバは、呼び出し元が使用している型とは無関係に呼び出されるので、基本クラスの明白な動作を派生クラスで完全に変更できるようにします。

仮想メンバは、それを最初に宣言したクラスとの間でどれほど多くのクラスが宣言されても、いつまでも仮想のままです。たとえば、クラス A が仮想メンバを宣言し、クラス B がクラス A から派生し、クラス C が クラス B から派生した場合、クラス C は仮想メンバを継承し、クラス B がその仮想メンバのオーバーライドを宣言した場合でも、そのメンバをオーバーライドできます。次に例を示します。

C#

```
public class A
{
    public virtual void DoWork() { }
}
public class B : A
{
    public override void DoWork() { }
}
```

C#

```
public class C : B
{
    public override void DoWork() { }
}
```

派生クラスでは、オーバーライドを sealed と宣言して仮想継承を中止できます。この場合、クラス メンバの宣言で、**override** キーワードの前に sealed キーワードを配置する必要があります。次に例を示します。

C#

```
public class C : B
{
    public sealed override void DoWork() { }
}
```

上の例では、DoWork メソッドは C から派生したすべてのクラスに対して仮想でなくなります。ただし、C のインスタンスに対しては、それが B 型や A 型にキャストされた場合でも、依然として仮想です。シール メソッドは、次のコード例に示すように、派生クラスで **new** キーワードを使用することで置き換えることができます。

C#

```
public class D : C
{
    public new void DoWork() { }
}
```

このコード例では、DoWork が、D 型の変数を使用して D で呼び出されると、新しい DoWork が呼び出されます。また、C 型、B 型、または A 型の変数を使用して D のインスタンスにアクセスした場合、DoWork への呼び出しは、仮想継承の規則に従って、クラス C の DoWork の実装に転送されます。

メソッドやプロパティを置き換えたり、オーバーライドしたりした派生クラスでは、base キーワードを使用することで、基本クラスのメソッドやプロパティにアクセスできます。次に例を示します。

C#

```
public class A
{
    public virtual void DoWork() { }
}
public class B : A
{
    public override void DoWork() { }
}
```

C#

```
public class C : B
{
    public override void DoWork()
    {
        // Call DoWork on B to get B's behavior:
        base.DoWork();

        // DoWork behavior specific to C goes here:
        // ...
    }
}
```

詳細については、「base」を参照してください。

メモ :

仮想メンバの場合、その固有の実装で **base** を使用して、その仮想メンバの基本クラス実装を呼び出すことをお勧めします。基本クラスの動作を実現することで、派生クラスは、その固有の動作を実装することに集中できます。基本クラス実装を呼び出さない場合は、基本クラスの動作と互換性のある動作を派生クラスで実現する必要があります。

このセクションの内容

詳細については以下を参照してください。

- Override キーワードと New キーワードによるバージョン管理 (C# プログラミング ガイド)
- Override キーワードと New キーワードを使用する場合について (C# プログラミング ガイド)
- 方法 : ToString メソッドをオーバーライドする (C# プログラミング ガイド)

参照

関連項目

継承 (C# プログラミング ガイド)
抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)
メソッド (C# プログラミング ガイド)
プロパティ (C# プログラミング ガイド)
インデクサ (C# プログラミング ガイド)

概念

C# プログラミング ガイド
C# プログラミング ガイド
イベント (C# プログラミング ガイド)

Override キーワードと New キーワードによるバージョン管理 (C# プログラミング ガイド)

C# 言語は、異なるライブラリの基本クラスと派生クラスの間でも、下位互換性を維持して改良できるようにバージョン管理がデザインされています。たとえば、C# では、派生クラスのメンバと同じ名前の新しいメンバを基本クラスに導入することが完全にサポートされているため、予期しない動作が起こることはありません。また、メソッドが継承メソッドをオーバーライドするかどうか、またはメソッドが同じ名前の継承メソッドを単純に隠す新しいメソッドかどうかをクラスに明示的に記述する必要があります。

C# では、基本クラスのメソッドと同じ名前のメソッドを派生クラスに含めることができます。

- 基本クラスのメソッドは、仮想と定義する必要があります。
- 派生クラスのメソッドの前に new キーワードや override キーワードが記述されていない場合、コンパイラは警告メッセージを表示し、そのメソッドは **new** キーワードが存在するように動作します。
- 派生クラスのメソッドの前に **new** キーワードが記述されている場合、そのメソッドは、基本クラスのメソッドから独立したものとして定義されます。
- 派生クラスのメソッドの前に **override** キーワードが記述されている場合、派生クラスのオブジェクトは、基本クラスのメソッドではなく、派生クラスのメソッドを呼び出します。
- **base** キーワードを使用すると、派生クラスの中から基本クラスのメソッドを呼び出すことができます。
- **override**、**virtual**、および **new** の各キーワードは、プロパティ、インデクサ、およびイベントにも適用できます。

既定では、C# メソッドは仮想でありません。メソッドが仮想と宣言されている場合、そのメソッドを継承するクラスでは、固有のバージョンを実装できます。メソッドを仮想メソッドにするには、**virtual** 修飾子を基本クラスのメソッド宣言で使用します。その場合、派生クラスでは、**override** キーワードを使用して基本クラスの仮想メソッドをオーバーライドすることも、**new** キーワードを使用して基本クラスの仮想メソッドを隠すこともできます。**override** キーワードも **new** キーワードも指定しない場合、コンパイラは警告メッセージを表示し、派生クラスのメソッドは基本クラスのメソッドを隠します。詳細については、「コンパイラの警告 (レベル 2) CS0108」を参照してください。

これを実際に示すために、`GraphicsClass` というクラスを A 社で作成しており、このクラスを自分のプログラムで使用するとします。`GraphicsClass` は、次のように記述されています。

C#

```
class GraphicsClass
{
    public virtual void DrawLine() { }
    public virtual void DrawPoint() { }
}
```

会社でこのクラスを使用しているので、このクラスから次のように独自のクラスを派生し、新しいメソッドを追加します。

C#

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public void DrawRectangle() { }
}
```

作成したアプリケーションは、A 社が次のような `GraphicsClass` の新バージョンをリリースするまで、問題なく使用できます。

C#

```
class GraphicsClass
{
    public virtual void DrawLine() { }
    public virtual void DrawPoint() { }
    public virtual void DrawRectangle() { }
}
```

GraphicsClass の新バージョンには、DrawRectangle というメソッドが含まれています。最初は、何の問題も発生しません。新バージョンは、以前のバージョンとバイナリ互換であり、新しいクラスがコンピュータ システムにインストールされても、配置済みのソフトウェアは引き続き正常に動作します。DrawRectangle メソッドへの既存の呼び出しは、派生クラスのバージョンを参照し続けます。

ただし、GraphicsClass の新バージョンを使ってアプリケーションを再コンパイルすると、コンパイラが警告メッセージを表示します。詳細については、「コンパイラの警告 (レベル 2) CS0108」を参照してください。

この警告の内容は、DrawRectangle メソッドをアプリケーションでどのように動作させるかを検討する必要があるというものです。

自分のメソッドで新しい基本クラスのメソッドをオーバーライドする場合は、次のように **override** キーワードを使用します。

C#

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public override void DrawRectangle() { }
}
```

override キーワードにより、YourDerivedGraphicsClass から派生したオブジェクトは、確実に DrawRectangle の派生クラス バージョンを使用します。YourDerivedGraphicsClass から派生したオブジェクトは、次のように base キーワードを使用して、DrawRectangle の基本クラス バージョンに引き続きアクセスできます。

C#

```
base.DrawRectangle();
```

自分のメソッドが、新しい基本クラスのメソッドをオーバーライドしないようにする場合は、2 つのメソッドを混同しないようにするために、自分のメソッドの名前を変更します。この作業には時間がかかり、エラーが発生しやすいので、状況によっては適切でないことがありますが、プロジェクトが比較的小規模である場合は、Visual Studio のリファクタリング オプションを使用して、メソッドの名前を変更できます。詳細については、「クラスおよび型のリファクタリング」を参照してください。

または、次のように派生クラスの定義で **new** キーワードを使用して警告を表示させないようにすることもできます。

C#

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public new void DrawRectangle() { }
}
```

new キーワードを使用すると、基本クラスに含まれている定義が派生クラスの定義によって隠されることがコンパイラに通知されます。これが既定の動作です。

オーバーライドとメソッドの選択

クラスでメソッドに名前を付けると、名前が同じで、渡されるパラメータと互換のパラメータを持つ 2 つのメソッドが存在する場合など、複数のメソッドが呼び出しに対応する場合に、呼び出すのに最適なメソッドを C# コンパイラが選択します。次の 2 つのメソッドは互換です。

C#

```
public class Derived : Base
{
    public override void DoWork(int param) { }
    public void DoWork(double param) { }
}
```

Derived のインスタンスで DoWork が呼び出されると、C# コンパイラは最初に、Derived で当初宣言された DoWork のバージョンと互換性のある呼び出しを実行します。オーバーライド メソッドは、クラスで宣言されたものと見なされません。これらは、基本クラスで宣言されたメソッドの新しい実装です。C# コンパイラは、Derived の元のメソッドにメソッド呼び出しを一致させることができない場合に限り、名前が同じで互換のパラメータを持つ、オーバーライドされたメソッドに呼び出しを一致させようとします。次に例を示します。

C#

```
int val = 5;
```

```
Derived d = new Derived();
d.DoWork(val);  // Calls DoWork(double).
```

変数 `val` は、暗黙的に double に変換できるので、C# コンパイラは、`DoWork(int)` の代わりに `DoWork(double)` を呼び出します。これを回避する方法は 2 つあります。1 つは、仮想メソッドと同じ名前を付けて新しいメソッドを宣言することを避ける方法です。もう 1 つは、`Derived` のインスタンスを `Base` にキャストすることにより、C# コンパイラに対して、基本クラスのメソッド リストを検索して、仮想メソッドを呼び出すように指示する方法です。メソッドが仮想であるため、`Derived` の `DoWork(int)` の実装が呼び出されます。次に例を示します。

C#

```
((Base)d).DoWork(val);  // Calls DoWork(int) on Derived.
```

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
メソッド (C# プログラミング ガイド)
継承 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

Override キーワードと New キーワードを使用する場合について (C# プログラミング ガイド)

C# では、新しいメソッドの取り扱い方を明確に規定している場合は、派生クラスのメソッドに基本クラスのメソッドと同じ名前を付けることができます。new と override の 2 つのキーワードの使用例を次に示します。

以下のコードでは、最初に `Car` という基本クラスと、そこから派生する `ConvertibleCar` クラスと `Minivan` クラスの、合わせて 3 つのクラスを宣言しています。基本クラスには、`DescribeCar` という 1 つのメソッドが含まれています。このメソッドは、自動車の説明をコンソールに送ります。また、派生クラスのメソッドにも `DescribeCar` というメソッドがあります。このメソッドは、それぞれに固有のプロパティを表示します。これらのメソッドも基本クラスの `DescribeCar` メソッドを呼び出し、`Car` クラスのプロパティをどのように継承しているかを示します。

この例では、違いを強調するために、`ConvertibleCar` クラスを **new** キーワードで定義し、`Minivan` クラスを **override** キーワードで定義しています。

C#

```
// Define the base class
class Car
{
    public virtual void DescribeCar()
    {
        System.Console.WriteLine("Four wheels and an engine.");
    }
}

// Define the derived classes
class ConvertibleCar : Car
{
    public new virtual void DescribeCar()
    {
        base.DescribeCar();
        System.Console.WriteLine("A roof that opens up.");
    }
}

class Minivan : Car
{
    public override void DescribeCar()
    {
        base.DescribeCar();
        System.Console.WriteLine("Carries seven people.");
    }
}
```

ここで、次のようなコードを記述できます。このコードでは、これらのクラスのインスタンスを宣言し、それぞれのメソッドを呼び出して、オブジェクトがオブジェクト自体を説明できるようにします。

C#

```
public static void TestCars1()
{
    Car car1 = new Car();
    car1.DescribeCar();
    System.Console.WriteLine("----------");

    ConvertibleCar car2 = new ConvertibleCar();
    car2.DescribeCar();
    System.Console.WriteLine("----------");

    Minivan car3 = new Minivan();
    car3.DescribeCar();
    System.Console.WriteLine("----------");
}
```

このコードの出力は次のようになります。

```
Four wheels and an engine.

----------

Four wheels and an engine.

A roof that opens up.

----------

Four wheels and an engine.

Carries seven people.

----------
```

次のコード セクションでは、`Car` 基本クラスから派生したオブジェクトの配列を宣言します。この配列には、`Car`、`ConvertibleCar`、`Minivan` の 3 つのオブジェクトを格納できます。この配列は、次のように宣言します。

C#

```
public static void TestCars2()
{
    Car[] cars = new Car[3];
    cars[0] = new Car();
    cars[1] = new ConvertibleCar();
    cars[2] = new Minivan();
}
```

これで、次のように **foreach** ループを使用して、配列内の各 `Car` オブジェクトにアクセスし、`DescribeCar` メソッドを呼び出すことができます。

C#

```
foreach (Car vehicle in cars)
{
    System.Console.WriteLine("Car object: " + vehicle.GetType());
    vehicle.DescribeCar();
    System.Console.WriteLine("----------");
}
```

このループの出力は次のようになります。

```
Car object: YourApplication.Car

Four wheels and an engine.

----------

Car object: YourApplication.ConvertibleCar

Four wheels and an engine.

----------

Car object: YourApplication.Minivan

Four wheels and an engine.

Carries seven people.

----------
```

`ConvertibleCar` の説明が、予想とは異なることに注意してください。このメソッドは、**new** キーワードで定義されているので、派生クラスのメソッドが呼び出されず、代わりに基本クラスのメソッドが呼び出されます。`Minivan` オブジェクトは、オーバーライドされたメソッドを正常に呼び出し、予想どおりの結果を生成します。

`Car` から派生したすべてクラスに `DescribeCar` メソッドを実装させる規則を適用する場合は、`DescribeCar` メソッドを **abstract** として定義する新しい基本クラスを作成する必要があります。抽象メソッドには、コードが含まれず、メソッド シグネチャだけが含まれます。この基本クラスから派生したクラスは、`DescribeCar` の実装を提供する必要があります。詳細については、「abstract (C# リファレンス)」を参照してください。

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

Override キーワードと New キーワードによるバージョン管理 (C# プログラミング ガイド)

方法：ToString メソッドをオーバーライドする (C# プログラミング ガイド)

C# では、すべてのオブジェクトが ToString メソッドを継承します。このメソッドは、該当するオブジェクトの文字列形式を返します。たとえば、int 型の変数はすべて **ToString** メソッドを持ち、次のようにその変数の内容を文字列として返すことができます。

C#

```
int x = 42;
string strx = x.ToString();
System.Console.WriteLine(strx);
```

カスタムのクラスまたは構造体を作成するときは、クライアント コードにカスタム型の情報を提供するため、**ToString** メソッドをオーバーライドする必要があります。

セキュリティに関するメモ：
このメソッドを使用して提供する情報を決定するときは、作成したクラスまたは構造体が信頼関係のないコードによって使用されるかどうかを考慮します。悪意があるコードで利用される可能性がある情報を提供しないように注意してください。

クラスまたは構造体内の OnString メソッドをオーバーライドするには

1. 次の修飾子および戻り値の値を指定して、**ToString** メソッドを定義します。

   ```
   public override string ToString(){}
   ```

2. 文字列を返すようにメソッドを実装します。

   次の例では、クラス名だけでなく、特定のクラス インスタンスに固有のデータも返されます。また、age 変数に対して **ToString** メソッドを実行し、**int** を出力可能な文字列に変換します。

   ```
   class Person
   {
     string name;
     int age;
     SampleObject(string name, int age)
     {
     this.name = name;
     this.age = age;
     }
     public override string ToString()
     {
        string s = age.ToString();
        return "Person: " + name + " " + s;
     }
   }
   ```

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
new (C# リファレンス)
override (C# リファレンス)
virtual (C# リファレンス)

概念

C# プログラミング ガイド

インターフェイス (C# プログラミング ガイド)

インターフェイスは、interface キーワードを使用して定義します。次に例を示します。

C#

```
interface IComparable
{
    int CompareTo(object obj);
}
```

インターフェイスは、任意のクラスまたは構造体に属する関連動作のグループを表します。インターフェイスは、メソッド、プロパティ、イベント、インデクサ、またはこれら 4 つのメンバの型を自由に組み合わせて構成できます。インターフェイスには、フィールドを含めることができません。インターフェイス メンバは自動的にパブリックになります。

クラスが基本クラスや構造体を継承できるのと同じように、クラスも構造体もインターフェイスを継承できますが、この場合、次の 2 点が異なります。

- クラスや構造体は、複数のインターフェイスを継承できます。
- クラスや構造体は、インターフェイスを継承するとき、メソッド名とシグネチャだけを継承します。これは、インターフェイス自体には実装が含まれていないからです。次に例を示します。

  C#

  ```
  public class Minivan : Car, IComparable
  {
      public int CompareTo(object obj)
      {
          //implementation of CompareTo
          return 0;  //if the Minivans are equal
      }
  }
  ```

インターフェイス メンバを実装するには、クラスの対応するメンバは、パブリックかつ非静的であり、その名前とシグネチャがインターフェイス メンバと一致する必要があります。クラスのプロパティとインデクサでは、インターフェイスで定義されているプロパティやインデクサに追加のアクセサを定義できます。たとえば、インターフェイスでは、get アクセサを持つプロパティを宣言できますが、このインターフェイスを実装するクラスでは、**get** と set の両方のアクセサを持つ同じプロパティを宣言できます。ただし、プロパティやインデクサで明示的な実装を使用する場合は、アクセサを一致させる必要があります。

インターフェイスとインターフェイス メンバは抽象であり、インターフェイスは既定の実装を提供しません。詳細については、
「抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)」を参照してください。

IComparable インターフェイスは、オブジェクトがそれ自体を同じ型の他のオブジェクトと比較できることをオブジェクトのユーザーに通知します。このインターフェイスのユーザーは、このような比較がどのように実装されるかを知る必要がありません。

インターフェイスは、他のインターフェイスを継承できます。クラスでは、継承した基本クラスやインターフェイスを介して、インターフェイスを繰り返し継承できます。このとき、クラスは、インターフェイスが新しいクラスの一部として宣言されている場合、インターフェイスを 1 回のみ実装できます。継承されたインターフェイスが新しいクラスの一部として宣言されていない場合、インターフェイスの実装は、それを宣言した基本クラスによって提供されます。基本クラスでは、仮想メンバを使ってインターフェイス メンバを実装できます。その場合、インターフェイスを継承するクラスでは、仮想メンバをオーバーライドしてインターフェイスの動作を変更できます。仮想メンバの詳細については、
「ポリモーフィズム (C# プログラミング ガイド)」を参照してください。

インターフェイスの概要

インターフェイスには、次の特徴があります。

- インターフェイスは抽象基本クラスに似ています。インターフェイスを継承する非抽象型は、すべてのインターフェイス メンバを実装する必要があります。
- インターフェイスは直接インスタンス化できません。

- インターフェイスには、イベント、インデクサ、メソッド、およびプロパティを含めることができます。
- インターフェイスには、メソッドの実装が含まれません。
- クラスと構造体は、複数のインターフェイスを継承できます。
- インターフェイス自体が複数のインターフェイスを継承できます。

このセクションの内容

- 明示的なインターフェイスの実装 (C# プログラミング ガイド)
- インターフェイスのプロパティ (C# プログラミング ガイド)
- インターフェイスのインデクサ (C# プログラミング ガイド)
- 方法 : インターフェイス イベントを実装する (C# プログラミング ガイド)
- 方法 : インターフェイス メンバを明示的に実装する (C# プログラミング ガイド)
- 方法 : 継承を使用してインターフェイス メンバを明示的に実装する (C# プログラミング ガイド)

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
継承 (C# プログラミング ガイド)
メソッド (C# プログラミング ガイド)
抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)
メソッド (C# プログラミング ガイド)
プロパティ (C# プログラミング ガイド)
インデクサ (C# プログラミング ガイド)

概念

C# プログラミング ガイド
ポリモーフィズム (C# プログラミング ガイド)
イベント (C# プログラミング ガイド)

明示的なインターフェイスの実装 (C# プログラミング ガイド)

同じシグネチャを持つメンバがそれぞれ存在する 2 つのインターフェイスをクラスが実装した場合、そのメンバをクラスで実装すると、両方のインターフェイスがそのメンバをそれぞれの実装として使用することになります。次に例を示します。

C#

```
interface IControl
{
    void Paint();
}
interface ISurface
{
    void Paint();
}
class SampleClass : IControl, ISurface
{
    // Both ISurface.Paint and IControl.Paint call this method.
    public void Paint()
    {
    }
}
```

ただし、2 つのインターフェイス メンバが同一の機能を実行しない場合は、これらのインターフェイスの一方または両方の実装が不適切になる可能性があります。そこで、インターフェイス メンバを明示的に実装し、特定のインターフェイス経由でのみ呼び出され、そのインターフェイスに固有のクラス メンバを作成できます。このように実装するには、インターフェイスの名前とピリオドを使ってクラス メンバに名前を付けます。次に例を示します。

C#

```
public class SampleClass : IControl, ISurface
{
    void IControl.Paint()
    {
        System.Console.WriteLine("IControl.Paint");
    }
    void ISurface.Paint()
    {
        System.Console.WriteLine("ISurface.Paint");
    }
}
```

クラス メンバ **IControl.Paint** は、**IControl** インターフェイス経由でのみ呼び出され、**ISurface.Paint** は **ISurface** 経由でのみ呼び出されます。これら 2 つのメソッド実装はそれぞれ独立しており、いずれもクラスで直接呼び出すことができません。次に例を示します。

C#

```
SampleClass obj = new SampleClass();
//obj.Paint();  // Compiler error.

IControl c = (IControl)obj;
c.Paint();  // Calls IControl.Paint on SampleClass.

ISurface s = (ISurface)obj;
s.Paint(); // Calls ISurface.Paint on SampleClass.
```

明示的な実装は、次のように、プロパティやメソッドなどの同じ名前を持つ別々のメンバを 2 つのインターフェイスがそれぞれ宣言するケースを解決する場合にも使用されます。

C#

```
interface ILeft
```

```
{
    int P { get;}
}
interface IRight
{
    int P();
}
```

これら両方のインターフェイスを実装する場合、クラスでは、プロパティ P またはメソッド P の一方または両方に明示的な実装を適用して、コンパイラのエラーを回避する必要があります。次に例を示します。

C#

```
class Middle : ILeft, IRight
{
    public int P() { return 0; }
    int ILeft.P { get { return 0; } }
}
```

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

インターフェイス (C# プログラミング ガイド)

継承 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

方法：インターフェイス メンバを明示的に実装する (C# プログラミング ガイド)

この例では、IDimensions インターフェイスと Box クラスが宣言されています。このクラスは、getLength と getWidth の各インターフェイス メンバを明示的に実装しています。メンバには、dimensions インターフェイス インスタンスを使ってアクセスします。

使用例

C#

```
interface IDimensions
{
    float getLength();
    float getWidth();
}

class Box : IDimensions
{
    float lengthInches;
    float widthInches;

    Box(float length, float width)
    {
        lengthInches = length;
        widthInches = width;
    }
    // Explicit interface member implementation:
    float IDimensions.getLength()
    {
        return lengthInches;
    }
    // Explicit interface member implementation:
    float IDimensions.getWidth()
    {
        return widthInches;
    }

    static void Main()
    {
        // Declare a class instance box1:
        Box box1 = new Box(30.0f, 20.0f);

        // Declare an interface instance dimensions:
        IDimensions dimensions = (IDimensions)box1;

        // The following commented lines would produce compilation
        // errors because they try to access an explicitly implemented
        // interface member from a class instance:
        //System.Console.WriteLine("Length: {0}", box1.getlength());
        //System.Console.WriteLine("Width: {0}", box1.getwidth());

        // Print out the dimensions of the box by calling the methods
        // from an instance of the interface:
        System.Console.WriteLine("Length: {0}", dimensions.getLength());
        System.Console.WriteLine("Width: {0}", dimensions.getWidth());
    }
}
```

出力

```
Length: 30
Width: 20
```

堅牢性の高いプログラム

- **Main** メソッド内の次の行は、コンパイル エラーを生じるため、コメント アウトされます。明示的に実装されるインターフェイス メンバには、クラス インスタンスからはアクセスできません。

  C#

  ```
  //System.Console.WriteLine("Length: {0}", box1.getlength());
  //System.Console.WriteLine("Width: {0}", box1.getwidth());
  ```

- **Main** メソッドの以下の行では、メソッドがインターフェイスのインスタンスから呼び出されるため、ボックスの寸法を正常に出力できます。

  C#

  ```
  System.Console.WriteLine("Length: {0}", dimensions.getLength());
  System.Console.WriteLine("Width: {0}", dimensions.getWidth());
  ```

参照

処理手順

方法 : 継承を使用してインターフェイス メンバを明示的に実装する (C# プログラミング ガイド)

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

インターフェイス (C# プログラミング ガイド)

概念

C# プログラミング ガイド

方法：継承を使用してインターフェイス メンバを明示的に実装する (C# プログラミング ガイド)

明示的なインターフェイス実装では、メンバ名が同じ 2 つのインターフェイスを実装し、各インターフェイス メンバに別々の実装を与えることもできます。この例では、ボックスの大きさをメートル法とヤード ポンド法の両方の単位で表示します。`Box` クラスは、異なる測定方式を表す `IEnglishDimensions` と `IMetricDimensions` の 2 つのインターフェイスを実装します。両方のインターフェイスは、`Length` と `Width` という同じメンバ名を持ちます。

使用例

C#

```
// Declare the English units interface:
interface IEnglishDimensions
{
    float Length();
    float Width();
}

// Declare the metric units interface:
interface IMetricDimensions
{
    float Length();
    float Width();
}

// Declare the Box class that implements the two interfaces:
// IEnglishDimensions and IMetricDimensions:
class Box : IEnglishDimensions, IMetricDimensions
{
    float lengthInches;
    float widthInches;

    public Box(float length, float width)
    {
        lengthInches = length;
        widthInches = width;
    }

    // Explicitly implement the members of IEnglishDimensions:
    float IEnglishDimensions.Length()
    {
        return lengthInches;
    }

    float IEnglishDimensions.Width()
    {
        return widthInches;
    }

    // Explicitly implement the members of IMetricDimensions:
    float IMetricDimensions.Length()
    {
        return lengthInches * 2.54f;
    }

    float IMetricDimensions.Width()
    {
        return widthInches * 2.54f;
    }

    static void Main()
    {
        // Declare a class instance box1:
        Box box1 = new Box(30.0f, 20.0f);
```

```
        // Declare an instance of the English units interface:
        IEnglishDimensions eDimensions = (IEnglishDimensions)box1;

        // Declare an instance of the metric units interface:
        IMetricDimensions mDimensions = (IMetricDimensions)box1;

        // Print dimensions in English units:
        System.Console.WriteLine("Length(in): {0}", eDimensions.Length());
        System.Console.WriteLine("Width (in): {0}", eDimensions.Width());

        // Print dimensions in metric units:
        System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());
        System.Console.WriteLine("Width (cm): {0}", mDimensions.Width());
    }
}
```

出力

```
Length(in): 30
Width (in): 20
Length(cm): 76.2
Width (cm): 50.8
```

堅牢性の高いプログラム

既定の測定値を英語単位系にする場合は、普通に `Length` メソッドと `Width` メソッドを実装し、次のように `IMetricDimensions` インターフェイスから Length メソッドと Width メソッドを明示的に実装します。

C#

```
// Normal implementation:
public float Length()
{
    return lengthInches;
}
public float Width()
{
    return widthInches;
}

// Explicit implementation:
float IMetricDimensions.Length()
{
    return lengthInches * 2.54f;
}
float IMetricDimensions.Width()
{
    return widthInches * 2.54f;
}
```

この場合、ヤード ポンド単位にはクラス インスタンスからアクセスでき、メートル単位にはインターフェイス インスタンスからアクセスできます。

C#

```
public static void Test()
{
    Box box1 = new Box(30.0f, 20.0f);
    IMetricDimensions mDimensions = (IMetricDimensions)box1;

    System.Console.WriteLine("Length(in): {0}", box1.Length());
    System.Console.WriteLine("Width (in): {0}", box1.Width());
    System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());
    System.Console.WriteLine("Width (cm): {0}", mDimensions.Width());
}
```

参照

処理手順

方法 : インターフェイス メンバを明示的に実装する (C# プログラミング ガイド)

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

インターフェイス (C# プログラミング ガイド)

概念

C# プログラミング ガイド

メンバ (C# プログラミング ガイド)

クラスと構造体には、そのデータおよび動作を表すメンバがあります。これらのメンバは以下のとおりです。

フィールド (C# プログラミング ガイド)

フィールドは、一般にクラス データを保持し、クラスの一部と見なされるオブジェクトのインスタンスです。たとえば、Calender クラスには、現在の日付を格納するフィールドがあります。

プロパティ (C# プログラミング ガイド)

プロパティはクラスのメソッドで、そのクラスのフィールドのようにアクセスされます。プロパティは、クラスのフィールドを保護し、オブジェクトが認識することなくフィールドが変更されるのを防止できます。

メソッド (C# プログラミング ガイド)

メソッドは、クラスが実行できるアクションを定義します。メソッドは、入力データを提供するパラメータを受け取り、パラメータを通じて出力データを返すことができます。メソッドは、パラメータを使用せずに値を直接返すこともできます。

イベント (C# プログラミング ガイド)

イベントは、ボタンのクリックやメソッドの正常な終了などの発生に関する通知を他のオブジェクトに提供する手段です。イベントを定義し、トリガするには、デリゲートを使用します。詳細については、「イベントとデリゲート」を参照してください。

演算子 (C# プログラミング ガイド)

演算子は、オペランドの演算を実行する用語または記号です (+、*、< など)。演算子は、カスタム データ型の演算を実行するよう再定義できます。詳細については、「オーバーロードされた演算子 (C# プログラミング ガイド)」を参照してください。

インデクサ (C# プログラミング ガイド)

インデクサを使用すると、配列と同じようにオブジェクトにインデックスを付けることができます。

コンストラクタ (C# プログラミング ガイド)

コンストラクタは、オブジェクトを初めて作成するときに呼び出されるメソッドです。コンストラクタは、一般にオブジェクトのデータを初期化するために使用します。

デストラクタ (C# プログラミング ガイド)

デストラクタは、オブジェクトをメモリから削除するときにランタイム実行エンジンによって呼び出されるメソッドです。デストラクタは、通常、解放する必要があるリソースが適切に処理されるようにするために使用します。

入れ子にされた型 (C# プログラミング ガイド)

入れ子にされた型は、クラスや構造体の中で宣言された型です。入れ子にされた型は、通常、それを格納している型だけで使用されるオブジェクトを表すために使用します。

参照

関連項目

メソッド (C# プログラミング ガイド)
メソッド (C# プログラミング ガイド)
デストラクタ (C# プログラミング ガイド)
プロパティ (C# プログラミング ガイド)
フィールド (C# プログラミング ガイド)
インデクサ (C# プログラミング ガイド)
入れ子にされた型 (C# プログラミング ガイド)
演算子 (C# プログラミング ガイド)
オーバーロードされた演算子 (C# プログラミング ガイド)

概念

C# プログラミング ガイド
クラス (C# プログラミング ガイド)
イベント (C# プログラミング ガイド)
イベントとデリゲート

その他の技術情報

メンバのデザインのガイドライン
コンストラクタ (C# プログラミング ガイド)

メソッド (C# プログラミング ガイド)

メソッドは、一連のステートメントを含むコード ブロックです。C# では、実行されるすべての命令がメソッドのコンテキストで実行されます。

メソッドは、クラスまたは構造体の中で、アクセス レベル、戻り値、メソッドの名前、およびメソッド パラメータを指定して宣言します。メソッド パラメータはかっこで囲み、コンマで区切って指定します。メソッドでパラメータが不要な場合は、かっこ内を空にします。次に 3 つのメソッドを含むクラスの例を示します。

C#

```
class Motorcycle
{
    public void StartEngine() { }
    public void AddGas(int gallons) { }
    public int Drive(int miles, int speed) { return 0; }
}
```

オブジェクトでメソッドを呼び出すのは、フィールドにアクセスするのと似ています。オブジェクト名の後に、ピリオド、メソッド名、およびかっこを追加します。引数はかっこの中に記述し、コンマで区切ります。そのため、たとえば、`Motorcycle` クラスのメソッドは、次のように呼び出すことができます。

C#

```
Motorcycle moto = new Motorcycle();

moto.StartEngine();
moto.AddGas(15);
moto.Drive(5, 20);
```

メソッド パラメータ

上のコード スニペットに示されているように、メソッドに引数を渡す場合は、メソッドを呼び出すときにかっこの中に引数を入力するだけで済みます。呼び出されるメソッドの側から見た場合、入ってくる引数をパラメータと言います。

メソッドが受け取るパラメータもかっこ内に指定しますが、パラメータごとに型と名前を指定する必要があります。この名前は、引数と同じにする必要はありません。次に例を示します。

C#

```
public static void PassesInteger()
{
    int fortyFour = 44;
    TakesInteger(fortyFour);
}
static void TakesInteger(int i)
{
    i = 33;
}
```

この例では、`PassesInteger` というメソッドが `TakesInteger` というメソッドに引数を渡します。`PassesInteger` では、引数は `fortyFour` という名前ですが、`TakeInteger` では、この引数は `i` という名前のパラメータです。このパラメータは、`TakesInteger` メソッドの中だけに存在します。このメソッドの内部で宣言されたパラメータまたは変数でない限り、他の任意の数の変数に `i` という名前を付け、任意の型を指定できます。

`TakesInteger` は、指定された引数に新しい値を代入します。この変更は、`TakeInteger` から制御が戻ると `PassesInteger` メソッドに反映するように思われますが、実際には、`fortyFour` 変数の値は変更されません。これは、**int** が値型であるためです。既定では、値型がメソッドに渡されるときは、オブジェクト自体ではなく、そのコピーが渡されます。そのため、パラメータに対して行われた変更は、呼び出し元のメソッドに影響しません。値型という名前は、オブジェクト自体ではなく、オブジェクトのコピーが渡されるという事実に基づくものです。値は渡されますが、同じオブジェクトではありません。

値型の引き渡しの詳細については、「値型のパラメータの引き渡し (C# プログラミング ガイド)」を参照してください。C# に不可欠な値型の一覧については、「値型の一覧表 (C# リファレンス)」を参照してください。

値型は、参照渡しされる参照型と異なります。参照型に基づくオブジェクトがメソッドに渡される場合、オブジェクトのコピーは作成されません。代わりに、メソッドの引数として使用されるオブジェクトへの参照が作成され、渡されます。そのため、この参照を通じて行われた変更は、呼び出し元のメソッドに反映されます。参照型を作成するときは、**class** キーワードを使用します。この例を次に示します。

C#

```
public class SampleRefType
{
    public int value;
}
```

この型に基づくオブジェクトがメソッドに渡される場合は、参照渡しされます。次に例を示します。

C#

```
public static void TestRefType()
{
    SampleRefType rt = new SampleRefType();
    rt.value = 44;
    ModifyObject(rt);
    System.Console.WriteLine(rt.value);
}
static void ModifyObject(SampleRefType obj)
{
    obj.value = 33;
}
```

この例は、基本的に前の例と同じです。しかし、参照型を使用しているため、`ModifyObject` によって行われた変更は、`TestRefType` メソッドで作成されたオブジェクトに反映されます。そのため、`TestRefType` メソッドは、値として 33 を表示します。

詳細については、「参照型のパラメータの引き渡し (C# プログラミング ガイド)」および「参照型 (C# リファレンス)」を参照してください。

戻り値

メソッドは、呼び出し元に値を返すことができます。戻り値の型 (メソッド名の前に記述されている型) が **void** でない場合、メソッドは、**return** キーワードを使用して値を返すことができます。return キーワードと、その後に戻り値の型に一致する値が記述されたステートメントは、その値をメソッドの呼び出し元に返します。また、**return** キーワードは、メソッドの実行を中止します。戻り値の型が **void** の場合でも、値を持たない **return** ステートメントは、メソッドの実行を中止するのに役立ちます。**return** キーワードを使用しないと、メソッドは、コード ブロックの最後に到達したときに実行を中止します。戻り値の型が void 以外のメソッドで値を返すには、**return** キーワードを使用する必要があります。たとえば、次の 2 つのメソッドは、**return** キーワードを使用して整数を返します。

C#

```
class SimpleMath
{
    public int AddTwoNumbers(int number1, int number2)
    {
        return number1 + number2;
    }

    public int SquareANumber(int number)
    {
        return number * number;
    }
}
```

メソッドから返された値を使用するために、呼び出し元のメソッドは、同じ型の値であれば、メソッド呼び出し自体を使用できます。戻り値は、変数に代入することもできます。たとえば、次の 2 つのコード例では、同様の結果が得られます。

C#

```
int result = obj.AddTwoNumbers(1, 2);
obj.SquareANumber(result);
```

C#

```
obj.SquareANumber(obj.AddTwoNumbers(1, 2));
```

中間変数 (上の例では、`result`) を使用して値を格納する処理は、省略可能です。中間変数を使用すると、コードが読みやすくなり、また値を繰り返し使用する場合は、必要になることもあります。

詳細については、「return (C# リファレンス)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.5 メソッド
- 10.5 メソッド

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
アクセス修飾子 (C# プログラミング ガイド)
静的クラスと静的クラス メンバ (C# プログラミング ガイド)
継承 (C# プログラミング ガイド)
params (C# リファレンス)
return (C# リファレンス)
out (C# リファレンス)
ref (C# リファレンス)
パラメータの引き渡し (C# プログラミング ガイド)

概念

C# プログラミング ガイド

パラメータの引き渡し (C# プログラミング ガイド)

C# では、パラメータは、値または参照で渡されます。パラメータを参照で渡すと、関数メンバ (メソッド、プロパティ、インデクサ、演算子、およびコンストラクタ) はパラメータの値を変更し、その変更を永続化できます。パラメータを参照で渡すには、**ref** キーワードまたは **out** キーワードを使用します。ここでは、説明を簡単にするために、例に **ref** キーワードだけを使用しています。**ref** と **out** の違いの詳細については、「ref」、「out」、および「ref と out を使用した配列の受け渡し」を参照してください。たとえば、次のようにします。

C#

```
// Passing by value
static void Square(int x)
{
    // code...
}
```

C#

```
// Passing by reference
static void Square(ref int x)
{
    // code...
}
```

ここでは、次の内容について説明します。

- 値型のパラメータの引き渡し
- 参照型のパラメータの引き渡し

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.5.1 パラメータ
- 5.1.4 値パラメータ
- 5.1.5 参照パラメータ
- 5.1.6 出力パラメータ
- 10.5.1 メソッド パラメータ

参照
関連項目
メソッド (C# プログラミング ガイド)
概念
C# プログラミング ガイド

値型のパラメータの引き渡し (C# プログラミング ガイド)

参照型の変数はデータへの参照を持つのに対し、値型の変数はデータを直接格納します。このため、値型変数をメソッドに渡すことは、メソッドに変数のコピーを渡すことを意味します。メソッド内でパラメータが変更されても、変数に格納されている元のデータには影響しません。呼び出されたメソッドでパラメータの値を変更する場合は、ref キーワードまたは out キーワードを使用して、パラメータを参照で渡す必要があります。説明を簡単にするために、次の例では **ref** だけを使用しています。

例 : 値による値型の引き渡し

値によって値型を渡す方法を次の例で示します。変数 `n` は、`SquareIt` メソッドに値で渡されます。メソッド内で変更があっても、元の変数値には影響しません。

C#

```
class PassingValByVal
{
    static void SquareIt(int x)
    // The parameter x is passed by value.
    // Changes to x will not affect the original value of x.
    {
        x *= x;
        System.Console.WriteLine("The value inside the method: {0}", x);
    }
    static void Main()
    {
        int n = 5;
        System.Console.WriteLine("The value before calling the method: {0}", n);

        SquareIt(n);  // Passing the variable by value.
        System.Console.WriteLine("The value after calling the method: {0}", n);
    }
}
```

出力

```
The value before calling the method: 5

The value inside the method: 25

The value after calling the method: 5
```

コードの説明

値型の変数 `n` には、データ (値は `5`) が格納されています。`SquareIt` が呼び出されると、`n` の内容がパラメータ `x` にコピーされ、値はメソッド内で 2 乗されます。ただし、`Main` では `SquareIt` メソッドの呼び出し前後で `n` の値は変わりません。実際、メソッド内での変更は、ローカル変数 `x` だけに影響します。

例 : 参照による値型の引き渡し

ref キーワードを使用してパラメータを渡すことを除けば、次の例は前の例と同じです。パラメータの値は、メソッドの呼び出し後に変更されています。

C#

```
class PassingValByRef
{
    static void SquareIt(ref int x)
    // The parameter x is passed by reference.
    // Changes to x will affect the original value of x.
    {
        x *= x;
        System.Console.WriteLine("The value inside the method: {0}", x);
    }
    static void Main()
    {
        int n = 5;
        System.Console.WriteLine("The value before calling the method: {0}", n);
```

```
            SquareIt(ref n);  // Passing the variable by reference.
            System.Console.WriteLine("The value after calling the method: {0}", n);
        }
    }
```

出力

```
The value before calling the method: 5

The value inside the method: 25

The value after calling the method: 25
```

コードの説明

この例では、`n` の値ではなく、`n` への参照が渡されています。パラメータ `x` は `int` ではなく、**int** への参照 (例では、`n` への参照) です。このため、メソッド内で `x` が 2 乗されると、`x` が参照している `n` が 2 乗されます。

例 : 値型の交換

渡されたパラメータの値を変更する例として一般的なのは、`Swap` メソッドです。このメソッドに `x` と `y` の 2 つの変数を渡すと、内容が交換されます。`Swap` メソッドにはパラメータを参照で渡す必要があります。参照で渡さないと、メソッド内ではパラメータのローカル コピーを処理することになります。参照のパラメータを使用する `Swap` メソッドの例を次に示します。

C#

```
    static void SwapByRef(ref int x, ref int y)
    {
        int temp = x;
        x = y;
        y = temp;
    }
```

このメソッドを呼び出すときは、**ref** キーワードを次のように使用します。

C#

```
    static void Main()
    {
        int i = 2, j = 3;
        System.Console.WriteLine("i = {0}  j = {1}" , i, j);

        SwapByRef (ref i, ref j);

        System.Console.WriteLine("i = {0}  j = {1}" , i, j);
    }
```

出力

```
i = 2 j = 3

i = 3 j = 2
```

参照

関連項目

パラメータの引き渡し (C# プログラミング ガイド)

参照型のパラメータの引き渡し (C# プログラミング ガイド)

概念

C# プログラミング ガイド

参照型のパラメータの引き渡し (C# プログラミング ガイド)

参照型の変数には、データが直接格納されず、データへの参照が格納されます。参照型のパラメータを値で渡すときには、クラス メンバの値など、参照によって指されるデータを変更できます。ただし、参照自身の値は変更できません。つまり、同じ参照を使用して、新しいクラスのメモリを割り当て、ブロックの外で永続化させることはできません。参照自身の値を変更するには、ref キーワードまたは out キーワードを使用してパラメータを渡します。説明を簡単にするために、次の例では **ref** だけを使用しています。

例 : 値による参照型の引き渡し

参照型のパラメータ `arr` を値で `Change` メソッドに渡す方法を、次の例で示します。パラメータは `arr` への参照であるため、配列要素の値を変更できます。ただし、他のメモリ位置へのパラメータの再割り当ては、メソッド内だけで有効です。元の変数 `arr` には影響しません。

C#

```
class PassingRefByVal
{
    static void Change(int[] pArray)
    {
        pArray[0] = 888;  // This change affects the original element.
        pArray = new int[5] {-3, -1, -2, -3, -4};   // This change is local.
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0])
;
    }

    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine("Inside Main, before calling the method, the first element
is: {0}", arr [0]);

        Change(arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element
is: {0}", arr [0]);
    }
}
```

出力

```
Inside Main, before calling the method, the first element is: 1

Inside the method, the first element is: -3

Inside Main, after calling the method, the first element is: 888
```

コードの説明

上の例では、参照型の配列 `arr` は、*ref* パラメータを指定せずにメソッドに渡されています。このような場合は、`arr` を指す参照のコピーがメソッドに渡されます。出力は、メソッドが配列要素の内容を (この場合は `1` から `888` へ) 変更できることを示しています。ただし、`Change` メソッド内で new 演算子を使用して新しいメモリ領域を割り当てると、変数 `pArray` は新しい配列を参照します。したがって、新しい領域の割り当ての後に変更があっても、`Main` 内で作成されている元の配列 `arr` は影響を受けません。この例では `Main` メソッド内と `Change` メソッド内で 2 つの配列が作成されています。

例 : 参照による参照型の引き渡し

メソッド ヘッダーと呼び出しで **ref** キーワードを使用していることを除けば、この例は前の例と同じです。メソッド内での変更は、呼び出しプログラム内の元の変数に影響します。

C#

```
class PassingRefByRef
{
    static void Change(ref int[] pArray)
    {
        // Both of the following changes will affect the original variables:
        pArray[0] = 888;
        pArray = new int[5] {-3, -1, -2, -3, -4};
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0])
```

```
;
    }

    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine("Inside Main, before calling the method, the first element
is: {0}", arr[0]);

        Change(ref arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element
is: {0}", arr[0]);
    }
}
```

出力

```
Inside Main, before calling the method, the first element is: 1

Inside the method, the first element is: -3

Inside Main, after calling the method, the first element is: -3
```

コードの説明

メソッド内でのすべての変更が、`Main` 内の元の配列に影響します。実際、元の配列は **new** 演算子を使用して再割り当てされています。つまり、`Change` メソッドの呼び出し後は、`arr` へのすべての参照が、`Change` メソッドで作成された 5 つの要素を持つ配列を指しています。

例 : 2 つの文字列の交換

文字列の交換は、参照によって参照型のパラメータを渡す方法の良い例です。次の例では、`str1` と `str2` の 2 つの文字列が `Main` で初期化され、**ref** キーワードを使用してパラメータとして `SwapStrings` メソッドに渡されています。2 つの文字列はこのメソッド内で交換され、`Main` 内でもその結果が反映されています。

C#

```
class SwappingStrings
{
    static void SwapStrings(ref string s1, ref string s2)
    // The string parameter is passed by reference.
    // Any changes on parameters will affect the original variables.
    {
        string temp = s1;
        s1 = s2;
        s2 = temp;
        System.Console.WriteLine("Inside the method: {0} {1}", s1, s2);
    }

    static void Main()
    {
        string str1 = "John";
        string str2 = "Smith";
        System.Console.WriteLine("Inside Main, before swapping: {0} {1}", str1, str2);

        SwapStrings(ref str1, ref str2);   // Passing strings by reference
        System.Console.WriteLine("Inside Main, after swapping: {0} {1}", str1, str2);
    }
}
```

出力

```
Inside Main, before swapping: John Smith

Inside the method: Smith John

Inside Main, after swapping: Smith John
```

コードの説明

この例では、呼び出しプログラムの変数に変更を反映するために、パラメータを参照で渡す必要があります。メソッド ヘッダーとメソッド呼び出しの

両方から **ref** キーワードを削除すると、呼び出しプログラムには変更が反映されません。

文字列の詳細については、「string (C# リファレンス)」を参照してください。

参照
関連項目
パラメータの引き渡し (C# プログラミング ガイド)
ref と out を使用した配列の引き渡し (C# プログラミング ガイド)
ref (C# リファレンス)
参照型 (C# リファレンス)
概念
C# プログラミング ガイド

コンストラクタ (C# プログラミング ガイド)

クラスまたは構造体を作成する場合は、そのコンストラクタが必ず呼び出されます。クラスや構造体には、異なる引数を受け取る複数のコンストラクタがある場合があります。コンストラクタを使用すると、既定値の設定、インスタンス化の制限、柔軟で読み取りやすいコードの記述などを行うことができます。

オブジェクトのコンストラクタを指定していない場合、C# は、オブジェクトをインスタンス化し、「既定値の一覧表 (C# リファレンス)」に記載されている既定値をすべてのメンバ変数に設定するコンストラクタを既定で生成します。静的クラスおよび構造体も、コンストラクタを持つことができます。

このセクションの内容

関連するセクション

コンストラクタの使用 (C# プログラミング ガイド)

コンストラクタは、特定の型のオブジェクトを作成するときに実行されるクラス メソッドです。コンストラクタはクラスと同じ名前を持ち、通常、新しいオブジェクトのデータ メンバを初期化します。

次の例では、`Taxi` というクラスを簡単なコンストラクタで定義しています。このクラスは、次に new 演算子によってインスタンス化されます。新しいオブジェクトにメモリが割り当てられるとすぐに、`Taxi` コンストラクタが **new** 演算子によって呼び出されます。

C#

```
public class Taxi
{
    public bool isInitialized;
    public Taxi()
    {
        isInitialized = true;
    }
}

class TestTaxi
{
    static void Main()
    {
        Taxi t = new Taxi();
        System.Console.WriteLine(t.isInitialized);
    }
}
```

パラメータを受け取らないコンストラクタを既定のコンストラクタと呼びます。既定のコンストラクタは、**new** 演算子を使用してオブジェクトをインスタンス化する際に **new** に引数が渡されない場合に呼び出されます。詳細については、「インスタンス コンストラクタ (C# プログラミング ガイド)」を参照してください。

クラスが静的である場合を除き、コンストラクタが存在しないクラスには、C# コンパイラによりパブリックな既定のコンストラクタが割り当てられ、クラスをインスタンス化できるようになります。詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」を参照してください。

次のようにコンストラクタをプライベートにすると、クラスがインスタンス化されないようにできます。

C#

```
class NLog
{
    // Private Constructor:
    private NLog() { }

    public static double e = System.Math.E;  //2.71828...
}
```

詳細については、「プライベート コンストラクタ (C# プログラミング ガイド)」を参照してください。

struct 型のコンストラクタはクラス コンストラクタに似ていますが、**structs** には、既定のコンストラクタがコンパイラによって自動的に提供されるため、明示的な既定のコンストラクタを含めることができません。このコンストラクタは、構造体の各フィールドを、「既定値の一覧表 (C# リファレンス)」に掲載されている既定値に初期化します。ただし、この既定のコンストラクタは、構造体が **new** によってインスタンス化される場合にのみ呼び出されます。たとえば、次のコードでは、Int32 に対して既定のコンストラクタが使用されるため、確実に整数を初期化できます。

```
int i = new int();
Console.WriteLine(i);
```

しかし、次のコードでは、**new** を使用せず、さらに初期化されていないオブジェクトの使用を試みるため、コンパイラ エラー CS0165 が生成されます。

```
int i;
```

```
Console.WriteLine(i);
```

これに対して、**structs** に基づくオブジェクトは次のように初期化や代入ができるため、使用できます。

```
int a = 44;  // Initialize the value type...
int b;
b = 33;      // Or assign it before using it.
Console.WriteLine("{0}, {1}", a, b);
```

そのため、値型の既定のコンストラクタを呼び出す必要がありません。

クラスも **structs** も共に、パラメータを受け取るコンストラクタを定義できます。パラメータを受け取るコンストラクタは、**new** ステートメントまたは base ステートメントを使用して呼び出す必要があります。クラスと **structs** では複数のコンストラクタも定義でき、クラスと構造体のどちらも既定のコンストラクタを定義する必要はありません。次に例を示します。

C#

```
public class Employee
{
    public int salary;

    public Employee(int annualSalary)
    {
        salary = annualSalary;
    }

    public Employee(int weeklySalary, int numberOfWeeks)
    {
        salary = weeklySalary * numberOfWeeks;
    }
}
```

このクラスは、次のいずれかのステートメントを使用して作成できます。

C#

```
Employee e1 = new Employee(30000);
Employee e2 = new Employee(500, 52);
```

コンストラクタでは、**base** キーワードを使用して、基本クラスのコンストラクタを呼び出すことができます。次に例を示します。

C#

```
public class Manager : Employee
{
    public Manager(int annualSalary)
        : base(annualSalary)
    {
        //Add further instructions here.
    }
}
```

この例では、コンストラクタのブロックを実行する前に基本クラスのコンストラクタを呼び出しています。**base** キーワードは、パラメータの有無に関係なく使用することもできます。コンストラクタのパラメータは、**base** のパラメータまたは、式の一部として使用できます。詳細については、「base」を参照してください。

派生クラスでは、**base** キーワードを使用して基本クラスのコンストラクタを明示的に呼び出さないと、既定のコンストラクタ (存在する場合) が暗黙的に呼び出されます。そのため、次に示すコンストラクタの宣言も実質的に同じです。

C#

```
public Manager(int initialdata)
{
    //Add further instructions here.
```

```
}
```

C#

```
public Manager(int initialdata) : base()
{
    //Add further instructions here.
}
```

基本クラスが既定のコンストラクタを提供しない場合、派生クラスでは、**base** を使って基本コンストラクタを明示的に呼び出す必要があります。

コンストラクタで this キーワードを使用すると、同じオブジェクトで別のコンストラクタを呼び出すことができます。**base** と同様に、**this** もパラメータの有無に関係なく使用でき、コンストラクタのパラメータはいずれも **this** のパラメータとしても、式の一部としても使用できます。たとえば、上の例の 2 番目のコンストラクタは、**this** を使用して次のように書き直すことができます。

C#

```
public Employee(int weeklySalary, int numberOfWeeks)
    : this(weeklySalary * numberOfWeeks)
{
}
```

上の **this** キーワードを使用すると、次のコンストラクタが呼び出されます。

C#

```
public Employee(int annualSalary)
{
    salary = annualSalary;
}
```

コンストラクタは、public、private、protected、internal、または **protected internal** とマークできます。これらのアクセス修飾子により、クラスのユーザーがクラスを作成する方法が定義されます。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

コンストラクタは、static キーワードを使用して静的と宣言できます。静的コンストラクタは、静的フィールドがアクセスされる直前に自動的に呼び出され、一般に静的なクラス メンバを初期化するために使用されます。詳細については、「静的コンストラクタ (C# プログラミング ガイド)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.1 コンストラクタ
- 10.10 インスタンス コンストラクタ (クラス)
- 10.11 インスタンス コンストラクタ (クラス)
- 11.3.8 コンストラクタ (構造体)
- 11.3.10 静的コンストラクタ (構造体)

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

デストラクタ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

コンストラクタ (C# プログラミング ガイド)

インスタンス コンストラクタ (C# プログラミング ガイド)

インスタンス コンストラクタは、インスタンスを作成および初期化するために使用します。次の例に示すように新しいオブジェクトを作成すると、クラス コンストラクタが呼び出されます。

C#

```
class CoOrds
{
    public int x, y;

    // constructor
    public CoOrds()
    {
        x = 0;
        y = 0;
    }
}
```

メモ :

このクラスには、パブリック データ メンバが含まれていますが、これはわかりやすくするためであり、推奨されるプログラミング手法ではありません。この場合、プログラム内のすべてのメソッドに、オブジェクトの内部処理への無制限で検証されないアクセスが許可されるからです。通常、データ メンバはプライベートにし、クラス メソッドとプロパティのみを介してアクセスする必要があります。

このコンストラクタは、`CoOrds` クラスからオブジェクトを作成するときに呼び出されます。引数を取らないこのようなコンストラクタを既定のコンストラクタと呼びます。これは、コンストラクタを追加する場合に便利です。たとえば、`CoOrds` クラスに、データ メンバの初期値を指定できるコンストラクタを追加できます。

C#

```
// A constructor with two arguments:
public CoOrds(int x, int y)
{
    this.x = x;
    this.y = y;
}
```

これにより、次のように、既定値や特定の初期値を使って `CoOrd` オブジェクトを作成できます。

C#

```
CoOrds p1 = new CoOrds();
CoOrds p2 = new CoOrds(5, 3);
```

クラスに既定のコンストラクタが存在しない場合は、コンストラクタが自動的に生成され、既定値を使ってオブジェクト フィールドが初期化されます (たとえば、int は 0 (ゼロ) に初期化されます)。既定値の詳細については、「既定値の一覧表 (C# リファレンス)」を参照してください。したがって、`CoOrds` クラスの既定のコンストラクタはすべてのデータ メンバをゼロに初期化するので、クラスの機能を変更せずに完全に削除できます。下の「例 1」は複数のコンストラクタを使用する完全な例で、「例 2」は自動的に生成されるコンストラクタの例を示しています。

また、インスタンス コンストラクタを使用すると、基本クラスのインスタンス コンストラクタを呼び出すこともできます。クラス コンストラクタは、初期化子を通じて基本クラスのコンストラクタを呼び出すことができます。次にその例を示します。

C#

```
class Circle : Shape
{
    public Circle(double radius)
        : base(radius, 0)
    {
    }
```

```
}
```

この例の `Circle` クラスは、`Circle` の派生元の `Shape` によって提供されるコンストラクタに、半径と高さを表す値を渡します。下の「例 3」は、`Shape` と `Circle` を使用する完全な例を示しています。

例 1

次の例は、クラス コンストラクタを 2 つ使用するクラスを示しています。1 つのコンストラクタには引数がなく、もう 1 つのコンストラクタには引数が 2 つあります。

C#

```
class CoOrds
{
    public int x, y;

    // Default constructor:
    public CoOrds()
    {
        x = 0;
        y = 0;
    }

    // A constructor with two arguments:
    public CoOrds(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    // Override the ToString method:
    public override string ToString()
    {
        return (System.String.Format("({0},{1})", x, y));
    }
}

class MainClass
{
    static void Main()
    {
        CoOrds p1 = new CoOrds();
        CoOrds p2 = new CoOrds(5, 3);

        // Display the results using the overriden ToString method:
        System.Console.WriteLine("CoOrds #1 at {0}", p1);
        System.Console.WriteLine("CoOrds #2 at {0}", p2);
    }
}
```

出力

```
CoOrds #1 at (0,0)
CoOrds #2 at (5,3)
```

例 2

次の例の `Person` クラスにはコンストラクタがありません。この場合は、既定のコンストラクタが自動的に使用され、フィールドは既定値に初期化されます。

C#

```
public class Person
{
    public int age;
    public string name;
}
```

```
class TestPerson
{
    static void Main()
    {
        Person p = new Person();

        System.Console.Write("Name: {0}, Age: {1}", p.name, p.age);
    }
}
```

出力

```
Name: , Age: 0
```

`age` の既定値は `0`、`name` の既定値は **null** です。既定値の詳細については、「既定値の一覧表 (C# リファレンス)」を参照してください。

例 3

次の例は、基本クラスの初期化子の使い方を示しています。`Circle` クラスは一般的なクラス `Shape` から派生しており、`Cylinder` クラスは `Circle` クラスから派生しています。どちらの派生クラスのコンストラクタも、基本クラスの初期化子を使用しています。

C#

```
abstract class Shape
{
    public const double pi = System.Math.PI;
    protected double x, y;

    public Shape(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public abstract double Area();
}

class Circle : Shape
{
    public Circle(double radius)
        : base(radius, 0)
    {
    }
    public override double Area()
    {
        return pi * x * x;
    }
}

class Cylinder : Circle
{
    public Cylinder(double radius, double height)
        : base(radius)
    {
        y = height;
    }

    public override double Area()
    {
        return (2 * base.Area()) + (2 * pi * x * y);
    }
}

class TestShapes
{
    static void Main()
    {
```

```
            double radius = 2.5;
            double height = 3.0;

            Circle ring = new Circle(radius);
            Cylinder tube = new Cylinder(radius, height);

            System.Console.WriteLine("Area of the circle = {0:F2}", ring.Area());
            System.Console.WriteLine("Area of the cylinder = {0:F2}", tube.Area());
        }
    }
```

出力

```
Area of the circle = 19.63

Area of the cylinder = 86.39
```

基本クラスのコンストラクタを呼び出すその他の例については、「virtual (C# リファレンス)」、「override (C# リファレンス)」、および「base (C# リファレンス)」を参照してください。

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
デストラクタ (C# プログラミング ガイド)
static (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

コンストラクタ (C# プログラミング ガイド)

プライベート コンストラクタ (C# プログラミング ガイド)

プライベート コンストラクタは、特別なインスタンス コンストラクタです。通常は、静的メンバだけを含むクラスで使用されます。クラスに 1 つ以上のプライベート コンストラクタがあり、パブリック コンストラクタがない場合、他のクラス (入れ子になったクラスを除く) はこのクラスのインスタンスを作成できません。以下にサンプルを示します。

C#

```
class NLog
{
    // Private Constructor:
    private NLog() { }

    public static double e = System.Math.E;  //2.71828...
}
```

空のコンストラクタを宣言すると、既定コンストラクタの自動生成は行われません。コンストラクタにアクセス修飾子を指定しない場合でも、コンストラクタは既定でプライベートになります。しかし、通常は、private 修飾子を明示的に使って、クラスをインスタンス化できないことを明確に示します。

プライベート コンストラクタは、Math クラスなどのようにインスタンス フィールドやメソッドが存在しない場合や、クラスのインスタンスを取得するためにメソッドが呼び出される場合に、クラスのインスタンスが作成されないようにするために使用します。クラス内のすべてのメソッドが静的な場合は、クラス全体を静的にすることを検討してください。詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」を参照してください。

使用例

プライベート コンストラクタを使用するクラスの例を次に示します。

C#

```
public class Counter
{
    private Counter() { }
    public static int currentCount;
    public static int IncrementCount()
    {
        return ++currentCount;
    }
}

class TestCounter
{
    static void Main()
    {
        // If you uncomment the following statement, it will generate
        // an error because the constructor is inaccessible:
        // Counter aCounter = new Counter();   // Error

        Counter.currentCount = 100;
        Counter.IncrementCount();
        System.Console.WriteLine("New count: {0}", Counter.currentCount);
    }
}
```

出力

```
New count: 101
```

この例で次のステートメントのコメントを解除すると、保護レベルが原因でコンストラクタにアクセスできなくなり、エラーが発生します。

C#

```
// Counter aCounter = new Counter();   // Error
```

C# 言語仕様

詳細については、「C# 言語仕様」の以下のセクションを参照してください。

- 10.10.5 プライベート コンストラクタ
- 25.2 静的クラス

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
デストラクタ (C# プログラミング ガイド)
private (C# リファレンス)
public (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

コンストラクタ (C# プログラミング ガイド)

静的コンストラクタ (C# プログラミング ガイド)

静的コンストラクタは、静的データを初期化したり、1 回限りの特定の処理を実行したりする際に使用されます。静的コンストラクタは、最初のインスタンスを作成する前、または静的メンバが参照される前に自動的に呼び出されます。

C#

```
class SimpleClass
{
    // Static constructor
    static SimpleClass()
    {
        //...
    }
}
```

静的コンストラクタには、次のような特徴があります。

- 静的コンストラクタはアクセス修飾子をとらず、パラメータはありません。
- 最初のインスタンスが作成される前、または静的メンバが参照される前に、静的コンストラクタが自動的に呼び出されてクラスを初期化します。
- 静的コンストラクタを直接呼び出すことはできません。
- プログラム内で静的コンストラクタが実行されるタイミングを制御することはできません。
- 静的コンストラクタは、通常、クラスがログ ファイルを使用しているときに使われ、このファイルにエントリを書き込みます。
- 静的コンストラクタは、アンマネージ コードのラッパー クラスを作成するときにも役立ちます。その際、静的コンストラクタは、**LoadLibrary** メソッドを呼び出すことができます。

使用例

次の例の `Bus` クラスには、静的コンストラクタと 1 つの静的メンバ `Drive()` があります。`Drive()` が呼び出されると、静的コンストラクタが呼び出されてクラスを初期化します。

C#

```
public class Bus
{
    // Static constructor:
    static Bus()
    {
        System.Console.WriteLine("The static constructor invoked.");
    }

    public static void Drive()
    {
        System.Console.WriteLine("The Drive method invoked.");
    }
}

class TestBus
{
    static void Main()
    {
        Bus.Drive();
    }
}
```

出力

```
The static constructor invoked.
```

```
The Drive method invoked.
```

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

デストラクタ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

コンストラクタ (C# プログラミング ガイド)

方法：コピー コンストラクタを記述する (C# プログラミング ガイド)

一部の言語とは異なり、C# ではコピー コンストラクタが用意されていません。新しいオブジェクトを作成し、既存のオブジェクトから値をコピーする場合は、自分で適切なメソッドを記述する必要があります。

使用例

この例では、**Person** クラスは、**Person** 型の別のオブジェクトを引数とするコンストラクタを含みます。このオブジェクトのフィールドの内容は、新しいオブジェクトのフィールドに割り当てられます。

C#

```
class Person
{
    private string name;
    private int age;

    // Copy constructor.
    public Person(Person previousPerson)
    {
        name = previousPerson.name;
        age = previousPerson.age;
    }

    // Instance constructor.
    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }

    // Get accessor.
    public string Details
    {
        get
        {
            return name + " is " + age.ToString();
        }
    }
}

class TestPerson
{
    static void Main()
    {
        // Create a new person object.
        Person person1 = new Person("George", 40);

        // Create another new object, copying person1.
        Person person2 = new Person(person1);
        System.Console.WriteLine(person2.Details);
    }
}
```

出力

```
George is 40
```

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
デストラクタ (C# プログラミング ガイド)
ICloneable

概念

C# プログラミング ガイド

その他の技術情報

コンストラクタ (C# プログラミング ガイド)

デストラクタ (C# プログラミング ガイド)

デストラクタは、クラスのインスタンスを消滅させるために使用します。

解説

- デストラクタは、構造体には定義できません。クラスでだけ使用します。
- クラスで使用できるデストラクタは 1 つだけです。
- デストラクタを継承またはオーバーロードすることはできません。
- デストラクタを呼び出すことはできません。デストラクタは自動的に起動されます。
- デストラクタは修飾子をとらず、パラメータはありません。

次に示すのは、`Car` クラスに対するデストラクタの宣言の例です。

C#

```
class Car
{
    ~ Car()  // destructor
    {
        // cleanup statements...
    }
}
```

デストラクタは、オブジェクトの基本クラスで Finalize を暗黙的に呼び出します。したがって、前記のデストラクタのコードは、暗黙的に次のように解釈されます。

```
protected override void Finalize()
{
    try
    {
        // cleanup statements...
    }
    finally
    {
        base.Finalize();
    }
}
```

つまり、最派生クラスから最低派生クラスまで、継承チェーンのすべてのインスタンスに対して、`Finalize` メソッドが再帰的に呼び出されます。

メモ :

空のデストラクタは使用しないでください。デストラクタがクラスに存在するときは、エントリが `Finalize` キューで作成されます。デストラクタを呼び出すと、ガベージ コレクタが呼び出され、このキューを処理します。デストラクタが空の場合は、これによってパフォーマンスが余計に低下します。

デストラクタがいつ呼び出されるかはガベージ コレクタによって決定されるため、プログラマは制御できません。ガベージ コレクタは、アプリケーションが使用していないオブジェクトをチェックします。消滅できるオブジェクトと考えられる場合、デストラクタ (存在する場合) を呼び出し、オブジェクトの格納に使用されているメモリをクリアします。デストラクタは、プログラムの終了時にも呼び出されます。

Collect を呼び出すことによって、ガベージ コレクションを強制的に行うことができます。ただし、パフォーマンスに問題が発生する可能性があるため、通常はこの処理を避けます。詳細については、「ガベージ コレクションの強制実行」を参照してください。

デストラクタを使ったリソースの解放

一般に C# では、ガベージ コレクションを使用しない言語で開発する場合ほど、メモリ管理を必要としません。.NET Framework のガベージ コレクタが、オブジェクトに対するメモリの割り当てと解放を暗黙的に管理します。ただし、ウィンドウ、ファイル、ネットワーク接続などのアンマネージ リソースをアプリケーションでカプセル化するときは、デストラクタを使ってこれらのリソースを解放する必要があります。オブジェクトを消滅させることができる場合、ガベージ コレクタはそのオブジェクトの `Finalize` メソッドを実行します。

リソースの明示的な解放

アプリケーションで貴重な外部リソースを使用している場合は、ガベージ コレクタがオブジェクトを解放する前にリソースを明示的に解放する手段を用意することをお勧めします。この処理を行うには、オブジェクトに対して必要なクリーンアップを実行する `Dispose` メソッドを IDisposable インターフェイスから実装します。これによって、アプリケーションのパフォーマンスを大幅に向上させることができます。このようにリソースを明示的に制御する場合でも、デストラクタは、`Dispose` メソッドの呼び出しが失敗したときにリソースをクリーンアップするための安全装置になります。

リソースのクリーンアップの詳細については、次のトピックを参照してください。

- アンマネージ リソースのクリーンアップ
- Dispose メソッドの実装
- using ステートメント (C# リファレンス)

使用例

次の例では、継承のチェインを形成する 3 つのクラスを作成します。`First` が基本クラスであり、`Second` は `First` から派生し、`Third` は `Second` から派生しています。3 つのクラスのいずれにもデストラクタがあります。`Main()` では、最派生クラスのインスタンスが作成されます。プログラムを実行すると、3 つのクラスのデストラクタが、最派生クラスから最低派生クラスの順に自動的に呼び出されます。

C#

```
class First
{
    ~First()
    {
        System.Console.WriteLine("First's destructor is called");
    }
}

class Second: First
{
    ~Second()
    {
        System.Console.WriteLine("Second's destructor is called");
    }
}

class Third: Second
{
    ~Third()
    {
        System.Console.WriteLine("Third's destructor is called");
    }
}

class TestDestructors
{
    static void Main()
    {
        Third t = new Third();
    }
}
```

出力

```
Third's destructor is called
Second's destructor is called
First's destructor is called
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.6 デストラクタ
- 10.2.7.4 デストラクタ用に予約されているメンバ名

- 10.12 デストラクタ (クラス)
- 11.3.9 デストラクタ (構造体)

参照
関連項目
IDisposable
概念
C# プログラミング ガイド
その他の技術情報
コンストラクタ (C# プログラミング ガイド)
ガベージ コレクション

フィールド (C# プログラミング ガイド)

このトピックでは、フィールドについて説明します。フィールドとは、クラスまたは構造体に含まれるオブジェクトや値のことです。フィールドを使用すると、クラスおよび構造体でデータをカプセル化できます。

簡略にするために、以下の例では **public** のフィールドを使用しますが、この方法を実際に使用することはお勧めしません。フィールドは通常、**private** にしてください。外部クラスによるフィールドへのアクセスは、メソッド、プロパティ、またはインデクサを使用した間接的なアクセスにする必要があります。詳細については、「メソッド (C# プログラミング ガイド)」、「プロパティ (C# プログラミング ガイド)」、および「インデクサ (C# プログラミング ガイド)」を参照してください。

フィールド

フィールドは、クラスのデザインを実現する上で必要なデータを格納します。たとえば、暦の日付を表すクラスには、月、日、年を表す 3 つの整数フィールドが存在します。フィールドは、フィールドのアクセス レベル、フィールドの型、フィールドの名前を順に指定して、クラス ブロック内で宣言します。次に例を示します。

C#

```
public class CalendarDate
{
    public int month;
    public int day;
    public int year;
}
```

オブジェクト内のフィールドにアクセスするには、`objectname.fieldname` のように、オブジェクト名の後にピリオドを追加し、その後にフィールド名を続けます。次に例を示します。

C#

```
CalendarDate birthday = new CalendarDate();
birthday.month = 7;
```

フィールドには、フィールドの宣言時に代入演算子を使用して初期値を設定できます。たとえば、`month` フィールドに自動的に `7` を代入するには、次のように `month` を宣言します。

C#

```
public class CalendarDateWithInitialization
{
    public int month = 7;
    //...
}
```

フィールドは、オブジェクト インスタンスのコンストラクタが呼び出される直前に初期化されるので、コンストラクタがフィールドの値を代入すると、フィールドの宣言中に指定された値はすべて上書きされます。詳細については、「コンストラクタの使用 (C# プログラミング ガイド)」を参照してください。

メモ :
フィールド初期化子は、他のインスタンス フィールドを参照できません。

フィールドは、public、private、protected、internal、または **protected internal** とマークできます。これらのアクセス修飾子により、クラスのユーザーがフィールドにどのようにアクセスできるかが定義されます。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

フィールドは、static と宣言することもできます。このように宣言すると、クラスのインスタンスが存在しない場合でも、呼び出し元がいつでもフィールドにアクセスできます。詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」を参照してください。

フィールドは、readonly として宣言できます。読み取り専用フィールドには、初期化時またはコンストラクタでしか値を代入できません。**static readonly** フィールドは基本的に定数と同じですが、C# コンパイラは、このフィールドの値にはコンパイル時にアクセスできず、実行時にしかアク

セスできません。詳細については、「定数 (C# プログラミング ガイド)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.4 フィールド
- 10.4 フィールド

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
コンストラクタの使用 (C# プログラミング ガイド)
継承 (C# プログラミング ガイド)
アクセス修飾子 (C# プログラミング ガイド)
抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

定数 (C# プログラミング ガイド)

クラスと構造体では、定数をメンバとして宣言できます。定数とは、コンパイル時に既知であり、変更されない値です (実行時に初期化される定数値を作成するには、readonly キーワードを使用します)。定数は、フィールドの型の前に const キーワードを使用して、フィールドとして宣言します。定数は、宣言するときに初期化する必要があります。次に例を示します。

C#

```
class Calendar1
{
    public const int months = 12;
}
```

この例では、定数 `months` は常に 12 になり、クラス自体によっても変更できません。定数は、整数型 (**sbyte**、**byte**、**short**、**ushort**、**int**、**uint**、**long**、**ulong**、**char**、**float**、**double**、**decimal**、**bool**、**string**)、列挙型、または **null** への参照である必要があります。

同じ型の複数の定数を、次のように同時に宣言できます。

C#

```
class Calendar2
{
    const int months = 12, weeks = 52, days = 365;
}
```

定数の初期化に使用する式は、循環参照を形成しない限り別の定数を参照できます。次に例を示します。

C#

```
class Calendar3
{
    const int months = 12;
    const int weeks = 52;
    const int days = 365;

    const double daysPerWeek = days / weeks;
    const double daysPerMonth = days / months;
}
```

定数は、public、private、protected、internal、または **protected internal** とマークできます。これらのアクセス修飾子によって、クラスのユーザーが定数にアクセスする方法が定義されます。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

定数は、静的フィールドのようにアクセスされますが、定数で **static** キーワードを使用することはできません。定数を定義しているクラスに含まれていない式で定数を使用する場合は、クラス名、ピリオド、および定数の名前を使用する必要があります。次に例を示します。

C#

```
int birthstones = Calendar.months;
```

C# 言語仕様

詳細については、「C# 言語仕様」の以下のセクションを参照してください。

- 10.3 定数

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
プロパティ (C# プログラミング ガイド)
データ型 (C# プログラミング ガイド)

入れ子にされた型 (C# プログラミング ガイド)

クラスや構造体の中で定義された型は、入れ子にされた型と呼ばれます。次に例を示します。

C#

```
class Container
{
    class Nested
    {
        Nested() { }
    }
}
```

外側の型がクラスまたは構造体のいずれであっても、入れ子にされた型は既定で private になりますが、public、protected internal、protected、internal、または private にすることもできます。上の例の `Nested` は、外部の型からアクセスできませんが、次のように public にもできます。

C#

```
class Container
{
    public class Nested
    {
        Nested() { }
    }
}
```

入れ子にされた型 (内側の型) は、包含する型 (外側の型) にアクセスできます。包含する型にアクセスするには、その型を入れ子にされた型にコンストラクタとして渡します。次に例を示します。

C#

```
public class Container
{
    public class Nested
    {
        private Container m_parent;

        public Nested()
        {
        }
        public Nested(Container parent)
        {
            m_parent = parent;
        }
    }
}
```

入れ子にされた型は、継承されたプライベート メンバとプロテクト メンバを含む、包含する型のプライベート メンバとプロテクト メンバにアクセスできます。

上記の宣言では、クラス `Nested` の完全名は `Container.Nested` です。これは、次のように入れ子になったクラスの新しいインスタンスを作成するときに使用される名前です。

C#

```
Container.Nested nest = new Container.Nested();
```

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
アクセス修飾子 (C# プログラミング ガイド)
概念
C# プログラミング ガイド
その他の技術情報
コンストラクタ (C# プログラミング ガイド)

アクセス修飾子 (C# プログラミング ガイド)

クラスや構造体は、それらを宣言したプログラムや名前空間だけが使用できるように制限できます。またクラス メンバも、派生クラスだけが使用できるように制限したり、現在の名前空間またはプログラム内のクラスだけが使用できるように制限したりできます。アクセス修飾子は、クラス、構造体、またはメンバの宣言に追加して、それぞれへのアクセス制限を指定するキーワードです。これらのキーワードは、public、private、protected、および internal です。アクセス修飾子の例を次に示します。

C#

```
public class Bicycle
{
    public void Pedal() { }
}
```

クラスと構造体のアクセシビリティ

他のクラスや構造体に入れ子にされていないクラスや構造体には、public または internal を指定できます。public と宣言された型には、他のすべての型がアクセスできます。internal と宣言された型には、同じアセンブリに所属する型だけがアクセスできます。クラスや構造体は、上の例のように **public** キーワードがクラス定義に追加されている場合を除き、既定で internal と宣言されます。クラス定義や構造体定義に **internal** キーワードを追加すると、それぞれのアクセス レベルを明示的に指定できます。アクセス修飾子は、クラスや構造体自体には影響しません。クラスや構造体は常にそれ自体にアクセスでき、それ自体のすべてのメンバにもアクセスできます。

クラス メンバと構造体メンバのアクセシビリティ

クラス メンバや構造体メンバは、5 種類のうちのいずれかのアクセス修飾子を使って宣言できます。これらのメンバは、クラス自体や構造体自体と同じように public にも internal にもできます。クラス メンバは、**protected** キーワードを使用して、protected として宣言できます。この場合、同じクラスを基本クラスとして使用している派生型だけがこのメンバにアクセスできます。**protected** と **internal** の 2 つのキーワードを合わせて使用すると、クラス メンバを protected internal に指定できます。この場合、派生型または同じアセンブリ内の型だけがこのメンバにアクセスできます。最後に、クラス メンバや構造体メンバは、**private** キーワードを使用して private と宣言できます。この場合は、このメンバを宣言したクラスや構造体だけにアクセスが許可されます。

クラス メンバや構造体メンバのアクセス レベルを設定するには、適切なキーワードをメンバ宣言に追加します。次にいくつかの例を示します。

C#

```
// public class:
public class Tricycle
{
    // protected method:
    protected void Pedal() { }

    // private field:
    private int m_wheels = 3;

    // protected internal property:
    protected internal int Wheels
    {
        get { return m_wheels; }
    }
}
```

その他の型

クラスと同様に、インターフェイスも public または internal として宣言できます。ただし、クラスと違って、インターフェイスには、既定で internal が設定されます。インターフェイス メンバは常に public になり、アクセス修飾子を適用できません。

名前空間と列挙型メンバは常に public になり、アクセス修飾子を適用できません。

デリゲートには、既定で internal が指定されます。

名前空間の内部やコンパイル単位の最上位レベル (名前空間、クラス、または構造体などの外側) で宣言された型はいずれも既定では internal になりますが、public にもできます。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 10.2.3 アクセス修飾子

参照

関連項目

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)
インターフェイス (C# プログラミング ガイド)
private (C# リファレンス)
public (C# リファレンス)
internal (C# リファレンス)
protected (C# リファレンス)
class (C# リファレンス)
struct (C# リファレンス)
インターフェイス (C# リファレンス)

概念

C# プログラミング ガイド

部分クラス定義 (C# プログラミング ガイド)

クラスや構造体またはインターフェイスの定義は、複数のソース ファイルに分割できます。各ソース ファイルには、クラス定義のセクションが含まれ、分割されたすべての部分はアプリケーションのコンパイル時に結合されます。クラス定義を分割するのが望ましいのは、次のような場合です。

- 大型プロジェクトを開発する際に、クラスを別個のファイルに分割すると、複数のプログラマが同時にそのクラスの作業を行うことができます。
- 自動生成ソースを使用する際に、ソース ファイルを再作成せずにコードをクラスに追加できます。Visual Studio では、Windows フォームや Web サービス ラッパー コードを作成するときにこのアプローチを使用します。Visual Studio によって作成されたファイルを編集せずに、これらのクラスを使用するコードを作成できます。
- クラス定義を分割するには、次のように partial キーワード修飾子を使用します。

C#

```
public partial class Employee
{
    public void DoWork()
    {
    }
}

public partial class Employee
{
    public void GoToLunch()
    {
    }
}
```

解説

partial キーワードを使用すると、クラス、構造体、またはインターフェイスの他の部分を名前空間内で定義できるようになります。**partial** キーワードは、すべての部分で使用する必要があります。最終的な型を形成するためには、コンパイル時にすべての部分を使用できる必要があります。また、すべての部分で同じアクセシビリティ (パブリックやプライベートなど) を使用する必要があります。

抽象と宣言された部分がある場合、型全体が抽象と見なされます。シールと宣言された部分がある場合は、型全体がシールと見なされます。また、基本型を宣言する部分がある場合は、型全体が該当するクラスを継承します。

基本クラスを指定する部分はすべて一致する必要がありますが、基本クラスを省略する部分も基本型を継承します。部分は別の基本インターフェイスを指定でき、すべての部分宣言で示されたすべてのインターフェイスが最終的な型によって実装されます。部分定義で宣言されたクラス、構造体、インターフェイスの各メンバは、他のすべての部分で利用できます。最終的な型は、コンパイル時にすべての部分を結合して形成されます。

メモ：
部分識別子は、デリゲートや列挙宣言では使用できません。

入れ子にされた型は、それを包含する型自体が partial でない場合でも、partial にできます。この例を次に示します。

C#

```
class Container
{
    partial class Nested
    {
        void Test() { }
    }
    partial class Nested
    {
        void Test2() { }
    }
}
```

- 部分型定義の属性は、コンパイル時に結合されます。たとえば、次のような宣言があるとします。

C#

```
[System.SerializableAttribute]
partial class Moon { }

[System.ObsoleteAttribute]
partial class Moon { }
```

この宣言は、次の宣言と等価です。

C#

```
[System.SerializableAttribute]
[System.ObsoleteAttribute]
class Moon { }
```

- 部分型定義に含まれる次の要素はすべて結合されます。
- XML コメント
- インターフェイス
- ジェネリック型パラメータ属性
- クラス属性
- メンバ

たとえば、次のような宣言があるとします。

C#

```
partial class Earth : Planet, IRotate { }
partial class Earth : IRevolve { }
```

この宣言は、次の宣言と等価です。

C#

```
class Earth : Planet, IRotate, IRevolve { }
```

制限事項

部分クラス定義の使用時には、以下の規則が適用されます。

- 同じ型の部分である部分型定義はすべて **partial** で修飾する必要があります。たとえば、次のクラス宣言はエラーになります。

  C#

  ```
  public partial class A { }
  //public class A { }  // Error, must also be marked partial
  ```

- **partial** 修飾子は、**class**、**struct**、または **interface** キーワードの直前にのみ配置できます。
- 入れ子にされた部分型は、次のように部分型定義で宣言できます。

  C#

  ```
  partial class ClassWithNestedClass
  {
  ```

```
        partial class NestedClass { }
    }

    partial class ClassWithNestedClass
    {
        partial class NestedClass { }
    }
```

- 同じ型の部分である部分型定義はすべて同じアセンブリまたは同じモジュール (.exe ファイルまたは .dll ファイル) 内で定義する必要があります。部分定義は、複数のモジュールにまたがることができません。
- クラス名とジェネリック型パラメータはすべての部分型定義で一致する必要があります。ジェネリック型は partial にできます。部分宣言では、それぞれ同じパラメータ名を同じ順序で使用する必要があります。
- 以下のキーワードは、部分型定義では省略できますが、ある 1 つの部分型定義に存在する場合は、同じ型の別の部分定義で指定されているキーワードと競合できません。
    - public
    - private
    - protected
    - internal
    - abstract
    - sealed
    - 基本クラス
    - new 修飾子 (入れ子にされた部分)
    - ジェネリック制約 (詳細については、「型パラメータの制約 (C# プログラミング ガイド)」を参照してください。)

例 1

次の例では、クラス `CoOrds` のフィールドとコンストラクタを 1 つの部分クラス定義で宣言し、メンバ `PrintCoOrds` を別の部分クラス定義で宣言しています。

C#

```
public partial class CoOrds
{
    private int x;
    private int y;

    public CoOrds(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

public partial class CoOrds
{
    public void PrintCoOrds()
    {
        System.Console.WriteLine("CoOrds: {0},{1}", x, y);
    }

}

class TestCoOrds
{
    static void Main()
    {
        CoOrds myCoOrds = new CoOrds(10, 15);
```

```
            myCoOrds.PrintCoOrds();
        }
    }
```

出力

```
CoOrds: 10,15
```

例 2

次の例は、部分構造体と部分インターフェイスも開発できることを示しています。

C#

```
    partial interface ITest
    {
        void Interface_Test();
    }

    partial interface ITest
    {
        void Interface_Test2();
    }

    partial struct S1
    {
        void Struct_Test() { }
    }

    partial struct S1
    {
        void Struct_Test2() { }
    }
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 23 部分型

参照
関連項目
インターフェイス (C# プログラミング ガイド)
partial (C# リファレンス)
概念
C# プログラミング ガイド
クラス (C# プログラミング ガイド)
構造体 (C# プログラミング ガイド)

静的クラスと静的クラス メンバ (C# プログラミング ガイド)

静的クラスと静的クラス メンバは、クラスのインスタンスを作成せずにアクセスできるデータや関数を作成する際に使用します。静的クラス メンバを使用すると、オブジェクト ID に依存しないデータと動作を分離できます。分離されたデータと関数は、オブジェクトにどのような処理が行われたかにかかわらず変更されません。静的クラスは、オブジェクト ID に依存するデータや動作がクラスに存在しないときに使用できます。

静的クラス

クラスは、静的メンバだけを含むことを示す静的クラスとして宣言できます。new キーワードを使用して静的クラスのインスタンスを作成することはできません。静的クラスは、クラスを含むプログラムや名前空間が読み込まれると、.NET Framework 共通言語ランタイム (CLR) によって自動的に読み込まれます。

特定のオブジェクトに関連付けられていないメソッドを含めるには、静的クラスを使用します。たとえば、インスタンス データで機能せず、コード内の特定のオブジェクトに関連付けられていないメソッドのセットを作成することがよく必要になります。静的クラスは、このようなメソッドを保持するために使用できます。

静的クラスの主要な特徴は次のとおりです。

- 静的メンバのみを含みます。
- インスタンス化できません。
- シールされます。
- インスタンス コンストラクタ (C# プログラミング ガイド) を含むことができません。

そのため、静的クラスを作成することと、静的メンバとプライベート コンストラクタのみを含むクラスを作成することはほぼ同じです。プライベート コンストラクタは、クラスのインスタンス化を防止します。

静的クラスを使用する利点は、インスタンス メンバが誤って追加されないことをコンパイラで確認できるという点です。コンパイラによって、このクラスのインスタンスを作成できないことが保証されます。

静的クラスはシールされるため、継承できません。静的クラスはコンストラクタを含むことができませんが、静的コンストラクタを宣言して初期値を割り当てたり、静的状態を設定したりすることはできます。詳細については、「静的コンストラクタ (C# プログラミング ガイド)」を参照してください。

静的クラスを使用する場合

たとえば、会社名と住所に関する情報を取得する次のメソッドが `CompanyInfo` クラスに含まれているとします。

C#

```
class CompanyInfo
{
    public string GetCompanyName() { return "CompanyName"; }
    public string GetCompanyAddress() { return "CompanyAddress"; }
    //...
}
```

これらのメソッドは、このクラスの特定のインスタンスに割り当てる必要がありません。そのため、不必要なインスタンスを作成する代わりに、このクラスを次のように静的クラスとして宣言できます。

C#

```
static class CompanyInfo
{
    public static string GetCompanyName() { return "CompanyName"; }
    public static string GetCompanyAddress() { return "CompanyAddress"; }
    //...
}
```

静的クラスは、特定のオブジェクトに関連付けられていないメソッドの編成単位として使用します。また、静的クラスを使用すると、メソッドを呼び出すためにオブジェクトを作成する必要がないので、実装を簡素化し、迅速化できます。これは、System 名前空間に Math クラスのメソッドを編成するなど、クラス内にメソッドを適切に編成するためにも役立ちます。

静的メンバ

静的なメソッド、フィールド、プロパティ、またはイベントは、クラスのインスタンスが作成されていないときでもクラスで呼び出すことができます。クラスのインスタンスが作成されている場合は、これらを使用して静的メンバにアクセスすることはできません。静的フィールドと静的イベントのコピーは 1 つのみ存在し、静的メソッドと静的プロパティは、静的フィールドと静的イベントにしかアクセスできません。静的メンバは、多くの場合、オブジェクトの状態に応じて変化しないデータや計算を表すために使用されます。たとえば、数値演算ライブラリには、サインとコサインを計算する静的メソッドを含めることができます。

静的クラス メンバは、次のようにメンバの戻り値の型の前に **static** キーワードを配置して宣言します。

C#

```
public class Automobile
{
    public static int NumberOfWheels = 4;
    public static int SizeOfGasTank
    {
        get
        {
            return 15;
        }
    }
    public static void Drive() { }
    public static event EventType RunOutOfGas;

    //other non-static fields and properties...
}
```

静的メンバは初めてアクセスされる前に初期化されます。また、静的コンストラクタが呼び出された場合は、静的コンストラクタが実行される前に初期化されます。静的クラス メンバにアクセスするには、変数名の代わりにクラスの名前を使用して、そのメンバの位置を指定します。以下にサンプルを示します。

C#

```
Automobile.Drive();
int i = Automobile.NumberOfWheels;
```

使用例

次に、摂氏と華氏の間で温度を変換する 2 つのメソッドを含む静的クラスのコード例を示します。

C#

```
public static class TemperatureConverter
{
    public static double CelsiusToFahrenheit(string temperatureCelsius)
    {
        // Convert argument to double for calculations.
        double celsius = System.Double.Parse(temperatureCelsius);

        // Convert Celsius to Fahrenheit.
        double fahrenheit = (celsius * 9 / 5) + 32;

        return fahrenheit;
    }

    public static double FahrenheitToCelsius(string temperatureFahrenheit)
    {
        // Convert argument to double for calculations.
        double fahrenheit = System.Double.Parse(temperatureFahrenheit);

        // Convert Fahrenheit to Celsius.
        double celsius = (fahrenheit - 32) * 5 / 9;

        return celsius;
    }
}
```

```
class TestTemperatureConverter
{
    static void Main()
    {
        System.Console.WriteLine("Please select the convertor direction");
        System.Console.WriteLine("1. From Celsius to Fahrenheit.");
        System.Console.WriteLine("2. From Fahrenheit to Celsius.");
        System.Console.Write(":");

        string selection = System.Console.ReadLine();
        double F, C = 0;

        switch (selection)
        {
            case "1":
                System.Console.Write("Please enter the Celsius temperature: ");
                F = TemperatureConverter.CelsiusToFahrenheit(System.Console.ReadLine());
                System.Console.WriteLine("Temperature in Fahrenheit: {0:F2}", F);
                break;

            case "2":
                System.Console.Write("Please enter the Fahrenheit temperature: ");
                C = TemperatureConverter.FahrenheitToCelsius(System.Console.ReadLine());
                System.Console.WriteLine("Temperature in Celsius: {0:F2}", C);
                break;

            default:
                System.Console.WriteLine("Please select a convertor.");
                break;
        }
    }
}
```

入力

```
2

98.6
```

出力例

```
Please select the convertor

1. From Celsius to Fahrenheit.

2. From Fahrenheit to Celsius.

:2

Please enter the Fahrenheit temperature: 98.6

Temperature in Celsius: 37.00
```

他の出力例は次のようになります。

```
Please select the convertor

1. From Celsius to Fahrenheit.

2. From Fahrenheit to Celsius.

:1

Please enter the Celsius temperature: 37.00

Temperature in Fahrenheit: 98.60
```

C# 言語仕様

詳細については、「C# 言語仕様」の以下のセクションを参照してください。

- 25.2 静的クラス

参照

関連項目

class (C# リファレンス)
インスタンス コンストラクタ (C# プログラミング ガイド)
概念
C# プログラミング ガイド
静的クラスのデザイン
クラス (C# プログラミング ガイド)

方法：メソッドに構造体を渡すこととクラス参照を渡すことの違いを理解する (C# プログラミング ガイド)

構造体がメソッドに渡されるときは構造体のコピーが渡されますが、クラス インスタンスが渡されるときは参照が渡されます。次にその例を示します。

次の例の出力は、クラス インスタンスを `ClassTaker` メソッドに渡すと、クラス フィールドの値だけが変更されることを示しています。構造体フィールドは、そのインスタンスを `StructTaker` メソッドに渡しても変更されません。これは、クラス参照が `ClassTaker` メソッドに渡されるのに対し、構造体のコピーが `StructTaker` メソッドに渡されるからです。

使用例

C#

```
class TheClass
{
    public string willIChange;
}

struct TheStruct
{
    public string willIChange;
}

class TestClassAndStruct
{
    static void ClassTaker(TheClass c)
    {
        c.willIChange = "Changed";
    }

    static void StructTaker(TheStruct s)
    {
        s.willIChange = "Changed";
    }

    static void Main()
    {
        TheClass testClass = new TheClass();
        TheStruct testStruct = new TheStruct();

        testClass.willIChange = "Not Changed";
        testStruct.willIChange = "Not Changed";

        ClassTaker(testClass);
        StructTaker(testStruct);

        System.Console.WriteLine("Class field = {0}", testClass.willIChange);
        System.Console.WriteLine("Struct field = {0}", testStruct.willIChange);
    }
}
```

出力

```
Class field = Changed
Struct field = Not Changed
```

参照

関連項目

パラメータの引き渡し (C# プログラミング ガイド)

概念

C# プログラミング ガイド

クラス (C# プログラミング ガイド)

構造体 (C# プログラミング ガイド)

プロパティ (C# プログラミング ガイド)

プロパティは、プライベート フィールドの値の読み取り、書き込み、または計算を行う、柔軟な機構が用意されたメンバです。プロパティは、パブリック データのメンバと同様に使用できますが、実際はアクセサという特殊なメソッドが使用されます。メソッドの安全性と柔軟性を維持しながら、簡単にデータへアクセスできます。

この例では、`TimePeriod` クラスに期間が格納されています。内部的に、クラスに秒単位で期間が格納されていますが、クライアントが時間単位で期間を指定できる `Hours` というプロパティも用意されています。`Hours` プロパティのアクセサでは、時間と秒の変換が実行されます。

使用例

C#

```
class TimePeriod
{
    private double seconds;

    public double Hours
    {
        get { return seconds / 3600; }
        set { seconds = value * 3600; }
    }
}

class Program
{
    static void Main()
    {
        TimePeriod t = new TimePeriod();

        // Assigning the Hours property causes the 'set' accessor to be called.
        t.Hours = 24;

        // Evaluating the Hours property causes the 'get' accessor to be called.
        System.Console.WriteLine("Time in hours: " + t.Hours);
    }
}
```

出力

```
Time in hours: 24
```

プロパティの概要

- プロパティによって、クラスの実装や検査のコードを隠ぺいしたままで、値の取得と設定を行うことができるようになります。
- get プロパティ アクセサはプロパティ値を返すときに使用され、set アクセサは新しい値を割り当てるときに使用されます。2 つのアクセサには異なるアクセス レベルを指定できます。詳細については、「アクセサのアクセシビリティ」を参照してください。
- value キーワードは、**set** インデクサで割り当てられている値を定義するときに使用されます。
- **set** メソッドが実装されないプロパティは、読み取り専用です。

関連項目

- プロパティの使用 (C# プログラミング ガイド)
- インターフェイスのプロパティ (C# プログラミング ガイド)
- プロパティとインデクサの比較 (C# プログラミング ガイド)
- 非対称アクセサのアクセシビリティ (C# プログラミング ガイド)
- プロパティのサンプル

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.2 プロパティ
- 10.2.7.1 プロパティ用に予約されているメンバ名
- 10.6 プロパティ

参照
関連項目
プロパティの使用 (C# プログラミング ガイド)
インデクサ (C# プログラミング ガイド)
概念
C# プログラミング ガイド
プロパティのデザイン

プロパティの使用 (C# プログラミング ガイド)

プロパティは、フィールドとメソッドの両方の側面を兼ね備えています。オブジェクトを使用する側から見ると、プロパティはフィールドのように見えます。プロパティへのアクセス方法はフィールドとまったく同じです。クラスを実装する側から見ると、プロパティは、get アクセサと set アクセサのいずれか、または両方を表すコード ブロックです。**get** アクセサのコード ブロックは、プロパティが読み込まれたときに実行されます。**set** アクセサのコード ブロックは、プロパティに新しい値が割り当てられたときに実行されます。**set** アクセサなしのプロパティは、読み取り専用と見なされます。**get** アクセサなしのプロパティは、書き込み専用と見なされます。両方のアクセサを備えるプロパティは、読み書きが可能です。

フィールドとは異なり、プロパティは変数には分類されません。したがって、ref (C# リファレンス) パラメータまたは out (C# リファレンス) パラメータとしてプロパティを渡すことはできません。

プロパティの用途はさまざまです。たとえば、変更を許可する前にデータを検証できます。データベースなど、他のソースからクラス上のデータが取得されている場合、そのデータを透過的に公開できます。データが変更されたときに、イベントを発行したり他のフィールド値を変更したりするなど、アクションを実行できます。

クラス ブロック内でプロパティを宣言するには、フィールドのアクセス レベル、プロパティの種類、プロパティの名前の順に宣言し、その後に **get** アクセサと **set** アクセサのいずれかまたは両方を宣言します。たとえば、次のようにします。

C#

```
public class Date
{
    private int month = 7;  //"backing store"

    public int Month
    {
        get
        {
            return month;
        }
        set
        {
            if ((value > 0) && (value < 13))
            {
                month = value;
            }
        }
    }
}
```

この例では、`Month` がプロパティとして宣言されています。これは、**set** アクセサで `Month` の値が 1 ～ 12 の間に設定されるようにするためです。`Month` プロパティでは、プライベート フィールドを使用して実際の値を追跡します。プロパティ データの実際の位置は、プロパティの "バッキング ストア" と呼ばれます。プロパティでは、プライベート フィールドをバッキング ストアとして使用することは一般的です。プロパティの呼び出しによってのみフィールドを変更できるように、フィールドはプライベートとマークされます。パブリック アクセスとプライベート アクセスの制約の詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

get アクセサ

get アクセサの本体は、メソッドの本体と似ています。アクセサは、プロパティの型の値を返す必要があります。**get** アクセサを実行することは、フィールドの値を読み取ることに相当します。たとえば、**get** アクセサからプライベート値が戻ったときに最適化が有効な場合、**get** アクセサ メソッドの呼び出しは、コンパイルでインライン展開されます。そのため、メソッド呼び出しのオーバーヘッドはありません。ただし、仮想 **get** アクセサ メソッドは、インライン展開できません。これは、コンパイル時点では、実行時に実際に呼び出されたメソッドをコンパイラで判断できないためです。次に示すのは、プライベート フィールド `name` の値を返す **get** アクセサです。

C#

```
class Person
{
    private string name;  // the name field
    public string Name    // the Name property
    {
        get
        {
            return name;
        }
```

```
    }
}
```

代入の対象になる場合を除き、プロパティを参照すると、プロパティの値を読み取るために **get** アクセサが呼び出されます。次に例を示します。

C#

```
Person p1 = new Person();
//...

System.Console.Write(p1.Name);  // the get accessor is invoked here
```

get アクセサは return ステートメントまたは throw ステートメントで終了する必要があり、さらに、制御がアクセサの本体から離れないようにします。

get アクセサを使ってオブジェクトの状態を変更するのは、不適切なプログラミング スタイルです。たとえば、次のアクセサには、`number` フィールドにアクセスするたびにオブジェクトの状態が変化するという副作用があります。

C#

```
private int number;
public int Number
{
    get
    {
        return number++;   // Don't do this
    }
}
```

get アクセサを使えば、フィールドの値を返したり、フィールドの値を計算して返したりできます。次に例を示します。

C#

```
class Employee
{
    private string name;
    public string Name
    {
        get
        {
            return name != null ? name : "NA";
        }
    }
}
```

前のコード例では、`Name` プロパティに値を代入しないと、NA という値を返します。

set アクセサ

set アクセサは、戻り値が void のメソッドと似ています。プロパティの型の *value* という名前の暗黙のパラメータを使用します。次の例では、**set** アクセサを `Name` プロパティに追加しています。

C#

```
class Person
{
    private string name;  // the name field
    public string Name    // the Name property
    {
        get
        {
            return name;
        }
        set
```

```
        {
            name = value;
        }
    }
}
```

プロパティに値を代入すると、新しい値を渡す引数を指定して **set** アクセサが呼び出されます。次に例を示します。

C#

```
Person p1 = new Person();
p1.Name = "Joe";  // the set accessor is invoked here

System.Console.Write(p1.Name);  // the get accessor is invoked here
```

set アクセサでローカル変数の宣言に暗黙のパラメータ名 (*value*) を使用するとエラーになります。

解説

プロパティは、**public**、**private**、**protected**、**internal**、または **protected internal** とマークできます。これらのアクセス修飾子により、クラスのユーザーがプロパティにどのようにアクセスできるかが定義されます。同じプロパティでも、**get** アクセサと **set** アクセサとでアクセス修飾子が異なることがあります。たとえば、**get** は、その型の外部からは読み取り専用アクセスのみを許可する **public** で、**set** は **private** または **protected** のことがあります。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

プロパティは、**static** キーワードを使用して静的プロパティと宣言することもできます。このように宣言すると、クラスのインスタンスが存在しない場合でも、呼び出し元がいつでもプロパティにアクセスできます。詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」を参照してください。

プロパティは、virtual キーワードを使用して仮想プロパティとマークすることもできます。この場合、派生クラスでは、override キーワードを使用して、プロパティの動作をオーバーライドできます。これらのオプションの詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

仮想プロパティをオーバーライドするプロパティは、sealed にすることもできます。この場合、派生クラスでは、プロパティが仮想でなくなります。最後に、プロパティは abstract と宣言できます。この場合、クラスには実装が存在しないので、派生クラスで独自の実装を記述する必要があります。これらのオプションの詳細については、「抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)」を参照してください。

メモ :

静的プロパティのアクセサで virtual (C# リファレンス)、abstract (C# リファレンス)、または override (C# リファレンス) のいずれかの修飾子を使うと、エラーになります。

例 1

次の例では、インスタンス プロパティ、静的プロパティ、および読み取り専用プロパティが使われています。キーボードから入力された従業員の名前を受け取り、`NumberOfEmployees` の値を 1 だけインクリメントし、従業員の名前と番号を表示します。

C#

```
public class Employee
{
    public static int NumberOfEmployees;
    private static int counter;
    private string name;

    // A read-write instance property:
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    // A read-only static property:
    public static int Counter
    {
        get { return counter; }
    }
```

```
    // A Constructor:
    public Employee()
    {
        // Calculate the employee's number:
        counter = ++counter + NumberOfEmployees;
    }
}

class TestEmployee
{
    static void Main()
    {
        Employee.NumberOfEmployees = 100;
        Employee e1 = new Employee();
        e1.Name = "Claude Vige";

        System.Console.WriteLine("Employee number: {0}", Employee.Counter);
        System.Console.WriteLine("Employee name: {0}", e1.Name);
    }
}
```

出力 1

```
Employee number: 101
Employee name: Claude Vige
```

例 2

基本クラスのプロパティは、派生クラスにある同じ名前の別のプロパティによって隠ぺいされています。次の例は、この基本クラスのプロパティにアクセスする方法を示しています。

C#

```
public class Employee
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}

public class Manager : Employee
{
    private string name;

    // Notice the use of the new modifier:
    public new string Name
    {
        get { return name; }
        set { name = value + ", Manager"; }
    }
}

class TestHiding
{
    static void Main()
    {
        Manager m1 = new Manager();

        // Derived class property.
        m1.Name = "John";

        // Base class property.
        ((Employee)m1).Name = "Mary";

        System.Console.WriteLine("Name in the derived class is: {0}", m1.Name);
```

```
            System.Console.WriteLine("Name in the base class is: {0}", ((Employee)m1).Name);
        }
    }
```

出力 2

```
Name in the derived class is: John, Manager
Name in the base class is: Mary
```

コードの説明

上記の例で重要な点は次のとおりです。

- 派生クラスのプロパティ `Name` は、基本クラスのプロパティ `Name` を隠ぺいしています。この場合、派生クラスのプロパティの宣言では **new** 修飾子が使われます。

  C#

  ```
  public new string Name
  ```

- 基本クラスの隠ぺいされたプロパティにアクセスするには、キャスト `(Employee)` を使用します。

  C#

  ```
  ((Employee)m1).Name = "Mary";
  ```

  メンバを隠ぺいする方法の詳細については、「new 修飾子 (C# リファレンス)」を参照してください。

例 3

次の例では、2 つのクラス `Cube` と `Square` が抽象クラス `Shape` を実装しており、その抽象プロパティ `Area` をオーバーライドしています。プロパティに対する override 修飾子の使用に注意してください。プログラムは、入力として 1 辺の長さ (side) を受け取り、正方形の面積 (square) と立方体の表面積 (cube) を計算します。また、入力として面積を受け取り、正方形の 1 辺の長さと立方体の 1 辺の長さを計算することもできます。

C#

```
abstract class Shape
{
    public abstract double Area
    {
        get;
        set;
    }
}

class Square : Shape
{
    public double side;

    public Square(double s)  //constructor
    {
        side = s;
    }

    public override double Area
    {
        get
        {
            return side * side;
        }
        set
        {
            side = System.Math.Sqrt(value);
```

```
        }
    }
}

class Cube : Shape
{
    public double side;

    public Cube(double s)
    {
        side = s;
    }

    public override double Area
    {
        get
        {
            return 6 * side * side;
        }
        set
        {
            side = System.Math.Sqrt(value / 6);
        }
    }
}

class TestShapes
{
    static void Main()
    {
        // Input the side:
        System.Console.Write("Enter the side: ");
        double side = double.Parse(System.Console.ReadLine());

        // Compute the areas:
        Square s = new Square(side);
        Cube c = new Cube(side);

        // Display the results:
        System.Console.WriteLine("Area of the square = {0:F2}", s.Area);
        System.Console.WriteLine("Area of the cube = {0:F2}", c.Area);
        System.Console.WriteLine();

        // Input the area:
        System.Console.Write("Enter the area: ");
        double area = double.Parse(System.Console.ReadLine());

        // Compute the sides:
        s.Area = area;
        c.Area = area;

        // Display the results:
        System.Console.WriteLine("Side of the square = {0:F2}", s.side);
        System.Console.WriteLine("Side of the cube = {0:F2}", c.side);
    }
}
```

入力

```
4
24
```

出力 3

```
Enter the side: 4

Area of the square = 16.00

Area of the cube = 96.00
```

```
Enter the area: 24

Side of the square = 4.90

Side of the cube = 2.00
```

参照

処理手順

プロパティのサンプル

関連項目

プロパティ (C# プログラミング ガイド)

インターフェイスのプロパティ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

インターフェイスのプロパティ (C# プログラミング ガイド)

インターフェイス (C# リファレンス) に対してプロパティを宣言できます。次に示すのは、インターフェイス インデクサのアクセサの例です。

C#

```
public interface ISampleInterface
{
    // Property declaration:
    string Name
    {
        get;
        set;
    }
}
```

インターフェイス プロパティのアクセサには、本体がありません。したがって、アクセサの目的は、プロパティが読み取り/書き込み、読み取り専用、または書き込み専用のいずれであるかを示すことです。

使用例

次の例では、インターフェイス `IEmployee` には、読み取り/書き込みプロパティ `Name` と読み取り専用プロパティ `Counter` があります。`Employee` クラスは `IEmployee` インターフェイスを実装し、この 2 つのプロパティを使用します。プログラムは、新しい従業員の名前と現在の従業員数を読み取り、従業員の名前と計算した従業員番号を表示します。

プロパティの完全修飾名を使用して、メンバが宣言されているインターフェイスを参照しています。次に例を示します。

C#

```
string IEmployee.Name
{
    get { return "Employee Name"; }
    set { }
}
```

これは、「明示的なインターフェイスの実装 (C# プログラミング ガイド)」と呼ばれます。たとえば、`Employee` クラスが 2 つのインターフェイス `ICitizen` と `IEmployee` を実装し、両方のインターフェイスに `Name` プロパティがある場合は、明示的なインターフェイス メンバの実装が必要になります。つまり、次のプロパティ宣言では `IEmployee` インターフェイスの `Name` プロパティが実装されます。

C#

```
string IEmployee.Name
{
    get { return "Employee Name"; }
    set { }
}
```

一方、次の宣言では `ICitizen` インターフェイス `Name` プロパティが実装されます。

C#

```
string ICitizen.Name
{
    get { return "Citizen Name"; }
    set { }
}
```

C#

```
interface IEmployee
{
    string Name
```

```
        {
            get;
            set;
        }

        int Counter
        {
            get;
        }
    }

    public class Employee : IEmployee
    {
        public static int numberOfEmployees;

        private string name;
        public string Name  // read-write instance property
        {
            get
            {
                return name;
            }
            set
            {
                name = value;
            }
        }

        private int counter;
        public int Counter  // read-only instance property
        {
            get
            {
                return counter;
            }
        }

        public Employee()  // constructor
        {
            counter = ++counter + numberOfEmployees;
        }
    }

    class TestEmployee
    {
        static void Main()
        {
            System.Console.Write("Enter number of employees: ");
            Employee.numberOfEmployees = int.Parse(System.Console.ReadLine());

            Employee e1 = new Employee();
            System.Console.Write("Enter the name of the new employee: ");
            e1.Name = System.Console.ReadLine();

            System.Console.WriteLine("The employee information:");
            System.Console.WriteLine("Employee number: {0}", e1.Counter);
            System.Console.WriteLine("Employee name: {0}", e1.Name);
        }
    }
```

入力

```
210
Hazem Abolrous
```

出力例

```
Enter number of employees: 210
```

```
Enter the name of the new employee: Hazem Abolrous

The employee information:

Employee number: 211

Employee name: Hazem Abolrous
```

参照

関連項目

プロパティ (C# プログラミング ガイド)
プロパティの使用 (C# プログラミング ガイド)
プロパティとインデクサの比較 (C# プログラミング ガイド)
インデクサ (C# プログラミング ガイド)
インターフェイス (C# プログラミング ガイド)

概念

C# プログラミング ガイド

非対称アクセサのアクセシビリティ (C# プログラミング ガイド)

プロパティやインデクサの get および set の部分は、アクセサと呼ばれます。既定では、これらのアクセサの参照範囲、つまりアクセス レベルは、それが属するプロパティまたはインデクサのアクセス レベルと同じになります。詳細については、「アクセシビリティ レベル」を参照してください。ただし、これらのアクセサのいずれかへのアクセスを制限すると便利な場合があります。この場合は、通常、**set** アクセサのアクセシビリティを制限し、**get** アクセサはパブリックにアクセス可能にしておきます。次に例を示します。

C#

```
public string Name
{
    get
    {
        return name;
    }
    protected set
    {
        name = value;
    }
}
```

この例では、Name というプロパティが **get** アクセサおよび **set** アクセサを定義しています。**get** アクセサは、プロパティ自体のアクセシビリティ レベル (この場合は **public**) を受け取りますが、**set** アクセサは、protected アクセス修飾子をアクセサ自体に適用することにより、明示的に制限されています。

アクセス修飾子によるアクセサの制限

プロパティやインデクサでアクセス修飾子を使用する際には、以下の条件が適用されます。

- アクセス修飾子は、インターフェイスや明示的なインターフェイス メンバ実装で使用できません。
- アクセス修飾子を使用できるのは、プロパティやインデクサが **set** と **get** の両方のアクセサを備えている場合に限られます。この場合、修飾子は、これら 2 つのアクセスのうちの一方でのみ許可されます。
- プロパティやインデクサにオーバーライド修飾子がある場合、アクセス修飾子は、オーバーライドされたアクセサのアクセサに一致する必要があります。
- アクセサのアクセシビリティ レベルは、プロパティやインデクサ自体のアクセシビリティ レベルよりも制限する必要があります。

アクセサのオーバーライド時のアクセス修飾子

プロパティやインデクサをオーバーライドした場合、オーバーライドされたアクセサは、オーバーライド側のコードにアクセスできる必要があります。また、プロパティとインデクサの両方のアクセシビリティ レベル、およびアクセサのアクセシビリティ レベルは、オーバーライドされた、対応するプロパティとインデクサ、およびアクセサに適合する必要があります。次に例を示します。

C#

```
public class Parent
{
    public virtual int TestProperty
    {
        // Notice the accessor accessibility level.
        protected set { }

        // No access modifier is used here.
        get { return 0; }
    }
}
public class Kid : Parent
{
    public override int TestProperty
    {
        // Use the same accessibility level as in the overridden accessor.
        protected set { }

        // Cannot use access modifier here.
```

```
            get { return 0; }
        }
    }
```

インターフェイスの実装

アクセサを使用してインターフェイスを実装するときは、アクセサでアクセス修飾子を使用できません。ただし、**get** などの 1 つのアクセサを使用してインターフェイスを実装する場合は、次の例に示すように、もう 1 つのアクセサでアクセス修飾子を使用できます。

C#

```
public interface ISomeInterface
{
    int TestProperty
    {
        // No access modifier allowed here
        // because this is an interface.
        get;
    }
}

public class TestClass : ISomeInterface
{
    public int TestProperty
    {
        // Cannot use access modifier here because
        // this is an interface implementation.
        get { return 10; }

        // Interface property does not have set accessor,
        // so access modifier is allowed.
        protected set { }
    }
}
```

アクセサのアクセシビリティ ドメイン

アクセサでアクセス修飾子を使用した場合は、この修飾子によって、アクセサのアクセシビリティ ドメインが決定されます。

アクセサでアクセス修飾子を使用しない場合は、プロパティやインデクサのアクセシビリティ レベルによって、アクセサのアクセシビリティ ドメインが決定されます。

使用例

次の例には、`BaseClass`、`DerivedClass`、`MainClass` の 3 つのクラスがあります。`BaseClass` には `Name` と `Id` の 2 つのプロパティがあり、これらのプロパティは派生クラスにもあります。この例は、protected や private などの制限付きのアクセス修飾子を使用したときに、`DerivedClass` の `Id` プロパティを `BaseClass` の `Id` プロパティによってどのように隠ぺいできるかを示しています。そのため、前者のプロパティに値を割り当てると、代わりに `BaseClass` クラスのプロパティが呼び出されます。アクセス修飾子を public に置き換えると、このプロパティにアクセスできるようになります。

また、次の例は、**private** や **protected** などの制限付きのアクセス修飾子を、`DerivedClass` の `Name` プロパティの **set** アクセサに指定すると、このアクセサにアクセスできなくなり、このプロパティに値を割り当てると、エラーが生成されることも示しています。

C#

```
public class BaseClass
{
    private string name = "Name-BaseClass";
    private string id = "ID-BaseClass";

    public string Name
    {
        get { return name; }
        set { }
    }

    public string Id
```

```
    {
        get { return id; }
        set { }
    }
}

public class DerivedClass : BaseClass
{
    private string name = "Name-DerivedClass";
    private string id = "ID-DerivedClass";

    new public string Name
    {
        get
        {
            return name;
        }

        // Using "protected" would make the set accessor not accessible.
        set
        {
            name = value;
        }
    }

    // Using private on the following property hides it in the Main Class.
    // Any assignment to the property will use Id in BaseClass.
    new private string Id
    {
        get
        {
            return id;
        }
        set
        {
            id = value;
        }
    }
}

class MainClass
{
    static void Main()
    {
        BaseClass b1 = new BaseClass();
        DerivedClass d1 = new DerivedClass();

        b1.Name = "Mary";
        d1.Name = "John";

        b1.Id = "Mary123";
        d1.Id = "John123";  // The BaseClass.Id property is called.

        System.Console.WriteLine("Base: {0}, {1}", b1.Name, b1.Id);
        System.Console.WriteLine("Derived: {0}, {1}", d1.Name, d1.Id);
    }
}
```

出力

```
Name and ID in the base class: Name-BaseClass, ID-BaseClass

Name and ID in the derived class: John, ID-BaseClass
```

説明

宣言 `new private string Id` を `new public string Id` に置き換えると、次の出力が得られます。

```
Name and ID in the base class: Name-BaseClass, ID-BaseClass
```

```
Name and ID in the derived class: John, John123
```

参照

関連項目

プロパティ (C# プログラミング ガイド)

インデクサ (C# プログラミング ガイド)

アクセス修飾子 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

方法：読み取り/書き込みプロパティを宣言および使用する (C# プログラミング ガイド)

プロパティは、オブジェクトのデータへの保護されていない、制御されず未確認のアクセスに伴うリスクなしにパブリック データ メンバの利便性を提供します。このような機能は、アクセサを通じて実現されます。アクセサは、基になるデータ メンバの値を割り当てたり、取得したりする特殊なメソッドです。set アクセサは、データ メンバの割り当てを可能にし、get アクセサは、データ メンバの値を取得します。

次の例は、`Name` (string 型) と `Age` (int 型) の 2 つのプロパティを持つ `Person` クラスを示しています。次のプロパティは共に **get** アクセサと **set** アクセサを備えているので、読み書き可能プロパティと見なされます。

使用例

C#

```
class Person
{
    private string m_name = "N/A";
    private int m_Age = 0;

    // Declare a Name property of type string:
    public string Name
    {
        get
        {
            return m_name;
        }
        set
        {
            m_name = value;
        }
    }

    // Declare an Age property of type int:
    public int Age
    {
        get
        {
            return m_Age;
        }

        set
        {
            m_Age = value;
        }
    }

    public override string ToString()
    {
        return "Name = " + Name + ", Age = " + Age;
    }
}

class TestPerson
{
    static void Main()
    {
        // Create a new Person object:
        Person person = new Person();

        // Print out the name and the age associated with the person:
        System.Console.WriteLine("Person details - {0}", person);

        // Set some values on the person object:
        person.Name = "Joe";
        person.Age = 99;
        System.Console.WriteLine("Person details - {0}", person);
```

```
        // Increment the Age property:
        person.Age += 1;
        System.Console.WriteLine("Person details - {0}", person);
    }
}
```

出力

```
Person details - Name = N/A, Age = 0
Person details - Name = Joe, Age = 99
Person details - Name = Joe, Age = 100
```

堅牢性の高いプログラム

上の例の `Name` プロパティと `Age` プロパティはパブリックであり、**get** アクセサと **set** アクセサの両方を含んでいます。このため、任意のオブジェクトがこれらのプロパティを読み書きできます。ただし、これらのアクセサのうち一方のみの実行が必要になる場合もあります。たとえば、**set** アクセサを省略すると、プロパティは読み取り専用になります。この例を次に示します。

C#

```
public string Name
{
    get
    {
        return m_name;
    }
}
```

また、一方のアクセサをパブリックに公開し、もう一方をプライベートまたはプロテクトにすることもできます。詳細については、「非対称アクセサのアクセシビリティ (C# プログラミング ガイド)」を参照してください。

プロパティを宣言すると、プロパティをクラスのフィールドのように使用できます。このため、プロパティ値の取得と設定の両方で、次のように自然な構文を使用できます。

C#

```
person.Name = "Joe";
person.Age = 99;
```

プロパティの **set** メソッドでは、特殊な `value` 変数を使用できます。この変数には、ユーザーが指定した値が含まれます。たとえば、次のように指定します。

C#

```
m_name = value;
```

`Person` オブジェクトの `Age` プロパティの値を増加させるには、次のような簡潔な構文を使用できます。

C#

```
person.Age += 1;
```

set メソッドと **get** メソッドがそれぞれ使用されてプロパティがモデル化されている場合、上記と同じ内容のコードは次のようになります。

```
person.SetAge(person.GetAge() + 1);
```

この例では、**ToString** メソッドが次のようにオーバーライドされています。

C#

```
public override string ToString()
{
    return "Name = " + Name + ", Age = " + Age;
}
```

このプログラムでは、**ToString** メソッドが明示的に使用されていないことに注意してください。このメソッドは、既定で **WriteLine** 呼び出しによって呼び出されます。

参照

関連項目

プロパティ (C# プログラミング ガイド)

オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

インデクサ (C# プログラミング ガイド)

インデクサを使うと、クラスまたは構造体のインスタンスに、配列と同じ方法でインデックスを付けることができます。インデクサはプロパティと似ていますが、インデクサのアクセサはパラメータを受け取る点が異なります。

次の例では、ジェネリック クラスが定義され、値の割り当てと取得を行う手段として単純な get アクセサと set アクセサのメソッドが指定されています。クラス `Program` では、このクラスのインスタンスが作成され、文字列を格納できます。

C#

```
class SampleCollection<T>
{
    private T[] arr = new T[100];
    public T this[int i]
    {
        get
        {
            return arr[i];
        }
        set
        {
            arr[i] = value;
        }
    }
}

// This class shows how client code uses the indexer
class Program
{
    static void Main(string[] args)
    {
        SampleCollection<string> stringCollection = new SampleCollection<string>();
        stringCollection[0] = "Hello, World";
        System.Console.WriteLine(stringCollection[0]);
    }
}
```

インデクサの概要

- インデクサを使用すると、配列と同じようにオブジェクトにインデックスを付けることができます。
- **get** アクセサは値を返します。**set** アクセサは値を割り当てます。
- this キーワードは、インデクサの定義に使用されます。
- value キーワードは、**set** インデクサで割り当てられている値を定義するときに使用されます。
- インデクサは、整数値でインデックスを指定する必要はありません。検索方法をどのように定義するかによって決めてください。
- インデクサはオーバーロードできません。
- インデクサには複数の仮パラメータを指定できます。たとえば、2 次元の配列にアクセスする場合などです。

関連項目

- インデクサの使用 (C# プログラミング ガイド)
- インターフェイスのインデクサ (C# プログラミング ガイド)
- プロパティとインデクサの比較 (C# プログラミング ガイド)
- 非対称アクセサのアクセシビリティ (C# プログラミング ガイド)
- インデクサのサンプル
- インデックス付きプロパティのサンプル

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.3 インデクサ
- 10.2.7.3 インデクサ用に予約されているメンバ名
- 10.8 インデクサ
- 13.2.4 インターフェイスのインデクサ

参照
関連項目
プロパティ (C# プログラミング ガイド)
概念
C# プログラミング ガイド

インデクサの使用 (C# プログラミング ガイド)

インデクサを使用すると、配列と同じ方法で、クラス、構造体、またはインターフェイスにインデックスを付けることができます。インターフェイスでインデクサを使用する方法の詳細については、「インターフェイスのインデクサ (C# プログラミング ガイド)」を参照してください。

クラスまたは構造体でインデクサを宣言するには、次の例のように、this キーワードを使用します。

```
public int this[int index]    // Indexer declaration
{
    // get and set accessors
}
```

解説

インデクサの型とパラメータの型は、少なくともインデクサ自体と同程度のアクセシビリティが必要です。アクセシビリティのレベルの詳細については、「アクセス修飾子」を参照してください。

インデクサのシグネチャは、番号と仮パラメータの型で構成されています。インデクサの型または仮パラメータの名前は含まれません。同じクラスで複数のインデクサを宣言する場合は、異なるシグネチャを指定する必要があります。

インデクサの値は変数には分類されないので、インデクサの値を ref パラメータまたは out パラメータとして渡すことはできません。

インデクサに他の言語で使用できる名前を指定するには、宣言に `name` 属性を使用します。たとえば、次のようにします。

```
[System.Runtime.CompilerServices.IndexerName("TheItem")]
public int this [int index]   // Indexer declaration
{
}
```

このインデクサの名前は `TheItem` になります。name 属性を指定しないと、`Item` が既定の名前になります。

例 1

次の例は、プライベートな配列フィールド `arr` とインデクサの宣言方法を示しています。インデクサを使うと、インスタンス `test[i]` に直接アクセスできます。インデクサを使わない場合は、配列を public メンバとして宣言し、そのメンバ `arr[i]` に直接アクセスします。

C#

```
class IndexerClass
{
    private int[] arr = new int[100];
    public int this[int index]   // Indexer declaration
    {
        get
        {
            // Check the index limits.
            if (index < 0 || index >= 100)
            {
                return 0;
            }
            else
            {
                return arr[index];
            }
        }
        set
        {
            if (!(index < 0 || index >= 100))
            {
                arr[index] = value;
            }
        }
    }
}

class MainClass
```

```
{
    static void Main()
    {
        IndexerClass test = new IndexerClass();
        // Call the indexer to initialize the elements #3 and #5.
        test[3] = 256;
        test[5] = 1024;
        for (int i = 0; i <= 10; i++)
        {
            System.Console.WriteLine("Element #{0} = {1}", i, test[i]);
        }
    }
}
```

出力

```
Element #0 = 0
Element #1 = 0
Element #2 = 0
Element #3 = 256
Element #4 = 0
Element #5 = 1024
Element #6 = 0
Element #7 = 0
Element #8 = 0
Element #9 = 0
Element #10 = 0
```

Console.Write ステートメントなどでインデクサのアクセスが評価されると、get アクセサが呼び出されることに注意してください。したがって、**get** アクセサがない場合は、コンパイル エラーが発生します。

他の値を使用したインデックス作成

C# では、インデックスの型は整数に限定されません。たとえば、文字列をインデクサに使用することが有効なこともあります。このようなインデクサは、コレクション内の文字列を検索し、適切な値を返す場合に実装されることがあります。アクセサをオーバーロードできるため、文字列と整数のバージョンは共存できます。

例 2

この例では、曜日を格納するクラスが宣言されています。**get** アクセサは、曜日の名前を示す文字列を取得すると、対応する整数を返すように宣言されています。たとえば、Sunday の場合は 0、Monday の場合は 1 などの値を返します。

C#

```
// Using a string as an indexer value
class DayCollection
{
    string[] days = { "Sun", "Mon", "Tues", "Wed", "Thurs", "Fri", "Sat" };

    // This method finds the day or returns -1
    private int GetDay(string testDay)
    {
        int i = 0;
        foreach (string day in days)
        {
            if (day == testDay)
            {
                return i;
            }
            i++;
        }
        return -1;
    }
```

```
        // The get accessor returns an integer for a given string
        public int this[string day]
        {
            get
            {
                return (GetDay(day));
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            DayCollection week = new DayCollection();
            System.Console.WriteLine(week["Fri"]);
            System.Console.WriteLine(week["Made-up Day"]);
        }
    }
```

出力

```
5

-1
```

プログラミングの注意点

インデクサのセキュリティと信頼性を改善するには、主に 2 つの方法があります。

- インデクサからアクセスするバッファまたは配列で、値の設定および取得を行う場合、コードで範囲と型のチェックを必ず実行するようにします。
- **get** アクセサと set アクセサのアクセシビリティを設定し、適切な制限を指定します。これは、**set** アクセサの場合、特に重要です。詳細については、「非対称アクセサのアクセシビリティ (C# プログラミング ガイド)」を参照してください。

参照

処理手順

インデクサのサンプル

関連項目

インデクサ (C# プログラミング ガイド)

プロパティ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

インターフェイスのインデクサ (C# プログラミング ガイド)

インターフェイス (C# リファレンス) に対してインデクサを宣言できます。インターフェイス インデクサのアクセサは、クラスのインデクサと次の点が異なります。

- インターフェイスのアクセサは修飾子を使用しません。
- インターフェイスのアクセサには本体がありません。

したがって、アクセサの目的は、インデクサが読み取り/書き込み、読み取り専用、または書き込み専用のいずれであるかを示すことです。

次に示すのは、インターフェイス インデクサのアクセサの例です。

C#

```
public interface ISomeInterface
{
    //...

    // Indexer declaration:
    string this[int index]
    {
        get;
        set;
    }
}
```

インデクサのシグネチャは、同じインターフェイスで宣言されている他のすべてのインデクサのシグネチャとは異なるシグネチャである必要があります。

使用例

次の例は、インターフェイスのインデクサの実装方法を示しています。

C#

```
// Indexer on an interface:
public interface ISomeInterface
{
    // Indexer declaration:
    int this[int index]
    {
        get;
        set;
    }
}

// Implementing the interface.
class IndexerClass : ISomeInterface
{
    private int[] arr = new int[100];
    public int this[int index]   // indexer declaration
    {
        get
        {
            // Check the index limits.
            if (index < 0 || index >= 100)
            {
                return 0;
            }
            else
            {
                return arr[index];
            }
        }
        set
```

```
            {
                if (!(index < 0 || index >= 100))
                {
                    arr[index] = value;
                }
            }
        }
    }

    class MainClass
    {
        static void Main()
        {
            IndexerClass test = new IndexerClass();
            // Call the indexer to initialize the elements #2 and #5.
            test[2] = 4;
            test[5] = 32;
            for (int i = 0; i <= 10; i++)
            {
                System.Console.WriteLine("Element #{0} = {1}", i, test[i]);
            }
        }
    }
```

出力

```
Element #0 = 0
Element #1 = 0
Element #2 = 4
Element #3 = 0
Element #4 = 0
Element #5 = 32
Element #6 = 0
Element #7 = 0
Element #8 = 0
Element #9 = 0
Element #10 = 0
```

前の例では、インターフェイス メンバの完全限定名を使うことで、明示的なインターフェイス メンバの実装を使用できます。たとえば、次のようにします。

```
public string ISomeInterface.this
{
}
```

ただし、完全限定名は、同じインデクサ シグネチャを持つ複数のインターフェイスがクラスで実装されている場合に、あいまいさを避けるためだけに必要です。たとえば、 `Employee` クラスが 2 つのインターフェイス `ICitizen` と `IEmployee` を実装し、両方のインターフェイスに同じインデクサ シグネチャがある場合は、明示的なインターフェイス メンバの実装が必要になります。つまり、次のインデクサ宣言では、`IEmployee` インターフェイスのインデクサが実装されます。

```
public string IEmployee.this
{
}
```

一方、次の宣言では `ICitizen` インターフェイスのインデクサが実装されます。

```
public string ICitizen.this
{
}
```

参照

処理手順

インデクサのサンプル

関連項目
インデクサ (C# プログラミング ガイド)
プロパティ (C# プログラミング ガイド)
インターフェイス (C# プログラミング ガイド)
概念
C# プログラミング ガイド

プロパティとインデクサの比較 (C# プログラミング ガイド)

インデクサはプロパティと似ています。次の表で示す相違点を除けば、プロパティのアクセサに対して定義されているすべての規則が、インデクサのアクセサにも同じように適用されます。

プロパティ	インデクサ
メソッドをパブリック データ メンバのように呼び出すことができます。	オブジェクトのメソッドを、オブジェクトが配列のように呼び出すことができます。
簡易名でアクセスされます。	インデックスでアクセスされます。
静的メンバまたはインスタンス メンバになることができます。	インスタンス メンバである必要があります。
プロパティの get アクセサにはパラメータがありません。	インデクサの **get** アクセサには、インデクサと同じ仮パラメータ リストがあります。
プロパティの set アクセサには、暗黙の **value** パラメータがあります。	インデクサの **set** アクセサには、value パラメータの他に、インデクサと同じ仮パラメータ リストがあります。

参照

関連項目

インデクサ (C# プログラミング ガイド)

プロパティ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

デリゲート (C# プログラミング ガイド)

デリゲートは、メソッドを参照する型です。デリゲートにメソッドを代入すると、デリゲートはそのメソッドとまったく同じように動作します。デリゲート メソッドは、パラメータと戻り値を指定して他のメソッドと同じように使用できます。この例を次に示します。

C#

```
public delegate int PerformCalculation(int x, int y);
```

デリゲートのシグネチャ (シグネチャは、戻り値の型とパラメータで構成されます) に一致するメソッドは、デリゲートに代入できます。このため、メソッド呼び出しをプログラムによって変更でき、また新しいコードを既存のクラスに接続することもできます。デリゲートのシグネチャを知っている限り、独自のデリゲート メソッドを割り当てることができます。

このようにメソッドをパラメータとして参照できるため、デリゲートはコールバック メソッドを定義するのに最適です。たとえば、並べ替えアルゴリズムには、2 つのオブジェクトを比較するメソッドへの参照を渡すことができます。比較コードを分離することによって、アルゴリズムをより一般的な形で記述できます。

デリゲートの概要

デリゲートには、次の特徴があります。

- デリゲートは、C++ の関数ポインタに似ていますが、タイプ セーフです。
- デリゲートを使用すると、メソッドをパラメータとして渡すことができます。
- デリゲートは、コールバック メソッドを定義するのに使用できます。
- デリゲートは連結でき、たとえば、複数のメソッドを 1 つのイベントで呼び出すことができます。
- メソッドは、デリゲートのシグネチャに正確に一致する必要がありません。詳細については、「デリゲートの共変性と反変性 (C# プログラミング ガイド)」を参照してください。
- C# Version 2.0 では、匿名メソッドという概念を導入し、別個に定義されたメソッドの代わりにコード ブロックをパラメータとして渡せるようにしています。

このセクションの内容

- デリゲートの使用 (C# プログラミング ガイド)
- インターフェイスの代わりにデリゲートを使用する場合 (C# プログラミング ガイド)
- 名前付きメソッド (C# プログラミング ガイド)
- 匿名メソッド
- デリゲートの共変性と反変性 (C# プログラミング ガイド)
- 方法 : デリゲートを結合する (マルチキャスト デリゲート) (C# プログラミング ガイド)
- 方法 : デリゲートを宣言し、インスタンス化して使用する (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.11 デリゲート
- 4.2.6 デリゲート型
- 7.5.5.2 デリゲートの呼び出し
- 15 デリゲート

参照

関連項目

Delegate

概念

C# プログラミング ガイド
イベント (C# プログラミング ガイド)

デリゲートの使用 (C# プログラミング ガイド)

デリゲートは、C や C++ の関数ポインタと同じように、メソッドを安全にカプセル化する型です。デリゲートはオブジェクト指向であり、タイプ セーフであり、さらに安全である点が、C の関数ポインタと異なります。デリゲートの型は、デリゲートの名前によって定義されます。次の例では、文字列を引数として受け取り void を返すメソッドをカプセル化できる、`Del` という名前のデリゲートを宣言しています。

C#

```
public delegate void Del(string message);
```

デリゲート オブジェクトは、通常、デリゲートがラップするメソッドの名前を指定することによって、または匿名メソッドを使用して構築されます。デリゲートがインスタンス化されると、デリゲートに対して行われたメソッド呼び出しは、デリゲートによってそのメソッドに渡されます。呼び出し元によってデリゲートに渡されたパラメータはメソッドに渡され、メソッドからの戻り値 (存在する場合) は、デリゲートによって呼び出し元に返されます。これをデリゲートの呼び出しと言います。インスタンス化されたデリゲートは、ラップされたメソッドそのもののように呼び出すことができます。次に例を示します。

C#

```
// Create a method for a delegate.
public static void DelegateMethod(string message)
{
    System.Console.WriteLine(message);
}
```

C#

```
// Instantiate the delegate.
Del handler = DelegateMethod;

// Call the delegate.
handler("Hello World");
```

デリゲート型は、.NET Framework の Delegate クラスから派生します。デリゲート型は sealed であるため派生できず、**Delegate** からカスタム クラスを派生させることはできません。インスタンス化されたデリゲートはオブジェクトなので、パラメータとして渡したり、プロパティに代入したりできます。このため、メソッドはデリゲートをパラメータとして受け入れ、後ほどそのデリゲートを呼び出すことができます。これを非同期コールバックと言い、長いプロセスが完了したときに呼び出し元に通知するための一般的な方法です。デリゲートをこのように使用する場合、デリゲートを使用するコードでは、使用されているメソッドの実装を認識する必要がありません。この機能は、インターフェイスが提供するカプセル化に似ています。詳細については、「インターフェイスの代わりにデリゲートを使用する場合 (C# プログラミング ガイド)」を参照してください。

これ以外に、コールバックは、カスタム比較メソッドを定義し、そのデリゲートを並べ替えメソッドに渡すためにもよく使用されます。これにより、呼び出し元のコードを並べ替えアルゴリズムの一部にできます。次のメソッドの例では、`Del` 型をパラメータとして使用しています。

C#

```
public void MethodWithCallback(int param1, int param2, Del callback)
{
    callback("The number is: " + (param1 + param2).ToString());
}
```

上で作成したデリゲートは、次のようにメソッドに渡すことができます。

C#

```
MethodWithCallback(1, 2, handler);
```

そして、次の出力をコンソールに送ります。

```
The number is: 3
```

デリゲートを抽象として使用すると、MethodWithCallback は、コンソールを直接呼び出す必要がないので、コンソールを念頭に入れて設計する必要がありません。MethodWithCallback が実行することは、文字列を準備し、それを別のメソッドに渡すだけです。このことが特に有効な理由は、デリゲート メソッドが任意の数のパラメータを使用できるという事実にあります。

インスタンス メソッドをラップするようにデリゲートを構築すると、デリゲートは、インスタンスとメソッドの両方を参照します。デリゲートは、インスタンス型がラップしているメソッド以外の情報を持たないので、デリゲート シグネチャに一致するメソッドがオブジェクトに存在する限り、あらゆる型のオブジェクトを参照できます。静的メソッドをラップするようにデリゲートを構築すると、デリゲートは静的メソッドだけを参照します。次に宣言の例を示します。

C#

```
public class MethodClass
{
    public void Method1(string message) { }
    public void Method2(string message) { }
}
```

これで、上の静的 DelegateMethod と合わせて 3 つのメソッドを Del インスタンスによってラップできるようになりました。

デリゲートを呼び出すと、デリゲートは複数のメソッドを呼び出すことができます。これをマルチキャスティングと言います。デリゲートのメソッド リスト (呼び出しリスト) に新しいメソッドを追加するには、加算演算子 ("+") または加算代入演算子 ("+=") を使用して 2 つのデリゲートを加算します。次に例を示します。

C#

```
MethodClass obj = new MethodClass();
Del d1 = obj.Method1;
Del d2 = obj.Method2;
Del d3 = DelegateMethod;

//Both types of assignment are valid.
Del allMethodsDelegate = d1 + d2;
allMethodsDelegate += d3;
```

この時点で、allMethodsDelegate の呼び出しリストには、Method1、Method2、DelegateMethod の 3 つのメソッドが含まれています。元の 3 つのデリゲート (d1、d2、および d3) は変更されません。allMethodsDelegate を呼び出すと、3 つのメソッドがすべて順に呼び出されます。デリゲートで参照パラメータを使用している場合、参照は、これら 3 つのメソッドに順に渡され、1 つのメソッドによって行われた変更が次のメソッドに対して可視になります。これらのメソッドのいずれかが、メソッド内でキャッチされない例外をスローした場合、その例外はデリゲートの呼び出し元に渡され、呼び出しリスト内の後続のメソッドが呼び出されなくなります。デリゲートに戻り値や out パラメータが存在する場合は、最後に呼び出されたメソッドの戻り値とパラメータを返します。呼び出しリストからメソッドを削除するには、デクリメント演算子 ("-") またはデクリメント代入演算子 ("-=") を使用します。次に例を示します。

C#

```
//remove Method1
allMethodsDelegate -= d1;

// copy AllMethodsDelegate while removing d2
Del oneMethodDelegate = allMethodsDelegate - d2;
```

デリゲート型は **System.Delegate** から派生するため、このクラスで定義されているメソッドとプロパティをデリゲートで呼び出すことができます。たとえば、デリゲートの呼び出しリストに含まれるメソッドの数を確認する場合は、次のようなコードを記述できます。

C#

```
int invocationCount = d1.GetInvocationList().GetLength(0);
```

呼び出しリストに複数のメソッドを含むデリゲートは、**System.Delegate** のサブクラスである MulticastDelegate から派生します。これらのクラスは共に **GetInvocationList** をサポートするので、上のコードはどちらの場合にも有効です。

マルチキャスト デリゲートは、イベント処理で広く使用されています。イベント ソース オブジェクトは、特定のイベントの受け取り先として登録されている受け取り側のオブジェクトにイベント通知を送信します。イベントに対して登録するには、受け取り側でイベントを処理できるメソッドを作成し、そのメソッドのデリゲートを作成して、イベント ソースにデリゲートを渡します。イベントが発生すると、ソースがデリゲートを呼び出します。デリ

ゲートは、受け取り側のイベント処理メソッドを呼び出し、イベント データを伝達します。特定のイベントのデリゲート型は、イベント ソースによって定義されます。詳細については、「イベント (C# プログラミング ガイド)」を参照してください。

2 つの異なる型が代入されたデリゲートをコンパイル時に比較すると、コンパイル エラーになります。デリゲート インスタンスが静的な **System.Delegate** 型の場合、比較は可能ですが、実行時に false を返します。次に例を示します。

C#

```
delegate void Delegate1();
delegate void Delegate2();

static void method(Delegate1 d, Delegate2 e, System.Delegate f)
{
    // Compile-time error.
    //Console.WriteLine(d == e);

    // OK at compile-time. False if the run-time type of f
    //is not the same as that of d.
    System.Console.WriteLine(d == f);
}
```

参照

関連項目

デリゲートの共変性と反変性 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

デリゲート (C# プログラミング ガイド)

イベント (C# プログラミング ガイド)

名前付きメソッド (C# プログラミング ガイド)

デリゲートは、名前付きメソッドに関連付けることができます。名前付きメソッドを使用してデリゲートをインスタンス化するときは、次のようにメソッドをパラメータとして渡します。

C#

```
// Declare a delegate:
delegate void Del(int x);

// Define a named method:
void DoWork(int k) { /* ... */ }

// Instantiate the delegate using the method as a parameter:
Del d = obj.DoWork;
```

これを名前付きメソッドの使用と言います。名前付きメソッドで構築されたデリゲートは、静的メソッドもインスタンス化されたメソッドもカプセル化できます。C# の以前のバージョンでは、名前付きメソッドを使用するのが、デリゲートをインスタンス化する唯一の方法ですが、ただし、新しいメソッドを作成するのがオーバーヘッドの点で望ましくない場合、C# 2.0 では、デリゲートをインスタンス化し、デリゲートが呼び出されたときに処理するコード ブロックを直接指定できます。これを 匿名メソッド (C# プログラミング ガイド) と言います。

解説

デリゲート パラメータとして渡すメソッドは、デリゲート宣言と同じシグネチャを持つ必要があります。

デリゲート インスタンスは、静的メソッドもインスタンス メソッドもカプセル化できます。

デリゲートは、out パラメータを使用できますが、マルチキャスト イベント デリゲートでは、どのデリゲートが呼び出されるかわからないので、このパラメータを使用しないことをお勧めします。

例 1

デリゲートの宣言と使用について、簡単な例を次に示します。デリゲート `Del` とそれに関連付けられているメソッド `MultiplyNumbers` は共に同じシグネチャを持つことに注意してください。

C#

```
// Declare a delegate
delegate void Del(int i, double j);

class MathClass
{
    static void Main()
    {
        MathClass m = new MathClass();

        // Delegate instantiation using "MultiplyNumbers"
        Del d = m.MultiplyNumbers;

        // Invoke the delegate object.
        System.Console.WriteLine("Invoking the delegate using 'MultiplyNumbers':");
        for (int i = 1; i <= 5; i++)
        {
            d(i, 2);
        }
    }

    // Declare the associated method.
    void MultiplyNumbers(int m, double n)
    {
        System.Console.Write(m * n + " ");
    }
}
```

出力

```
Invoking the delegate using 'MultiplyNumbers':

2 4 6 8 10
```

例 2

次の例では、1 つのデリゲートを静的メソッドとインスタンス メソッドの両方に割り当て、各メソッドから特定の情報を戻します。

C#

```
// Declare a delegate
delegate void Del();

class SampleClass
{
    public void InstanceMethod()
    {
        System.Console.WriteLine("A message from the instance method.");
    }

    static public void StaticMethod()
    {
        System.Console.WriteLine("A message from the static method.");
    }
}

class TestSampleClass
{
    static void Main()
    {
        SampleClass sc = new SampleClass();

        // Map the delegate to the instance method:
        Del d = sc.InstanceMethod;
        d();

        // Map to the static method:
        d = SampleClass.StaticMethod;
        d();
    }
}
```

出力

```
A message from the instance method.

A message from the static method.
```

参照

処理手順

方法 : デリゲートを結合する (マルチキャスト デリゲート) (C# プログラミング ガイド)

概念

C# プログラミング ガイド

デリゲート (C# プログラミング ガイド)

イベント (C# プログラミング ガイド)

匿名メソッド (C# プログラミング ガイド)

C# 2.0 より前のバージョンでは、デリゲートを宣言するには名前付きメソッドを使用するしかありませんでした。C# 2.0 では、匿名メソッドが導入されています。

匿名メソッドは、基本的にコード ブロックをデリゲート パラメータとして渡すために作成します。この例を次に示します。

C#

```
// Create a handler for a click event
button1.Click += delegate(System.Object o, System.EventArgs e)
                    { System.Windows.Forms.MessageBox.Show("Click!"); };
```

または、次のようにも指定できます。

C#

```
// Create a delegate instance
delegate void Del(int x);

// Instantiate the delegate using an anonymous method
Del d = delegate(int k) { /* ... */ };
```

匿名メソッドを使用すると、デリゲートをインスタンス化する際に別のメソッドを作成する必要がなくなるため、コーディングのオーバーヘッドを削減できます。

たとえば、デリゲートの代わりにコード ブロックを指定すると、メソッドを作成するのが余分なオーバーヘッドと思われる場合に役立つことがあります。新しいスレッドを開始する場合などがそのよい例です。このクラスではスレッドを作成し、スレッドが実行するコードも含みますが、デリゲート用の追加のメソッドを作成する必要がありません。

C#

```
void StartThread()
{
    System.Threading.Thread t1 = new System.Threading.Thread
      (delegate()
            {
                System.Console.Write("Hello, ");
                System.Console.WriteLine("World!");
            });
    t1.Start();
}
```

解説

匿名メソッドのパラメータのスコープは *anonymous-method-block* です。

対象がブロックの外部にある匿名メソッド ブロックの内部でジャンプ ステートメント (goto、break、または continue) を使用するとエラーになります。また、対象がブロックの内部にある匿名メソッド ブロックの外部でジャンプ ステートメント (**goto**、**break**、**continue** など) を使用してもエラーになります。

匿名メソッドの宣言をスコープに含むローカル変数とパラメータは、匿名メソッドの外部変数または取り込まれた変数と呼ばれます。たとえば、次のコード セグメントに示されている `n` は外部変数です。

C#

```
int n = 0;
Del d = delegate() { System.Console.WriteLine("Copy #:{0}", ++n); };
```

ローカル変数と違って、外部変数の有効期間は、匿名メソッドを参照するデリゲートがガベージ コレクションの対象になるまで継続します。`n` への参照は、デリゲートが作成されたときに取り込まれます。

匿名メソッドは、外部スコープの ref パラメータや out パラメータにアクセスできません。

anonymous-method-block 内のアンセーフ コードにはアクセスできません。

使用例

次の例は、デリゲートをインスタンス化する 2 つの方法を示しています。

- デリゲートと匿名メソッドを関連付ける。
- デリゲートと名前付きメソッド (`DoWork`) を関連付ける。

どちらの場合も、デリゲートを呼び出すと、メッセージが表示されます。

C#

```
// Declare a delegate
delegate void Printer(string s);

class TestClass
{
    static void Main()
    {
        // Instatiate the delegate type using an anonymous method:
        Printer p = delegate(string j)
        {
            System.Console.WriteLine(j);
        };

        // Results from the anonymous delegate call:
        p("The delegate using the anonymous method is called.");

        // The delegate instantiation using a named method "DoWork":
        p = new Printer(TestClass.DoWork);

        // Results from the old style delegate call:
        p("The delegate using the named method is called.");
    }

    // The method associated with the named delegate:
    static void DoWork(string k)
    {
        System.Console.WriteLine(k);
    }
}
```

出力

```
The delegate using the anonymous method is called.

The delegate using the named method is called.
```

参照

関連項目

メソッド (C# プログラミング ガイド)

概念

C# プログラミング ガイド

デリゲート (C# プログラミング ガイド)

unsafe コードとポインタ (C# プログラミング ガイド)

名前付きメソッド (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

インターフェイスの代わりにデリゲートを使用する場合 (C# プログラミング ガイド)

クラスの設計時には、デリゲートとインターフェイスのいずれを使用しても型の宣言と実装を分離できます。特定のインターフェイスは、任意のクラスまたは構造体で継承および実装でき、デリゲートは、メソッドがデリゲートのメソッド シグネチャに適合する限り、どのクラスのメソッドに対しても作成できます。また、インターフェイス参照もデリゲートも、インターフェイスやデリゲート メソッドを実装する、クラスを認識していないオブジェクトで使用できます。このようにインターフェイスとデリゲートは類似しているため、クラスの設計時には、デリゲートとインターフェイスをそれぞれどのような場合に使用するかが問題になります。

デリゲートを使用する場合

- イベント デザイン パターンを使用している場合。
- 静的メソッドをカプセル化するのが望ましい場合。
- 呼び出し側が、メソッドを実装しているオブジェクトの他のプロパティ、メソッド、またはインターフェイスにアクセスする必要がない場合。
- 簡単な結合を行う場合。
- クラスでメソッドを複数実装する必要がある場合。

インターフェイスを使用する場合

- 呼び出される可能性がある関連メソッドのグループがある場合。
- クラスでメソッドを 1 つのみ実装する必要がある場合。
- インターフェイスを使用しているクラスで、そのインターフェイスを他のインターフェイス型またはクラス型にキャストする場合。
- 比較メソッドなど、実装されているメソッドがクラスの型や ID にリンクしている場合。

デリゲートの代わりに単一メソッド インターフェイスを使用する適切な例として、IComparable や IComparable があります。**IComparable** では、CompareTo メソッドを宣言し、このメソッドは、同じ型の 2 つのオブジェクトの関係 (より小さい、等しい、より大きい) を示す整数値を返します。**IComparable** は、並べ替えアルゴリズムの基盤として使用できます。デリゲートによる比較メソッドも並べ替えアルゴリズムの基盤として使用できますが、望ましくありません。比較機能がクラスに属し、比較アルゴリズムが実行時に変更されないので、単一メソッド インターフェイスが最も適しています。

参照

関連項目

メソッド (C# プログラミング ガイド)
インターフェイス (C# プログラミング ガイド)

概念

C# プログラミング ガイド
イベント (C# プログラミング ガイド)

デリゲートの共変性と反変性 (C# プログラミング ガイド)

共変性と反変性により、メソッドのシグネチャをデリゲート型に柔軟に一致させることができます。共変性により、メソッドの戻り値の型の派生をデリゲートに定義されている型より強くすることができます。反変性により、メソッドのパラメータ型の派生をデリゲート型より弱くすることができます。

例 1 (共変性)

この例では、デリゲートと、デリゲートのシグネチャ内の戻り値の型から派生した戻り値の型を持つメソッドの使用方法を説明します。`SecondHandler` が返すデータ型は `Dogs` です。これは、デリゲートに定義された `Mammals` 型の派生型です。

C#

```
class Mammals
{
}

class Dogs : Mammals
{
}

class Program
{
    // Define the delegate.
    public delegate Mammals HandlerMethod();

    public static Mammals FirstHandler()
    {
        return null;
    }

    public static Dogs SecondHandler()
    {
        return null;
    }

    static void Main()
    {
        HandlerMethod handler1 = FirstHandler;

        // Covariance allows this delegate.
        HandlerMethod handler2 = SecondHandler;
    }
}
```

例 2 (反変性)

この例では、デリゲートと、デリゲートのシグネチャのパラメータ型の基本型のパラメータを持つメソッドの使用方法を説明します。反変性により、以前は複数のハンドラを使用する必要があった場所で単一のイベント ハンドラを使用できるようになりました。たとえば、EventArgs 入力パラメータを受け付けるイベント ハンドラを作成し、MouseEventArgs 型のパラメータを送信する Button.MouseClick イベントや KeyEventArgs パラメータを送信する TextBox.KeyDown イベントで使用できるようになりました。

C#

```
System.DateTime lastActivity;
public Form1()
{
    InitializeComponent();

    lastActivity = new System.DateTime();
    this.textBox1.KeyDown += this.MultiHandler; //works with KeyEventArgs
    this.button1.MouseClick += this.MultiHandler; //works with MouseEventArgs

}

// Event hander for any event with an EventArgs or
```

```
// derived class in the second parameter
private void MultiHandler(object sender, System.EventArgs e)
{
    lastActivity = System.DateTime.Now;
}
```

参照

関連項目

汎用デリゲート (C# プログラミング ガイド)

概念

C# プログラミング ガイド

イベント (C# プログラミング ガイド)

デリゲート (C# プログラミング ガイド)

方法：デリゲートを結合する (マルチキャスト デリゲート) (C# プログラミング ガイド)

次の例は、マルチキャスト デリゲートの結合方法を示しています。デリゲート オブジェクトには、+ 演算子を使用して 1 つのデリゲート インスタンスに代入し、マルチキャストにできるという特長があります。結合されたデリゲートは、結合元の 2 つのデリゲートを呼び出します。結合できるのは同じ型のデリゲートだけです。

- 演算子を使用すると、結合されたデリゲートからコンポーネントのデリゲートを削除できます。

使用例

C#

```
delegate void Del(string s);

class TestClass
{
    static void Hello(string s)
    {
        System.Console.WriteLine("  Hello, {0}!", s);
    }

    static void Goodbye(string s)
    {
        System.Console.WriteLine("  Goodbye, {0}!", s);
    }

    static void Main()
    {
        Del a, b, c, d;

        // Create the delegate object a that references
        // the method Hello:
        a = Hello;

        // Create the delegate object b that references
        // the method Goodbye:
        b = Goodbye;

        // The two delegates, a and b, are composed to form c:
        c = a + b;

        // Remove a from the composed delegate, leaving d,
        // which calls only the method Goodbye:
        d = c - a;

        System.Console.WriteLine("Invoking delegate a:");
        a("A");
        System.Console.WriteLine("Invoking delegate b:");
        b("B");
        System.Console.WriteLine("Invoking delegate c:");
        c("C");
        System.Console.WriteLine("Invoking delegate d:");
        d("D");
    }
}
```

出力

```
Invoking delegate a:
  Hello, A!
Invoking delegate b:
  Goodbye, B!
Invoking delegate c:
  Hello, C!
```

```
  Goodbye, C!
Invoking delegate d:
  Goodbye, D!
```

参照

関連項目

MulticastDelegate

概念

C# プログラミング ガイド

イベント (C# プログラミング ガイド)

方法：デリゲートを宣言し、インスタンス化して使用する (C# プログラミング ガイド)

デリゲートは、次のように宣言します。

C#

```
public delegate void Del<T>(T item);
public void Notify(int i) { }
```

C#

```
Del<int> d1 = new Del<int>(Notify);
```

C# 2.0 では、次に示す簡単な構文でデリゲートを宣言することもできます。

C#

```
Del<int> d2 = Notify;
```

この例では、デリゲートの宣言方法、インスタンス化方法、および使用方法を示します。`BookDB` クラスは、書籍のデータベースを管理する書店データベースをカプセル化します。このクラスは、`ProcessPaperbackBooks` メソッドを公開します。このメソッドは、データベースからすべてのペーパーバックを検索し、それぞれについてデリゲートを呼び出します。使用されている **delegate** 型は、`ProcessBookDelegate` に呼び出されます。`Test` クラスはこのクラスを使用して、ペーパーバックの書名と平均価格を出力します。

デリゲートを使用すると、書店データベースとクライアント コードの機能の分担を適切に行うことができます。クライアント コードは、書籍の在庫状況や書店コードがペーパーバックを検索する方法については関知しません。書店コードは、ペーパーバックの検索後の処理については関知しません。

使用例

C#

```
// A set of classes for handling a bookstore:
namespace Bookstore
{
    using System.Collections;

    // Describes a book in the book list:
    public struct Book
    {
        public string Title;        // Title of the book.
        public string Author;       // Author of the book.
        public decimal Price;       // Price of the book.
        public bool Paperback;      // Is it paperback?

        public Book(string title, string author, decimal price, bool paperBack)
        {
            Title = title;
            Author = author;
            Price = price;
            Paperback = paperBack;
        }
    }

    // Declare a delegate type for processing a book:
    public delegate void ProcessBookDelegate(Book book);

    // Maintains a book database.
    public class BookDB
    {
        // List of all books in the database:
```

```
        ArrayList list = new ArrayList();

        // Add a book to the database:
        public void AddBook(string title, string author, decimal price, bool paperBack)
        {
            list.Add(new Book(title, author, price, paperBack));
        }

        // Call a passed-in delegate on each paperback book to process it:
        public void ProcessPaperbackBooks(ProcessBookDelegate processBook)
        {
            foreach (Book b in list)
            {
                if (b.Paperback)
                    // Calling the delegate:
                    processBook(b);
            }
        }
    }
}


// Using the Bookstore classes:
namespace BookTestClient
{
    using Bookstore;

    // Class to total and average prices of books:
    class PriceTotaller
    {
        int countBooks = 0;
        decimal priceBooks = 0.0m;

        internal void AddBookToTotal(Book book)
        {
            countBooks += 1;
            priceBooks += book.Price;
        }

        internal decimal AveragePrice()
        {
            return priceBooks / countBooks;
        }
    }

    // Class to test the book database:
    class TestBookDB
    {
        // Print the title of the book.
        static void PrintTitle(Book b)
        {
            System.Console.WriteLine("   {0}", b.Title);
        }

        // Execution starts here.
        static void Main()
        {
            BookDB bookDB = new BookDB();

            // Initialize the database with some books:
            AddBooks(bookDB);

            // Print all the titles of paperbacks:
            System.Console.WriteLine("Paperback Book Titles:");

            // Create a new delegate object associated with the static
            // method Test.PrintTitle:
            bookDB.ProcessPaperbackBooks(PrintTitle);
```

```
            // Get the average price of a paperback by using
            // a PriceTotaller object:
            PriceTotaller totaller = new PriceTotaller();

            // Create a new delegate object associated with the nonstatic
            // method AddBookToTotal on the object totaller:
            bookDB.ProcessPaperbackBooks(totaller.AddBookToTotal);

            System.Console.WriteLine("Average Paperback Book Price: ${0:#.##}",
                    totaller.AveragePrice());
        }

        // Initialize the book database with some test books:
        static void AddBooks(BookDB bookDB)
        {
            bookDB.AddBook("The C Programming Language", "Brian W. Kernighan and Dennis M.
Ritchie", 19.95m, true);
            bookDB.AddBook("The Unicode Standard 2.0", "The Unicode Consortium", 39.95m, tr
ue);
            bookDB.AddBook("The MS-DOS Encyclopedia", "Ray Duncan", 129.95m, false);
            bookDB.AddBook("Dogbert's Clues for the Clueless", "Scott Adams", 12.00m, true)
;
        }
    }
}
```

出力

```
Paperback Book Titles:
   The C Programming Language
   The Unicode Standard 2.0
   Dogbert's Clues for the Clueless
Average Paperback Book Price: $23.97
```

堅牢性の高いプログラム

- デリゲートの宣言

  次のステートメントは、新しいデリゲート型を宣言します。

  C#

  ```
  public delegate void ProcessBookDelegate(Book book);
  ```

  各デリゲート型は、引数の数、引数型、およびカプセル化できるメソッドの戻り値の型を表します。引数型または戻り値の型の新しいセットが必要になった場合には、新しいデリゲート型を宣言する必要があります。

- デリゲートのインスタンス化

  デリゲート型を宣言したら、デリゲート オブジェクトを作成して、特定のメソッドに関連付ける必要があります。上の例では、これは `PrintTitle` メソッドを `ProcessPaperbackBooks` メソッドに渡すことで行われます。次に例を示します。

  C#

  ```
  bookDB.ProcessPaperbackBooks(PrintTitle);
  ```

  このコードは、静的メソッド `Test.PrintTitle` と関連付けられている新しいデリゲート オブジェクトを作成します。同様に、`totaller` オブジェクトの静的でないメソッド `AddBookToTotal` は、次のように渡されます。

  C#

  ```
  bookDB.ProcessPaperbackBooks(totaller.AddBookToTotal);
  ```

どちらの場合も、新しいデリゲート オブジェクトは `ProcessPaperbackBooks` メソッドに渡されます。

一度デリゲートが作成されると、デリゲートに関連付けられたメソッドは変更できません。つまり、デリゲート オブジェクトは不変であることに注意してください。

- デリゲートの呼び出し

  作成されたデリゲート オブジェクトは、通常、デリゲートを呼び出す他のコードに渡されます。デリゲート オブジェクトを呼び出すには、デリゲート オブジェクトの名前を使用します。名前の後ろには、デリゲートへ渡す引数をかっこで囲んで指定します。デリゲートの呼び出し例は、次のとおりです。

  C#

```
processBook(b);
```

  この例のように、デリゲートは、同期的に呼び出すことも、**BeginInvoke** メソッドと **EndInvoke** メソッドを使用して非同期的に呼び出すこともできます。

参照

概念

C# プログラミング ガイド
イベント (C# プログラミング ガイド)
デリゲート (C# プログラミング ガイド)

イベント (C# プログラミング ガイド)

クラスやオブジェクトは、何か重要なことが起こった場合に、イベントを使用して他のクラスまたはオブジェクトに通知を送ります。イベントを送信する (発生させる) クラスをパブリッシャ、イベントを受信する (処理する) クラスをサブスクライバと呼びます。

通常、C# Windows フォームや Web アプリケーションでは、ボタンやリスト ボックスのようなコントロールによって発生したイベントをサブスクライブします。Visual C# 統合開発環境 (IDE) を使用して、コントロールによって発行されたイベントを表示し、処理対象のイベントを選択できます。イベント サブスクリプションを実行するため、IDE は空のイベント ハンドラ メソッドとコードを自動的に追加します。詳細については、「方法 : イベント サブスクリプションとサブスクリプションの解除 (C# プログラミング ガイド)」を参照してください。

イベントの概要

イベントには、次の特徴があります。

- パブリッシャは、イベントがいつ発生するかを特定します。サブスクライバは、イベントに対してどのようなアクションを実行するかを特定します。
- 1 つのイベントに対して、複数のサブスクライバが存在できます。1 つのサブスクライバで、複数のパブリッシャの複数のイベントを処理できます。
- サブスクライバを持たないイベントは呼び出されません。
- イベントは、通常、グラフィカル ユーザー インターフェイスのボタン クリックやメニュー選択などのユーザー アクションを通知するために使用します。
- イベントに複数のサブスクライバが存在する場合は、イベントの発生時に複数のイベント ハンドラが同時に呼び出されます。イベントを非同期で呼び出す方法については、「同期メソッドの非同期呼び出し」を参照してください。
- イベントを使用して、スレッドを同期させることができます。
- .NET Framework クラス ライブラリでは、イベントは EventHandler デリゲートと EventArgs 基本クラスに基づいています。

関連項目

詳細については、次のトピックを参照してください。

- 方法 : イベント サブスクリプションとサブスクリプションの解除 (C# プログラミング ガイド)
- 方法 : .NET Framework ガイドラインに準拠したイベントを発行する (C# プログラミング ガイド)
- 方法 : 派生クラスから基本クラス イベントを発生させる (C# プログラミング ガイド)
- 方法 : インターフェイス イベントを実装する (C# プログラミング ガイド)
- スレッドの同期 (C# プログラミング ガイド)
- 方法 : ディクショナリを使用してイベント インスタンスを格納する (C# プログラミング ガイド)
- イベントのサンプル
- イベントのデザイン

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.4 イベント
- 10.2.7.2 イベント用に予約されているメンバ名
- 10.7 イベント
- 13.2.3 インターフェイスのイベント

参照

関連項目

EventHandler

概念

C# プログラミング ガイド
デリゲート (C# プログラミング ガイド)
その他の技術情報
Windows フォーム内でのイベント ハンドラの作成

方法：イベント サブスクリプションとサブスクリプションの解除 (C# プログラミング ガイド)

イベントの発生時に呼び出されるカスタム コードを作成するときは、別のクラスによって発行されたイベントをサブスクライブします。たとえば、ユーザーのボタン クリックでアプリケーションに何らかの処理を実行させるには、ボタンの "クリック" イベントをサブスクライブします。

Visual Studio 2005 IDE を使用してイベントをサブスクライブするには

1. [プロパティ] ウィンドウが表示されていない場合は、デザイン ビューで、イベント ハンドラを作成するフォームまたはコントロールを右クリックし、[プロパティ] をクリックします。
2. [プロパティ] ウィンドウの上部にある [イベント] アイコンをクリックします。
3. `Load` イベントなど、作成するイベントをダブルクリックします。

   Visual C# により空のイベント ハンドラ メソッドが作成され、コードに追加されます。コード ビューを使用して手動でコードを追加することもできます。たとえば次のコード行では、`Form` クラスが `Load` イベントを発生させたときに呼び出されるイベント ハンドラ メソッドを宣言しています。

   C#

   ```
   private void Form1_Load(object sender, System.EventArgs e)
   {
       // Add your form load event handling code here.
   }
   ```

   プロジェクトの Form1.Designer.cs ファイルの `InitializeComponent` メソッド内に、イベントをサブスクライブする必要があるコード行も自動生成されます。次に例を示します。

   ```
   this.Load += new System.EventHandler(this.Form1_Load);
   ```

プログラムを使用してイベントをサブスクライブするには

1. シグネチャがイベントのデリゲート シグネチャと一致するイベント ハンドラ メソッドを定義します。たとえばイベントがデリゲート型 EventHandler に基づいている場合は、次のコードがメソッド スタブになります。

   ```
   void HandleCustomEvent(object sender, CustomEventArgs a)
   {
      // Do something useful here.
   }
   ```

2. 加算代入演算子 (+=) を使用して、イベントにイベント ハンドラをアタッチします。次の例では、オブジェクト `publisher` に `RaiseCustomEvent` という名前のイベントがあることを想定しています。サブスクライバ クラスがイベントをサブスクライブするには、パブリッシャ クラスの参照が必要です。

   ```
   publisher.RaiseCustomEvent += HandleCustomEvent;
   ```

これは、C# 2.0 で新しく導入された構文です。新しいキーワードを使用してカプセル化する側のデリゲートを明示的に作成する必要がある点では、C# 1.0 の構文と同じです。

```
publisher.RaiseCustomEvent += new CustomEventHandler(HandleCustomEvent);
```

匿名メソッドを使用してイベントをサブスクライブするには

- 加算代入演算子 (+=) を使用して、イベントに匿名メソッドをアタッチします。次の例では、オブジェクト `publisher` に

`RaiseCustomEvent` という名前のイベントがあり、特別なイベント情報を格納する `CustomEventArgs` クラスが定義されていることを想定しています。サブスクライバ クラスがイベントをサブスクライブするには、`publisher` の参照が必要です。

```
publisher.RaiseCustomEvent += delegate(object o, CustomEventArgs e)
{
  string s = o.ToString() + " " + e.ToString();
  Console.WriteLine(s);
};
```

匿名メソッドを使用してイベントをサブスクライブした場合は、簡単にサブスクリプションを解除できない点に注意してください。この場合、サブスクリプションを解除するには、イベントをサブスクライブしたコードに戻り、匿名メソッドをデリゲート変数に格納して、このデリゲートをイベントに追加します。

サブスクリプション解除

イベントの発生時にイベント ハンドラが呼び出されるのを防ぐには、イベント サブスクリプションを解除します。リソースのリークを防ぐには、サブスクライバ オブジェクトを破棄する前にイベント サブスクリプションを解除する必要があります。イベント サブスクリプションを解除しない限り、パブリッシャ オブジェクト内のイベントの基礎となるマルチキャスト デリゲートは、サブスクライバのイベント ハンドラをカプセル化するデリゲートを参照します。パブリッシャ オブジェクトに参照が含まれている限り、サブスクライバ オブジェクトはガベージ コレクションされません。

イベント サブスクリプションを解除するには

- 減算代入演算子 (-=) を使用して、イベント サブスクリプションを解除します。

```
publisher.RaiseCustomEvent -= HandleCustomEvent;
```

すべてのサブスクライバのサブスクリプションが解除されたら、パブリッシャ クラス内のイベント インスタンスが null に設定されます。

参照

処理手順

方法 : .NET Framework ガイドラインに準拠したイベントを発行する (C# プログラミング ガイド)

関連項目

event (C# リファレンス)

-= 演算子 (C# プログラマーズ リファレンス)

+= 演算子 (C# リファレンス)

概念

イベント (C# プログラミング ガイド)

方法：.NET Framework ガイドラインに準拠したイベントを発行する (C# プログラミング ガイド)

次の例では、.NET Framework の標準のパターンに従うイベントを固有のクラスおよび構造体に追加する方法を示します。.NET Framework クラス ライブラリ内のすべてのイベントは、次のように定義されている EventHandler デリゲートに基づいています。

```
public delegate void EventHandler(object sender, EventArgs e);
```

メモ：

.NET Framework 2.0 には、このデリゲートのジェネリック バージョンである EventHandler<T> が導入されています。次の例では、両方のバージョンの使用方法を示します。

ユーザー定義のクラス内のイベントは、値を返すデリゲートを含む、あらゆる有効なデリゲートを基にすることができますが、一般には、次の例のように **EventHandler** を使用して、.NET Framework のパターンを基にすることをお勧めします。

EventHandler パターンに基づいてイベントを発行するには

1. イベントと共にカスタム データを送信する必要がない場合は、この手順を省略し、手順 3a. に進んでください。パブリッシャ クラスとサブスクライバ クラスの両方から認識できるスコープでクラスを宣言し、カスタム イベント データを格納するために必要なメンバを追加します。この例では、単純な文字列が 1 つ返されます。

   ```
   public class CustomEventArgs : EventArgs
   {
       public CustomEventArgs(string s)
       {
           msg = s;
       }
       private string msg;
       public string Message
       {
           get { return msg; }
       }
   }
   ```

2. ジェネリック バージョンの **EventHandler** を使用する場合、この手順は省略します。パブリッシャ クラス内にデリゲートを宣言します。EventHandler で終わる名前を指定します。2 番目のパラメータに、カスタムの EventArgs 型を指定します。

   ```
   public delegate void CustomEventHandler(object sender, CustomEventArgs a);
   ```

3. 次のいずれかの手順で、パブリッシャ クラス内にイベントを宣言します。

   a. カスタムの EventArgs クラスがない場合、Event 型は非ジェネリック バージョンの EventHandler デリゲートになります。このデリゲートは、C# プロジェクトに既定で含まれている System 名前空間内に既に宣言されているため、ここで宣言する必要はありません。

      ```
      public event EventHandler RaiseCustomEvent;
      ```

   b. 非ジェネリック バージョンの **EventHandler** を使用し、EventArgs から派生したカスタム クラスがある場合は、パブリッシャ クラス内のイベントを宣言し、デリゲートを型として使用します。

      ```
      class Publisher
      {
          public event CustomEventHandler RaiseCustomEvent;
      }
      ```

c. ジェネリック バージョンを使用する場合、カスタム デリゲートは不要です。代わりに、イベントの型として、`EventHandler<CustomEventArgs>` を指定します。角かっこの部分は、実際のクラス名で置き換えます。

```
public event EventHandler<CustomEventArgs> RaiseCustomEvent;
```

使用例

次に示すのは、前の手順の具体例です。カスタムの EventArgs クラスを使用し、イベントの型としては `EventHandler<T>` を指定します。

C#

```
namespace DotNetEvents
{
    using System;
    using System.Collections.Generic;

    // Define a class to hold custom event info
    public class CustomEventArgs : EventArgs
    {
        public CustomEventArgs(string s)
        {
            message = s;
        }
        private string message;

        public string Message
        {
            get { return message; }
            set { message = value; }
        }
    }

    // Class that publishes an event
    class Publisher
    {

        // Declare the event using EventHandler<T>
        public event EventHandler<CustomEventArgs> RaiseCustomEvent;

        public void DoSomething()
        {
            // Write some code that does something useful here
            // then raise the event. You can also raise an event
            // before you execute a block of code.
            OnRaiseCustomEvent(new CustomEventArgs("Did something"));

        }

        // Wrap event invocations inside a protected virtual method
        // to allow derived classes to override the event invocation behavior
        protected virtual void OnRaiseCustomEvent(CustomEventArgs e)
        {
            // Make a temporary copy of the event to avoid possibility of
            // a race condition if the last subscriber unsubscribes
            // immediately after the null check and before the event is raised.
            EventHandler<CustomEventArgs> handler = RaiseCustomEvent;

            // Event will be null if there are no subscribers
            if (handler != null)
            {
                // Format the string to send inside the CustomEventArgs parameter
                e.Message += String.Format(" at {0}", DateTime.Now.ToString());

                // Use the () operator to raise the event.
                handler(this, e);
            }
        }
    }
```

```
    //Class that subscribes to an event
    class Subscriber
    {
        private string id;
        public Subscriber(string ID, Publisher pub)
        {
            id = ID;
            // Subscribe to the event using C# 2.0 syntax
            pub.RaiseCustomEvent += HandleCustomEvent;
        }

        // Define what actions to take when the event is raised.
        void HandleCustomEvent(object sender, CustomEventArgs e)
        {
            Console.WriteLine(id + " received this message: {0}", e.Message);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Publisher pub = new Publisher();
            Subscriber sub1 = new Subscriber("sub1", pub);
            Subscriber sub2 = new Subscriber("sub2", pub);

            // Call the method that raises the event.
            pub.DoSomething();

            // Keep the console window open
            Console.WriteLine("Press Enter to close this window.");
            Console.ReadLine();

        }
    }
}
```

参照

関連項目

Delegate

概念

C# プログラミング ガイド
イベント (C# プログラミング ガイド)
イベントのデザイン
デリゲート (C# プログラミング ガイド)

方法：派生クラスから基本クラス イベントを発生させる (C# プログラミング ガイド)

ここでは、単純な例を使用して、基本クラスで宣言したイベントを派生クラスから発生させる標準的な方法を説明します。このパターンは、.NET Framework 基本クラス ライブラリの Windows フォーム クラスでのみ使用できます。

他のコンポーネントの基本クラスとして使用できるクラスを作成するときは、イベントは宣言元のクラスからしか呼び出せない特別なデリゲートであることを考慮する必要があります。派生クラスは、基本クラスの中で宣言されたイベントを直接呼び出せません。常に基本クラスからイベントを発生させることもできますが、ほとんどの場合、派生クラスから基本クラス イベントを発生させることができるようにします。このためには、イベントをラップする基本クラス内に保護された呼び出しメソッドを作成します。この起動メソッドを呼び出すかオーバーライドすることによって、派生クラスから間接的にイベントを呼び出せます。

使用例

C#

```
namespace BaseClassEvents
{
    using System;
    using System.Collections.Generic;

    // Special EventArgs class to hold info about Shapes.
    public class ShapeEventArgs : EventArgs
    {
        private double newArea;

        public ShapeEventArgs(double d)
        {
            newArea = d;
        }
        public double NewArea
        {
            get { return newArea; }
        }
    }

    // Base class event publisher
    public abstract class Shape
    {
        protected double area;

        public double Area
        {
            get { return area; }
            set { area = value; }
        }
        // The event. Note that by using the generic EventHandler<T> event type
        // we do not need to declare a separate delegate type.
        public event EventHandler<ShapeEventArgs> ShapeChanged;

        public abstract void Draw();

        //The event-invoking method that derived classes can override.
        protected virtual void OnShapeChanged(ShapeEventArgs e)
        {
            // Make a temporary copy of the event to avoid possibility of
            // a race condition if the last subscriber unsubscribes
            // immediately after the null check and before the event is raised.
            EventHandler<ShapeEventArgs> handler = ShapeChanged;
            if (handler != null)
            {
                handler(this, e);
            }
        }
```

```
    }

    public class Circle : Shape
    {
        private double radius;
        public Circle(double d)
        {
            radius = d;
            area = 3.14 * radius;
        }
        public void Update(double d)
        {
            radius = d;
            area = 3.14 * radius;
            OnShapeChanged(new ShapeEventArgs(area));
        }
        protected override void OnShapeChanged(ShapeEventArgs e)
        {
            // Do any circle-specific processing here.

            // Call the base class event invocation method.
            base.OnShapeChanged(e);
        }
        public override void Draw()
        {
            Console.WriteLine("Drawing a circle");
        }
    }

    public class Rectangle : Shape
    {
        private double length;
        private double width;
        public Rectangle(double length, double width)
        {
            this.length = length;
            this.width = width;
            area = length * width;
        }
        public void Update(double length, double width)
        {
            this.length = length;
            this.width = width;
            area = length * width;
            OnShapeChanged(new ShapeEventArgs(area));
        }
        protected override void OnShapeChanged(ShapeEventArgs e)
        {
            // Do any rectangle-specific processing here.

            // Call the base class event invocation method.
            base.OnShapeChanged(e);
        }
        public override void Draw()
        {
            Console.WriteLine("Drawing a rectangle");
        }

    }

    // Represents the surface on which the shapes are drawn
    // Subscribes to shape events so that it knows
    // when to redraw a shape.
    public class ShapeContainer
    {
        List<Shape> _list;

        public ShapeContainer()
```

```
        {
            _list = new List<Shape>();
        }

        public void AddShape(Shape s)
        {
            _list.Add(s);
            // Subscribe to the base class event.
            s.ShapeChanged += HandleShapeChanged;
        }

        // ...Other methods to draw, resize, etc.

        private void HandleShapeChanged(object sender, ShapeEventArgs e)
        {
            Shape s = (Shape)sender;

            // Diagnostic message for demonstration purposes.
            Console.WriteLine("Received event. Shape area is now {0}", e.NewArea);

            // Redraw the shape here.
            s.Draw();
        }
    }

    class Test
    {

        static void Main(string[] args)
        {
            //Create the event publishers and subscriber
            Circle c1 = new Circle(54);
            Rectangle r1 = new Rectangle(12, 9);
            ShapeContainer sc = new ShapeContainer();

            // Add the shapes to the container.
            sc.AddShape(c1);
            sc.AddShape(r1);

            // Cause some events to be raised.
            c1.Update(57);
            r1.Update(7, 7);

            // Keep the console window open.
            Console.WriteLine();
            Console.WriteLine("Press Enter to exit");
            Console.ReadLine();
        }
    }
}
```

出力

```
Received event. Shape area is now 178.98
Drawing a circle
Received event. Shape area is now 49
Drawing a rectangle
```

参照

関連項目

アクセス修飾子 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

イベント (C# プログラミング ガイド)

デリゲート (C# プログラミング ガイド)

その他の技術情報

Windows フォーム内でのイベント ハンドラの作成

方法：インターフェイス イベントを実装する (C# プログラミング ガイド)

インターフェイスはイベントを宣言できます。次の例は、クラスにインターフェイス イベントを実装する方法を示しています。基本的な方法は、インターフェイスのメソッドやプロパティを実装する方法と同じです。

クラスにインターフェイス イベントを実装するには

- クラス内にイベントを宣言し、適切な場所で呼び出します。

```
public interface IDrawingObject
{
    event EventHandler ShapeChanged;
}
public class MyEventArgs : EventArgs {…}
public class Shape : IDrawingObject
{
    event EventHandler ShapeChanged;
    void ChangeShape()
    {
        // Do something before the event…
        OnShapeChanged(new MyEventsArgs(…));
        // or do something after the event.
    }
    protected virtual void OnShapeChanged(MyEventArgs e)
    {
        if(ShapeChanged != null)
        {
           ShapeChanged(this, e);
        }
    }
}
```

使用例

次の例では、クラスが複数のインターフェイスを継承し、各インターフェイスが同じ名前のイベントを持っているという、あまり一般的ではない状態の処理方法を示します。この例では、1 つ以上のイベントに対する明示的なインターフェイス実装が必要です。イベントの明示的なインターフェイス実装を記述するときは、イベント アクセサ **add** および **remove** も記述する必要があります。通常、これらのイベント アクセサはコンパイラによって提供されますが、この例では提供されません。

ユーザー固有のアクセサを指定して、2 つのイベントをクラス内の単一のイベントで表すか、別々のイベントで表すかを指定できます。たとえば、インターフェイスの仕様により、イベントを複数回発生させる必要がある場合、各イベントをクラス内の別々の実装に関連付けることができます。次の例では、サブスクライバで `IShape` または `IDrawingObject` の図形参照をキャストすることにより、受信する `OnDraw` イベントを確認します。

C#

```
namespace WrapTwoInterfaceEvents
{
    using System;

    public interface IDrawingObject
    {
        // Raise this event before drawing
        // the object.
        event EventHandler OnDraw;
    }
    public interface IShape
    {
        // Raise this event after drawing
        // the shape.
        event EventHandler OnDraw;
    }
```

```
// Base class event publisher inherits two
// interfaces, each with an OnDraw event
public class Shape : IDrawingObject, IShape
{
    // Create an event for each interface event
    event EventHandler PreDrawEvent;
    event EventHandler PostDrawEvent;

    // Explicit interface implementation required.
    // Associate IDrawingObject's event with
    // PreDrawEvent
    event EventHandler IDrawingObject.OnDraw
    {
        add { PreDrawEvent += value; }
        remove { PreDrawEvent -= value; }
    }
    // Explicit interface implementation required.
    // Associate IShape's event with
    // PostDrawEvent
    event EventHandler IShape.OnDraw
    {
        add { PostDrawEvent += value; }
        remove { PostDrawEvent -= value; }
    }

    // For the sake of simplicity this one method
    // implements both interfaces.
    public void Draw()
    {
        // Raise IDrawingObject's event before the object is drawn.
        EventHandler handler = PreDrawEvent;
        if (handler != null)
        {
            handler(this, new EventArgs());
        }
        Console.WriteLine("Drawing a shape.");

        // RaiseIShape's event after the object is drawn.
        handler = PostDrawEvent;
        if (handler != null)
        {
            handler(this, new EventArgs());
        }
    }
}
public class Subscriber1
{
    // References the shape object as an IDrawingObject
    public Subscriber1(Shape shape)
    {
        IDrawingObject d = (IDrawingObject)shape;
        d.OnDraw += new EventHandler(d_OnDraw);
    }

    void d_OnDraw(object sender, EventArgs e)
    {
        Console.WriteLine("Sub1 receives the IDrawingObject event.");
    }
}
// References the shape object as an IShape
public class Subscriber2
{
    public Subscriber2(Shape shape)
    {
        IShape d = (IShape)shape;
        d.OnDraw += new EventHandler(d_OnDraw);
```

```
        }

        void d_OnDraw(object sender, EventArgs e)
        {
            Console.WriteLine("Sub2 receives the IShape event.");
        }
    }


    public class Program
    {
        static void Main(string[] args)
        {
            Shape shape = new Shape();
            Subscriber1 sub = new Subscriber1(shape);
            Subscriber2 sub2 = new Subscriber2(shape);
            shape.Draw();

            Console.WriteLine("Press Enter to close this window.");
            Console.ReadLine();
        }
    }

}
```

出力

```
Sub1 receives the IDrawingObject event.
Drawing a shape.
Sub2 receives the IShape event.
```

参照

処理手順

方法 : 派生クラスから基本クラス イベントを発生させる (C# プログラミング ガイド)

関連項目

明示的なインターフェイスの実装 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

イベント (C# プログラミング ガイド)

デリゲート (C# プログラミング ガイド)

方法：ディクショナリを使用してイベント インスタンスを格納する (C# プログラミング ガイド)

数多くのイベントを公開する場合に **accessor-declarations** を使用すると、各イベントにフィールドを割り当てる代わりにディクショナリを使用してイベント インスタンスを格納できます。これは、多数のイベントがあるものの、それらのほとんどが実装されそうもない場合にだけ便利です。

使用例

C#

```
public delegate void EventHandler1(int i);
public delegate void EventHandler2(string s);

public class PropertyEventsSample
{
    private System.Collections.Generic.Dictionary<string, System.Delegate> eventTable;

    public PropertyEventsSample()
    {
        eventTable = new System.Collections.Generic.Dictionary<string, System.Delegate>();
        eventTable.Add("Event1", null);
        eventTable.Add("Event2", null);
    }

    public event EventHandler1 Event1
    {
        add
        {
            eventTable["Event1"] = (EventHandler1)eventTable["Event1"] + value;
        }
        remove
        {
            eventTable["Event1"] = (EventHandler1)eventTable["Event1"] - value;
        }
    }

    public event EventHandler2 Event2
    {
        add
        {
            eventTable["Event2"] = (EventHandler2)eventTable["Event2"] + value;
        }
        remove
        {
            eventTable["Event2"] = (EventHandler2)eventTable["Event2"] - value;
        }
    }

    internal void RaiseEvent1(int i)
    {
        EventHandler1 handler1;
        if (null != (handler1 = (EventHandler1)eventTable["Event1"]))
        {
            handler1(i);
        }
    }

    internal void RaiseEvent2(string s)
    {
        EventHandler2 handler2;
        if (null != (handler2 = (EventHandler2)eventTable["Event2"]))
        {
            handler2(s);
        }
    }
```

```
}

public class TestClass
{
    public static void Delegate1Method(int i)
    {
        System.Console.WriteLine(i);
    }

    public static void Delegate2Method(string s)
    {
        System.Console.WriteLine(s);
    }

    static void Main()
    {
        PropertyEventsSample p = new PropertyEventsSample();

        p.Event1 += new EventHandler1(TestClass.Delegate1Method);
        p.Event1 += new EventHandler1(TestClass.Delegate1Method);
        p.Event1 -= new EventHandler1(TestClass.Delegate1Method);
        p.RaiseEvent1(2);

        p.Event2 += new EventHandler2(TestClass.Delegate2Method);
        p.Event2 += new EventHandler2(TestClass.Delegate2Method);
        p.Event2 -= new EventHandler2(TestClass.Delegate2Method);
        p.RaiseEvent2("TestString");
    }
}
```

出力

```
2
TestString
```

参照

概念

C# プログラミング ガイド
イベント (C# プログラミング ガイド)
デリゲート (C# プログラミング ガイド)

イベント ベースの非同期パターンを使用したマルチスレッド プログラミング

非同期機能をクライアント コードに公開する方法は数多くあります。イベント ベースの非同期パターンは、非同期動作を示すクラスに対して推奨される方法を規定します。

このセクションの内容

イベントベースの非同期パターンの概要

イベント ベースの非同期パターンによって、マルチスレッド デザイン固有の多くの複雑な問題を気にせずに、マルチスレッド アプリケーションの利点を活用できるしくみを説明します。

イベントベースの非同期パターンの実装

非同期機能を持つクラスをパッケージ化するための標準的な方法について説明します。

イベントベースの非同期パターンを実装するための推奨される手順

イベント ベースの非同期パターンに従って非同期機能を公開するための要件について説明します。

イベントベースの非同期パターンをいつ実装するかの決定

どのような場合に、IAsyncResult パターンではなく、イベント ベースの非同期パターンの実装を選択するかを判断する方法について説明します。

チュートリアル : イベントベースの非同期パターンをサポートするコンポーネントの実装

イベント ベースの非同期パターンを実装するコンポーネントの作成方法を示します。これは、System.ComponentModel 名前空間のヘルパー クラスを使用して実装します。これにより、コンポーネントは任意のアプリケーション モデルで正常に動作します。

方法 : イベントベースの非同期パターンをサポートするコンポーネントを使用する

イベント ベースの非同期パターンをサポートするコンポーネントの使用方法について説明します。

参照

AsyncOperation

AsyncOperation クラスについて説明し、すべてのメンバへのリンクの一覧を示します。

AsyncOperationManager

AsyncOperationManager クラスについて説明し、すべてのメンバへのリンクの一覧を示します。

BackgroundWorker

BackgroundWorker コンポーネントについて説明し、すべてのメンバへのリンクの一覧を示します。

関連するセクション

イベントベースの非同期パターンの技術サンプル

イベント ベースの非同期パターンを使用して一般的な非同期操作を実行する方法を示します。

Visual Basic におけるマルチスレッド

.NET Framework のマルチスレッド機能について説明します。

参照

概念

マネージ スレッド処理の実施

イベントとデリゲート

その他の技術情報

コンポーネントのマルチスレッド

非同期プログラミングのデザイン パターン

ジェネリック (C# プログラミング ガイド)

ジェネリックは、C# 言語の Version 2.0 と共通言語ランタイム (CLR) の新機能です。ジェネリックは、.NET Framework に型パラメータという概念を導入します。型パラメータを使用すると、クラスやメソッドがクライアント コードで宣言され、インスタンス化されるまで、1 つ以上の型の指定を遅延させるクラスとメソッドを設計できます。たとえば、ジェネリック型パラメータ T を使用すると、次に示すようにランタイムのキャストやボックス化操作のコストやリスクを負わずに他のクライアント コードで使用できる単一のクラスを記述できます。

C#

```
// Declare the generic class
public class GenericList<T>
{
    void Add(T input) { }
}
class TestGenericList
{
    private class ExampleClass { }
    static void Main()
    {
        // Declare a list of type int
        GenericList<int> list1 = new GenericList<int>();

        // Declare a list of type string
        GenericList<string> list2 = new GenericList<string>();

        // Declare a list of type ExampleClass
        GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();
    }
}
```

ジェネリックの概要

- ジェネリック型は、コードの再利用性、タイプ セーフ性、およびパフォーマンスを最大化するために使用します。
- ジェネリックの最も一般的な用途は、コレクション クラスの作成です。
- .NET Framework クラス ライブラリには、複数の新しいジェネリック コレクション クラスが System.Collections.Generic 名前空間に含まれています。これらのクラスは、System.Collections 名前空間の ArrayList などのクラスの代わりとして、できる限り使用してください。
- 独自のジェネリック インターフェイス、クラス、メソッド、イベント、およびデリゲートを作成できます。
- ジェネリック クラスは、特定のデータ型のメソッドのみへのアクセスを許可するように制約できます。
- ジェネリック データ型で使用される型に関する情報は、実行時にリフレクションを使用して取得できます。

関連項目

詳細情報

- ジェネリックの概要 (C# プログラミング ガイド)
- ジェネリックの利点 (C# プログラミング ガイド)
- ジェネリック型の型パラメータ (C# プログラミング ガイド)
- 型パラメータの制約 (C# プログラミング ガイド)
- ジェネリック クラス (C# プログラミング ガイド)
- ジェネリック インターフェイス (C# プログラミング ガイド)
- ジェネリック メソッド (C# プログラミング ガイド)
- 汎用デリゲート (C# プログラミング ガイド)
- ジェネリック コードの default キーワード (C# プログラミング ガイド)

- C++ テンプレートと C# ジェネリックの違い (C# プログラミング ガイド)
- ジェネリックとリフレクション (C# プログラミング ガイド)
- ランタイムのジェネリック (C# プログラミング ガイド)
- .NET Framework クラス ライブラリのジェネリック (C# プログラミング ガイド)
- ジェネリックのサンプル (C#)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 20 ジェネリック

参照

関連項目

データ型 (C# プログラミング ガイド)
<typeparam> (C# プログラミング ガイド)
<typeparamref> (C# プログラミング ガイド)
System.Collections.Generic

概念

C# プログラミング ガイド

ジェネリックの概要 (C# プログラミング ガイド)

ジェネリック クラスとジェネリック メソッドは、それぞれの非ジェネリック バージョンでは実現できない形で再利用性、タイプ セーフ、および効率性を兼ね備えています。通常、ジェネリックは、コレクションおよびコレクションに対して動作するメソッドと共に使用されます。Version 2.0 の .NET Framework クラス ライブラリには、System.Collections.Generic という新しい名前空間が用意されています。この名前空間には、ジェネリック ベースの新しいコレクション クラスがあります。Version 2.0 を対象にするアプリケーションでは、常に ArrayList などの以前の非ジェネリック クラスの代わりに新しいジェネリック コレクション クラスを使用することをお勧めします。詳細については、「.NET Framework クラス ライブラリのジェネリック (C# プログラミング ガイド)」を参照してください。

また、カスタム ジェネリック型やカスタム ジェネリック メソッドを作成して、独自の汎用ソリューションを提供したり、タイプ セーフで効率的なパターンを設計したりすることもできます。以下のコード例は、簡単な汎用リンク リスト クラスを示しています。通常は、独自のクラスを作成するよりも、.NET Framework クラス ライブラリに用意されている List<T> クラスを使用することをお勧めします。この例では、通常、具体的な型を使用して、リストに格納する項目の型を示すところで、型パラメータ *T* を使用しています。このパラメータは、次のように使用されています。

- **AddHead** メソッドのメソッド パラメータの型として使用されています。
- **GetNext** パブリック メソッドの戻り値の型、および入れ子にされた **Node** クラスの **Data** プロパティとして使用されています。
- 入れ子になったクラスのプライベート メンバ データの型として使用されています。

T は、入れ子になった **Node** クラスで使用できることに注意してください。`GenericList<T>` を、たとえば、`GenericList<int>` のように具体的な型でインスタンス化すると、`T` の各出現箇所は、*int* に置き換えられます。

C#

```
// type parameter T in angle brackets
public class GenericList<T>
{
    // The nested class is also generic on T
    private class Node
    {
        // T used in non-generic constructor
        public Node(T t)
        {
            next = null;
            data = t;
        }

        private Node next;
        public Node Next
        {
            get { return next; }
            set { next = value; }
        }

        // T as private member data type
        private T data;

        // T as return type of property
        public T Data
        {
            get { return data; }
            set { data = value; }
        }
    }

    private Node head;

    // constructor
    public GenericList()
    {
        head = null;
    }

    // T as method parameter type:
    public void AddHead(T t)
```

```
        {
            Node n = new Node(t);
            n.Next = head;
            head = n;
        }

        public IEnumerator<T> GetEnumerator()
        {
            Node current = head;

            while (current != null)
            {
                yield return current.Data;
                current = current.Next;
            }
        }
    }
```

次のコード例は、クライアント コードで `GenericList<T>` ジェネリック クラスを使用して整数のリストを作成する方法を示しています。このコードの型引数を変更するだけで、文字列や他の任意のカスタム型のリストを生成するように簡単に修正できます。

C#

```
    class TestGenericList
    {
        static void Main()
        {
            // int is the type argument
            GenericList<int> list = new GenericList<int>();

            for (int x = 0; x < 10; x++)
            {
                list.AddHead(x);
            }

            foreach (int i in list)
            {
                System.Console.Write(i + " ");
            }
            System.Console.WriteLine("\nDone");
        }
    }
```

参照

関連項目

System.Collections.Generic

概念

C# プログラミング ガイド

ジェネリック (C# プログラミング ガイド)

ジェネリックの利点 (C# プログラミング ガイド)

ジェネリックを使用することによって、汎用基本型 Object との値で型をキャストして一般化を行う、共通言語ランタイムや C# 言語の以前のバージョンの制限を解決できます。ジェネリック クラスを作成すると、コンパイル時にタイプ セーフなコレクションを作成できます。

ジェネリック以外のコレクション クラスを使用する際の制限は、.NET Framework 基本クラス ライブラリの ArrayList コレクション クラスを利用する短いプログラムを作成してみるとわかります。**ArrayList** は、変更なしで参照型や値型を格納できるたいへん便利なコレクション クラスです。

C#

```
// The .NET Framework 1.1 way to create a list:
System.Collections.ArrayList list1 = new System.Collections.ArrayList();
list1.Add(3);
list1.Add(105);

System.Collections.ArrayList list2 = new System.Collections.ArrayList();
list2.Add("It is raining in Redmond.");
list2.Add("It is snowing in the mountains.");
```

ただし、この利便性の一方でマイナス面もあります。**ArrayList** に追加される参照型や値型は暗黙的に **Object** にアップキャストされます。項目が値型の場合は、リストに追加するときにボックス化し、取得するときにボックス化解除する必要があります。キャスト操作もボックス化およびボックス化解除操作もパフォーマンスに影響します。ボックス化およびボックス化解除の影響は、大型のコレクションの反復処理が必要な場合に非常に大きくなります。

また、コンパイル時の型チェックがないという制限もあります。**ArrayList** はすべてを **Object** にキャストするため、コンパイル時にクライアント コードが次のような処理を行うのを防止できません。

C#

```
System.Collections.ArrayList list = new System.Collections.ArrayList();
// Add an integer to the list.
list.Add(3);
// Add a string to the list. This will compile, but may cause an error later.
list.Add("It is raining in Redmond.");

int t = 0;
// This causes an InvalidCastException to be returned.
foreach (int x in list)
{
    t += x;
}
```

異種コレクションを作成することは有効であり、意図的に行うこともありますが、その場合、文字列と **ints** を 1 つの **ArrayList** に組み合わせると、プログラミング エラーになる可能性があります。このエラーは、実行時まで検出されません。

C# 言語の Version 1.0 および 1.1 では、.NET Framework 基本クラス ライブラリのコレクション クラスにおける一般化されたコードの問題は、独自の型に固有のコレクションを記述することによってのみ回避できました。もちろん、このようなクラスは複数のデータ型で再利用できないため、一般化のメリットがなく、格納する型ごとにクラスを書き直す必要があります。

ArrayList や他の同じようなクラスで実際に必要なのは、クライアント コードで、使用する特定のデータ型をインスタンスごとに指定する方法です。このような方法があれば、**T:System.Object** にアップキャストする必要性がなくなり、またコンパイラが型チェックを実行することもできるようになります。言い換えれば、**ArrayList** には *type parameter* が必要なのです。これを実現するのがジェネリックです。**N:System.Collections.Generic** 名前空間の `List<T>` ジェネリック コレクションでは、コレクションに項目を追加するという同じ操作が次のようになります。

C#

```
// The .NET Framework 2.0 way to create a list
List<int> list1 = new List<int>();

// No boxing, no casting:
list1.Add(3);

// Compile-time error:
```

```
// list1.Add("It is raining in Redmond.");
```

クライアント コードでは、**ArrayList** と比較される `List<T>` と共に追加される構文は、宣言とインスタンス化での型引数だけです。このようにコーディングは若干複雑になりますが、それと引き換えに、**ArrayList** よりも安全であるだけでなく、特にリスト項目が値型のときには処理時間が大幅に縮小するリストを作成できます。

参照

関連項目

ジェネリックの概要 (C# プログラミング ガイド)

System.Collections.Generic

概念

C# プログラミング ガイド

ジェネリック型の型パラメータ (C# プログラミング ガイド)

ジェネリック型またはジェネリック メソッドの定義では、型パラメータは、ジェネリック型の変数をインスタンス化するときにクライアントが指定する、特定の型のプレースホルダです。「ジェネリックの概要 (C# プログラミング ガイド)」に紹介されている `GenericList<T>` などのジェネリック クラスは、実際のところ型ではなく、型の設計図のようなものなので、そのままでは使用できません。`GenericList<T>` を使用するには、クライアント コードで、山かっこ内に型の引数を指定して構築型を宣言し、インスタンス化する必要があります。この特定クラスの型の引数には、コンパイラで認識される任意の型を使用できます。構築型のインスタンスはいくつでも作成できます。また、それぞれに対して、次のように異なる型の引数を使用できます。

C#

```
GenericList<float> list1 = new GenericList<float>();
GenericList<ExampleClass> list2 = new GenericList<ExampleClass>();
GenericList<ExampleStruct> list3 = new GenericList<ExampleStruct>();
```

この `GenericList<T>` の各インスタンスでは、クラスで `T` が発生すると、実行時に型の引数で置換されます。この置換を利用して、単一のクラス定義を使用したタイプ セーフで効率的なオブジェクトを 3 つ作成しました。この置換を CLR で実行する方法の詳細については、「ランタイムのジェネリック (C# プログラミング ガイド)」を参照してください。

型パラメータの名前付けのガイドライン

- 1 文字の名前だけで理解でき、追加の文字が必要ない場合を除き、ジェネリック型の型パラメータには、わかりやすい名前を付けます。

  C#

  ```
  public interface ISessionChannel<TSession> { /*...*/ }
  public delegate TOutput Converter<TInput, TOutput>(TInput from);
  public class List<T> { /*...*/ }
  ```

- 1 文字の型パラメータを持つ型である場合、型パラメータ名として T を使用することを検討します。

  C#

  ```
  public int IComparer<T>() { return 0; }
  public delegate bool Predicate<T>(T item);
  public struct Nullable<T> where T : struct { /*...*/ }
  ```

- わかりやすい型パラメータ名の前に、"T" を付けます。

  C#

  ```
  public interface ISessionChannel<TSession>
  {
      TSession Session { get; }
  }
  ```

- パラメータの名前に型パラメータに関する制約を指定することを検討します。たとえば、`ISession` に制約されるパラメータである場合、`TSession` と指定します。

参照

関連項目

C++ テンプレートと C# ジェネリックの違い (C# プログラミング ガイド)

System.Collections.Generic

概念

C# プログラミング ガイド

ジェネリック (C# プログラミング ガイド)

型パラメータの制約 (C# プログラミング ガイド)

ジェネリック クラスを定義するときは、クライアント コードでクラスをインスタンス化するときに型引数に使用できる型の種類に制限を適用できます。クライアント コードで、制限で許可されていない型を使ってクラスをインスタンス化しようとすると、コンパイル時にエラーが発生します。このような制限を制約と呼びます。制約は、**where** コンテキスト キーワードで指定します。6 種類の制約を次に示します。

制約	説明
where T: 構造体	型引数は、値型である必要があります。Nullable 以外のすべての値型を指定できます。詳細については、「Null 許容型の使用 (C# プログラミング ガイド)」を参照してください。
where T : クラス	型引数は、クラス型、インターフェイス型、デリゲート型、配列型などの参照型である必要があります。
where T : new()	型引数は、パラメータなしのパブリック コンストラクタを持つ必要があります。他の制約と併用する場合、**new()** 制約は最後に指定する必要があります。
where T : <基本クラス名>	型引数は、指定した基本クラスであるか、または指定した基本クラスから派生する必要があります。
where T : <インターフェイス名>	型引数は、指定したインターフェイスであるか、または指定したインターフェイスを実装する必要があります。インターフェイス制約は複数指定できます。制約元のインターフェイスもジェネリックにできます。
where T : U	T の位置にある型引数は、U の位置にある引数であるか、またはその引数から派生する必要があります。この制約は、生の型制約と呼ばれます。

制約を使用する理由

ジェネリック リストの項目をチェックして、その項目が有効であるかどうかを確認したり、他の項目と比較したりする場合は、コンパイラが呼び出す必要がある演算子やメソッドが、クライアント コードで指定される可能性があるすべての型引数でサポートされるというある程度の保証をコンパイラに与える必要があります。この保証が、ジェネリック クラス定義に 1 つ以上の制約を適用することで得られます。たとえば、基本クラス制約では、この型のオブジェクトまたはこの型から派生したオブジェクトだけが型引数として使用されることをコンパイラに伝えます。この保証が得られると、コンパイラは、その型のメソッドをジェネリック クラス内で呼び出すことを許可できます。制約は、**where** コンテキスト キーワードを使用して適用します。基本クラス制約を適用することによって、「ジェネリックの概要 (C# プログラミング ガイド)」の `GenericList<T>` クラスに追加できる機能を次のコード例に示します。

C#

```
public class Employee
{
    private string name;
    private int id;

    public Employee(string s, int i)
    {
        name = s;
        id = i;
    }

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public int ID
    {
        get { return id; }
        set { id = value; }
    }
}

public class GenericList<T> where T : Employee
{
```

```
    private class Node
    {
        private Node next;
        private T data;

        public Node(T t)
        {
            next = null;
            data = t;
        }

        public Node Next
        {
            get { return next; }
            set { next = value; }
        }

        public T Data
        {
            get { return data; }
            set { data = value; }
        }
    }

    private Node head;

    public GenericList() //constructor
    {
        head = null;
    }

    public void AddHead(T t)
    {
        Node n = new Node(t);
        n.Next = head;
        head = n;
    }

    public IEnumerator<T> GetEnumerator()
    {
        Node current = head;

        while (current != null)
        {
            yield return current.Data;
            current = current.Next;
        }
    }

    public T FindFirstOccurrence(string s)
    {
        Node current = head;
        T t = null;

        while (current != null)
        {
            //The constraint enables access to the Name property.
            if (current.Data.Name == s)
            {
                t = current.Data;
                break;
            }
            else
            {
                current = current.Next;
            }
        }
        return t;
```

```
    }
}
```

制約により、T 型のすべての項目が `Employee` オブジェクト、または `Employee` から継承されたオブジェクトのいずれかであることが保証されるため、ジェネリック クラスは、`Employee.Name` プロパティを使用できます。

制約は、次のように同じ型パラメータに複数適用でき、制約自体をジェネリック型にできます。

C#

```
class EmployeeList<T> where T : Employee, IEmployee, System.IComparable<T>, new()
{
    // ...
}
```

型パラメータを制約することで、許容される操作の数と、制約している型とその継承階層内のすべての型でサポートされるメソッドへのメソッド呼び出しの数が増えます。そのため、ジェネリック クラスやジェネリック メソッドを設計するときに、簡単な代入を超える操作を汎用メンバに対して実行したり、**System.Object** でサポートされていないメソッドを呼び出したりする場合は、型パラメータに制約を適用する必要があります。

`where T : class` 制約を適用するときは、**==** キーワードと **!=** 演算子を型パラメータで使用しないことをお勧めします。これらの演算子は、値の等価性ではなく、参照 ID のみをテストするからです。これらの演算子が、引数として使用される型でオーバーロードされた場合でも同じです。次のコードは、この点を示しています。このコードの出力は、String クラスが **==** 演算子をオーバーロードしても false です。

C#

```
public static void OpTest<T>(T s, T t) where T : class
{
    System.Console.WriteLine(s == t);
}
static void Main()
{
    string s1 = "foo";
    System.Text.StringBuilder sb = new System.Text.StringBuilder("foo");
    string s2 = sb.ToString();
    OpTest<string>(s1, s2);
}
```

false が出力されるのは、コンパイル時に、コンパイラが、T が参照型であるということしか認識しないため、すべての参照型に有効な既定の演算子を使用する必要があるからです。値の等価性をテストする必要がある場合も、`where T : IComparable<T>` 制約を適用し、ジェネリック クラスの制約に使用されるすべてのクラスでそのインターフェイスを実装することをお勧めします。

非バインド型パラメータ

`SampleClass<T>{}` パブリック クラスの T など、制約を持たない型パラメータは、非バインド型パラメータと呼ばれます。非バインド型パラメータには、次の規則が適用されます。

- **!=** 演算子と **==** 演算子は、具象型引数がこれらの演算子をサポートするという保証がないため、使用できません。
- これらのパラメータは、**System.Object** との間で変換できます。またはインターフェイス型に明示的に変換できます。
- null と比較できます。非バインド パラメータを **null** に比較したときに型引数が値型の場合は、常に false が返されます。

生の型制約

ジェネリック型パラメータを制約として使用した場合、この制約は生の型制約と呼ばれます。生の型制約は、固有の型パラメータを持つメンバ関数で、そのパラメータを、メンバ関数を包含する側の型の型パラメータに制約する必要があるときに便利です。このコード例を次に示します。

C#

```
class List<T>
{
    void Add<U>(List<U> items) where U : T {/*...*/}
}
```

上の例で、T は、**Add** メソッドのコンテキストでは生の型制約であり、**List** クラスのコンテキストでは非バインド型パラメータです。

生の型制約も、ジェネリック クラスの定義で使用できます。生の型制約も他の型パラメータと同様に山かっこで囲んで宣言する必要があります。

C#

```
  //naked type constraint
  public class SampleClass<T, U, V> where T : V { }
```

ジェネリック クラスを使用した生の型制約の有効性は非常に限られます。というのも、生の型制約が **System.Object** から派生したことしかコンパイラが認識できないからです。ジェネリック クラスの生の型制約は、2 つの型パラメータの間に継承関係を適用する場合に使用します。

参照

関連項目

ジェネリックの概要 (C# プログラミング ガイド)

new 制約 (C# リファレンス)

System.Collections.Generic

概念

C# プログラミング ガイド

ジェネリック クラス (C# プログラミング ガイド)

ジェネリック クラスは、特定のデータ型に固有ではない操作をカプセル化します。一般に、ジェネリック クラスは、リンクされたリスト、ハッシュ テーブル、スタック、キュー、ツリーなどのコレクションと共に使用されます。ジェネリック クラスでは、コレクションに格納されているデータ型にかかわらず、項目の追加と削除などの操作がほぼ同じ方法で実行されます。

コレクション クラスが必要な場合、一般に、.NET Framework 2.0 クラス ライブラリで提供されているものを使用することをお勧めします。これらのクラスの使い方の詳細については、「.NET Framework クラス ライブラリのジェネリック (C# プログラミング ガイド)」を参照してください。

通常、ジェネリック クラスを作成するには、既存の具象クラスから始め、生成と使いやすさが最適なバランスになるまで、1 つずつ、型を型パラメータに変更します。独自にジェネリック クラスを作成する場合は、次の点を考慮する必要があります。

- 型パラメータに一般化する型。

  一般則として、パラメータ化できる型が増えると、コードの柔軟性と再利用できる度合いが向上します。ただし、コードの一般化が多すぎると、他の開発者が読んだり理解したりするのが困難になります。

- 型パラメータに適用する制約 (存在する場合) の内容 (「型パラメータの制約 (C# プログラミング ガイド)」を参照してください)。

  望ましい規則は、必要に応じて型を処理できる範囲で、最大の制約を適用することです。たとえば、参照型でのみジェネリック クラスを使用することがわかっていれば、そのクラスの制約を適用します。こうすることで、値型でクラスを使うという意図しない用法を回避できます。また、`T` で **as** 演算子を使用し、null 値をチェックできるようになります。

- ジェネリックの動作を基本クラスとサブクラスにファクタリングするかどうか。

  ジェネリック クラスは基本クラスとして機能するため、非ジェネリック クラスと同様なデザインの考慮事項が適用されます。一般的な基本クラスから継承する場合の規則を次に示します。

- 1 つ以上のジェネリック インターフェイスを実装するかどうか。

  たとえば、ジェネリック ベースのコレクションに項目を作成するときに使用するクラスを設計している場合、`T` がそのクラスの型である `IComparable<T>` などのインターフェイスの実装が必要になることがあります。

単純なジェネリック クラスの例については、「ジェネリックの概要 (C# プログラミング ガイド)」を参照してください。

型パラメータの規則と制約には、ジェネリック クラスの動作 (特に、継承とメンバのアクセシビリティ) について、暗示的な意味合いがあります。話を先に進める前に、いくつかの用語について理解しておくことが必要です。ジェネリック クラスで、`Node<T>`, クライアント コードでクラスを参照するには、型の引数を指定してクローズ構築型 (`Node<int>`) を作成するか、型パラメータを指定せずにオープン構築型 (`Node<T>`) を作成します。ジェネリック クラスは、具象、クローズ構築、またはオープン構築の各基本クラスから継承できます。

C#

```
class BaseNode { }
class BaseNodeGeneric<T> { }

// concrete type
class NodeConcrete<T> : BaseNode { }

//closed constructed type
class NodeClosed<T> : BaseNodeGeneric<int> { }

//open constructed type
class NodeOpen<T> : BaseNodeGeneric<T> { }
```

非ジェネリック (具象) クラスは、クローズ構築の基本クラスからは継承できますが、オープン構築のクラスや修飾されない型パラメータからは継承できません。実行時に、クライアント コードから基本クラスをインスタンス化するときに必要な型の引数を提示する方法がないためです。

C#

```
//No error
class Node1 : BaseNodeGeneric<int> { }

//Generates an error
//class Node2 : BaseNodeGeneric<T> {}
```

```
  //Generates an error
  //class Node3 : T {}
```

オープン構築型から継承するジェネリック クラスの場合、継承するクラスで共有されない基本クラスの型パラメータに対して、型の引数を提示する必要があります。次にコード例を示します。

C#

```
  class BaseNodeMultiple<T, U> { }

  //No error
  class Node4<T> : BaseNodeMultiple<T, int> { }

  //No error
  class Node5<T, U> : BaseNodeMultiple<T, U> { }

  //Generates an error
  //class Node6<T> : BaseNodeMultiple<T, U> {}
```

オープン構築型から継承するジェネリック クラスでは、基本型に対する制約のスーパーセットである制約、または基本型に対する制約を暗示する制約を指定する必要があります。

C#

```
  class NodeItem<T> where T : System.IComparable<T>, new() { }
  class SpecialNodeItem<T> : NodeItem<T> where T : System.IComparable<T>, new() { }
```

ジェネリック型では、次のように複数の型パラメータと制約を使用できます。

C#

```
  class SuperKeyType<K, V, U>
      where U : System.IComparable<U>
      where V : new()
  { }
```

オープン構築型とクローズ構築j型は、メソッドのパラメータとして使用できます。

C#

```
  void Swap<T>(List<T> list1, List<T> list2)
  {
      //code to swap items
  }

  void Swap(List<int> list1, List<int> list2)
  {
      //code to swap items
  }
```

ジェネリック クラスは不変です。つまり、入力パラメータで `List<BaseClass>` を使用していて、`List<DerivedClass>` を提示しようとすると、コンパイル時にエラーが発生します。

参照

関連項目

System.Collections.Generic

概念

C# プログラミング ガイド

ジェネリック (C# プログラミング ガイド)

ジェネリック インターフェイス (C# プログラミング ガイド)

多くの場合、ジェネリック コレクション クラス、またはコレクション内の項目を表すジェネリック クラスにインターフェイスを定義すると便利です。ジェネリック クラスがある場合、値型に対するボックス化またはボックス化解除の操作を回避するため、IComparable よりも IComparable<T> などのジェネリック インターフェイスを使用することをお勧めします。.NET Framework 2.0 クラス ライブラリでは、System.Collections.Generic 名前空間の新しいコレクション クラスと共に使用する、いくつかの新しいジェネリック インターフェイスが定義されています。

型パラメータに関する制約としてインターフェイスを指定すると、そのインターフェイスを実装する型のみを使用できます。次のコード例で、GenericList<T> クラスから派生する SortedList<T> クラスを示します。詳細については、「ジェネリックの概要 (C# プログラミング ガイド)」を参照してください。SortedList<T> では、制約 `where T : IComparable<T>` を追加しています。これによって、SortedList<T> の **BubbleSort** メソッドが、リスト要素でジェネリックの CompareTo メソッドを使用できるようになります。この例では、リスト要素は単純なクラス Person であり、IComparable<Person> を実装しています。

C#

```
//Type parameter T in angle brackets.
public class GenericList<T> : System.Collections.Generic.IEnumerable<T>
{
    protected Node head;
    protected Node current = null;

    // Nested class is also generic on T
    protected class Node
    {
        public Node next;
        private T data;  //T as private member datatype

        public Node(T t)  //T used in non-generic constructor
        {
            next = null;
            data = t;
        }

        public Node Next
        {
            get { return next; }
            set { next = value; }
        }

        public T Data  //T as return type of property
        {
            get { return data; }
            set { data = value; }
        }
    }

    public GenericList()  //constructor
    {
        head = null;
    }

    public void AddHead(T t)  //T as method parameter type
    {
        Node n = new Node(t);
        n.Next = head;
        head = n;
    }

    // Implementation of the iterator
    public System.Collections.Generic.IEnumerator<T> GetEnumerator()
    {
        Node current = head;
        while (current != null)
        {
            yield return current.Data;
            current = current.Next;
```

```
        }
    }

    // IEnumerable<T> inherits from IEnumerable, therefore this class
    // must implement both the generic and non-generic versions of
    // GetEnumerator. In most cases, the non-generic method can
    // simply call the generic method.
    System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}

public class SortedList<T> : GenericList<T> where T : System.IComparable<T>
{
    // A simple, unoptimized sort algorithm that
    // orders list elements from lowest to highest:

    public void BubbleSort()
    {
        if (null == head || null == head.Next)
        {
            return;
        }
        bool swapped;

        do
        {
            Node previous = null;
            Node current = head;
            swapped = false;

            while (current.next != null)
            {
                //  Because we need to call this method, the SortedList
                //  class is constrained on IEnumerable<T>
                if (current.Data.CompareTo(current.next.Data) > 0)
                {
                    Node tmp = current.next;
                    current.next = current.next.next;
                    tmp.next = current;

                    if (previous == null)
                    {
                        head = tmp;
                    }
                    else
                    {
                        previous.next = tmp;
                    }
                    previous = tmp;
                    swapped = true;
                }
                else
                {
                    previous = current;
                    current = current.next;
                }
            }
        } while (swapped);
    }
}

// A simple class that implements IComparable<T> using itself as the
// type argument. This is a common design pattern in objects that
// are stored in generic lists.
public class Person : System.IComparable<Person>
{
```

```
    string name;
    int age;

    public Person(string s, int i)
    {
        name = s;
        age = i;
    }

    // This will cause list elements to be sorted on age values.
    public int CompareTo(Person p)
    {
        return age - p.age;
    }

    public override string ToString()
    {
        return name + ":" + age;
    }

    // Must implement Equals.
    public bool Equals(Person p)
    {
        return (this.age == p.age);
    }
}

class Program
{
    static void Main()
    {
        //Declare and instantiate a new generic SortedList class.
        //Person is the type argument.
        SortedList<Person> list = new SortedList<Person>();

        //Create name and age values to initialize Person objects.
        string[] names = new string[]
        {
            "Franscoise",
            "Bill",
            "Li",
            "Sandra",
            "Gunnar",
            "Alok",
            "Hiroyuki",
            "Maria",
            "Alessandro",
            "Raul"
        };

        int[] ages = new int[] { 45, 19, 28, 23, 18, 9, 108, 72, 30, 35 };

        //Populate the list.
        for (int x = 0; x < 10; x++)
        {
            list.AddHead(new Person(names[x], ages[x]));
        }

        //Print out unsorted list.
        foreach (Person p in list)
        {
            System.Console.WriteLine(p.ToString());
        }
        System.Console.WriteLine("Done with unsorted list");

        //Sort the list.
        list.BubbleSort();
```

```
            //Print out sorted list.
            foreach (Person p in list)
            {
                System.Console.WriteLine(p.ToString());
            }
            System.Console.WriteLine("Done with sorted list");
        }
    }
```

単一の型に対する制約として、次のように複数のインターフェイスを指定できます。

C#

```
class Stack<T> where T : System.IComparable<T>, IEnumerable<T>
{
}
```

1 つのインターフェイスで、次のように複数の型パラメータを定義できます。

C#

```
interface IDictionary<K, V>
{
}
```

継承される同じ規則が、クラスに関するインターフェイスに適用されます。

C#

```
interface IMonth<T> { }

interface IJanuary     : IMonth<int> { }  //No error
interface IFebruary<T> : IMonth<int> { }  //No error
interface IMarch<T>    : IMonth<T> { }    //No error
//interface IApril<T>  : IMonth<T, U> {}  //Error
```

ジェネリック インターフェイスが非バリアントの場合 (つまり、型パラメータを戻り値としてのみ使用する場合)、そのジェネリック インターフェイスは、非ジェネリック インターフェイスから継承できます。.NET Framework クラス ライブラリでは、IEnumerable<T> は IEnumerable から継承されます。GetEnumerator の戻り値と Current プロパティの取得側では、IEnumerable<T> のみが T を使用するためです。

具象クラスでは、次のように閉じた構造のインターフェイスを実装できます。

C#

```
interface IBaseInterface<T> { }

class SampleClass : IBaseInterface<string> { }
```

ジェネリック クラスでは、インターフェイスで必要なすべての引数がクラスのパラメータ リストに指定されている場合、次のように、ジェネリック インターフェイスまたは閉じた構造のインターフェイスを実装できます。

C#

```
interface IBaseInterface1<T> { }
interface IBaseInterface2<T, U> { }

class SampleClass1<T> : IBaseInterface1<T> { }          //No error
class SampleClass2<T> : IBaseInterface2<T, string> { }  //No error
```

オーバーロードするメソッドを制御する規則は、ジェネリック クラス、ジェネリック構造体、またはジェネリック インターフェイス内のメソッドと同様です。詳細については、「ジェネリック メソッド (C# プログラミング ガイド)」を参照してください。

参照

関連項目

ジェネリックの概要 (C# プログラミング ガイド)

interface

概念

C# プログラミング ガイド

その他の技術情報

.NET Framework におけるジェネリック

ジェネリック メソッド (C# プログラミング ガイド)

ジェネリック メソッドは、型パラメータと共に次のように宣言されるメソッドです。

C#

```
static void Swap<T>(ref T lhs, ref T rhs)
{
    T temp;
    temp = lhs;
    lhs = rhs;
    rhs = temp;
}
```

次のコード例は、型引数として int を使用してメソッドを呼び出す方法を示しています。

C#

```
public static void TestSwap()
{
    int a = 1;
    int b = 2;

    Swap<int>(ref a, ref b);
    System.Console.WriteLine(a + " " + b);
}
```

型引数は省略することもできます。省略された型引数は、コンパイラが推論します。次に示す Swap の呼び出しは、上の呼び出しと同じです。

C#

```
Swap(ref a, ref b);
```

静的メソッドにもインスタンス メソッドにも同じ型の推定規則が適用されます。コンパイラは、渡されたメソッド引数に基づいて型パラメータを推論できます。制約や戻り値のみから型パラメータを推論することはできません。そのため型の推定は、パラメータを持たないメソッドでは無効です。型の推定は、コンパイル時にコンパイラが、オーバーロードされたメソッド シグネチャを解決しようとする前に行われます。コンパイラは、同じ名前を持つすべてのジェネリック メソッドに型の推定ロジックを適用します。オーバーロード解決ステップで、コンパイラは、型の推定が成功したジェネリック メソッドのみを取り込みます。

ジェネリック クラス内の非ジェネリック メソッドは、クラスレベルの型パラメータに次のようにアクセスできます。

C#

```
class SampleClass<T>
{
    void Swap(ref T lhs, ref T rhs) { }
}
```

格納する側のクラスと同じ型パラメータを受け取るジェネリック メソッドを定義した場合、コンパイラは警告 CS0693 を生成します。これは、メソッドのスコープ内で、内側の T に指定された引数が、外側の T に指定された引数を隠ぺいするからです。クラスをインスタンス化したときに指定した以外の型引数を使ってジェネリック クラス メソッドを柔軟に呼び出す必要がある場合は、次の例の `GenericList2<T>` に示されているように、メソッドの型引数に別の識別子を指定することを検討してください。

C#

```
class GenericList<T>
{
    // CS0693
    void SampleMethod<T>() { }
}
```

```
class GenericList2<T>
{
    //No warning
    void SampleMethod<U>() { }
}
```

メソッドの型パラメータでより特殊な演算を実現するには、制約を使用します。次に示す `SwapIfGreater<T>` という `Swap<T>` のバージョンは、`IComparable<T>` を実装する型引数でのみ使用できます。

C#

```
void SwapIfGreater<T>(ref T lhs, ref T rhs) where T : System.IComparable<T>
{
    T temp;
    if (lhs.CompareTo(rhs) > 0)
    {
        temp = lhs;
        lhs = rhs;
        rhs = temp;
    }
}
```

ジェネリック メソッドは、いくつかの型パラメータでオーバーロードできます。たとえば、次のメソッドは、同じクラスにすべて存在できます。

C#

```
void DoWork() { }
void DoWork<T>() { }
void DoWork<T, U>() { }
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 20.6.4 型引数の推論

参照

関連項目

ジェネリックの概要 (C# プログラミング ガイド)
メソッド (C# プログラミング ガイド)
System.Collections.Generic

概念

C# プログラミング ガイド

ジェネリックと配列 (C# プログラミング ガイド)

C# 2.0 では、下限がゼロの 1 次元配列には、IList<T> が自動的に実装されます。これによって、ジェネリック メソッドを作成できるようになり、配列やその他のコレクション型の反復処理を実行するときに同じコードを使用できます。この技法は、主にコレクションのデータを読み込むときに有効です。配列で要素の追加または削除を行うときに **IList<T>** インターフェイスは使用できません。このコンテキストの配列上で RemoveAt などの **IList<T>** メソッドを呼び出そうとすると、例外がスローされます。

次のコード例では、**IList<T>** 入力パラメータを使用する単一のジェネリック メソッドで、リストと配列 (この場合は整数の配列) の両方に対して反復処理を実行する方法を示します。

C#

```
class Program
{
    static void Main()
    {
        int[] arr = { 0, 1, 2, 3, 4 };
        List<int> list = new List<int>();

        for (int x = 5; x < 10; x++)
        {
            list.Add(x);
        }

        ProcessItems<int>(arr);
        ProcessItems<int>(list);
    }

    static void ProcessItems<T>(IList<T> coll)
    {
        foreach (T item in coll)
        {
            System.Console.Write(item.ToString() + " ");
        }
        System.Console.WriteLine();
    }
}
```

メモ :

`ProcessItems` メソッドで項目の追加または削除を行うことができなくても、**IsReadOnly** プロパティは `ProcessItems` 内の `T[]` に false を返します。これは、配列自体が ReadOnly 属性で宣言されていなかったためです。

参照

関連項目

System.Collections.Generic

概念

C# プログラミング ガイド

ジェネリック (C# プログラミング ガイド)

配列 (C# プログラミング ガイド)

その他の技術情報

.NET Framework におけるジェネリック

汎用デリゲート (C# プログラミング ガイド)

delegate (C# リファレンス)は、独自に型パラメータを定義できます。汎用デリゲートを参照するコードでは、型の引数を指定して、クローズ構築型を作成できます。ジェネリック クラスをインスタンス化するときや、ジェネリック メソッドを呼び出すときと同様です。次に例を示します。

C#

```
public delegate void Del<T>(T item);
public static void Notify(int i) { }

Del<int> m1 = new Del<int>(Notify);
```

C# Version 2.0 には、メソッド グループの変換という新機能があります。この機能は、汎用デリゲート型と同様に具象にも適用されるため、前述のコード行を次の簡単な構文で記述できます。

C#

```
Del<int> m2 = Notify;
```

ジェネリック クラス内で定義されるデリゲートでは、クラスのメソッドと同様に、ジェネリック クラスの型パラメータを使用できます。

C#

```
class Stack<T>
{
    T[] items;
    int index;

    public delegate void StackDelegate(T[] items);
}
```

デリゲートを参照するコードでは、次のように、含まれるクラスの型の引数を指定する必要があります。

C#

```
private static void DoWork(float[] items) { }

public static void TestStack()
{
    Stack<float> s = new Stack<float>();
    Stack<float>.StackDelegate d = DoWork;
}
```

汎用デリゲートが特に有効であるのは、標準的なデザイン パターンに基づいてイベントを定義する場合です。送信元の引数は厳密に型指定され、Object との間でキャストを実行する必要がなくなるためです。

C#

```
delegate void StackEventHandler<T, U>(T sender, U eventArgs);

class Stack<T>
{
    public class StackEventArgs : System.EventArgs { }
    public event StackEventHandler<Stack<T>, StackEventArgs> stackEvent;

    protected virtual void OnStackChanged(StackEventArgs a)
    {
        stackEvent(this, a);
    }
}
```

```
class SampleClass
{
    public void HandleStackChange<T>(Stack<T> stack, Stack<T>.StackEventArgs args) { }
}

public static void Test()
{
    Stack<double> s = new Stack<double>();
    SampleClass o = new SampleClass();
    s.stackEvent += o.HandleStackChange;
}
```

参照

関連項目

ジェネリックの概要 (C# プログラミング ガイド)
ジェネリック メソッド (C# プログラミング ガイド)
ジェネリック クラス (C# プログラミング ガイド)
ジェネリック インターフェイス (C# プログラミング ガイド)
System.Collections.Generic

概念

C# プログラミング ガイド
デリゲート (C# プログラミング ガイド)

その他の技術情報

.NET Framework におけるジェネリック

ジェネリック コードの default キーワード (C# プログラミング ガイド)

ジェネリック クラスとジェネリック メソッドでは、あらかじめ以下の情報を把握していない場合に、パラメータ化された型 T に既定値を割り当てる方法が 1 つの問題となります。

- T が参照型か値型か
- T が値型の場合、数値か構造体か

パラメータ化された型 T の変数がある場合、ステートメント t = null は、T が参照型のときにのみ有効です。また、t = 0 は、数値では機能しますが、構造体では機能しません。この問題を解決するには、**default** キーワードを使用します。このキーワードは、参照型の場合には null を返し、数値の値型にはゼロを返します。構造体の場合、ゼロまたは null (値型か参照型かによって変わります) に初期化された構造体の各メンバを返します。`GenericList<T>` クラスで **default** キーワードを使用する方法の例を次に示します。詳細については、「ジェネリックの概要 (C# プログラミング ガイド)」を参照してください。

C#

```
public class GenericList<T>
{
    private class Node
    {
        //...

        public Node Next;
        public T Data;
    }

    private Node head;

    //...

    public T GetNext()
    {
        T temp = default(T);

        Node current = head;
        if (current != null)
        {
            temp = current.Data;
            current = current.Next;
        }
        return temp;
    }
}
```

参照

関連項目

ジェネリック メソッド (C# プログラミング ガイド)

System.Collections.Generic

概念

C# プログラミング ガイド

ジェネリック (C# プログラミング ガイド)

その他の技術情報

.NET Framework におけるジェネリック

C++ テンプレートと C# ジェネリックの違い (C# プログラミング ガイド)

C# ジェネリックも C++ テンプレートもいずれもパラメータ化された型のサポートを提供する言語機能です。ただし、これら 2 つにはさまざまな違いがあります。たとえば、構文レベルでは、C# ジェネリックは、パラメータ化された型により単純にアプローチでき、C++ テンプレートのような複雑さがありません。さらに、C# では、C++ テンプレートで提供されるすべての機能が提供されるわけではありません。また、実装レベルでは、C# ジェネリック型の置換は実行時に行われ、その結果、インスタンス化されたオブジェクトのジェネリック型情報が保存されるという点が主に異なります。詳細については、「ランタイムのジェネリック (C# プログラミング ガイド)」を参照してください。

C# ジェネリックと C++ テンプレートの主な違いを以下に示します。

- C# ジェネリックは、C++ テンプレートほど柔軟ではありません。たとえば、C# ジェネリック クラスでは、ユーザー定義演算子は呼び出すことができますが、算術演算子を呼び出すことはできません。
- C# では、`template C<int i> {}` などの非型テンプレート パラメータを使用できません。
- C# は、明示的な特殊化 (特定の型のテンプレートのカスタム実装) をサポートしません。
- C# は、部分的な特殊化 (型引数のサブセットのカスタム実装) をサポートしません。
- C# では、型パラメータをジェネリック型の基本クラスとして使用できません。
- C# では、型パラメータに既定の型を割り当てることができません。
- C# では、構築された型はジェネリックとして使用できますが、ジェネリック型パラメータ自体はジェネリックにできません。C++ では、テンプレート パラメータを使用できます。
- C++ では、テンプレート内のすべての型パラメータに対して有効でない可能性のあるコードを使用できます。このようなコードは、型パラメータとして使用されている特定の型に対してチェックされます。C# では、制約を満たすすべての型で正常に動作するようにクラスのコードを記述する必要があります。たとえば、C++ では、型パラメータのオブジェクトで算術演算子 + および - を使用し、これらの演算子をサポートしない型を使ってテンプレートをインスタンス化するとエラーを生成する関数を記述できます。C# では、このような関数は許可されません。許可される言語構成要素は、制約から推定できるものだけに限られます。

参照

関連項目

ジェネリックの概要 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Templates

ランタイムのジェネリック (C# プログラミング ガイド)

ジェネリック型またはジェネリック メソッドを Microsoft Intermediate Language (MSIL) にコンパイルすると、型パラメータを持つと識別されるメタデータが組み込まれます。ジェネリック型の MSIL の使用方法は、指定した型パラメータが値型か参照型かによって異なります。

パラメータとして値型を持つジェネリック型を初めて構築すると、指定したパラメータ、または MSIL の適切な位置で置換されたパラメータを持つ、特殊なジェネリック型がランタイムで作成されます。特殊なジェネリック型は、パラメータとして使用される一意の値型ごとに、1 回作成されます。

たとえば、プログラミング コードで、次のように整数で構築されたスタックが宣言されているとします。

C#

```
Stack<int> stack;
```

この時点で、パラメータの代わりに、整数型を持つ特殊なバージョンの Stack<T> クラスがランタイムによって生成されます。これで、プログラミング コードで整数のスタックを使用すると、ランタイムでは、この生成された特殊な **Stack<T>** クラスが必ず再利用されるようになります。次の例では、整数スタックの 2 つのインスタンスを作成し、これらのインスタンスが `Stack<int>` コードの単一のインスタンスを共有します。

C#

```
Stack<int> stackOne = new Stack<int>();
Stack<int> stackTwo = new Stack<int>();
```

ただし、プログラミング コードの別の部分で別の **Stack<T>** クラスが作成されていて、**long** などの異なる値型やユーザー定義の構造体がパラメータとして存在する場合、別バージョンのジェネリック型がランタイムで生成されます。このとき、MSIL で適切な位置にある **long** が置換されます。この特殊なジェネリック クラスにはそれぞれネイティブに値型が含まれるため、変換は必要ありません。

参照型の場合、ジェネリックの機能はやや異なります。ジェネリック型を参照型で初めて構築すると、MSIL のパラメータを置換するオブジェクト参照を持つ、特殊なジェネリック型がランタイムで作成されます。次に、構築された型がパラメータとして参照型でインスタンス化されるたびに、その型の種類にかかわらず、以前に作成した特殊なバージョンのジェネリック型がランタイムで再利用されます。参照はいずれも同じサイズのため、このように処理されます。

たとえば、`Customer` クラスと `Order` クラスという 2 つの参照型があり、さらに `Customer` 型のスタックを作成したとします。

C#

```
class Customer { }
class Order { }
```

C#

```
Stack<Customer> customers;
```

この時点で、データではなく、後で入力されるオブジェクト参照を格納した、特殊なバージョンの **Stack<T>** クラスがランタイムで生成されます。次のコード行で、`Order` という別の参照型のスタックが作成されるとします。

C#

```
Stack<Order> orders = new Stack<Order>();
```

値型とは異なり、`Order` 型用に特殊なバージョンの **Stack<T>** クラスは新たに作成されません。そうではなく、特殊なバージョンの **Stack<T>** クラスのインスタンスが作成され、`orders` 変数はそのインスタンスを参照するように設定されます。このとき、`Customer` 型のスタックを作成するコード行があるとします。

C#

```
customers = new Stack<Customer>();
```

前述した、`Order` 型で作成された **Stack<T>** クラスを使用する場合と同様に、特殊な **Stack<T>** クラスのインスタンスが新たに作成され、インスタンスに含まれるポインタは、`Customer` 型のサイズを示すメモリ領域を参照するように設定されます。参照型の数はプログラムによって大幅に異なるため、C# でジェネリックを実装する場合、コード サイズを抑えるには、参照型のジェネリック クラスのためにコンパイラで作成される特殊なクラスの数を 1 つに減らします。

さらに、ジェネリック C# クラスを型パラメータでインスタンス化する場合、(値型でも参照型でも) 実行時にクエリを実行するにはリフレクションを使用します。また、実際の型と型パラメータを確認できます。

参照

関連項目

ジェネリックの概要 (C# プログラミング ガイド)

System.Collections.Generic

概念

C# プログラミング ガイド

その他の技術情報

.NET Framework におけるジェネリック

.NET Framework クラス ライブラリのジェネリック (C# プログラミング ガイド)

Version 2.0 の .NET Framework クラス ライブラリには、System.Collections.Generic という新しい名前空間があります。この名前空間には、すぐに使用できるジェネリック コレクション クラスと関連インターフェイスが含まれています。System などの他の名前空間も、IComparable<T> のような新しいジェネリック インターフェイスを提供します。これらのクラスやインターフェイスは、.NET Framework の以前のリリースに用意されていた非ジェネリック コレクション クラスよりも処理効率に優れ、タイプ セーフです。独自のカスタム コレクション クラスを設計し実装する前に、基本クラス ライブラリに用意されているクラスを使用できるか、またはいずれかのクラスからクラスを派生できるかを検討してください。

参照

関連項目

ジェネリックの概要 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

ジェネリック コレクションを使用する状況

その他の技術情報

コレクションとデータ構造体

ジェネリックとリフレクション (C# プログラミング ガイド)

共通言語ランタイム (CLR) は、実行時にジェネリック型情報にアクセスできるため、非ジェネリック型の場合と同じように、リフレクションを使用してジェネリック型の情報を取得できます。詳細については、「ランタイムのジェネリック (C# プログラミング ガイド)」を参照してください。

.NET Framework 2.0 では、ジェネリック型のランタイム情報を有効にするために Type クラスに新しいメンバが追加されています。新しいメソッドとプロパティの使い方の詳細については、これらのクラスに関するドキュメントを参照してください。また、System.Reflection.Emit 名前空間にも、ジェネリックをサポートする新しいメンバが追加されています。方法 : リフレクション出力を使用してジェネリック型を定義する を参照してください。

ジェネリック リフレクションで使用する用語に関する一定の条件の一覧については、IsGenericType プロパティの解説を参照してください。

System.Type メンバ名	**説明**
IsGenericType	型がジェネリックの場合、true を返します。
GetGenericArguments	構築された型に対して指定された型引数、またはジェネリック型定義の型パラメータを表す **Type** オブジェクトの配列を返します。
GetGenericTypeDefinition	現在の構築された型の基になっているジェネリック型定義を返します。
GetGenericParameterConstraints	現在のジェネリック型パラメータの制約を表す **Type** オブジェクトの配列を返します。
ContainsGenericParameters	型、またはその型を囲む型またはメソッドのいずれかに、特定の型が指定されていない型パラメータが含まれている場合、true を返します。
GenericParameterAttributes	現在のジェネリック型パラメータの特別な制約を表す **GenericParameterAttributes** フラグの組み合わせを取得します。
GenericParameterPosition	型パラメータを表す **Type** オブジェクトの場合、型パラメータを宣言したジェネリック型定義またはジェネリック メソッド定義の型パラメータ リストで型パラメータの位置を取得します。
IsGenericParameter	現在の **Type** が、ジェネリック型定義またはジェネリック メソッド定義の型パラメータを表しているかどうかを示す値を取得します。
IsGenericTypeDefinition	現在の **Type** が、他のジェネリック型を構築できるジェネリック型定義を表しているかどうかを示す値を取得します。型がジェネリック型定義を表している場合、true を返します。
DeclaringMethod	現在のジェネリック型パラメータを定義したジェネリック メソッドを返します。型パラメータがジェネリック メソッドによって定義されていない場合は null を返します。
MakeGenericType	現在のジェネリック型定義の型パラメータを型の配列要素に置き換え、その結果構築された型を表す **Type** オブジェクトを返します。

さらに、ジェネリック メソッドのランタイム情報を有効にする新しいメンバが MethodInfo クラスに追加されています。ジェネリック メソッドのリフレクションで使用する用語に関する一定の条件の一覧については、IsGenericMethod プロパティの解説を参照してください。

System.Reflection.MemberInfo メンバ名	**説明**
IsGenericMethod	メソッドがジェネリックの場合、true を返します。
GetGenericArguments	構築されたジェネリック メソッドの型引数、またはジェネリック メソッド定義の型パラメータを表す Type オブジェクトの配列を返します。
GetGenericMethodDefinition	現在の構築されたメソッドの基になっているジェネリック メソッド定義を返します。

ContainsGenericParameters	メソッド、またはそれを囲む型のいずれかに、特定の型が指定されていない型パラメータが含まれている場合、true を返します。
IsGenericMethodDefinition	現在の **MethodInfo** がジェネリック メソッド定義を表している場合、true を返します。
MakeGenericMethod	現在のジェネリック メソッド定義の型パラメータを型の配列要素に置き換え、その結果構築されるメソッドを表す **MethodInfo** オブジェクトを返します。

参照

概念

C# プログラミング ガイド

ジェネリック (C# プログラミング ガイド)

リフレクションとジェネリックの概要

その他の技術情報

.NET Framework におけるジェネリック

ジェネリックと属性 (C# プログラミング ガイド)

属性は、非ジェネリック型と同じ方法でジェネリック型に適用できます。属性の適用の詳細については、「属性 (C# プログラミング ガイド)」を参照してください。

カスタム属性は、オープン ジェネリック型のみを参照できます。オープン ジェネリック型は、型引数が与えられていないジェネリック型です。クローズ構築ジェネリック型は、すべての型パラメータに引数が与えられています。

次の例では、このカスタム属性を使用しています。

C#

```
class CustomAttribute : System.Attribute
{
    public System.Object info;
}
```

属性は、次のようにオープン ジェネリック型を参照できます。

C#

```
public class GenericClass1<T> { }

[CustomAttribute(info = typeof(GenericClass1<>))]
class ClassA { }
```

適切な数のコンマを使用して複数の型パラメータを指定します。この例では、`GenericClass2` に 2 つの型パラメータがあります。

C#

```
public class GenericClass2<T, U> { }

[CustomAttribute(info = typeof(GenericClass2<,>))]
class ClassB { }
```

属性は、クローズ構築ジェネリック型を参照できます。

C#

```
public class GenericClass3<T, U, V> { }

[CustomAttribute(info = typeof(GenericClass3<int, double, string>))]
class ClassC { }
```

ジェネリック型のパラメータを参照する属性は、コンパイル時のエラーを発生させます。

C#

```
//[CustomAttribute(info = typeof(GenericClass3<int, T, string>))]  //Error
class ClassD<T> { }
```

ジェネリック型は Attribute から継承できません。

C#

```
//public class CustomAtt<T> : System.Attribute {}  //Error
```

ジェネリック型または型パラメータに関する情報を実行時に取得するには、System.Reflection というメソッドを使用できます。詳細については、「ジェネリックとリフレクション (C# プログラミング ガイド)」を参照してください。

参照

概念

C# プログラミング ガイド

ジェネリック (C# プログラミング ガイド)

属性の概要

ジェネリック型の分散 (C# プログラミング ガイド)

C# にジェネリックが追加されたことによる主な利点の 1 つは、System.Collections.Generic 名前空間の型を使用して、厳密に型指定されたコレクションを容易に作成できることです。たとえば、List<int> 型の変数を作成した場合、コンパイラでこの変数へのすべてのアクセスをチェックできるため、コレクションには **ints** だけを追加できます。この点は、型指定のないコレクションを使用する C# Version 1.0 と比べ、使いやすさが大幅に向上しています。

残念ながら、厳密に型指定されたコレクションにも欠点があります。たとえば、厳密に型指定された List<object> を使用している場合に、List<int> のすべての要素を List<object> に追加する必要が生じたとします。そこで、次のようなコードを記述します。

C#

```
List<int> ints = new List<int>();
ints.Add(1);
ints.Add(10);
ints.Add(42);
List<object> objects = new List<object>();

// doesnt compile ints is not a IEnumerable<object>
//objects.AddRange(ints);
```

この場合、IEnumerable<int> でもある List<int> を IEnumerable<object> として扱う必要があります。**int** は object に変換できるため、この点は特に問題がないように思えます。これは string[] を object[] として扱うことができるのと非常に似ており、この処理は現在は可能です。このような場合に必要となる機能をジェネリックの分散といいます。ジェネリックの分散では、ジェネリック型のインスタンス化 (このケースでは IEnumerable<int>) を同じ型の別のインスタンス化 (このケースでは IEnumerable<object>) として扱います。

C# はジェネリック型の分散をサポートしていないため、このような処理が必要になった場合、いくつかの技法のいずれかを使用して問題を回避する必要があります。最も単純なケース、たとえば上記の例の AddRange のように単一のメソッドを使用する場合は、単純なヘルパー メソッドを宣言して自動的に変換を行うことができます。たとえば、次のようなメソッドを記述できます。

C#

```
// Simple workaround for single method
// Variance in one direction only
public static void Add<S, D>(List<S> source, List<D> destination)
    where S : D
{
    foreach (S sourceElement in source)
    {
        destination.Add(sourceElement);
    }
}
```

このメソッドを使用すると、次の処理が可能になります。

C#

```
// does compile
VarianceWorkaround.Add<int, object>(ints, objects);
```

この例には、単純な分散の回避策の特性がいくつか示されています。このヘルパー メソッドは、変換元と変換先の 2 つの型パラメータを受け取ります。変換元の型パラメータ *S* は、変換先の型パラメータが *D* に制約されています。つまり、読み込み元の List<> には、挿入先の List<> の要素型に変換可能な要素が含まれている必要があります。これによりコンパイラは確実に **int** を object に変換できるようになります。型パラメータを別の型パラメータから強制的に派生させることを、生の型パラメータ制約といいます。

単一のメソッドを定義することによって分散の問題を回避する方法は、ある程度まで有効ですが、残念なことに、分散の問題は急速に複雑化する場合があります。あるインスタンス化のインターフェイスを別のインスタンス化のインターフェイスとして扱おうとした場合に、さらに複雑になります。たとえば、IEnumerable<int> を IEnumerable<object> だけを受け取るメソッドに渡す必要があるような場合です。この場合も、IEnumerable<object> を一連の object、IEnumerable<int> を一連の **ints** であると見なすことができるため、問題はないように思えます。**ints** は object であるため、一連の **ints** は一連の object として扱うことができると推測されます。次に例を示します。

C#

```
static void PrintObjects(IEnumerable<object> objects)
{
    foreach (object o in objects)
    {
        Console.WriteLine(o);
    }
}
```

これを次のように記述すると、処理に失敗します。

C#

```
// would like to do this, but cant ...
// ... ints is not an IEnumerable<object>
//PrintObjects(ints);
```

インターフェイスの代替手段として、このインターフェイスの各メンバについて変換を実行するラッパー オブジェクトを作成します。これは次のようになります。

C#

```
// Workaround for interface
// Variance in one direction only so type expressinos are natural
public static IEnumerable<D> Convert<S, D>(IEnumerable<S> source)
    where S : D
{
    return new EnumerableWrapper<S, D>(source);
}

private class EnumerableWrapper<S, D> : IEnumerable<D>
    where S : D
{
```

これを使用すると、次の処理が可能になります。

C#

```
PrintObjects(VarianceWorkaround.Convert<int, object>(ints));
```

この場合も、ラッパー クラスとヘルパー メソッドにおける生の型パラメータ制約に注目してください。この処理はかなり複雑なものになりますが、ラッパー クラスのコードは非常に単純です。つまり、ラップされたインターフェイスのメンバに処理を渡すだけで、単純な型変換以外には何も行いません。コンパイラで直接 `IEnumerable<int>` から `IEnumerable<object>` への変換を行わない理由はどこにあるのでしょうか。

分散は、コレクションの読み取り操作だけを行う場合はタイプ セーフですが、読み取り操作と書き込み操作の両方を伴う場合はタイプ セーフではなくなります。たとえば、`IList<>` インターフェイスは自動的に処理できない場合があります。タイプ セーフになるように、`IList<>` のすべての読み取り操作をラップするヘルパーを記述することは可能ですが、読み取り操作はそれほど簡単にはラップできません。

`IList<T>` インターフェイスの分散を処理するためのラッパーの一部を次に示します。読み取りと書き込みの双方向において分散に関係する問題が生じることがわかります。

C#

```
private class ListWrapper<S, D> : CollectionWrapper<S, D>, IList<D>
    where S : D
{
    public ListWrapper(IList<S> source) : base(source)
    {
        this.source = source;
    }

    public int IndexOf(D item)
    {
        if (item is S)
        {
            return this.source.IndexOf((S) item);
        }
        else
        {
```

```
            return -1;
        }
    }

    // variance the wrong way ...
    // ... can throw exceptions at runtime
    public void Insert(int index, D item)
    {
        if (item is S)
        {
            this.source.Insert(index, (S)item);
        }
        else
        {
            throw new Exception("Invalid type exception");
        }
    }
```

ラッパーの `Insert` メソッドに問題があります。このメソッドは引数として `D` を受け取りますが、これを `IList<S>` に挿入する必要があります。`D` は `S` の基本型であるため、すべての `D` が `S` であるとは限りません。したがって、`Insert` 操作は失敗します。この例は配列の分散と似ています。オブジェクトを `object[]` に挿入する場合、`object[]` が実際には `string[]` である可能性が実行時にはあるため、動的な型チェックが実行されます。次に例を示します。

C#

```
object[] objects = new string[10];

// no problem, adding a string to a string[]
objects[0] = "hello";

// runtime exception, adding an object to a string[]
objects[1] = new object();
```

`IList<>` の例では、実行時に実際の型が目的の型と一致しない場合は単に `Insert` メソッドのラッパーをスローできます。ここでも、プログラマに代わってコンパイラが自動的にラッパーを生成すると考えることはできます。しかし、このようなポリシーではうまくいかないケースが存在します。`IndexOf` メソッドは指定した項目をコレクションで検索し、項目が見つかった場合はコレクションのインデックスを返します。ただし、項目が見つからなかった場合、`IndexOf` メソッドは `-1` を返すだけで、ラッパーはスローされません。この種のラップは、自動的に生成されるラッパーでは提供されません。

ここまでは、ジェネリックの分散に関する問題の回避策として `2` つの単純な方法を説明しました。しかし、分散の問題はいっそう複雑化する可能性があります。たとえば、`List<IEnumerable<int>>` を `List<IEnumerable<object>>` として扱ったり、`List<IEnumerable<IEnumerable<int>>>` を `List<IEnumerable<IEnumerable<object>>>` として扱ったりする場合などです。

分散の問題を回避するためにコードでこのようなラッパーを生成すると、コードに著しいオーバーヘッドが生じることがあります。また、参照 `ID` に関連した問題が生じることもあります。これは、各ラッパーが元のコレクションと同じ `ID` を持たないためで、軽度のバグが発生する可能性があります。ジェネリックを使用する場合は、密に結合されたコンポーネント間での不一致が少なくなるように、型のインスタンス化を選択する必要があります。そのためには、コードのデザインにおいてある程度の妥協が必要になることもあります。当然のことながら、デザインには競合する要件の間でのトレードオフが付き物であり、デザイン プロセスでは当該言語における型システムの制約を考慮に入れる必要があります。

その言語の中核の部分としてジェネリックの分散が含まれるような型システムも存在します。`Eiffel` などはその主な例です。ただし、型システムの中核の部分としてジェネリックの分散が含まれた場合、`C#` の型システムは、分散を伴わない比較的単純なシナリオにおいてさえ著しく複雑になってしまいます。その結果、`C#` の設計では、分散を含めないことが正しい選択であると考えられました。

上記の例の完全なソース コードを次に示します。

C#

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;

static class VarianceWorkaround
{
    // Simple workaround for single method
    // Variance in one direction only
    public static void Add<S, D>(List<S> source, List<D> destination)
        where S : D
    {
        foreach (S sourceElement in source)
        {
            destination.Add(sourceElement);
        }
```

```
    }

    // Workaround for interface
    // Variance in one direction only so type expressinos are natural
    public static IEnumerable<D> Convert<S, D>(IEnumerable<S> source)
        where S : D
    {
        return new EnumerableWrapper<S, D>(source);
    }

    private class EnumerableWrapper<S, D> : IEnumerable<D>
        where S : D
    {
        public EnumerableWrapper(IEnumerable<S> source)
        {
            this.source = source;
        }

        public IEnumerator<D> GetEnumerator()
        {
            return new EnumeratorWrapper(this.source.GetEnumerator());
        }

        IEnumerator System.Collections.IEnumerable.GetEnumerator()
        {
            return this.GetEnumerator();
        }

        private class EnumeratorWrapper : IEnumerator<D>
        {
            public EnumeratorWrapper(IEnumerator<S> source)
            {
                this.source = source;
            }

            private IEnumerator<S> source;

            public D Current
            {
                get { return this.source.Current; }
            }

            public void Dispose()
            {
                this.source.Dispose();
            }

            object IEnumerator.Current
            {
                get { return this.source.Current; }
            }

            public bool MoveNext()
            {
                return this.source.MoveNext();
            }

            public void Reset()
            {
                this.source.Reset();
            }
        }

        private IEnumerable<S> source;
    }

    // Workaround for interface
    // Variance in both directions, causes issues
```

```
    // similar to existing array variance
    public static ICollection<D> Convert<S, D>(ICollection<S> source)
        where S : D
    {
        return new CollectionWrapper<S, D>(source);
    }


    private class CollectionWrapper<S, D>
        : EnumerableWrapper<S, D>, ICollection<D>
        where S : D
    {
        public CollectionWrapper(ICollection<S> source)
            : base(source)
        {
        }

        // variance going the wrong way ...
        // ... can yield exceptions at runtime
        public void Add(D item)
        {
            if (item is S)
            {
                this.source.Add((S)item);
            }
            else
            {
                throw new Exception(@"Type mismatch exception, due to type hole introduced
by variance.");
            }
        }

        public void Clear()
        {
            this.source.Clear();
        }

        // variance going the wrong way ...
        // ... but the semantics of the method yields reasonable semantics
        public bool Contains(D item)
        {
            if (item is S)
            {
                return this.source.Contains((S)item);
            }
            else
            {
                return false;
            }
        }

        // variance going the right way ...
        public void CopyTo(D[] array, int arrayIndex)
        {
            foreach (S src in this.source)
            {
                array[arrayIndex++] = src;
            }
        }

        public int Count
        {
            get { return this.source.Count; }
        }

        public bool IsReadOnly
        {
            get { return this.source.IsReadOnly; }
```

```
        }

        // variance going the wrong way ...
        // ... but the semantics of the method yields reasonable  semantics
        public bool Remove(D item)
        {
            if (item is S)
            {
                return this.source.Remove((S)item);
            }
            else
            {
                return false;
            }
        }

        private ICollection<S> source;
    }

    // Workaround for interface
    // Variance in both directions, causes issues similar to existing array variance
    public static IList<D> Convert<S, D>(IList<S> source)
        where S : D
    {
        return new ListWrapper<S, D>(source);
    }

    private class ListWrapper<S, D> : CollectionWrapper<S, D>, IList<D>
        where S : D
    {
        public ListWrapper(IList<S> source) : base(source)
        {
            this.source = source;
        }

        public int IndexOf(D item)
        {
            if (item is S)
            {
                return this.source.IndexOf((S) item);
            }
            else
            {
                return -1;
            }
        }

        // variance the wrong way ...
        // ... can throw exceptions at runtime
        public void Insert(int index, D item)
        {
            if (item is S)
            {
                this.source.Insert(index, (S)item);
            }
            else
            {
                throw new Exception("Invalid type exception");
            }
        }

        public void RemoveAt(int index)
        {
            this.source.RemoveAt(index);
        }

        public D this[int index]
        {
```

```
            get
            {
                return this.source[index];
            }
            set
            {
                if (value is S)
                    this.source[index] = (S)value;
                else
                    throw new Exception("Invalid type exception.");
            }
        }

        private IList<S> source;
    }
}

namespace GenericVariance
{
    class Program
    {
        static void PrintObjects(IEnumerable<object> objects)
        {
            foreach (object o in objects)
            {
                Console.WriteLine(o);
            }
        }

        static void AddToObjects(IList<object> objects)
        {
            // this will fail if the collection provided is a wrapped collection
            objects.Add(new object());
        }
        static void Main(string[] args)
        {
            List<int> ints = new List<int>();
            ints.Add(1);
            ints.Add(10);
            ints.Add(42);
            List<object> objects = new List<object>();

            // doesnt compile ints is not a IEnumerable<object>
            //objects.AddRange(ints);

            // does compile
            VarianceWorkaround.Add<int, object>(ints, objects);

            // would like to do this, but cant ...
            // ... ints is not an IEnumerable<object>
            //PrintObjects(ints);

            PrintObjects(VarianceWorkaround.Convert<int, object>(ints));

            AddToObjects(objects); // this works fine
            AddToObjects(VarianceWorkaround.Convert<int, object>(ints));
        }
        static void ArrayExample()
        {
            object[] objects = new string[10];

            // no problem, adding a string to a string[]
            objects[0] = "hello";

            // runtime exception, adding an object to a string[]
            objects[1] = new object();
        }
    }
```

```
    }
```

参照

概念

C# プログラミング ガイド

ジェネリック (C# プログラミング ガイド)

反復子 (C# プログラミング ガイド)

反復子は、C# 2.0 の新機能です。反復子は、クラスまたは構造体で foreach 反復処理のサポートを可能にするメソッド、get アクセサ、または演算子であり、これにより IEnumerable インターフェイス全体を実装する必要はありません。この実装の代わりに、反復子を提供するだけで、反復子がクラス内のデータ構造を走査します。コンパイラは、反復子を検出すると、**IEnumerable** インターフェイスまたは IEnumerable<T> インターフェイスの `Current` メソッド、`MoveNext` メソッド、および `Dispose` メソッドを自動的に生成します。

反復子の概要

- 反復子は、順序付けされた同じ型の一連の値を返すコードのセクションです。
- 反復子は、メソッド、演算子、または **get** アクセサの本体として使用できます。
- 反復子コードでは、**yield return** ステートメントを使用して、各要素を順に返します。反復処理は、**yield break** で終了します。詳細については、「yield (C# リファレンス)」を参照してください。
- 複数の反復子を 1 つのクラスに実装できます。各反復子は、他のクラス メンバと同様に一意の名前を持つ必要があり、"`foreach(int x in SampleClass.Iterator2){}`" のように、**foreach** ステートメント内でクライアント コードによって呼び出すことができます。
- 反復子の戻り値の型は、**IEnumerable**、IEnumerator、**IEnumerable**<T>、または IEnumerator<T> である必要があります。

yield キーワードを使用して、単数または複数の戻り値を指定します。**yield return** ステートメントに到達すると、現在の位置が保存されます。次回、反復子が呼び出されると、この位置から実行が再開されます。

反復子は、特にコレクション クラスで役立ち、バイナリ ツリーなどの重要なデータ構造体を簡単に反復処理できるようにします。

関連項目

詳細情報

- 反復子の使用 (C# プログラミング ガイド)
- 方法 : 整数リストの反復子ブロックを作成する (C# プログラミング ガイド)
- 方法 : ジェネリック リストの反復子ブロックを作成する (C# プログラミング ガイド)
- ジェネリック インターフェイス (C# プログラミング ガイド)

例

次の例の `DaysOfTheWeek` クラスは、曜日を文字列として格納する単純なコレクション クラスです。**foreach** ループを反復処理するたびに、コレクション内の次の文字列が返されます。

C#

```
public class DaysOfTheWeek : System.Collections.IEnumerable
{
    string[] m_Days = { "Sun", "Mon", "Tue", "Wed", "Thr", "Fri", "Sat" };

    public System.Collections.IEnumerator GetEnumerator()
    {
        for (int i = 0; i < m_Days.Length; i++)
        {
            yield return m_Days[i];
        }
    }
}

class TestDaysOfTheWeek
{
    static void Main()
    {
        // Create an instance of the collection class
        DaysOfTheWeek week = new DaysOfTheWeek();

        // Iterate with foreach
        foreach (string day in week)
```

```
            {
                System.Console.Write(day + " ");
            }
        }
    }
```

出力

```
Sun Mon Tue Wed Thr Fri Sat
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 22 反復子

参照

概念

C# プログラミング ガイド

ジェネリック (C# プログラミング ガイド)

反復子の使用 (C# プログラミング ガイド)

反復子を作成する場合、次のように `IEnumerable` インターフェイスで `GetEnumerator` メソッドを実装するのが最も一般的な方法です。

C#

```
public System.Collections.IEnumerator GetEnumerator()
{
    for (int i = 0; i < max; i++)
    {
        yield return i;
    }
}
```

`GetEnumerator` メソッドが存在するため、型は列挙型になり、foreach ステートメントを使用できます。上記のメソッドが `ListClass` のクラス定義の一部である場合、次のようにクラスで **foreach** を使用できます。

C#

```
static void Main()
{
    ListClass listClass1 = new ListClass();

    foreach (int i in listClass1)
    {
        System.Console.WriteLine(i);
    }
}
```

foreach ステートメントは `ListClass.GetEnumerator()` を呼び出し、返された列挙子を使用して、値を反復処理します。IEnumerator インターフェイスを返すジェネリック反復子を作成する方法の例については、
「方法 : ジェネリック リストの反復子ブロックを作成する (C# プログラミング ガイド)」を参照してください。

同じデータ コレクションを反復処理する別の方法をサポートするために、名前付き反復子を使用することもできます。たとえば、要素を昇順で返す反復子と、降順で返す反復子を作成できます。また、反復動作の全部または一部をクライアントが制御できるようにするパラメータを反復子に設定することもできます。次の反復子は、名前付き反復子 `SampleIterator` を使用して `IEnumerable` インターフェイスを実装します。

C#

```
// Implementing the enumerable pattern
public System.Collections.IEnumerable SampleIterator(int start, int end)
{
    for (int i = start; i <= end; i++)
    {
        yield return i;
    }
}
```

名前付き反復子は、次のように呼び出します。

C#

```
ListClass test = new ListClass();
foreach (int n in test.SampleIterator(1, 10))
{
    System.Console.WriteLine(n);
}
```

次のように、同じ反復子で複数の yield ステートメントを使用できます。

C#

```
public System.Collections.IEnumerator GetEnumerator()
{
    yield return "With an iterator, ";
    yield return "more than one ";
    yield return "value can be returned";
    yield return ".";
}
```

結果を出力するには、次の **foreach** ステートメントを使用します。

C#

```
foreach (string element in new TestClass())
{
    System.Console.Write(element);
}
```

この例を実行すると、次のテキストが表示されます。

```
With an iterator, more than one value can be returned.
```

foreach ループを反復するごとに (または `IEnumerator.MoveNext` を直接呼び出すと)、前の **yield** ステートメントの後で次の反復子コード本体が再開され、反復子本体の最後に到達するか、または **yield break** ステートメントに達するまで、このプロセスが継続されます。

参照

処理手順

方法 : 整数リストの反復子ブロックを作成する (C# プログラミング ガイド)

方法 : ジェネリック リストの反復子ブロックを作成する (C# プログラミング ガイド)

関連項目

yield (C# リファレンス)

配列での foreach の使用 (C# プログラミング ガイド)

foreach、in (C# リファレンス)

概念

C# プログラミング ガイド

方法：整数リストの反復子ブロックを作成する (C# プログラミング ガイド)

この例では、整数の配列を使用して `SampleCollection` リストを構築します。for ループは、コレクションを反復処理して各項目の値を生成します。次に、**foreach** ループを使用してコレクションの項目を表示します。

使用例

C#

```
// Declare the collection:
public class SampleCollection
{
    public int[] items;

    public SampleCollection()
    {
        items = new int[5] { 5, 4, 7, 9, 3 };
    }

    public System.Collections.IEnumerable BuildCollection()
    {
        for (int i = 0; i < items.Length; i++)
        {
            yield return items[i];
        }
    }
}

class MainClass
{
    static void Main()
    {
        SampleCollection col = new SampleCollection();

        // Display the collection items:
        System.Console.WriteLine("Values in the collection are:");
        foreach (int i in col.BuildCollection())
        {
            System.Console.Write(i + " ");
        }
    }
}
```

出力

```
Values in the collection are:
5 4 7 9 3
```

参照

処理手順

方法 : ジェネリック リストの反復子ブロックを作成する (C# プログラミング ガイド)

関連項目

反復子の使用 (C# プログラミング ガイド)

yield (C# リファレンス)

Array

概念

C# プログラミング ガイド

反復子 (C# プログラミング ガイド)

方法：ジェネリック リストの反復子ブロックを作成する (C# プログラミング ガイド)

この例では、`Stack<T>` ジェネリック クラスが IEnumerator<T> ジェネリック インターフェイスを実装します。`Push` メソッドを使って、T 型の配列が宣言され、値が割り当てられます。`GetEnumerator` メソッドでは、配列の値が **yield return** ステートメントを使用して戻されます。

`IEnumerable<T>` は `IEnumerable` を継承するため、非ジェネリックの GetEnumerator も実装されます。この例は、非ジェネリック メソッドが呼び出しをジェネリック メソッドに単に転送する一般的な実装を示しています。

使用例

C#

```
using System.Collections;
using System.Collections.Generic;

namespace GenericIteratorExample
{
    public class Stack<T> : IEnumerable<T>
    {
        private T[] values = new T[100];
        private int top = 0;

        public void Push(T t) { values[top++] = t; }
        public T Pop() { return values[--top]; }

        // These make Stack<T> implement IEnumerable<T> allowing
        // a stack to be used in a foreach statement.
        public IEnumerator<T> GetEnumerator()
        {
            for (int i = top; --i >= 0; )
            {
                yield return values[i];
            }
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }

        // Iterate from top to bottom.
        public IEnumerable<T> TopToBottom
        {
            get
            {
                // Since we implement IEnumerable<T>
                // and the default iteration is top to bottom,
                // just return the object.
                return this;
            }
        }

        // Iterate from bottom to top.
        public IEnumerable<T> BottomToTop
        {
            get
            {
                for (int i = 0; i < top; i++)
                {
                    yield return values[i];
                }
            }
        }

        //A parameterized iterator that return n items from the top
```

```
        public IEnumerable<T> TopN(int n)
        {
            // in this example we return less than N if necessary
            int j = n >= top ? 0 : top - n;

            for (int i = top; --i >= j; )
            {
                yield return values[i];
            }
        }
    }

    //This code uses a stack and the TopToBottom and BottomToTop properties
    //to enumerate the elements of the stack.
    class Test
    {
        static void Main()
        {
            Stack<int> s = new Stack<int>();
            for (int i = 0; i < 10; i++)
            {
                s.Push(i);
            }

            // Prints: 9 8 7 6 5 4 3 2 1 0
            // Foreach legal since s implements IEnumerable<int>
            foreach (int n in s)
            {
                System.Console.Write("{0} ", n);
            }
            System.Console.WriteLine();

            // Prints: 9 8 7 6 5 4 3 2 1 0
            // Foreach legal since s.TopToBottom returns IEnumerable<int>
            foreach (int n in s.TopToBottom)
            {
                System.Console.Write("{0} ", n);
            }
            System.Console.WriteLine();

            // Prints: 0 1 2 3 4 5 6 7 8 9
            // Foreach legal since s.BottomToTop returns IEnumerable<int>
            foreach (int n in s.BottomToTop)
            {
                System.Console.Write("{0} ", n);
            }
            System.Console.WriteLine();

            // Prints: 9 8 7 6 5 4 3
            // Foreach legal since s.TopN returns IEnumerable<int>
            foreach (int n in s.TopN(7))
            {
                System.Console.Write("{0} ", n);
            }
            System.Console.WriteLine();
        }
    }
}
```

出力

```
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3
```

参照

処理手順
方法 : 整数リストの反復子ブロックを作成する (C# プログラミング ガイド)
関連項目
反復子の使用 (C# プログラミング ガイド)
System.Collections.Generic
IEnumerable
概念
C# プログラミング ガイド
反復子 (C# プログラミング ガイド)

名前空間 (C# プログラミング ガイド)

C# プログラミングでは、名前空間が 2 つの点で盛んに使用されます。1 つは、.NET Framework では、次のように名前空間を使用して多くのクラスを編成します。

C#

```
System.Console.WriteLine("Hello World!");
```

System が名前空間で、**Console** が、この名前空間に含まれるクラスです。**using** キーワードを使用すると、次のように完全な名前を記述する必要がなくなります。

C#

```
using System;
```

C#

```
Console.WriteLine("Hello");
Console.WriteLine("World!");
```

詳細については、「using ディレクティブ (C# リファレンス)」を参照してください。

もう 1 つは、独自の名前空間を宣言すると、大型のプログラミング プロジェクトでクラス名とメソッド名のスコープの管理が容易になります。次の例に示すように、namespace キーワードを使用して名前空間を宣言します。

C#

```
namespace SampleNamespace
{
    class SampleClass
    {
        public void SampleMethod()
        {
            System.Console.WriteLine(
              "SampleMethod inside SampleNamespace");
        }
    }
}
```

名前空間の概要

名前空間には、次の特徴があります。

- 名前空間を使用することにより、大型のコード プロジェクトを整理できます。
- 名前空間は、**.** 演算子で区切ります。
- **using directive** を使用すると、クラスごとに名前空間の名前を指定する必要がなくなります。
- **global** 名前空間は "ルート" 名前空間です。**global::system** は、常に .NET Framework 名前空間の **System** を参照します。

関連項目

名前空間の詳細については、以下のトピックを参照してください。

- 名前空間の使用 (C# プログラミング ガイド)
- 方法 : 名前空間エイリアス修飾子を使用する (C# プログラミング ガイド)
- 方法 : My 名前空間を使用する (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 9 名前空間

参照

関連項目

名前空間キーワード (C# リファレンス)
using ディレクティブ (C# リファレンス)
:: 演算子 (C# リファレンス)
. 演算子 (C# リファレンス)

概念

C# プログラミング ガイド

名前空間の使用 (C# プログラミング ガイド)

C# プログラムでは、名前空間が 2 つの点で頻繁に使用されます。1 つは、.NET Framework クラスが多数のクラスを編成するために名前空間を使用するからであり、もう 1 つは、固有の名前空間を宣言すると、大型のプログラミング プロジェクトでクラス名とメソッド名のスコープを管理するのに役立つからです。

名前空間へのアクセス

C# アプリケーションは、一般に **using** ディレクティブのセクションから始まります。このセクションには、アプリケーションが頻繁に使用する名前空間が一覧表示され、包含されているメソッドを使用するたびに完全修飾名を指定しなくても済むようにします。

たとえば、次の行を記述したとします。

C#

```
using System;
```

この場合、プログラムの開始位置で、次のコードを使用できます。

C#

```
Console.WriteLine("Hello, World!");
```

このコードの本来の書式は次のとおりです。

C#

```
System.Console.WriteLine("Hello, World!");
```

名前空間エイリアス

「using ディレクティブ (C# リファレンス)」の内容に従って名前空間のエイリアスを作成することもできます。たとえば、入れ子になった名前空間を含む、以前に作成した名前空間を使用する場合は、次のようにエイリアスを宣言して、特定の名前空間を簡単に参照することもできます。

C#

```
using Co = Company.Proj.Nested;  // define an alias to represent a namespace
```

名前空間によるスコープの制御

namespace キーワードは、スコープの宣言に使用します。プロジェクト内でスコープを作成すると、コードの編成が容易になり、グローバルに一意の型を作成できます。次の例では、入れ子関係にある 2 つの名前空間で `SampleClass` というクラスを定義します。
「. 演算子 (C# リファレンス)」の内容に従って、呼び出されるメソッドを区別します。

C#

```
namespace SampleNamespace
{
    class SampleClass
    {
        public void SampleMethod()
        {
            System.Console.WriteLine(
              "SampleMethod inside SampleNamespace");
        }
    }

    // Create a nested namespace, and define another class.
    namespace NestedNamespace
    {
        class SampleClass
```

```
        {
            public void SampleMethod()
            {
                System.Console.WriteLine(
                  "SampleMethod inside NestedNamespace");
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Displays "SampleMethod inside SampleNamespace."
            SampleClass outer = new SampleClass();
            outer.SampleMethod();

            // Displays "SampleMethod inside SampleNamespace."
            SampleNamespace.SampleClass outer2 = new SampleNamespace.SampleClass();
            outer2.SampleMethod();

            // Displays "SampleMethod inside NestedNamespace."
            NestedNamespace.SampleClass inner = new NestedNamespace.SampleClass();
            inner.SampleMethod();
        }
    }
}
```

完全修飾名

名前空間と型には、論理階層を示す完全修飾名で表された一意のタイトルが割り当てられています。たとえば、ステートメント `A.B` は、`A` が名前空間または型の名前であり、`B` が A の入れ子であることを意味します。

入れ子にされたクラスと名前空間を次の例に示します。完全修飾名は、各エンティティの後のコメントとして示されています。

C#

```
namespace N1     // N1
{
    class C1      // N1.C1
    {
        class C2   // N1.C1.C2
        {
        }
    }
    namespace N2  // N1.N2
    {
        class C2   // N1.N2.C2
        {
        }
    }
}
```

前のコードは、次のような関係になっています。

- 名前空間 `N1` は、グローバル名前空間のメンバです。この完全修飾名は `N1` です。
- 名前空間 `N2` は、`N1` のメンバです。この完全修飾名は `N1.N2` です。
- クラス `C1` は `N1` のメンバです。この完全修飾名は `N1.C1` です。
- クラス名 `C2` は、このコードで 2 回使われています。ただし、完全修飾名は異なります。最初の名前は `C1` の内部で宣言されているので、完全修飾名は `N1.C1.C2` です。2 番目の名前は `N2` の内部で宣言されているので、完全修飾名は `N1.N2.C2` です。

上のコード セグメントを使用して、次のように新しいクラス メンバ `C3` を名前空間 `N1.N2` に追加できます。

C#

```
namespace N1.N2
{
    class C3   // N1.N2.C3
    {
    }
}
```

通常、:: は、名前空間エイリアスを参照する際に使用し、**global::** は、グローバル名前空間を参照する際に使用します。. は、型またはメンバを修飾する際に使用します。

名前空間ではなく型を参照するエイリアスで :: を使用するのは誤りです。次に例を示します。

C#

```
using Alias = System.Console;
```

C#

```
class TestClass
{
    static void Main()
    {
        // Error
        //Alias::WriteLine("Hi");

        // OK
        Alias.WriteLine("Hi");
    }
}
```

global という語は定義済みのエイリアスではないので、`global.X` には特別な意味がないことに注意してください。この語は、:: と共に使用したときにのみ特別な意味が与えられます。

global:: は常にグローバル名前空間を参照し、エイリアスを参照しないので、global という名前のエイリアスを定義すると、警告 (「コンパイラの警告 (レベル 2) CS0440」を参照) が生成されます。たとえば、次の行では警告が生成されます。

C#

```
using global = System.Collections;   // Warning
```

エイリアスで :: を使用すると、追加の型が予想外に導入されることを防ぐことができます。次に例を示します。

C#

```
using Alias = System;
```

C#

```
namespace Library
{
    public class C : Alias.Exception { }
}
```

このコードは動作しますが、`Alias` という型が後で導入されると、`Alias.` は代わりにその型にバインドされます。`Alias::Exception` を使用すると、`Alias` は名前空間エイリアスとして扱われ、型と間違われることがなくなります。

global エイリアスの詳細については、「方法 : 名前空間エイリアス修飾子を使用する (C# プログラミング ガイド)」を参照してください。

参照

関連項目

名前空間キーワード (C# リファレンス)
. 演算子 (C# リファレンス)
:: 演算子 (C# リファレンス)
extern (C# リファレンス)

概念

C# プログラミング ガイド
名前空間 (C# プログラミング ガイド)

方法：名前空間エイリアス修飾子を使用する (C# プログラミング ガイド)

グローバル名前空間のメンバにアクセスできると、そのメンバが同名の別のエンティティによって隠される可能性がある場合に役立ちます。

たとえば、次のコードでは、`Console` は、System 名前空間の **Console** 型ではなく、`TestApp.Console` に解決されます。

C#

```
using System;
```

C#

```
class TestApp
{
    // Define a new class called 'System' to cause problems.
    public class System { }

    // Define a constant called 'Console' to cause more problems.
    const int Console = 7;
    const int number = 66;

    static void Main()
    {
        // Error  Accesses TestApp.Console
        //Console.WriteLine(number);
    }
}
```

System.Console を使用すると、**System** 名前空間が `TestApp.System` クラスによって隠されているため、依然としてエラーになります。

C#

```
// Error  Accesses TestApp.System
System.Console.WriteLine(number);
```

ただし、次のように `global::System.Console` を使用すると、このエラーを回避できます。

C#

```
// OK
global::System.Console.WriteLine(number);
```

左の識別子が **global** の場合、右の識別子の検索はグローバル名前空間から開始されます。たとえば、次の宣言は、グローバル空間のメンバとして `TestApp` を参照します。

C#

```
class TestClass : global::TestApp
```

`System` という名前の独自の名前空間を作成することはお勧めしません。そのようなコードに遭遇することはほとんどありません。ただし、大型のプロジェクトでは、フォームによっては名前空間の重複が実際に発生する可能性があります。そのような場合は、グローバル名前空間修飾子によって、ルート名前空間を指定できます。

使用例

次の例では、**System** 名前空間を使用して `TestClass` クラスを格納しています。そのため、**System** 名前空間によって隠されている **System.Console** クラスを参照するために `global::System.Console` を使用する必要があります。また、**System.Collections** 名前空間を参照するのに `colAlias` エイリアスを使用するため、名前空間の代わりにこのエイリアスを使用して System.Collections.Hashtable のインスタンスを作成しています。

C#

```
using colAlias = System.Collections;
namespace System
{
    class TestClass
    {
        static void Main()
        {
            // Searching the alias:
            colAlias::Hashtable test = new colAlias::Hashtable();

            // Add items to the table.
            test.Add("A", "1");
            test.Add("B", "2");
            test.Add("C", "3");

            foreach (string name in test.Keys)
            {
                // Seaching the gloabal namespace:
                global::System.Console.WriteLine(name + " " + test[name]);
            }
        }
    }
}
```

サンプル出力

```
A 1
B 2
C 3
```

参照

関連項目

. 演算子 (C# リファレンス)
:: 演算子 (C# リファレンス)
extern (C# リファレンス)

概念

C# プログラミング ガイド
名前空間 (C# プログラミング ガイド)

方法：My 名前空間を使用する (C# プログラミング ガイド)

Microsoft.VisualBasic.MyServices 名前空間 (Visual Basic では **My**) を使用すると、いくつもの .NET Framework クラスに簡単かつ直感的にアクセスできるため、コンピュータ、アプリケーション、設定、リソースなどと対話するコードを記述できます。**MyServices** 名前空間は、もともとは Visual Basic で使用するものとして設計されましたが、C# アプリケーションでも使用できます。

Visual Basic で **MyServices** 名前空間を使用する方法の詳細については、「My による開発」を参照してください。

参照の追加

MyServices クラスをソリューションで使用する前に、Visual Basic ライブラリへの参照を追加する必要があります。

Visual Basic ライブラリへの参照を追加するには

1. ソリューション エクスプローラで、[参照設定] ノードを右クリックし、[参照の追加] をクリックします。
2. [参照の追加] ダイアログ ボックスが表示されたら、一覧を下にスクロールし、Microsoft.VisualBasic.dll を選択します。

   プログラムの先頭の **using** セクションに次の行を追加することもできます。

   C#

```
using Microsoft.VisualBasic.Devices;
```

使用例

次の例では、**MyServices** 名前空間に含まれているさまざまな静的メソッドを呼び出します。このコードをコンパイルするには、Microsoft.VisualBasic.DLL への参照をプロジェクトに追加する必要があります。

C#

```
using System;
using Microsoft.VisualBasic.Devices;

class TestMyServices
{
    static void Main()
    {
        // Play a sound with the Audio class:
        Audio myAudio = new Audio();
        Console.WriteLine("Playing sound...");
        myAudio.Play(@"c:\WINDOWS\Media\chimes.wav");

        // Display time information with the Clock class:
        Clock myClock = new Clock();
        Console.Write("Current day of the week: ");
        Console.WriteLine(myClock.LocalTime.DayOfWeek);
        Console.Write("Current date and time: ");
        Console.WriteLine(myClock.LocalTime);

        // Display machine information with the Computer class:
        Computer myComputer = new Computer();
        Console.WriteLine("Computer name: " + myComputer.Name);

        if (myComputer.Network.IsAvailable)
        {
            Console.WriteLine("Computer is connected to network.");
        }
        else
        {
            Console.WriteLine("Computer is not connected to network.");
        }
    }
}
```

MyServices 名前空間のクラスの中には C# アプリケーションから呼び出すことができないクラスもあります。たとえば、FileSystemProxy クラスは、C# と互換性がありません。そのような場合は、同様に VisualBasic.dll に含まれている FileSystem を構成する静的メソッドを代わりに使用できます。このようなメソッドを使用してディレクトリを複製する方法を次に示します。

C#

```
// Duplicate a directory
Microsoft.VisualBasic.FileIO.FileSystem.CopyDirectory(
    @"C:\original_directory",
    @"C:\copy_of_original_directory");
```

参照

概念

C# プログラミング ガイド

名前空間 (C# プログラミング ガイド)

名前空間の使用 (C# プログラミング ガイド)

null 許容型 (C# プログラミング ガイド)

null 許容型は、System.Nullable 構造体のインスタンスです。null 許容型は、基礎となる値型の通常範囲の値だけでなく、**null** 値も表すことができます。たとえば、Nullable<Int32> ("Int32 の Null 許容" と読みます) には、-2147483648 から 2147483647 の任意の値、または **null** 値を割り当てることができます。Nullable<bool> には、**true**、**false**、または **null** の値を割り当てることができます。数値型と Boolean 型に **null** を割り当てる機能が便利なのは、値を割り当てることができない要素を含むデータベースや他のデータ型を処理するときです。たとえば、データベースの Boolean フィールドには、値 **true** または **false** を格納するか、未定義にすることが可能です。

C#

```
class NullableExample
{
    static void Main()
    {
        int? num = null;
        if (num.HasValue == true)
        {
            System.Console.WriteLine("num = " + num.Value);
        }
        else
        {
            System.Console.WriteLine("num = Null");
        }

        //y is set to zero
        int y = num.GetValueOrDefault();

        // num.Value throws an InvalidOperationException if num.HasValue is false
        try
        {
            y = num.Value;
        }
        catch (System.InvalidOperationException e)
        {
            System.Console.WriteLine(e.Message);
        }
    }
}
```

上のコードで以下の出力が表示されます。

```
num = Null

Nullable object must have a value.
```

null 許容型の概要

null 許容型には次の特性があります。

- null 許容型は、**null** の値を割り当てることができる、値型の変数を表します。参照型に基づいた null 許容型は作成できません (参照型では **null** 値が既にサポートされています)。
- 構文 **T?** は、System.Nullable<T> の省略表現です。ここで、**T** は値型です。この 2 つの形式はどちらでも使用できます。
- null 許容型に値を割り当てる方法は、`int? x = 10;` や `double? d = 4.108;` など、通常の値型と同様です。
- 値が **null** である場合、割り当てられた値または基礎となる型の既定値を返すには、System.Nullable.GetValueOrDefault プロパティを使用します。たとえば、 `int j = x.GetValueOrDefault();` と指定します。
- null をテストし、値を取得するには、HasValue と Value の読み取り専用プロパティを使用します。たとえば、`if(x.HasValue) j = x.Value;` と指定します。
    - **HasValue** プロパティは、変数に値が含まれる場合は true を返し、null の場合は false を返します。
    - **Value** プロパティは、値が割り当てられている場合はその値を返し、それ以外の場合は System.InvalidOperationException をスローします。

    - null 許容型の変数の既定値では、**HasValue** が **false** に設定されています。**Value** は未定義です。
- 現在の値が null である null 許容型に、null 非許容型を割り当てる場合、既定値を割り当てるには、**??** 演算子を使用します。たとえば、`int? x = null; int y = x ?? -1;` と指定します。
- 入れ子になった null 許容型は許可されていません。`Nullable<Nullable<int>> n;` の行はコンパイルされません。

関連項目

詳細情報

- Null 許容型の使用 (C# プログラミング ガイド)
- null 許容型のボックス化 (C# プログラミング ガイド)
- ?? 演算子 (C# リファレンス)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 24 Null 許容型

参照

処理手順
null 許容のサンプル
関連項目
Nullable
概念
C# プログラミング ガイド
その他の技術情報
Visual C#
C# リファレンス

Null 許容型の使用 (C# プログラミング ガイド)

Null 許容型は、基底の型のすべての値に加えて、null 値を表すことができます。Null 許容型は、次のいずれかの形式で宣言します。

```
System.Nullable<T> variable
```

または

```
T? variable
```

T は、Null 許容型の基底の型です。T には、**struct** を含む任意の値型を指定できますが、参照型は指定できません。

Null 許容型を使用するときの例として、通常のブール型変数が、true と false の 2 つの値をどのように持つことができるかを考えてみましょう。この変数には、"未定義" を示す値はありません。多くのプログラミング アプリケーションの中でも特にデータベース操作では、変数が未定義の状態で存在することがあります。たとえば、データベース フィールドには、true や false の値が入力されている場合がありますが、値が入力されていない場合もあります。また、参照型に **null** を設定すると、その型が初期化されていないことを示すことができます。

このような違いから、ステータス情報を格納する変数を追加したり、特別な値を使用したりするような余分なプログラミング作業が生じることがあります。C# では、Null 許容型修飾子により、未定義の値を示す値型変数を作成できます。

Null 許容型の例

Null 許容型の基底の型には、任意の値型を使用できます。次に例を示します。

C#

```
int? i = 10;
double? d1 = 3.14;
bool? flag = null;
char? letter = 'a';
int?[] arr = new int?[10];
```

Null 許容型のメンバ

Null 許容型の各インスタンスには、次のような読み取り専用の 2 つのパブリック プロパティがあります。

- **HasValue**

  HasValue は **bool** 型です。変数が null 以外の値を格納している場合、このプロパティには **true** が設定されます。

- **Value**

  Value は、基底の型と同じ型です。HasValue が **true** の場合、Value は、有意な値を格納しています。HasValue が **false** の場合、Value にアクセスすると、InvalidOperationException がスローされます。

次の例では、HasValue メンバを使用して、値を表示する前に、変数に値が格納されているかどうかをテストします。

C#

```
int? x = 10;
if (x.HasValue)
{
    System.Console.WriteLine(x.Value);
}
else
{
    System.Console.WriteLine("Undefined");
}
```

値のテストは、次のように行うこともできます。

C#

```
int? y = 10;
if (y != null)
{
    System.Console.WriteLine(y.Value);
```

```
}
else
{
    System.Console.WriteLine("Undefined");
}
```

明示的な変換

Null 許容型は、キャストを明示的に使用するか、または `Value` プロパティを使用して通常の型にキャストできます。次に例を示します。

C#

```
int? n = null;

//int m1 = n;      // Will not compile.
int m2 = (int)n;   // Compiles, but will create an exception if x is null.
int m3 = n.Value;  // Compiles, but will create an exception if x is null.
```

2 つのデータ型の間でユーザー定義変換を定義している場合は、これらのデータ型の Null 許容バージョンを使用して、同じユーザー定義変換を実行することもできます。

暗黙の型変換

Null 許容型の変数には、次のように **null** キーワードを使用して null に設定できます。

C#

```
int? n1 = null;
```

通常の型から Null 許容型への変換は暗黙的です。

C#

```
int? n2;
n2 = 10;  // Implicit conversion.
```

演算子

定義済みの単項演算子と二項演算子およびユーザー定義演算子は、いずれも値型を対象にしていますが、Null 許容型でも使用できます。これらの演算子は、オペランドが null の場合は null 値を生成し、null 以外の場合は、包含されている値に基づいて結果を算出します。次に例を示します。

C#

```
int? a = 10;
int? b = null;

a++;         // Increment by 1, now a is 11.
a = a * 10;  // Multiply by 10, now a is 110.
a = a + b;   // Add b, now a is null.
```

2 つの Null 許容型を比較したときに、Null 許容型のいずれかが **null** の場合は、常に **false** と評価されます。そのため、比較した結果が **false** であっても、逆のケースが **true** とは限りません。次に例を示します。

C#

```
int? num1 = 10;
int? num2 = null;
if (num1 >= num2)
{
    System.Console.WriteLine("num1 is greater than or equal to num1");
}
else
```

```
{
    // num1 is NOT less than num2
}
```

上の **else** ステートメントの結果は、`num2` が **null** であり、値を含んでいないため、無効です。

どちらも **null** である 2 つの Null 許容型を比較すると、結果は **true** になります。

?? 演算子

?? 演算子は、Null 許容型が Null 許容型以外の型に割り当てられているときに返す既定値を定義します。

C#

```
int? c = null;

// d = c, unless c is null, in which case d = -1.
int d = c ?? -1;
```

この演算子は、複数の Null 許容型で使用することもできます。次に例を示します。

C#

```
int? e = null;
int? f = null;

// g = e or f, unless e and f are both null, in which case g = -1.
int g = e ?? f ?? -1;
```

bool? 型

Null 許容 bool? 型は、true、false、null の 3 つの異なる値を格納できます。そのため、この型は、**if**、**for**、**while** などの条件文で使用できません。たとえば、次のコードでは、コンパイラ エラー CS0266 が発生してコンパイルが失敗します。

```
bool? b = null;
if (b) // Error CS0266.
{
}
```

コンパイルできないのは、条件文のコンテキストで **null** の意味があいまいだからです。Null 許容ブール型は、条件文で使用するために明示的に bool にキャストできますが、オブジェクトの値が **null** の場合は、**InvalidOperationException** がスローされます。そのため、**bool** にキャストする前に HasValue プロパティをチェックする必要があります。

Null 許容ブール型は、SQL で使用するブール変数型と同じです。**&** 演算子と **|** 演算子によって生成される結果が SQL の 3 値のブール型と一致するようにするには、次の定義済みの演算子を指定します。

```
bool? operator &(bool? x, bool? y)
```

```
bool? operator |(bool? x, bool? y)
```

これらの演算子の結果を以下の表に示します。

x	**y**	**x&y**	**x|y**
True	true	True	true
True	false	False	true
True	null	Null	true
False	true	False	true
False	false	False	false

False	null	False	null
Null	true	Null	true
Null	false	False	null
Null	null	Null	null

参照

関連項目

null 許容型のボックス化 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

null 許容型 (C# プログラミング ガイド)

null 許容型のボックス化 (C# プログラミング ガイド)

null 許容型に基づくオブジェクトは、null 以外の場合にのみボックス化されます。HasValue が **false** の場合は、ボックス化ではなく、単にオブジェクト参照が **null** に代入されます。たとえば、次のように指定します。

```
bool? b = null;
object o = b;
// Now o is null.
```

オブジェクトが null 以外の場合 (**HasValue** が **true** の場合)、ボックス化が実行されますが、null 許容オブジェクトの基になる型のみがボックス化されます。null 以外の null 許容値型のボックス化では、その値型自体がボックス化され、その値型をラップする System.Nullable はボックス化されません。次に例を示します。

```
bool? b = false;
int? i = 44;
object bBoxed = b; // bBoxed contains a boxed bool.
object iBoxed = i; // iBoxed contains a boxed int.
```

上の例のボックス化された 2 つのオブジェクトは、null 非許容型のボックス化によって作成されたオブジェクトと同じです。また、ボックス化された null 非許容型と同様に、null 許容型にボックス化解除できます。この例を次に示します。

```
bool? b2 = (bool?)bBoxed;
int? i2 = (int?)iBoxed;
```

解説

ボックス化された場合の null 許容型の動作には、次の 2 つの利点があります。

1. null 許容オブジェクトとそのボックス化されたオブジェクトは、次のように null であるかどうかをテストできます。

   ```
   bool? b = null;
   object boxedB = b;
   if (b == null)
   {
     // True.
   }
   if (boxedB == null)
   {
     // Also true.
   }
   ```

2. ボックス化された null 許容型は、次のように基になる型の機能を完全にサポートします。

   ```
   double? d = 44.4;
   object iBoxed = d;
   // Access IConvertible interface implemented by double.
   IConvertible ic = (IConvertible)iBoxed;
   int i = ic.ToInt32(null);
   string str = ic.ToString();
   ```

ボックス化の動作を含む、null 許容型のその他の例については、「null 許容のサンプル」を参照してください。

参照

処理手順

方法 : Null 許容型を識別する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

null 許容型 (C# プログラミング ガイド)

方法：Null 許容型を識別する (C# プログラミング ガイド)

C# の typeof 演算子を使用して、Null 許容型を表す Type オブジェクトを作成できます。

```
System.Type type = typeof(int?);
```

System.Reflection 名前空間のクラスおよびメソッドを使用して、Null 許容型を表す **Type** オブジェクトを作成することもできます。ただし、実行時に GetType メソッドまたは **is** 演算子を使用して Null 許容型の変数から型情報を取得しようとすると、Null 許容型ではなく基になる型を表す **Type** オブジェクトが作成されます。

Null 許容型に対して **GetType** を呼び出すと、型が暗黙的に Object に変換されるときに、ボックス化操作が実行されます。このため、**GetType** は常に、Null 許容型ではなく基になる型を表す **Type** オブジェクトを返します。

```
int? i = 5;
Type t = i.GetType();
Console.WriteLine(t.FullName); //"System.Int32"
```

C# の is 演算子も Null 許容型に作用します。このため、変数が Null 許容型であるかどうかを確認する目的で **is** を使用することはできません。次の例の **is** 演算子は、Nullable<int> 変数を整数値として処理します。

```
static void Main(string[] args)
{
  int? i = 5;
  if (i is int) // true
    //…
}
```

使用例

次のコードを使用して、**Type** オブジェクトが Null 許容型を表しているかどうかを確認します。前に説明したように、**Type** オブジェクトが **GetType** の呼び出しから返された場合、このコードは常に false を返します。

```
if (type.IsGenericType && type.GetGenericTypeDefinition() == typeof(Nullable<>)) {…}
```

参照

関連項目

null 許容型のボックス化 (C# プログラミング ガイド)

概念

null 許容型 (C# プログラミング ガイド)

unsafe コードとポインタ (C# プログラミング ガイド)

型の安全性とセキュリティを維持するために、C# では既定でポインタ演算がサポートされていません。ただし、unsafe キーワードを使用すると、ポインタを使用できる unsafe コンテキストを定義できます。ポインタの詳細については、「ポインタ型 (C# プログラミング ガイド)」を参照してください。

メモ :

共通言語ランタイム (CLR) では、アンセーフ コードは検査できないコードと呼ばれます。C# の アンセーフ コードは、それほど危険ではありません。単に、CLR で安全性を検査できないコードです。そのため、CLR では、完全に信頼できるアセンブリ内にある場合にのみ、アンセーフ コードが実行されます。アンセーフ コードを使用する場合は、セキュリティ上のリスクやポインタ エラーが発生しないように注意してください。詳細については、「セキュリティ (C# プログラミング ガイド)」を参照してください。

アンセーフ コードの概要

アンセーフ コードには次の特性があります。

- メソッド、型、およびコード ブロックは、unsafe として定義できます。
- アンセーフ コードでアプリケーションのパフォーマンスが向上することがあります。これは、配列のバインド チェックが削除されるためです。
- アンセーフ コードは、ポインタを必要とするネイティブ関数を呼び出すときに必要です。
- アンセーフ コードを使用すると、セキュリティと安定性の面でリスクが高くなります。
- C# でアンセーフ コードをコンパイルするには、/unsafe を指定してアプリケーションをコンパイルする必要があります。

関連項目

詳細については、次のトピックを参照してください。

- ポインタ型 (C# プログラミング ガイド)
- 固定サイズ バッファ (C# プログラミング ガイド)
- 方法 : ポインタを使用してバイトの配列をコピーする (C# プログラミング ガイド)
- 方法 : Windows の ReadFile 関数を使用する (C# プログラミング ガイド)
- unsafe (C# リファレンス)
- アンセーフ コードのサンプル

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 18 アンセーフ コード
- B 3 アンセーフ コードのための文法

参照

概念

C# プログラミング ガイド

固定サイズ バッファ (C# プログラミング ガイド)

C# 2.0 では、fixed ステートメントを使用して、データ構造内に固定サイズの配列を作成できます。この機能は、別の言語で作成されたコード、既存の DLL プロジェクト、既存の COM プロジェクトなど、各種既存コードを操作するときに便利です。固定配列は、標準の構造体メンバのあらゆる属性または修飾子を使用できます。唯一の制限として、配列型を **bool**、**byte**、**char**、**short**、**int**、**long**、**sbyte**、**ushort**、**uint**、**ulong**、**float**、または **double** にする必要があります。

```
private fixed char name[30];
```

解説

C# の以前のバージョンでは、配列を格納する C# 構造体が配列要素を含まず、代わりに配列要素への参照を含むため、C++ スタイルの固定サイズの構造体を宣言するのが困難でした。

C# 2.0 では、安全でないコード ブロックの使用時に固定サイズの配列を構造体に埋め込む機能が追加されています。

たとえば、C# 2.0 より前のバージョンでは、次の **struct** はサイズが 8 バイトで、`pathName` 配列はヒープ割り当て配列への参照です。

C#

```
public struct MyArray
{
    public char[] pathName;
    private int reserved;
}
```

C# 2.0 では、**struct** は、埋め込まれた配列で次のように宣言できます。

C#

```
public struct MyArray // This code must appear in an unsafe block
{
    public fixed char pathName[128];
}
```

この構造体の `pathName` 配列は、サイズと位置が固定されるので、他のアンセーフ コードで使用できます。

128 要素の **char** 配列のサイズは 256 バイトです。固定サイズの char バッファは、エンコーディングとは無関係に、常に 1 文字につき 2 バイトを使用します。これは、`CharSet = CharSet.Auto` や `CharSet = CharSet.Ansi` によって char バッファが API メソッドや構造体にマーシャリングされる場合でも同じです。詳細については、「CharSet」を参照してください。

一般的な固定サイズの配列には、この他に bool 配列があります。**bool** 配列の要素は常にサイズが 1 バイトです。そのため、**bool** 配列は、ビット配列やバッファの作成には適していません。

メモ :

stackalloc で作成されたメモリを除き、C# コンパイラおよび共通言語ランタイム (CLR: Common Language Runtime) は、バッファ オーバーランのセキュリティ チェックを実行しません。固定サイズの配列には、すべてのアンセーフ コードと同様に注意してください。

アンセーフ バッファは、通常のバッファと次の点で異なります。

- アンセーフ バッファは、unsafe コンテキストでのみ使用できます。
- アンセーフ バッファは常にベクタ (1 次元配列) です。
- 配列の宣言には、`char id[8]` のように要素数を記述する必要があります。代わりに `char id[]` を使用することはできません。
- アンセーフ バッファは、unsafe コンテキストの構造体のインスタンス フィールドのみに限られます。

参照

処理手順

方法 : PInvoke を使用してマネージ コードからネイティブ DLL を呼び出す

方法 : PInvoke を使用して配列をマーシャリングする
方法 : PInvoke を使用して埋め込みポインタをマーシャリングする
方法 : PInvoke を使用して関数ポインタをマーシャリングする
方法 : PInvoke を使用して文字列をマーシャリングする

関連項目

fixed ステートメント (C# リファレンス)

概念

C# プログラミング ガイド
unsafe コードとポインタ (C# プログラミング ガイド)
配列に対する既定のマーシャリング

方法：Windows の ReadFile 関数を使用する (C# プログラミング ガイド)

この例では、テキスト ファイルの読み取りと表示を行うことにより、Windows の **ReadFile** 関数を示します。**ReadFile** 関数では、パラメータとしてポインタを必要とするため、**unsafe** コードを使用する必要があります。

Read 関数に渡されるバイト配列はマネージ型です。つまり、共通言語ランタイム (CLR: Common Language Runtime) のガベージ コレクタは、配列が使用するメモリを自由に再配置できます。これを防ぐために、fixed を使用して、メモリへのポインタを取得し、ガベージ コレクタがそれを移動しないようにマークします。**fixed** ブロックの末尾で、メモリはガベージ コレクションによる移動の対象に自動的に戻ります。

この機能は、宣言固定と呼ばれます。メモリの固定が優れている点は、ガベージ コレクションが **fixed** ブロック内で発生しない限り (ほとんど発生しません)、オーバーヘッドがきわめて少ないことです。

使用例

C#

```
class FileReader
{
    const uint GENERIC_READ = 0x80000000;
    const uint OPEN_EXISTING = 3;
    System.IntPtr handle;

    [System.Runtime.InteropServices.DllImport("kernel32", SetLastError = true)]
    static extern unsafe System.IntPtr CreateFile
    (
        string FileName,          // file name
        uint DesiredAccess,       // access mode
        uint ShareMode,           // share mode
        uint SecurityAttributes,  // Security Attributes
        uint CreationDisposition, // how to create
        uint FlagsAndAttributes,  // file attributes
        int hTemplateFile         // handle to template file
    );

    [System.Runtime.InteropServices.DllImport("kernel32", SetLastError = true)]
    static extern unsafe bool ReadFile
    (
        System.IntPtr hFile,      // handle to file
        void* pBuffer,            // data buffer
        int NumberOfBytesToRead,  // number of bytes to read
        int* pNumberOfBytesRead,  // number of bytes read
        int Overlapped            // overlapped buffer
    );

    [System.Runtime.InteropServices.DllImport("kernel32", SetLastError = true)]
    static extern unsafe bool CloseHandle
    (
        System.IntPtr hObject // handle to object
    );

    public bool Open(string FileName)
    {
        // open the existing file for reading
        handle = CreateFile
        (
            FileName,
            GENERIC_READ,
            0,
            0,
            OPEN_EXISTING,
            0,
            0
        );

        if (handle != System.IntPtr.Zero)
```

```
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public unsafe int Read(byte[] buffer, int index, int count)
    {
        int n = 0;
        fixed (byte* p = buffer)
        {
            if (!ReadFile(handle, p + index, count, &n, 0))
            {
                return 0;
            }
        }
        return n;
    }

    public bool Close()
    {
        return CloseHandle(handle);
    }
}

class Test
{
    static int Main(string[] args)
    {
        if (args.Length != 1)
        {
            System.Console.WriteLine("Usage : ReadFile <FileName>");
            return 1;
        }

        if (!System.IO.File.Exists(args[0]))
        {
            System.Console.WriteLine("File " + args[0] + " not found.");
            return 1;
        }

        byte[] buffer = new byte[128];
        FileReader fr = new FileReader();

        if (fr.Open(args[0]))
        {
            // Assume that an ASCII file is being read.
            System.Text.ASCIIEncoding Encoding = new System.Text.ASCIIEncoding();

            int bytesRead;
            do
            {
                bytesRead = fr.Read(buffer, 0, buffer.Length);
                string content = Encoding.GetString(buffer, 0, bytesRead);
                System.Console.Write("{0}", content);
            }
            while (bytesRead > 0);

            fr.Close();
            return 0;
        }
        else
        {
            System.Console.WriteLine("Failed to open requested file");
            return 1;
```

```
            }
        }
    }
```

参照

関連項目

ポインタ型 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

unsafe コードとポインタ (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

ガベージ コレクション

ポインタ型 (C# プログラミング ガイド)

unsafe コンテキストの型には、ポインタ型、値型、または参照型を設定できます。ポインタ型の宣言は、次のいずれかの形式になります。

```
type* identifier;
void* identifier; //allowed but not recommended
```

次の型はいずれもポインタ型になります。

- sbyte、byte、short、ushort、int、uint、long、ulong、char、float、double、decimal、または bool。
- 任意の列挙型。
- 任意のポインタ型。
- アンマネージ型のフィールドのみを含むユーザー定義の struct 型。

ポインタ型は object を継承せず、ポインタ型と **object** 間の変換は存在しません。また、ボックス化とボックス化解除もポインタをサポートしません。ただし、異なるポインタ型の間で変換したり、ポインタ型と整数型の間で変換したりすることはできます。

同じ 1 つの宣言で複数のポインタを宣言する場合、* は基底の型だけに記述し、各ポインタ名のプレフィックスとして使用しません。以下にサンプルを示します。

```
int* p1, p2, p3;   // Ok
int *p1, *p2, *p3;   // Invalid in C#
```

オブジェクト参照は、それを指すポインタがあってもガベージ コレクションされる場合があるため、ポインタが参照や参照を含む構造体を指すことはできません。GC は、オブジェクトを指すポインタ型があるかどうかを追跡しません。

`myType*` 型のポインタ変数の値は、`myType` 型の変数のアドレスです。ポインタ型の宣言の例を次に示します。

例	説明
`int* p`	p は、整数へのポインタです。
`int** p`	p は、整数へのポインタのポインタです。
`int*[] p`	p は、整数へのポインタの 1 次元配列です。
`char* p`	p は、char へのポインタです。
`void* p`	p は、未知の型へのポインタです。

ポインタ間接演算子 * を使用すると、ポインタ変数が指す位置にあるコンテンツにアクセスできます。ポインタ間接演算子を使用した宣言の例を次に示します。

```
int* myVariable;
```

この例の式 `*myVariable` は、`myVariable` に含まれているアドレスの位置にある **int** 変数を示しています。

間接演算子は、`void*` 型のポインタに適用できません。ただし、void ポインタと他のポインタ型はキャストを使用して相互に変換できます。

ポインタは、**null** にできます。null ポインタに間接演算子を適用すると、実装で定義されている動作が発生します。

ポインタをメソッド間で引き渡すと、未定義の動作が発生する可能性があります。たとえば、Out パラメータや Ref パラメータを介してポインタをローカル変数に返したり、関数の結果として返したりする場合です。ポインタが固定ブロックに設定されていた場合は、そのポインタが指す変数が既に固定されていない可能性があります。

次の表は、unsafe コンテキストでポインタに使用できる演算子とステートメントの一覧を示しています。

演算子/ステートメント	用途

*	ポインタの間接参照を実行します。
->	ポインタ経由で構造体のメンバにアクセスします。
[]	ポインタにインデックスを付けます。
&	変数のアドレスを取得します。
++ および --	ポインタをインクリメントおよびデクリメントします。
+ および -	ポインタ演算を実行します。
==、!=、<、>、<=、および >=	ポインタを比較します。
stackalloc	スタックにメモリを割り当てます。
fixed ステートメント	変数を一時的に固定して、そのアドレスを取得できるようにします。

C# 言語仕様

詳細については、「C# 言語仕様」の以下のセクションを参照してください。

- 18 アンセーフ コード

参照

関連項目

ポインタ変換 (C# プログラミング ガイド)
ポインタ式 (C# プログラミング ガイド)
unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド
unsafe コードとポインタ (C# プログラミング ガイド)
ボックス化とボックス化解除 (C# プログラミング ガイド)

その他の技術情報

型 (C# リファレンス)

ポインタ変換 (C# プログラミング ガイド)

定義済みの暗黙のポインタ変換を次の表に示します。暗黙の変換は、メソッドの呼び出しや代入ステートメントなど、多くの状況で発生することがあります。

暗黙のポインタ変換

変換前	変換後
任意のポインタ型	void*
null	任意のポインタ型

明示的なポインタ変換には暗黙の変換がなく、キャスト式を使用して変換を実行します。これらの変換を次に示します。

明示的なポインタ変換

変換前	変換後
任意のポインタ型	他の任意のポインタ型
sbyte、byte、short、ushort、int、uint、long、ulong	任意のポインタ型
任意のポインタ型	sbyte、byte、short、ushort、int、uint、long、ulong

使用例

次の例では、**int** へのポインタを **byte** へのポインタに変換しています。このポインタは、変数のアドレスの最下位バイトを指すことに注意してください。結果を **int** のサイズ (4 バイト) だけ連続してインクリメントすると、変数の残りのバイトを表示できます。

C#

```
// compile with: /unsafe
```

C#

```
class ClassConvert
{
    static void Main()
    {
        int number = 1024;

        unsafe
        {
            // Convert to byte:
            byte* p = (byte*)&number;

            System.Console.Write("The 4 bytes of the integer:");

            // Display the 4 bytes of the int variable:
            for (int i = 0 ; i < sizeof(int) ; ++i)
            {
                System.Console.Write(" {0:X2}", *p);
                // Increment the pointer:
                p++;
            }
            System.Console.WriteLine();
            System.Console.WriteLine("The value of the integer: {0}", number);
        }
    }
}
```

出力

```
The 4 bytes of the integer: 00 04 00 00

The value of the integer: 1024
```

参照

関連項目

ポインタ式 (C# プログラミング ガイド)
ポインタ型 (C# プログラミング ガイド)
unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

型 (C# リファレンス)

ポインタ式 (C# プログラミング ガイド)

ここでは、次のポインタ式について説明します。

ポインタ変数の値 (C# プログラミング ガイド)

変数のアドレスの取得 (C# プログラミング ガイド)

方法 : ポインタを使用してメンバにアクセスする (C# プログラミング ガイド)

方法 : ポインタを使用して配列要素にアクセスする (C# プログラミング ガイド)

ポインタの操作 (C# プログラミング ガイド)

参照

関連項目

ポインタ変換 (C# プログラミング ガイド)
ポインタ型 (C# プログラミング ガイド)
unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

型 (C# リファレンス)

方法：ポインタ変数の値を取得する (C# プログラミング ガイド)

ポインタが指している位置にある変数を取得するには、ポインタ間接演算子を使用します。この式は、次の形式になります。 `p` はポインタ型を表します。

```
*p;
```

単項間接演算子は、ポインタ型以外の型の式では使用できません。また、void ポインタに適用することもできません。

間接演算子を null ポインタに適用したときの結果は、実装によって異なります。

使用例

次の例では、異なる型のポインタを使用して `char` 型の変数にアクセスしています。変数に割り当てられる物理アドレスは一定ではないので、`theChar` のアドレスは、実行するたびに変化することに注意してください。

C#

```
// compile with: /unsafe
```

C#

```
unsafe class TestClass
{
    static void Main()
    {
        char theChar = 'Z';
        char* pChar = &theChar;
        void* pVoid = pChar;
        int* pInt = (int*)pVoid;

        System.Console.WriteLine("Value of theChar = {0}", theChar);
        System.Console.WriteLine("Address of theChar = {0:X2}",(int)pChar);
        System.Console.WriteLine("Value of pChar = {0}", *pChar);
        System.Console.WriteLine("Value of pInt = {0}", *pInt);
    }
}
```

サンプル出力

```
Value of theChar = Z
Address of theChar = 12F718
Value of pChar = Z
Value of pInt = 90
```

参照

関連項目

ポインタ式 (C# プログラミング ガイド)
ポインタ型 (C# プログラミング ガイド)
unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

型 (C# リファレンス)

方法：変数のアドレスを取得する (C# プログラミング ガイド)

固定変数に評価される単項式のアドレスを取得するには、アドレス演算子を使用します。

```
int number;
int* p = &number; //address-of operator &
```

アドレス演算子は、変数にのみ適用できます。変数が可変変数である場合は、固定ステートメントを使用して、アドレスを取得する前に一時的に変数を固定します。

変数は、確実に初期化してください。変数が初期化されていない場合、コンパイラはエラー メッセージを発行しません。

定数や値のアドレスは取得できません。

使用例

次の例では、**int** へのポインタ `p` を宣言し、整数の変数 `number` のアドレスを代入します。*p に代入することによって変数 `number` が初期化されます。この代入ステートメントをコメントにすると、変数 `number` の初期化が削除されますが、コンパイル時のエラーは発行されません。メンバ アクセス演算子 **->** を使用して、ポインタに格納されているアドレスを取得し、表示することに注意してください。

C#

```
// compile with: /unsafe
```

C#

```
class AddressOfOperator
{
    static void Main()
    {
        int number;

        unsafe
        {
            // Assign the address of number to a pointer:
            int* p = &number;

            // Commenting the following statement will remove the
            // initialization of number.
            *p = 0xffff;

            // Print the value of *p:
            System.Console.WriteLine("Value at the location pointed to by p: {0:X}", *p);

            // Print the address stored in p:
            System.Console.WriteLine("The address stored in p: {0}", p->ToString());
        }

        // Print the value of the variable number:
        System.Console.WriteLine("Value of the variable number: {0:X}", number);
    }
}
```

サンプル出力

```
Value at the location pointed to by p: FFFF
The address stored in p: 65535
Value of the variable number: FFFF
```

参照

関連項目

ポインタ式 (C# プログラミング ガイド)
ポインタ型 (C# プログラミング ガイド)

unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

型 (C# リファレンス)

方法：ポインタを使用してメンバにアクセスする (C# プログラミング ガイド)

unsafe コンテキストで宣言された構造体のメンバにアクセスするには、次の例に示すように、メンバ アクセス演算子を使用できます。p は、メンバ x を含む構造体のポインタになります。

```
CoOrds* p = &home;
p -> x = 25; //member access operator ->
```

使用例

次の例では、x と y の 2 つの座標を含む構造体である CoOrds を宣言し、インスタンス化します。-> メンバ アクセス演算子と、home インスタンスへのポインタを使用して、x と y に値を代入します。

メモ：

式 p->x と式 (*p).x は等価であり、どちらの式を使用しても同じ結果が得られます。

C#

```
// compile with: /unsafe
```

C#

```
struct CoOrds
{
    public int x;
    public int y;
}

class AccessMembers
{
    static void Main()
    {
        CoOrds home;

        unsafe
        {
            CoOrds* p = &home;
            p->x = 25;
            p->y = 12;

            System.Console.WriteLine("The coordinates are: x={0}, y={1}", p->x, p->y );
        }
    }
}
```

参照

関連項目

ポインタ式 (C# プログラミング ガイド)
ポインタ型 (C# プログラミング ガイド)
unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

型 (C# リファレンス)

方法：ポインタを使用して配列要素にアクセスする (C# プログラミング ガイド)

unsafe コンテキストでは、次の例のようにポインタ要素アクセスを使用して、メモリ内の要素にアクセスできます。

```
        char* charPointer = stackalloc char[123];
       for (int i = 65; i < 123; i++)
       {
           charPointer[i] = (char)i; //access array elements
       }
```

角かっこ内の式は、暗黙的に **int**、**uint**、**long**、または **ulong** に変換できる必要があります。演算 p[e] は *(p+e) と等価です。C や C++ の場合と同様に、ポインタ要素アクセスでは、範囲外のエラーをチェックしません。

使用例

次の例では、123 のメモリ位置を文字配列 `charPointer` に割り当てます。この配列を使用して、2 つの for ループ中の小文字と大文字を表示します。

式 `charPointer[i]` と式 `*(charPointer + i)` は等価であり、どちらの式を使用しても同じ結果が得られます。

C#

```
// compile with: /unsafe
```

C#

```
class Pointers
{
    unsafe static void Main()
    {
        char* charPointer = stackalloc char[123];

        for (int i = 65; i < 123; i++)
        {
            charPointer[i] = (char)i;
        }

        // Print uppercase letters:
        System.Console.WriteLine("Uppercase letters:");
        for (int i = 65; i < 91; i++)
        {
            System.Console.Write(charPointer[i]);
        }
        System.Console.WriteLine();

        // Print lowercase letters:
        System.Console.WriteLine("Lowercase letters:");
        for (int i = 97; i < 123; i++)
        {
            System.Console.Write(charPointer[i]);
        }
    }
}
```

サンプル出力

```
Uppercase letters:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Lowercase letters:
abcdefghijklmnopqrstuvwxyz
```

参照

関連項目

ポインタ式 (C# プログラミング ガイド)
ポインタ型 (C# プログラミング ガイド)
unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

型 (C# リファレンス)

ポインタの操作 (C# プログラミング ガイド)

ここでは、次のポインタの操作について説明します。

ポインタのインクリメントとデクリメント (C# プログラミング ガイド)

算術演算

ポインタ比較 (C# プログラミング ガイド)

参照

関連項目

ポインタ式 (C# プログラミング ガイド)
C# の演算子
ポインタ型 (C# プログラミング ガイド)
/unsafe (unsafe モードの有効化) (C# コンパイラ オプション)

概念

C# プログラミング ガイド

方法：ポインタのインクリメントとデクリメント (C# プログラミング ガイド)

pointer-type* 型のポインタの位置を sizeof (*pointer-type*) に従って変更するには、インクリメント演算子 ++ またはデクリメント演算子 -- を使用します。インクリメント式とデクリメント式には、次の書式を使用します。

```
++p;
P++;
--p;
p--;
```

インクリメント演算子とデクリメント演算子は、`void*` 以外のすべての型のポインタに適用できます。

pointer-type 型のポインタにインクリメント演算子を適用すると、ポインタ変数に格納されているアドレスに sizeof (*pointer-type*) が加算されます。

また、*pointer-type* 型のポインタにデクリメント演算子を適用すると、ポインタ変数に格納されているアドレスから **sizeof** (*pointer-type*) が減算されます。

演算がポインタのドメインをオーバーフローしても例外は生成されません。どのような結果が生じるかは実装によって異なります。

使用例

次の例では、ポインタを **int** のサイズだけインクリメントして、配列をステップ実行します。ステップごとに、配列要素のアドレスと内容を表示します。

C#

```
// compile with: /unsafe
```

C#

```
class IncrDecr
{
    unsafe static void Main()
    {
        int[] numbers = {0,1,2,3,4};

        // Assign the array address to the pointer:
        fixed (int* p1 = numbers)
        {
            // Step through the array elements:
            for(int* p2=p1; p2<p1+numbers.Length; p2++)
            {
                System.Console.WriteLine("Value:{0} @ Address:{1}", *p2, (long)p2);
            }
        }
    }
}
```

サンプル出力

```
Value:0 @ Address:12860272
Value:1 @ Address:12860276
Value:2 @ Address:12860280
Value:3 @ Address:12860284
Value:4 @ Address:12860288
```

参照

関連項目

ポインタ式 (C# プログラミング ガイド)
C# の演算子
ポインタの操作 (C# プログラミング ガイド)
ポインタ型 (C# プログラミング ガイド)

unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

型 (C# リファレンス)

ポインタに対する算術演算 (C# プログラミング ガイド)

ここでは、算術演算子 + および - を使用したポインタ操作について説明します。

メモ：
void ポインタには、算術演算を実行できません。

ポインタへの数値の加算と除算

型が int、uint、long、または ulong の値 `n` を `void*` 以外の任意の型のポインタ `p` に加算できます。結果の `p+n` は、`p` のアドレスに `n * sizeof(p)` を加算した結果のポインタです。同様に `p-n` は、`p` のアドレスから `n * sizeof(p)` を減算した結果のポインタです。

ポインタの除算

同じ型のポインタを除算することもできます。結果は常に **long** 型になります。たとえば、`p1` と `p2` が `pointer-type*` 型のポインタの場合、式 `p1-p2` の結果は次のようになります。

```
((long)p1 - (long)p2)/sizeof(pointer_type)
```

算術演算がポインタのドメインをオーバーフローしても例外は生成されません。どのような結果が生じるかは実装によって異なります。

使用例

C#

```
// compile with: /unsafe
```

C#

```
class PointerArithmetic
{
    unsafe static void Main()
    {
        int* memory = stackalloc int[30];
        long* difference;
        int* p1 = &memory[4];
        int* p2 = &memory[10];

        difference = (long*)(p2 - p1);

        System.Console.WriteLine("The difference is: {0}", (long)difference);
    }
}
```

出力

```
The difference is: 6
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 18.5.6 ポインタの算術演算

参照

関連項目

ポインタ式 (C# プログラミング ガイド)
C# の演算子
ポインタの操作 (C# プログラミング ガイド)
ポインタ型 (C# プログラミング ガイド)
unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド
unsafe コードとポインタ (C# プログラミング ガイド)

その他の技術情報

型 (C# リファレンス)

ポインタ比較 (C# プログラミング ガイド)

次の演算子を適用すると、あらゆる型のポインタを比較できます。

```
== != < > <= >=
```

比較演算子は、2 つのオペランドのアドレスを符号なし整数として比較します。

使用例

C#

```
// compile with: /unsafe
```

C#

```
class CompareOperators
{
    unsafe static void Main()
    {
        int x = 234;
        int y = 236;
        int* p1 = &x;
        int* p2 = &y;

        System.Console.WriteLine(p1 < p2);
        System.Console.WriteLine(p2 < p1);
    }
}
```

出力例

```
True

False
```

参照

関連項目

ポインタ式 (C# プログラミング ガイド)
C# の演算子
ポインタの操作 (C# プログラミング ガイド)
ポインタ型 (C# プログラミング ガイド)
unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)
stackalloc (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

型 (C# リファレンス)

方法：ポインタを使用してバイトの配列をコピーする (C# プログラミング ガイド)

次の例では、ポインタを使用して配列間でバイトをコピーします。

この例では、unsafe キーワードを使用して、Copy メソッド内でポインタを使用できるようにしています。fixed ステートメントを使用して、コピー元とコピー先の配列へのポインタを宣言します。これで、ガベージ コレクションによって移動されないように、メモリ内でコピー元とコピー先の位置が固定されます。**fixed** ブロックが完了すると、これらのメモリ ブロックの固定が解除されます。この例では、Copy 関数に **unsafe** キーワードを使用しているため、**/unsafe** コンパイラ オプションを指定してコンパイルする必要があります。

使用例

C#

```
// compile with: /unsafe
```

C#

```
class TestCopy
{
    // The unsafe keyword allows pointers to be used within the following method:
    static unsafe void Copy(byte[] src, int srcIndex, byte[] dst, int dstIndex, int count)
    {
        if (src == null || srcIndex < 0 ||
            dst == null || dstIndex < 0 || count < 0)
        {
            throw new System.ArgumentException();
        }

        int srcLen = src.Length;
        int dstLen = dst.Length;
        if (srcLen - srcIndex < count || dstLen - dstIndex < count)
        {
            throw new System.ArgumentException();
        }

        // The following fixed statement pins the location of the src and dst objects
        // in memory so that they will not be moved by garbage collection.
        fixed (byte* pSrc = src, pDst = dst)
        {
            byte* ps = pSrc;
            byte* pd = pDst;

            // Loop over the count in blocks of 4 bytes, copying an integer (4 bytes) at a
time:
            for (int i = 0 ; i < count / 4 ; i++)
            {
                *((int*)pd) = *((int*)ps);
                pd += 4;
                ps += 4;
            }

            // Complete the copy by moving any bytes that weren't moved in blocks of 4:
            for (int i = 0; i < count % 4 ; i++)
            {
                *pd = *ps;
                pd++;
                ps++;
            }
        }
    }

    static void Main()
    {
```

```
            byte[] a = new byte[100];
            byte[] b = new byte[100];

            for (int i = 0; i < 100; ++i)
            {
                a[i] = (byte)i;
            }

            Copy(a, 0, b, 0, 100);
            System.Console.WriteLine("The first 10 elements are:");

            for (int i = 0; i < 10; ++i)
            {
                System.Console.Write(b[i] + " ");
            }
            System.Console.WriteLine("\n");
        }
    }
```

出力

```
The first 10 elements are:
0 1 2 3 4 5 6 7 8 9
```

参照

処理手順

アンセーフ コードのサンプル

関連項目

/unsafe (unsafe モードの有効化) (C# コンパイラ オプション)

概念

C# プログラミング ガイド

unsafe コードとポインタ (C# プログラミング ガイド)

その他の技術情報

ガベージ コレクション

XML ドキュメント コメント (C# プログラミング ガイド)

Visual C# では、ソース コード内で、参照先のコード ブロックの直前の特別なコメント フィールドに XML タグを配置することで、コードのドキュメントを作成できます。この例を次に示します。

```
/// <summary>
///  This class performs an important function.
/// </summary>
public class MyClass{}
```

/doc を使用してコンパイルすると、コンパイラは、ソース コードですべての XML タグを検索し、XML ドキュメント ファイルを作成します。

メモ :

XML ドキュメント コメントはメタデータでなく、コンパイルされたアセンブリに含まれないため、リフレクションでアクセスできません。

このセクションの内容

- ドキュメント コメントとして推奨されるタグ
- XML ファイルの処理
- ドキュメント タグの区切り記号
- 方法 : XML ドキュメント機能を使用する (C# プログラミング ガイド)

関連項目

詳細については、次のトピックを参照してください。

- /doc (ドキュメント コメントの処理)
- XML ドキュメントのサンプル

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 付録 A ドキュメント コメント

参照

概念

C# プログラミング ガイド

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

C# コンパイラは、コード中のドキュメント コメントを処理して XML ファイルを生成します。ドキュメントを作成するために XML ファイルを処理する詳細は、サイトに実装する必要があります。

タグは、型や型メンバなどのコードの構成体に対して処理されます。

メモ :
ドキュメント コメントは、名前空間に適用できません。

C# コンパイラは、有効な XML のタグをすべて処理します。ユーザー ドキュメントで一般的に使用される機能を提供するタグを次の表に示します。

<c>	<para>	<see>*
<code>	<param>*	<seealso>*
<example>	<paramref>	<summary>
<exception>*	<permission>*	<typeparam>*
<include>*	<remarks>	<typeparamref>
<list>	<returns>	<value>

(* は、コンパイラが構文を検証することを示します)

ドキュメント コメントのテキストに山かっこ (< >) を表示する場合は、"<" と ">" を使用します。たとえば、"<テキスト>" のように記述します。

参照

処理手順

XML ドキュメントのサンプル

関連項目

/doc (ドキュメント コメントの処理) (C# コンパイラ オプション)

概念

C# プログラミング ガイド

XML ドキュメント コメント (C# プログラミング ガイド)

XML ドキュメント コメント (C# プログラミング ガイド)

<c> (C# プログラミング ガイド)

```
<c>text</c>
```

パラメータ

text

コードとして指定するテキスト。

解説

<c> タグを使用すると、説明内のテキストをコードとして指定できます。コードとして複数行を指定する場合は、<code> タグを使用します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <summary><c>DoWork</c> is a method in the <c>TestClass</c> class.
    /// </summary>
    public static void DoWork(int Int1)
    {
    }

    /// text for Main
    static void Main()
    {
    }
}
```

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<code> (C# プログラミング ガイド)

```
<code>content</code>
```

パラメータ
content

コードとして指定するテキスト。

解説

<code> タグは、複数行をコードとして指定する場合に使用します。説明内のテキストをコードとして指定する場合は、<c> タグを使用します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

<code> タグの使用例については、「<example>」を参照してください。

参照
関連項目
ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)
概念
C# プログラミング ガイド

<example> (C# プログラミング ガイド)

```
<example>description</example>
```

パラメータ
description

サンプル コードの説明。

解説

<example> タグでは、メソッドやその他のライブラリ メンバの使用例を指定します。通常は、<code> タグも使用します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例
C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <summary>
    /// The GetZero method.
    /// </summary>
    /// <example> This sample shows how to call the GetZero method.
    /// <code>
    /// class TestClass
    /// {
    ///     static int Main()
    ///     {
    ///         return GetZero();
    ///     }
    /// }
    /// </code>
    /// </example>
    public static int GetZero()
    {
        return 0;
    }

    /// text for Main
    static void Main()
    {
    }
}
```

参照
関連項目
ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)
概念
C# プログラミング ガイド

<exception> (C# プログラミング ガイド)

```
<exception cref="member">description</exception>
```

パラメータ

cref = "*member*"

現在のコンパイル環境から利用可能な例外への参照。コンパイラは、指定された例外が存在することを確認してから、*member* を出力先 XML 内で標準要素名に変換します。*member* は二重引用符 (" ") で囲みます。

ジェネリック型への cref 参照の作成方法の詳細については、「<see> (C# プログラミング ガイド)」を参照してください。

description

例外の説明。

解説

<exception> タグを使用すると、スローできる例外を指定できます。このタグは、メソッド、プロパティ、イベント、およびインデクサの定義に適用できます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

C#

```
// compile with: /doc:DocFileName.xml

/// comment for class
public class EClass : System.Exception
{
    // class definition...
}

/// comment for class
class TestClass
{
    /// <exception cref="System.Exception">Thrown when...</exception>
    public void DoSomething()
    {
        try
        {
        }
        catch (EClass)
        {
        }
    }
}
```

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<include> (C# プログラミング ガイド)

```
<include file='filename' path='tagpath[@name="id"]' />
```

パラメータ

filename

ドキュメントを含むファイルの名前。ファイル名にパスを指定することもできます。*filename* は、単一引用符 (' ') で囲みます。

tagpath

filename のタグのパス。その後ろにタグの *name* を指定します。パスは、単一引用符 (' ') で囲みます。

name

タグの名前指定子。その後ろにコメントを指定します。*name* には *id* を指定します。

id

タグの ID。その後ろにコメントを指定します。ID は、二重引用符 (" ") で囲みます。

解説

<include> タグを使用すると、ソース コード内の型およびメンバの説明として、別のファイル内のコメントを参照できます。これはソース コードのファイルにドキュメント コメントを直接記述しない方法です。

<include> タグは、XML の XPath 構文を使用します。<include> タグのカスタマイズ方法については、XPath に関するドキュメントを参照してください。

使用例

複数ファイルの例を次に示します。最初のファイルは、次のように <include> タグを使用しています。

C#

```
// compile with: /doc:DocFileName.xml

/// <include file='xml_include_tag.doc' path='MyDocs/MyMembers[@name="test"]/*' />
class Test
{
    static void Main()
    {
    }
}

/// <include file='xml_include_tag.doc' path='MyDocs/MyMembers[@name="test2"]/*' />
class Test2
{
    public void Test()
    {
    }
}
```

2 番目のファイル xml_include_tag.doc には、次のドキュメント コメントが記述されています。

```
<MyDocs>

<MyMembers name="test">
<summary>
The summary for this type.
</summary>
</MyMembers>

<MyMembers name="test2">
<summary>
```

```
  The summary for this other type.
  </summary>
  </MyMembers>

  </MyDocs>
```

プログラムの出力

```
<?xml version="1.0"?>
<doc>
<assembly>
<name>xml_include_tag</name>
</assembly>
<members>
<member name="T:Test">
<summary>
The summary for this type.
</summary>
</member>
<member name="T:Test2">
<summary>
The summary for this other type.
</summary>
</member>
</members>
</doc>
```

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<list> (C# プログラミング ガイド)

```
<list type="bullet" | "number" | "table">
    <listheader>
        <term>term</term>
        <description>description</description>
    </listheader>
    <item>
        <term>term</term>
        <description>description</description>
    </item>
</list>
```

パラメータ

term

定義する用語。定義は、*description* に記述されます。

description

箇条書きリストまたは番号付きリストの項目、あるいは *term* の定義のいずれか。

解説

<listheader> ブロックは、表または定義リストの見出し行の定義に使用します。表を定義する場合は、見出しには用語のエントリだけを指定します。

リストの各項目は、<item> ブロックで指定します。定義リストを作成するときは、*term* と *description* の両方を指定します。ただし、表、箇条書きリスト、または番号付きリストに指定する必要があるのは、*description* のエントリだけです。

リストや表には、必要な数だけ <item> ブロックを指定できます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <summary>Here is an example of a bulleted list:
    /// <list type="bullet">
    /// <item>
    /// <description>Item 1.</description>
    /// </item>
    /// <item>
    /// <description>Item 2.</description>
    /// </item>
    /// </list>
    /// </summary>
    static void Main()
    {
    }
}
```

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<para> (C# プログラミング ガイド)

```
<para>content</para>
```

パラメータ
content

段落のテキスト。

解説

<para> タグは、<summary>、<remarks>、<returns> などのタグの内部で使用します。<para> タグで、テキストを段落に分けることができます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

<para> タグの使用例については、「<summary>」を参照してください。

参照
関連項目
ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)
概念
C# プログラミング ガイド

<param> (C# プログラミング ガイド)

```
<param name='name'>description</param>
```

パラメータ

name

メソッド パラメータの名前。名前は、二重引用符 (" ") で囲みます。

description

パラメータの説明。

解説

<param> タグは、メソッド宣言のコメント内で使用してメソッドのパラメータの 1 つを説明します。

<param> タグのテキストは、IntelliSense、オブジェクト ブラウザ、およびコード コメント Web レポートに表示されます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <param name="Int1">Used to indicate status.</param>
    public static void DoWork(int Int1)
    {
    }
    /// text for Main
    static void Main()
    {
    }
}
```

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<paramref> (C# プログラミング ガイド)

```
<paramref name="name"/>
```

パラメータ

name

参照するパラメータの名前。名前は、二重引用符 (" ") で囲みます。

解説

<paramref> タグを使用すると、<summary> ブロックや <remarks> ブロックなどのコード コメント内の単語をパラメータの参照として指定できます。XML ファイルは、この単語に太字や斜体などの異なる書式を設定するように処理できます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <summary>DoWork is a method in the TestClass class.
    /// The <paramref name="Int1"/> parameter takes a number.
    /// </summary>
    public static void DoWork(int Int1)
    {
    }

    /// text for Main
    static void Main()
    {
    }
}
```

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<permission> (C# プログラミング ガイド)

```
<permission cref="member">description</permission>
```

パラメータ

cref = "*member*"

現在のコンパイル環境からの呼び出しに利用できる、メンバまたはフィールドへの参照。コンパイラは、指定されたコード要素が存在することを確認してから、*member* を出力先 XML 内で標準要素名に変換します。*member* は二重引用符 (" ") で囲みます。

ジェネリック型への cref 参照の作成方法については、「<see> (C# プログラミング ガイド)」を参照してください。

description

メンバへのアクセスの説明。

解説

<permission> タグを使用すると、メンバへのアクセスをドキュメントにできます。PermissionSet クラスは、アクセスをメンバに指定するために使用します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

C#

```
// compile with: /doc:DocFileName.xml

class TestClass
{
    /// <permission cref="System.Security.PermissionSet">Everyone can access this method.</
permission>
    public static void Test()
    {
    }

    static void Main()
    {
    }
}
```

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<remarks> (C# プログラミング ガイド)

```
<remarks>description</remarks>
```

パラメータ
Description

メンバの説明。

解説

<remarks> タグを使用して、型の情報を追加し、<summary> で指定された情報を補足します。この情報は オブジェクト ブラウザ に表示されます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例
C#

```
// compile with: /doc:DocFileName.xml

/// <summary>
/// You may have some primary information about this class.
/// </summary>
/// <remarks>
/// You may have some additional information about this class.
/// </remarks>
public class TestClass
{
    /// text for Main
    static void Main()
    {
    }
}
```

参照
関連項目
ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)
概念
C# プログラミング ガイド

<returns> (C# プログラミング ガイド)

```
<returns>description</returns>
```

パラメータ
description

戻り値の説明。

解説

<returns> タグは、メソッド宣言のコメント内で使用して、戻り値を説明します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例
C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <returns>Returns zero.</returns>
    public static int GetZero()
    {
        return 0;
    }

    /// text for Main
    static void Main()
    {
    }
}
```

参照
関連項目
ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)
概念
C# プログラミング ガイド

<see> (C# プログラミング ガイド)

```
<see cref="member"/>
```

パラメータ

cref = "*member*"

現在のコンパイル環境からの呼び出しに利用できる、メンバまたはフィールドへの参照。コンパイラは、指定されたコード要素が存在するかどうかを確認し、*member* を出力 XML 内の要素名に渡します。*member* は、二重引用符 (" ") で囲みます。

解説

<see> タグを使用すると、テキスト内でリンクを指定できます。テキストが参照セクションに配置されていることを示すには、<seealso> を使用します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

<see> タグの使用例については、「<summary>」を参照してください。

使用例

ジェネリック型への **cref** 参照を作成する方法を次の例に示します。

C#

```
// compile with: /doc:DocFileName.xml

// the following cref shows how to specify the reference, such that,
// the compiler will resolve the reference
/// <summary cref="C{T}">
/// </summary>
class A { }

// the following cref shows another way to specify the reference,
// such that, the compiler will resolve the reference
// <summary cref="C &lt; T &gt;">

// the following cref shows how to hard-code the reference
/// <summary cref="T:C`1">
/// </summary>
class B { }

/// <summary cref="A">
/// </summary>
/// <typeparam name="T"></typeparam>
class C<T> { }
```

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<seealso> (C# プログラミング ガイド)

```
<seealso cref="member"/>
```

パラメータ

cref = "*member*"

現在のコンパイル環境からの呼び出しに利用できる、メンバまたはフィールドへの参照。コンパイラは、指定されたコード要素が存在することを確認してから、*member* を出力先 XML 内の要素名に渡します。*member* は、二重引用符 (" ") で囲みます。

ジェネリック型への cref 参照の作成方法については、「<see> (C# プログラミング ガイド)」を参照してください。

解説

<seealso> タグを使用すると、「参照」のセクションに示すテキストを指定できます。テキスト内でリンクを指定するには、<see> タグを使用します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

<seealso> タグの使用例については、「<summary>」を参照してください。

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<summary> (C# プログラミング ガイド)

```
<summary>description</summary>
```

パラメータ
description

オブジェクトの要約。

解説

<summary> タグは、型または型のメンバの説明に使用します。型の説明に補足情報を追加するには、<remarks> タグを使用します。

<summary> タグのテキストは、IntelliSense で、型に関する唯一の情報源になり、オブジェクト ブラウザ にも表示されます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例
C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <summary>DoWork is a method in the TestClass class.
    /// <para>Here's how you could make a second paragraph in a description. <see cref="Sys
tem.Console.WriteLine(System.String)"/> for information about output statements.</para>
    /// <seealso cref="TestClass.Main"/>
    /// </summary>
    public static void DoWork(int Int1)
    {
    }

    /// text for Main
    static void Main()
    {
    }
}
```

参照
関連項目
ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)
概念
C# プログラミング ガイド

<typeparam> (C# プログラミング ガイド)

```
<typeparam name="name">description</typeparam>
```

パラメータ

name

型パラメータの名前です。名前は、二重引用符 (" ") で囲みます。

description

型パラメータの説明です。

解説

ジェネリック型またはジェネリック メソッドを宣言して型パラメータを説明する場合、コメントに **<typeparam>** タグを使用します。ジェネリック型またはジェネリック メソッドの型パラメータごとに、タグを追加します。

詳細については、「ジェネリック (C# プログラミング ガイド)」を参照してください。

<typeparam> タグのテキストは、オブジェクト ブラウザ コード コメント Web レポートである、IntelliSense に表示されます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

<typeparam> の使用例については、「<typeparamref> (C# プログラミング ガイド)」を参照してください。

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

<typeparamref> (C# プログラミング ガイド)

```
<typeparamref name="name"/>
```

パラメータ

name

型パラメータの名前です。名前は、二重引用符 (" ") で囲みます。

解説

ジェネリック型とジェネリック メソッドの型パラメータの詳細については、「ジェネリック (C# プログラミング ガイド)」を参照してください。

ドキュメント ファイルを使用するときに何らかの方法で単語の書式 (斜体など) を指定するには、このタグを使用します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

C#

```
// compile with: /doc:DocFileName.xml

/// comment for class
public class TestClass
{
    /// <summary>
    /// Creates a new array of arbitrary type <typeparamref name="T"/>
    /// </summary>
    /// <typeparam name="T">The element type of the array</typeparam>
    public static T[] mkArray<T>(int n)
    {
        return new T[n];
    }
}
```

参照

関連項目

ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

<value> (C# プログラミング ガイド)

```
<value>property-description</value>
```

パラメータ
property-description

プロパティの説明。

解説

<value> タグを使用すると、プロパティが表す値の説明を記述できます。Visual Studio .NET 開発環境のコード ウィザードで追加したプロパティには、<summary> タグが追加されます。そのプロパティが表す値を記述するには、<value> タグを手動で追加する必要があります。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例
C#

```
// compile with: /doc:DocFileName.xml

/// text for class Employee
public class Employee
{
    private string _name;

    /// <summary>The Name property represents the employee's name.</summary>
    /// <value>The Name property gets/sets the _name data member.</value>
    public string Name
    {
        get
        {
            return _name;
        }
        set
        {
            _name = value;
        }
    }
}

/// text for class MainClass
public class MainClass
{
    /// text for Main
    static void Main()
    {
    }
}
```

参照
関連項目
ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)
概念
C# プログラミング ガイド

XML ファイルの処理 (C# プログラミング ガイド)

コンパイラは、ドキュメントを生成するためにタグ付けされたコードの構成体ごとに、ID 文字列を生成します。コードにタグを付ける方法については、「ドキュメント コメントとして推奨されるタグ」を参照してください。ID 文字列によって、構成体は一意に識別されます。XML ファイルを処理するプログラムは、ID 文字列を使用して、ドキュメントの適用対象となる .NET Framework メタデータ/リフレクション項目を識別できます。

XML ファイルは、コードの階層的表現ではなく、要素ごとに ID が生成されるフラットなリストです。

コンパイラは、次の規則に基づいて ID 文字列を生成します。

- ホワイトスペースは含めない。
- ID 文字列の最初の部分には、1 文字とコロンで、識別されるメンバの種類を示す。使用されるメンバ型は次のとおりです。

文字	説明
N	名前空間 名前空間にはドキュメント コメントを追加できませんが、サポートされている場合は、ドキュメント コメントへの cref 参照を作成できます。
T	型 : class、interface、struct、enum、delegate
F	フィールド
P	プロパティ (インデクサまたはその他のインデックス付きプロパティを含む)
M	メソッド (コンストラクタ、演算子などの特殊なメソッドを含む)
E	イベント
!	エラー文字列 エラーに続く文字列で、エラーの内容を示します。C# コンパイラは、解決できないリンクについてのエラー情報を生成します。

- 文字列の 2 番目の部分は、項目の完全限定名です。名前は、名前空間のルートから始まります。項目の名前、項目が含まれている型、および名前空間は、ピリオドで区切られます。名前自体にピリオドがある場合、名前のピリオドはハッシュ記号 ('#') に置き換えられます。項目の名前にはハッシュ記号がないことが前提です。たとえば、String コンストラクタの完全限定名は、"System.String.#ctor" になります。
- プロパティおよびメソッドについては、メソッドに引数がある場合は、引数のリストをかっこで囲み、メソッドに続けて指定します。引数がない場合は、かっこはありません。引数の区切り文字には、コンマを使用します。各引数のエンコードは、次に示す .NET Framework のシグネチャでの引数のエンコーディング方法にそのまま従います。
    - 基本型。通常の型 (ELEMENT_TYPE_CLASS または ELEMENT_TYPE_VALUETYPE) は、型の完全限定名で表されます。
    - ELEMENT_TYPE_I4、ELEMENT_TYPE_OBJECT、ELEMENT_TYPE_STRING、ELEMENT_TYPE_TYPEDBYREF、ELEMENT_TYPE_VOID など、組み込みの型は、対応する完全な型の完全限定名で表されます。たとえば、System.Int32 や System.TypedReference です。
    - ELEMENT_TYPE_PTR は、修飾される型に続けて '*' と表されます。
    - ELEMENT_TYPE_BYREF は、修飾される型に続けて '@' と表されます。
    - ELEMENT_TYPE_PINNED は、修飾される型に続けて '^' と表されます。C# コンパイラでは生成されません。
    - ELEMENT_TYPE_CMOD_REQ は、修飾される型に続けて "|" と修飾子クラスの完全限定名で表されます。C# コンパイラでは生成されません。
    - ELEMENT_TYPE_CMOD_OPT は、修飾される型に続けて '!' と修飾子クラスの完全限定名で表されます。
    - ELEMENT_TYPE_SZARRAY は、配列の要素型に続けて "[]" と表されます。

- ELEMENT_TYPE_GENERICARRAY は、配列の要素型に続けて "[?]" と表されます。C# コンパイラでは生成されません。
- ELEMENT_TYPE_ARRAY は、[*lowerbound*:*size*,*lowerbound*:*size*] の形式で表されます。ここで、コンマの個数はランク -1 個であり、各次元の下限とサイズは明らかな場合は、10 進数で表されます。下限またはサイズを、指定しない場合は省略します。特定の次元で下限およびサイズが省略されている場合は、その次元の ':' も省略されます。たとえば、ある 2 次元配列の下限が 1 で、サイズの指定がない場合は、[1:,1:] と表されます。
- ELEMENT_TYPE_FNPTR は、"=FUNC:*type*(*signature*)" と表されます。ここで、*type* は戻り値の型であり、*signature* はメソッドの引数です。引数がない場合は、かっこが省略されます。C# コンパイラでは生成されません。

次に示すシグネチャ コンポーネントは、オーバーロードされるメソッドの区別には使用されることがないため、表されません。

- 呼び出し規約
- 戻り値の型
- ELEMENT_TYPE_SENTINEL

- 変換演算子 (op_Implicit および op_Explicit) だけは、上記のエンコードと同様に、メソッドの戻り値が '~' としてエンコードされ、それに続けて戻り値の型が表されます。
- ジェネリック型では、型の名前の後に、バック チック (`)、ジェネリック型パラメータの数を示す数値が順に続きます。この例を次に示します。

  `<member name="T:SampleClass`2">` は、`public class SampleClass<T, U>` として定義されている型のタグです。

  パラメータとしてジェネリック型を受け取るメソッドでは、ジェネリック型パラメータは、バック チック付きの数値 (`0、`1 など) として指定されます。各数値は、型のジェネリック パラメータに対する、インデックス番号が 0 から始まる配列表記を表しています。

例

クラスおよびそのメンバの ID 文字列が生成される例を次に示します。

C#

```
///
///
namespace N  // "N:N"
{
    ///
    ///
    public unsafe class X    // "T:N.X"
    {
        public X(){}
        //----------------------------
        // The result of the above is:
        // "M:N.X.#ctor"


        /// <param name="i"></param>
        public X(int i){}
        //----------------------------
        // The result of the above is:
        // "M:N.X.#ctor(System.Int32)"


        ~X(){}
        //----------------------------
        // The result of the above is:
        // "M:N.X.Finalize", destructor's representation in metadata


        public string q;
        //----------------------------
        // The result of the above is:
        // "F:N.X.q"


        /// <returns></returns>
        public const double PI = 3.14;
        //----------------------------
```

```
// The result of the above is:
// "F:N.X.PI"


/// <param name="s"></param>
/// <param name="y"></param>
/// <param name="z"></param>
/// <returns></returns>
public int f(){return 1;}
//----------------------------
// The result of the above is:
// "M:N.X.f"


/// <param name="array1"></param>
/// <param name="array"></param>
/// <returns></returns>
public int bb(string s, ref int y, void * z){return 1;}
//----------------------------
// The result of the above is:
// "M:N.X.bb(System.String,System.Int32@,=System.Void*)"


/// <param name="x"></param>
/// <param name="xx"></param>
/// <returns></returns>
public int gg(short[] array1, int[,] array){return 0;}
//----------------------------
// The result of the above is:
// "M:N.X.gg(System.Int16[], System.Int32[0:,0:])"


public static X operator+(X x, X xx){return x;}
//----------------------------
// The result of the above is:
// "M:N.X.op_Addition(N.X,N.X)"


public int prop {get{return 1;} set{}}
//----------------------------
// The result of the above is:
// "P:N.X.prop"


public event D d;
//----------------------------
// The result of the above is:
// "E:N.X.d"


public int this[string s]{get{return 1;}}
//----------------------------
// The result of the above is:
// "P:N.X.Item(System.String)"


public class Nested{}
//----------------------------
// The result of the above is:
// "T:N.X.Nested"


public delegate void D(int i);
//----------------------------
// The result of the above is:
// "T:N.X.D"
```

```
        /// <param name="x"></param>
        /// <returns></returns>
        public static explicit operator int(X x){return 1;}
        //---------------------------
        // The result of the above is:
        // "M:N.X.op_Explicit(N.X)~System.Int32"
    }
}
```

参照

処理手順

XML ドキュメントのサンプル

関連項目

/doc (ドキュメント コメントの処理) (C# コンパイラ オプション)

概念

C# プログラミング ガイド

XML ドキュメント コメント (C# プログラミング ガイド)

ドキュメント タグの区切り記号 (C# プログラミング ガイド)

XML ドキュメント コメントでは区切り記号を使用して、ドキュメント コメントの開始位置と終了位置をコンパイラに示す必要があります。XML ドキュメント タグでは、次の種類の区切り記号を使用できます。

`///`

ドキュメント例で使用されています。この区切り記号は、Visual C# プロジェクト テンプレートで使用されます。

> **メモ :**
>
> Visual Studio IDE には、スマート コメント編集と呼ばれる機能があります。コード エディタで `///` 区切り記号を入力すると、<summary> タグと </summary> タグが自動的に挿入され、これらのタグの内側にカーソルが移動します。この機能には、プロジェクト プロパティ ページの [書式設定] ([オプション] ダイアログ ボックス - [テキスト エディタ] - [C#]/[J#])からアクセスします。

`/** */`

複数行の区切り記号

`/**` と `*/` を使用する場合は、書式に関する次の規則が適用されます。

- `/**` 区切り記号がある行では、行の残りの部分が空白の場合、その行はコメントとして扱われません。最初の文字が空白の場合、その空白文字は無視され、行の残りの部分が処理されます。それ以外の場合は、`/**` 区切り記号の後にある行のテキスト全体が、コメントの一部として扱われます。
- `*/` 区切り記号がある行では、`*/` 区切り記号までの部分がすべて空白の場合、その行が無視されます。それ以外の場合は、以下の箇条書きで説明するパターン一致規則に従って、その行の `*/` 区切り記号までのテキストがコメントの一部として扱われます。
- `/**` 区切り記号で始まる行が検出されると、コンパイラはその次の行から、空白 (省略可能) とアスタリスク (*) と、それに続く空白 (省略可能) から成る共通パターンが各行の先頭にあるかどうかを調べます。共通の文字セットが各行の先頭で見つかった場合、`/**` 区切り記号から `*/` 区切り記号がある行までのすべての行について (`*/` 区切り記号のある行も含まれる可能性があります) 、コンパイラはそのパターンを無視します。

次にいくつかの例を示します。

- 次の例では、`<summary>` で始まる行だけがコメントの一部として扱われます。次の 2 とおりの形式のタグは、どちらも同じコメントを生成します。

```
/**
<summary>text</summary>
*/
/** <summary>text</summary> */
```

- コンパイラは、2 行目と 3 行目の先頭にパターン " * " を適用して無視します。

```
/**
* <summary>
* text </summary>*/
```

- このコメントでは、2 行目にアスタリスクがないため、パターンが見つかりません。このため、2 行目のすべてのテキストと 3 行目の `*/` までのすべてのテキストが、コメントの一部として扱われます。

```
/**
* <summary>
text </summary>*/
```

- このコメントでは、2 つの原因でパターンが見つかりません。第 1 に、各行の先頭で、アスタリスクの前の空白の数が一致していません。第 2 に、5 行目がタブで始まっています。空白とタブは一致しません。このため、2 行目から `*/` までのすべてのテキストが、コメントの一部として扱われます。

```
/**
* <summary>
* text
* text2
* </summary>
*/
```

参照

処理手順

XML ドキュメントのサンプル

関連項目

/doc (ドキュメント コメントの処理) (C# コンパイラ オプション)

概念

C# プログラミング ガイド

XML ドキュメント コメント (C# プログラミング ガイド)

XML ドキュメント コメント (C# プログラミング ガイド)

方法：XML ドキュメント機能を使用する (C# プログラミング ガイド)

ドキュメント化された型の基本的な概要を次の例で示します。

使用例

C#

```
// compile with: /doc:DocFileName.xml

/// <summary>
/// Class level summary documentation goes here.</summary>
/// <remarks>
/// Longer comments can be associated with a type or member
/// through the remarks tag</remarks>
public class TestClass
{
    /// <summary>
    /// Store for the name property</summary>
    private string _name = null;

    /// <summary>
    /// The class constructor. </summary>
    public TestClass()
    {
        // TODO: Add Constructor Logic here
    }

    /// <summary>
    /// Name property </summary>
    /// <value>
    /// A value tag is used to describe the property value</value>
    public string Name
    {
        get
        {
            if (_name == null)
            {
                throw new System.Exception("Name is null");
            }
            return _name;
        }
    }

    /// <summary>
    /// Description for SomeMethod.</summary>
    /// <param name="s"> Parameter description for s goes here</param>
    /// <seealso cref="System.String">
    /// You can use the cref attribute on any tag to reference a type or member
    /// and the compiler will check that the reference exists. </seealso>
    public void SomeMethod(string s)
    {
    }

    /// <summary>
    /// Some other method. </summary>
    /// <returns>
    /// Return results are described through the returns tag.</returns>
    /// <seealso cref="SomeMethod(string)">
    /// Notice the use of the cref attribute to reference a specific method </seealso>
    public int SomeOtherMethod()
    {
        return 0;
    }

    /// <summary>
    /// The entry point for the application.
```

```
        /// </summary>
        /// <param name="args"> A list of command line arguments</param>
        static int Main(System.String[] args)
        {
            // TODO: Add code to start application here
            return 0;
        }
    }
```

サンプル出力

```
// This .xml file was generated with the previous code sample.
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>xmlsample</name>
    </assembly>
    <members>
        <member name="T:SomeClass">
            <summary>
            Class level summary documentation goes here.</summary>
            <remarks>
            Longer comments can be associated with a type or member
            through the remarks tag</remarks>
        </member>
        <member name="F:SomeClass.m_Name">
            <summary>
            Store for the name property</summary>
        </member>
        <member name="M:SomeClass.#ctor">
            <summary>The class constructor.</summary>
        </member>
        <member name="M:SomeClass.SomeMethod(System.String)">
            <summary>
            Description for SomeMethod.</summary>
            <param name="s"> Parameter description for s goes here</param>
            <seealso cref="T:System.String">
            You can use the cref attribute on any tag to reference a type or member
            and the compiler will check that the reference exists. </seealso>
        </member>
        <member name="M:SomeClass.SomeOtherMethod">
            <summary>
            Some other method. </summary>
            <returns>
            Return results are described through the returns tag.</returns>
            <seealso cref="M:SomeClass.SomeMethod(System.String)">
            Notice the use of the cref attribute to reference a specific method </seeal
so>
        </member>
        <member name="M:SomeClass.Main(System.String[])">
            <summary>
            The entry point for the application.
            </summary>
            <param name="args"> A list of command line arguments</param>
        </member>
        <member name="P:SomeClass.Name">
            <summary>
            Name property </summary>
            <value>
            A value tag is used to describe the property value</value>
        </member>
    </members>
</doc>
```

コードのコンパイル方法

この例をコマンド ラインからコンパイルするには、次のように入力します。

```
csc XMLsample.cs /doc:XMLsample.xml
```

これで、XML ファイルの XMLsample.xml が作成されます。このファイルは、ブラウザや **TYPE** コマンドによって参照できます。

堅牢性の高いプログラム

XML ドキュメントは /// で始まります。新しいプロジェクトを作成すると、ウィザードによって /// で始まる行が数行表示されます。コメントの処理には、次の制限があります。

- ドキュメントは、適切な XML であることが必要です。適切な XML でない場合は警告が表示され、ドキュメント ファイルにはエラーが発生したことを表すコメントが記録されます。
- 開発者は、独自のタグ セットを自由に作成できます。推奨されるタグについては、関連項目を参照してください。推奨されるタグには、次のような特殊な意味を持つタグがあります。
    - <param> タグは、パラメータの記述に使用します。このタグを使用した場合、コンパイラはパラメータが存在するかどうか、およびすべてのパラメータがドキュメントに記述されているかどうかを検査します。検査が失敗すると、コンパイラは警告を発行します。
    - **cref** 属性をタグに追加すると、コード要素を参照できます。コンパイラは、該当するコード要素が存在するかどうかを検査します。検査が失敗すると、コンパイラは警告を発行します。コンパイラは、**cref** 属性に記述されている型を探すときに、**using** ステートメントについて考慮します。
    - <summary> タグは、型やメンバについての追加情報を表示するために、Visual Studio の IntelliSense で使用されています。

メモ：
XML ファイルでは、型やメンバに関する完全な情報は提供されません。たとえば、XML ファイルには型情報が含まれていません。型やメンバに関する完全な情報を得るには、ドキュメント ファイルを実際の型やメンバのリフレクションと共に使用する必要があります。

参照

処理手順

XML ドキュメントのサンプル

関連項目

/doc (ドキュメント コメントの処理) (C# コンパイラ オプション)

概念

C# プログラミング ガイド

XML ドキュメント コメント (C# プログラミング ガイド)

XML ドキュメント コメント (C# プログラミング ガイド)

アプリケーション ドメイン (C# プログラミング ガイド)

アプリケーション ドメインは、実行中のアプリケーションを分離するための柔軟で安全な方法として利用できます。

通常、アプリケーション ドメインは、ランタイム ホストによって作成および操作されます。アプリケーションを実行しているときには、時折、実行を中断せずにコンポーネントをアンロードするなど、アプリケーションとアプリケーション ドメインをプログラムによって対話させることが必要になる場合があります。

アプリケーション ドメインは、アプリケーションおよびそのデータを相互に分離してセキュリティを向上します。単一のプロセスで複数のアプリケーション ドメインを実行でき、個別のプロセスに存在する分離レベルは同じです。複数のアプリケーションを単一のプロセス内で実行すると、サーバーのスケーラビリティが向上します。

次のコード例では、新しいアプリケーション ドメインを作成し、C ドライブに格納されている、作成済みのアセンブリ `HelloWorld.exe` を読み込んで実行します。

C#

```
static void Main()
{
    // Create an Application Domain:
    System.AppDomain newDomain = System.AppDomain.CreateDomain("NewApplicationDomain");

    // Load and execute an assembly:
    newDomain.ExecuteAssembly(@"c:\HelloWorld.exe");

    // Unload the application domain:
    System.AppDomain.Unload(newDomain);
}
```

アプリケーション ドメインの概要

アプリケーション ドメインには、次の特徴があります。

- アセンブリは、実行する前にアプリケーション ドメインに読み込む必要があります。詳細については、「アセンブリとグローバル アセンブリ キャッシュ (C# プログラミング ガイド)」を参照してください。
- 1 つのアプリケーション ドメインで障害が発生しても、他のアプリケーション ドメインで動作している別のコードには影響しません。
- プロセス全体を停止せずに個々のアプリケーションを終了し、コードをアンロードできます。個々のアセンブリや型はアンロードできず、アプリケーション ドメイン全体のみをアンロードできます。

関連項目

- アプリケーション ドメインの概要
- アプリケーション ドメイン
- アプリケーション ドメインとアセンブリ
- アプリケーション ドメインを使用したプログラミング
- アプリケーション ドメインとアセンブリを使用したプログラミング
- 別のアプリケーション ドメインでのコードの実行 (C# プログラミング ガイド)
- 方法 : アプリケーション ドメインを作成し、使用する (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.1 アプリケーションの起動

参照
概念
C# プログラミング ガイド
アセンブリとグローバル アセンブリ キャッシュ (C# プログラミング ガイド)

別のアプリケーション ドメインでのコードの実行 (C# プログラミング ガイド)

アセンブリがアプリケーション ドメインに読み込まれると、そのアセンブリに含まれるコードを実行できます。この操作を行うには、AssemblyLoad を使用するのが最も簡単です。これを使用すると、アセンブリが現在のアプリケーション ドメインに読み込まれ、アセンブリの既定のエントリ ポイントでコードの実行が開始されます。

アセンブリを別のアプリケーション ドメインに読み込むには、ExecuteAssembly または ExecuteAssemblyByName、あるいはこれらのメソッドをオーバーロードした別のバージョンを使用します。

また、既定のエントリ ポイント以外から始まる他のアセンブリを実行する場合は、このリモート アセンブリで新しい型を定義し、MarshalByRefObject から派生します。次に、CreateInstance を使用して、その型のインスタンスをアプリケーションで作成します。

次のファイルは、1 つの名前空間と 2 つのクラスから成るアセンブリを作成します。このアセンブリが既に構築され、`HelloWorldRemote.exe` という名前で C ドライブに格納されていることを前提にしています。

C#

```
// This namespace contains code to be called.
namespace HelloWorldRemote
{
    public class RemoteObject : System.MarshalByRefObject
    {
        public RemoteObject()
        {
            System.Console.WriteLine("Hello, World! (RemoteObject Constructor)");
        }
    }
    class Program
    {
        static void Main()
        {
            System.Console.WriteLine("Hello, World! (Main method)");
        }
    }
}
```

別のアプリケーションからコードにアクセスするには、アセンブリを現在のアプリケーション ドメインに読み込むか、または新しいアプリケーション ドメインを作成し、そこにアセンブリを読み込みます。**Assembly.LoadFrom** を使用してアセンブリを現在のアプリケーション ドメインに読み込む場合は、**Assembly.CreateInstance** を使用して、**RemoteObject** クラスのインスタンスをインスタンス化します。これにより、オブジェクト コンストラクタが実行されます。

C#

```
static void Main()
{
    // Load the assembly into the current appdomain:
    System.Reflection.Assembly newAssembly = System.Reflection.Assembly.LoadFrom(@"c:\Hello
WorldRemote.exe");

    // Instantiate RemoteObject:
    newAssembly.CreateInstance("HelloWorldRemote.RemoteObject");
}
```

アセンブリを別のアプリケーション ドメインに読み込む場合は、**AppDomain.ExecuteAssembly** を使用して既定のエントリ ポイントにアクセスするか、または **AppDomain.CreateInstance** を使用して **RemoteObject** クラスのインスタンスを作成します。インスタンスを作成すると、コンストラクタが実行されます。

C#

```
static void Main()
{
    System.AppDomain NewAppDomain = System.AppDomain.CreateDomain("NewApplicationDomain");

    // Load the assembly and call the default entry point:
```

```
    NewAppDomain.ExecuteAssembly(@"c:\HelloWorldRemote.exe");

    // Create an instance of RemoteObject:
    NewAppDomain.CreateInstanceFrom(@"c:\HelloWorldRemote.exe", "HelloWorldRemote.RemoteObj
ect");
}
```

アセンブリをプログラムによって読み込まない場合は、ソリューション エクスプローラの [参照の追加] を使用してアセンブリ HelloWorldRemote.exe を指定します。次に、アプリケーションの using ブロックに `using HelloWorldRemote;` ディレクティブを追加し、プログラムの **RemoteObject** 型を使用して、次のように **RemoteObject** オブジェクトのインスタンスを宣言します。

C#

```
static void Main()
{
    // This code creates an instance of RemoteObject, assuming HelloWorldRemote has been ad
ded as a reference:
    HelloWorldRemote.RemoteObject o = new HelloWorldRemote.RemoteObject();
}
```

参照

関連項目

アプリケーション ドメイン (C# プログラミング ガイド)

概念

C# プログラミング ガイド

アプリケーション ドメインの概要

アプリケーション ドメインとアセンブリ

アプリケーション ドメインを使用したプログラミング

その他の技術情報

アプリケーション ドメイン

アプリケーション ドメインとアセンブリを使用したプログラミング

方法 : アプリケーション ドメインを作成し、使用する (C# プログラミング ガイド)

アプリケーション ドメインには、セキュリティとパフォーマンスの強化を目的にコードを分離するメソッドが用意されています。詳細については、「アプリケーション ドメイン (C# プログラミング ガイド)」を参照してください。

アプリケーション ドメインの使用

1. アプリケーション ドメインを作成します。共通言語ランタイム ホストは、アプリケーションに応じたアプリケーション ドメインを自動的に作成します。詳細については、「方法 : アプリケーション ドメインを作成する」のトピックを参照してください。
2. アプリケーション ドメインを構成します。詳細については、「方法 : アプリケーション ドメインを構成する」を参照してください。
3. アセンブリをアプリケーション ドメインに読み込みます。詳細については、「方法 : アプリケーション ドメインにアセンブリを読み込む」を参照してください。
4. その他のアセンブリのコンテンツにアクセスします。詳細については、「別のアプリケーション ドメインでのコードの実行 (C# プログラミング ガイド)」を参照してください。
5. アプリケーション ドメインをアンロードします。詳細については、「方法 : アプリケーション ドメインをアンロードする」を参照してください。

参照

関連項目

アプリケーション ドメイン (C# プログラミング ガイド)

概念

C# プログラミング ガイド
アプリケーション ドメインの概要
アプリケーション ドメインとアセンブリ
アプリケーション ドメインを使用したプログラミング

その他の技術情報

アプリケーション ドメイン
アプリケーション ドメインとアセンブリを使用したプログラミング

アセンブリとグローバル アセンブリ キャッシュ (C# プログラミング ガイド)

アセンブリは、.NET Framework アプリケーションの基本ビルド ブロックです。たとえば、簡単な C# アプリケーションを構築する場合、Visual Studio は、単一のポータブル実行可能 (PE) ファイルの形式 (具体的には EXE または DLL) でアセンブリを作成します。

アセンブリには、固有の内部バージョン番号と、格納されているすべてのデータとオブジェクト型の詳細を表すメタデータが含まれます。詳細については、「アセンブリ マニフェスト」を参照してください。

アセンブリは、必要なときにだけ読み込まれます。使用されない場合は、読み込まれません。このため、大型のプロジェクトでアセンブリを使用すると、リソースを効率的に管理できます。

アセンブリには、1 つ以上のモジュールを含めることができます。たとえば、大型のプロジェクトは、数人の開発者がそれぞれ個別のモジュールを担当し、すべてのモジュールを組み合わせて 1 つのアセンブリを作成するように計画できます。モジュールの詳細については、「方法 : マルチファイル アセンブリをビルドする」を参照してください。

アセンブリの概要

アセンブリには、次のような特徴があります。

- アセンブリは、.exe ファイルまたは .dll ファイルとして実装されます。
- アセンブリをグローバル アセンブリ キャッシュに配置すると、複数のアプリケーションで共有できます。
- アセンブリをグローバル アセンブリ キャッシュに配置する場合は、厳密な名前を付ける必要があります。詳細については、「厳密な名前付きアセンブリ」を参照してください。
- アセンブリは、必要な場合にのみメモリに読み込まれます。
- リフレクションを使用すると、アセンブリに関する情報をプログラムによって取得できます。詳細については、「リフレクション (C# プログラミング ガイド)」を参照してください。
- 検査だけに限定してアセンブリを読み込む場合は、ReflectionOnlyLoadFrom などのメソッドを使用します。
- 同じアセンブリの 2 つのバージョンを 1 つのアプリケーションで使用できます。詳細については、「extern エイリアス (C# リファレンス)」を参照してください。

関連項目

詳細については以下を参照してください。

- フレンド アセンブリ (C# プログラミング ガイド)
- 方法 : アセンブリを他のアプリケーションと共有する (C# プログラミング ガイド)
- 方法 : アセンブリを読み込み、アンロードする (C# プログラミング ガイド)
- 方法 : ファイルがアセンブリであるかどうかを確認する (C# プログラミング ガイド)
- extern (C# リファレンス)
- 共通言語ランタイムのアセンブリ
- アセンブリの概要

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.2 プログラム構造
- 9.1 コンパイル単位

参照

関連項目

アプリケーション ドメイン (C# プログラミング ガイド)

概念

C# プログラミング ガイド

メタデータと PE ファイルの構造

フレンド アセンブリ (C# プログラミング ガイド)

アセンブリの内部型や内部メンバには、別のアセンブリからアクセスできます。

解説

フレンド アセンブリ機能を使用すると、内部メンバへのアクセスが可能になります。ただし、プライベート型とプライベート メンバにはアクセスできません。

アセンブリ (アセンブリ A) の内部型および内部メンバへのアクセスを別のアセンブリ (アセンブリ B) に付与するには、アセンブリ A で InternalsVisibleToAttribute 属性を使用します。

メモ :

アセンブリ (アセンブリ A) の内部型または内部メンバにアクセスするアセンブリ (アセンブリ B) をコンパイルするときは、**/out** コンパイラ オプションを使用して、出力ファイル (.exe または .dll) の名前を明示的に指定する必要があります。詳細については、「/out (出力ファイル名の設定) (C# コンパイラ オプション)」を参照してください。この操作が必要なのは、コンパイラが外部参照にバインドした時点では構築するアセンブリの名前がまだ生成されていないからです。

StrongNameIdentityPermission クラスを使用した場合も型を共有できますが、次の点が異なります。

- **StrongNameIdentityPermission** は、個々の型に適用され、フレンド アセンブリはアセンブリ全体に適用されます。
- アセンブリ B と共有する必要がある型がアセンブリ A に数多く存在する場合、**StrongNameIdentityPermission** では、すべての型を修飾する必要がありますが、フレンド アセンブリを使用すると、フレンド関係を 1 回宣言するだけで済みます。
- **StrongNameIdentityPermission** を使用する場合は、共有が必要な型を public と宣言する必要があります。フレンド アセンブリを使用する場合は、共有する型を internal と宣言します。
- アセンブリ内の非パブリック型にアクセスできる .netmodule を構築する方法については、「/moduleassemblyname (モジュールに対するフレンド アセンブリの指定) (C# コンパイラ オプション)」を参照してください。

使用例

次の例では、アセンブリの内部型と内部メンバに cs_friend_assemblies_2 というアセンブリがアクセスできるようにしています。

```
// cs_friend_assemblies.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
using System;

[assembly:InternalsVisibleTo("cs_friend_assemblies_2")]

// internal by default
class Class1
{
    public void Test()
    {
        Console.WriteLine("Class1.Test");
    }
}

// public type with internal member
public class Class2
{
    internal void Test()
    {
        Console.WriteLine("Class2.Test");
    }
}
```

次の例では、アセンブリが cs_friend_assemblies.dll アセンブリの内部型と内部メンバを使用します。

出力ファイル (**/out:cs_friend_assemblies_2.exe**) の名前を明示的に指定する必要があることに注意してください。

このアセンブリが内部型と内部メンバへのアクセスを別のアセンブリ (アセンブリ C) に提供した場合でも、アセンブリ C は、自動的には cs_friend_assemblies.dll アセンブリのフレンドになりません。

```
// cs_friend_assemblies_2.cs
// compile with: /reference:cs_friend_assemblies.dll /out:cs_friend_assemblies_2.exe
public class M
{
    static void Main()
    {
        // access an internal type
        Class1 a = new Class1();
        a.Test();

        Class2 b = new Class2();
        // access an internal member of a public type
        b.Test();
    }
}
```

出力

```
Class1.Test
Class2.Test
```

次の例は、内部型と内部メンバへのアクセスを、厳密な名前を持つアセンブリに提供する方法を示しています。

キーファイルを生成してパブリック キーを表示するには、次の sn.exe コマンドのシーケンスを使用します。詳細については、「厳密名ツール (Sn.exe)」を参照してください。

- sn -k friend_assemblies.snk // 厳密な名前キーを生成します。
- sn -p friend_assemblies.snk key.publickey // key.snk から key.publickey にパブリック キーを抽出します。
- sn -tp key.publickey // ファイル key.publickey に格納されているパブリック キーを表示します。

/keyfile を使用して、キーファイルをコンパイラに渡します。

```
// cs_friend_assemblies_3.cs
// compile with: /target:library /keyfile:friend_assemblies.snk
using System.Runtime.CompilerServices;

[assembly:InternalsVisibleTo("cs_friend_assemblies_4, PublicKey=0024000004800000940000000602
000000240000525341310004000001000100031d7b6f3abc16c7de526fd67ec2926fe68ed2f9901afbc5f1b6b4
28bf6cd9086021a0b38b76bc340dc6ab27b65e4a593fa0e60689ac98dd71a12248ca025751d135df7b98c5f9d09
172f7b62dabdd302b2a1ae688731ff3fc7a6ab9e8cf39fb73c60667e1b071ef7da5838dc009ae0119a9cbff2c58
1fc0f2d966b77114b2c4")]
class Class1
{
    public void Test()
    {
        System.Console.WriteLine("Class1.Test");
    }
}
```

次の例は、厳密な名前を持つアセンブリがアクセスできる内部型と内部メンバを使用する方法を示しています。

```
// cs_friend_assemblies_4.cs
// compile with: /keyfile:friend_assemblies.snk /reference:cs_friend_assemblies_3.dll /out:
cs_friend_assemblies_4.exe
public class M
{
    static void Main()
    {
        Class1 a = new Class1();
        a.Test();
    }
}
```

出力

```
Class1.Test
```

参照
概念
C# プログラミング ガイド
アセンブリとグローバル アセンブリ キャッシュ (C# プログラミング ガイド)

方法：ファイルがアセンブリであるかどうかを確認する (C# プログラミング ガイド)

ファイルが管理されていて、そのメタデータにアセンブリ エントリが含まれている場合に限り、そのファイルはアセンブリです。アセンブリとメタデータの詳細については、「アセンブリ マニフェスト」を参照してください。

ファイルがアセンブリであるかどうかを手動で確認するには

1. MSIL 逆アセンブラ (Ildasm.exe) を起動します。
2. テストするファイルを読み込みます。
3. ILDASM で、そのファイルがポータブル実行可能 (PE) ファイルではないと報告された場合、そのファイルはアセンブリでありません。詳細については、「方法 : アセンブリの内容を表示する」を参照してください。

ファイルがアセンブリであるかどうかをプログラムによって確認するには

1. GetAssemblyName メソッドを呼び出し、テストするファイルの完全パスと名前を渡します。
2. BadImageFormatException 例外がスローされた場合、ファイルはアセンブリでありません。

使用例

次の例では、DLL がアセンブリであるかどうかをテストして確認します。

C#

```
class TestAssembly
{
    static void Main()
    {
        try
        {
            System.Reflection.AssemblyName testAssembly =
                System.Reflection.AssemblyName.GetAssemblyName(@"C:\WINDOWS\system\avicap.d
ll");

            System.Console.WriteLine("Yes, the file is an Assembly.");
        }

        catch (System.IO.FileNotFoundException e)
        {
            System.Console.WriteLine("The file cannot be found.");
        }

        catch (System.BadImageFormatException e)
        {
            System.Console.WriteLine("The file is not an Assembly.");
        }

        catch (System.IO.FileLoadException e)
        {
            System.Console.WriteLine("The Assembly has already been loaded.");
        }
    }
}
```

GetAssemblyName メソッドはテスト ファイルを読み込み、情報が読み取られた時点で解放します。

出力

```
The file is not an Assembly.
```

参照

処理手順
例外のトラブルシューティング : System.BadImageFormatException
関連項目
AssemblyName
概念
C# プログラミング ガイド
アセンブリとグローバル アセンブリ キャッシュ (C# プログラミング ガイド)

方法 : アセンブリを読み込み、アンロードする (C# プログラミング ガイド)

プログラムから参照されるアセンブリは、ビルド時に自動的に読み込まれますが、実行時に特定のアセンブリを現在のアプリケーション ドメインに読み込むこともできます。詳細については、「アプリケーション ドメイン (C# プログラミング ガイド)」を参照してください。

個々のアセンブリをアンロードするには、アセンブリを含むすべてのアプリケーション ドメインを必ずアンロードする必要があります。アセンブリがスコープの外にある場合であっても、実際のアセンブリ ファイルは、これらのアセンブリ ファイルを含むすべてのアプリケーション ドメインがアンロードされるまでは読み込まれたままになります。

一部のアセンブリだけをアンロードする場合は、新しいアプリケーション ドメインを作成して、そのドメイン内でコードを実行した後で、そのアプリケーション ドメインをアンロードしてください。詳細については、「別のアプリケーション ドメインでのコードの実行 (C# プログラミング ガイド)」を参照してください。

アセンブリをアプリケーション ドメインに読み込むには

- AppDomain クラスと System.Reflection クラスに含まれるいくつかの load メソッドの 1 つを使用します。詳細については、「方法 : アプリケーション ドメインにアセンブリを読み込む」を参照してください。

アプリケーション ドメインをアンロードするには

- 個々のアセンブリをアンロードするには、アセンブリを含むすべてのアプリケーション ドメインを必ずアンロードする必要があります。**AppDomain** の `Unload` メソッドを使用してアプリケーション ドメインをアンロードします。詳細については、「方法 : アプリケーション ドメインをアンロードする」を参照してください。

参照

処理手順

方法 : アプリケーション ドメインにアセンブリを読み込む

関連項目

アプリケーション ドメイン (C# プログラミング ガイド)

概念

C# プログラミング ガイド

アセンブリとグローバル アセンブリ キャッシュ (C# プログラミング ガイド)

方法：アセンブリを他のアプリケーションと共有する (C# プログラミング ガイド)

アセンブリは、プライベートにすることも、共有することもできます。多くの場合、単純な C# プログラムは、他のアプリケーションで使用することを目的としていないので、既定でプライベート アセンブリで構成されます。

アセンブリを他のアプリケーションと共有するには、そのアセンブリを グローバル アセンブリ キャッシュ (GAC) に配置する必要があります。

アセンブリの共有

1. アセンブリを作成します。詳細については、「アセンブリの作成」を参照してください。
2. アセンブリに厳密な名前を付けます。詳細については、「方法 : 厳密な名前でアセンブリに署名する」を参照してください。
3. アセンブリにバージョン情報を割り当てます。詳細については、「アセンブリのバージョン管理」を参照してください。
4. アセンブリをグローバル アセンブリ キャッシュに追加します。詳細については、「方法 : グローバル アセンブリ キャッシュにアセンブリをインストールする」を参照してください。
5. アセンブリに含まれている型に他のアプリケーションからアクセスします。詳細については、「方法 : 厳密な名前のアセンブリを参照する」を参照してください。

参照

概念

C# プログラミング ガイド
厳密な名前付きアセンブリ

その他の技術情報

アセンブリを使用したプログラミング
アセンブリとグローバル アセンブリ キャッシュの使用
厳密な名前付きアセンブリの作成と使用

属性 (C# プログラミング ガイド)

属性は、型、メソッド、プロパティなどの宣言に関係する情報を C# コードに関連付けます。プログラム要素に関連付けられた属性は、リフレクションと呼ばれる方法によって実行時に照会できます。

属性には 2 つの形式があります。1 つは、共通言語ランタイムの基本クラス ライブラリで定義されている属性で、もう 1 つは、ユーザーが作成するカスタム属性で、コードに特別な情報を追加できます。この情報は、後でプログラムによって取得できます。

次の例では、System.Reflection.TypeAttributes.Serializable 属性を使用して、クラスに特定の特性を適用しています。

C#

```
[System.Serializable]
public class SampleClass
{
    // Objects of this type can be serialized.
}
```

属性の概要

属性には、次の特徴があります。

- 属性は、プログラムにメタデータを追加します。メタデータは、コンパイラ命令やデータの説明など、プログラムに埋め込まれた情報のことです。
- プログラムでは、リフレクションを使用して固有のメタデータを調べることができます。
  「リフレクションによる属性へのアクセス (C# プログラミング ガイド)」を参照してください。
- 属性は、一般に COM と対話する際に使用されます。

関連項目

詳細については、次のトピックを参照してください。

- 属性の使用 (C# プログラミング ガイド)
- カスタム属性の作成 (C# プログラミング ガイド)
- 属性の対象の明確化 (C# プログラミング ガイド)
- リフレクションによる属性へのアクセス (C# プログラミング ガイド)
- 方法 : 属性を使用して C/C++ の共用体を作成する (C# プログラミング ガイド)
- 共通の属性 (C# プログラミング ガイド)
- 属性のサンプル

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.12 属性
- 17 属性

参照

概念

C# プログラミング ガイド
リフレクション (C# プログラミング ガイド)
属性の概要
属性の一般的な使用方法

属性の使用 (C# プログラミング ガイド)

属性は、ほとんどすべての宣言に使用できます。ただし、一部の属性は、特定の種類の宣言に対してのみ使用できます。属性を指定するときは、属性の名前を角かっこで囲み、適用するエンティティの宣言の前に置きます。たとえば、**DllImport** 属性をメソッドに適用する場合は、次のように宣言します。

C#

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
extern static void SampleMethod();
```

詳細については、「DllImportAttribute クラス」を参照してください。

属性の多くにはパラメータがあり、位置、名前なし、または名前で特定されます。位置で決まるパラメータは、特定の順序で指定する必要があり、省略できません。名前付きのパラメータは省略でき、自由な順序で指定できます。位置で決まるパラメータを最初に指定します。たとえば、次の 3 つの属性は同じものです。

```
[DllImport("user32.dll")]
[DllImport("user32.dll", SetLastError=false, ExactSpelling=false)]
[DllImport("user32.dll", ExactSpelling=false, SetLastError=false)]
```

1 番目のパラメータである DLL 名は位置で決まるので、常に最初に指定する必要があります。それ以外のパラメータは名前付きです。この場合、名前付きパラメータはどちらも既定で false なので、省略できます。既定のパラメータ値については、各属性のドキュメントを参照してください。

1 つの宣言で複数の属性を指定できます。その場合、個別に角かっこで囲むか、または同じ角かっこの中に入れます。

C#

```
void MethodA([In][Out] ref double x) { }
void MethodB([Out][In] ref double x) { }
void MethodC([In, Out] ref double x) { }
```

一部の属性は、特定のエンティティに対して 2 回以上指定できます。次の例では、Conditional 属性を 2 回使用しています。

C#

```
[Conditional("DEBUG"), Conditional("TEST1")]
void TraceMethod()
{
    // ...
}
```

メモ :

規則では、属性名の終わりにはすべて "Attribute" が付きます。これにより、属性と .NET Framework の他の項目を区別できます。ただし、属性を使用するときには、必ずしもこのサフィックスを使用する必要はありません。たとえば、`[DllImport]` は `[DllImportAttribute]` と等価ですが、.NET Framework での属性の実際の名前は `DllImportAttribute` です。

参照

関連項目

属性の対象の明確化 (C# プログラミング ガイド)
カスタム属性の作成 (C# プログラミング ガイド)
リフレクションによる属性へのアクセス (C# プログラミング ガイド)
Attribute
System.Reflection

概念

C# プログラミング ガイド
リフレクション (C# プログラミング ガイド)
属性 (C# プログラミング ガイド)

カスタム属性の作成 (C# プログラミング ガイド)

属性クラスを定義することで、独自のカスタム属性を作成できます。属性クラスは、Attribute の直接的または間接的な派生クラスです。Attribute により、メタデータの中で属性の定義をすばやく簡単に識別できます。クラスと構造体にそれを記述したプログラマの名前でタグを付けるものとします。`Author` というカスタム属性クラスを定義します。

C#

```
[System.AttributeUsage(System.AttributeTargets.Class |
                       System.AttributeTargets.Struct)
]
public class Author : System.Attribute
{
    private string name;
    public double version;

    public Author(string name)
    {
        this.name = name;
        version = 1.0;
    }
}
```

クラス名は属性の名前の `Author` です。このクラスは **System.Attribute** から派生しているので、カスタム属性クラスです。コンストラクタのパラメータはカスタム属性の位置パラメータ (この場合は `name`) で、パブリックな読み取り/書き込みフィールドまたはプロパティは名前付きパラメータ (この場合は `version` が唯一の名前付きパラメータ) です。**AttributeUsage** 属性を使用して、**class** と **struct** の宣言に対してのみ `Author` 属性を有効にしていることに注意してください。

この新しい属性の使用方法は次のとおりです。

C#

```
[Author("H. Ackerman", version = 1.1)]
class SampleClass
{
    // H. Ackerman's code goes here...
}
```

AttributeUsage には名前付きパラメータの **AllowMultiple** があり、これを使ってカスタム属性が 1 回しか指定できない属性か、または複数回指定できる属性かを設定できます。

C#

```
[System.AttributeUsage(System.AttributeTargets.Class |
                       System.AttributeTargets.Struct,
                       AllowMultiple = true)  // multiuse attribute
]
public class Author : System.Attribute
```

C#

```
[Author("H. Ackerman", version = 1.1)]
[Author("M. Knott", version = 1.2)]
class SampleClass
{
    // H. Ackerman's code goes here...
    // M. Knott's code goes here...
}
```

参照

関連項目

属性の使用 (C# プログラミング ガイド)
属性の対象の明確化 (C# プログラミング ガイド)
リフレクションによる属性へのアクセス (C# プログラミング ガイド)
System.Reflection

概念

C# プログラミング ガイド
リフレクション (C# プログラミング ガイド)
属性 (C# プログラミング ガイド)

属性の対象の明確化 (C# プログラミング ガイド)

場合によっては、属性の対象 (属性が適用されるエンティティ) が明確でないことがあります。たとえば、次に示すメソッドの宣言では、SomeAttr 属性は、メソッド自体にもメソッドの戻り値にも適用される可能性があります。

C#

```
public class SomeAttr : System.Attribute { }

[SomeAttr]
int Method()
{
    return 0;
}
```

このような状況は、マーシャリング時によく発生します。あいまいさを排除するため、C# では宣言の種類ごとに既定の対象が決められています。属性の対象を明示的に指定することで、既定の対象をオーバーライドできます。

C#

```
// default: applies to method
[SomeAttr]
int Method1() { return 0; }

// applies to method
[method: SomeAttr]
int Method2() { return 0; }

// applies to return value
[return: SomeAttr]
int Method3() { return 0; }
```

上の例では、SomeAttr が実際に適用される対象と、SomeAttr に対して定義されている有効な対象は無関係であることに注意してください。つまり、これが戻り値だけに適用される属性として定義されている場合でも、対象を示す **return** を指定する必要があります。コンパイラには、**AttributeUsage** の情報を使用して属性の対象のあいまいさを解決する機能はありません。詳細については、「AttributeUsage (C# プログラミング ガイド)」を参照してください。

属性対象指定の構文は、次のとおりです。

```
[target : attribute-list]
```

パラメータ

target

assembly、field、event、method、module、param、property、return、type のいずれかです。

attribute-list

適用する属性のリストです。

次の表は、属性を指定できるすべての宣言をまとめたものです。宣言ごとに、その宣言の属性の対象を右側の列に示してあります。太字で示した対象が既定値です。

宣言	指定できる対象
assembly	**assembly**
module	**module**
class	**type**
struct	**type**

interface	**type**
enum	**type**
delegate	**type**、return
method	**method**、return
parameter	**param**
Field	**field**
property - indexer	**property**
property - get accessor	**method**、return
property - set accessor	**method**、param、return
event - field	**event**、field、method
event - property	**event**、property
event - add	**method**、param
event - remove	**method**、param

assembly レベルと module レベルの属性には既定の対象がありません。詳細については、「グローバル属性」を参照してください。

使用例

C#

```
using System.Runtime.InteropServices;
```

C#

```
[Guid("12345678-1234-1234-1234-123456789abc"), InterfaceType(ComInterfaceType.InterfaceIsIU
nknown)]
interface ISampleInterface
{
    [DispId(17)]  // set the DISPID of the method
    [return: MarshalAs(UnmanagedType.Interface)]  // set the marshaling on the return type
    object DoWork();
}
```

参照

関連項目

属性の使用 (C# プログラミング ガイド)
カスタム属性の作成 (C# プログラミング ガイド)
リフレクションによる属性へのアクセス (C# プログラミング ガイド)
System.Reflection
Attribute

概念

C# プログラミング ガイド
リフレクション (C# プログラミング ガイド)
属性 (C# プログラミング ガイド)

リフレクションによる属性へのアクセス (C# プログラミング ガイド)

カスタム属性を定義してソース コードで使用できたとしても、その情報を取得し、それに基づいて処理を実行する手段がなくては、価値のある機能とはいえません。C# では、リフレクション システムを使用して、カスタム属性で定義された情報を取得できます。ここで重要となるメソッドは **GetCustomAttributes** です。このメソッドは、ソース コードの属性に対応するオブジェクトの配列を実行時に返します。このメソッドには、オーバーロードされたバージョンがいくつかあります。詳細については、「Attribute」を参照してください。

C#

```
[Author("H. Ackerman", version = 1.1)]
class SampleClass
```

上の属性宣言は、概念的には下の式と同等です。

C#

```
Author anonymousAuthorObject = new Author("H. Ackerman");
anonymousAuthorObject.version = 1.1;
```

ただし、`SampleClass` に対して属性を問い合わせるまで、コードは実行されません。`SampleClass` に対して **GetCustomAttributes** を呼び出すと、`Author` オブジェクトが生成されて、上記のように初期化されます。クラスに他の属性がある場合は、他の属性オブジェクトが同じように作成されます。作成後、**GetCustomAttributes** は、配列内の `Author` オブジェクトとその他の属性オブジェクトを返します。この配列に対して反復処理を行うことで、各配列要素の種類に基づいて適用された属性を特定し、属性オブジェクトから情報を取得できます。

使用例

完全な例を次に示します。この例では、カスタム属性を定義し、いくつかのエンティティに適用し、リフレクションを使って情報を取得しています。

C#

```
[System.AttributeUsage(System.AttributeTargets.Class |
                       System.AttributeTargets.Struct,
                       AllowMultiple = true)  // multiuse attribute
]
public class Author : System.Attribute
{
    string name;
    public double version;

    public Author(string name)
    {
        this.name = name;
        version = 1.0;  // Default value
    }

    public string GetName()
    {
        return name;
    }
}

[Author("H. Ackerman")]
private class FirstClass
{
    // ...
}

// No Author attribute
private class SecondClass
{
    // ...
}

[Author("H. Ackerman"), Author("M. Knott", version = 2.0)]
```

```
private class ThirdClass
{
    // ...
}

class TestAuthorAttribute
{
    static void Main()
    {
        PrintAuthorInfo(typeof(FirstClass));
        PrintAuthorInfo(typeof(SecondClass));
        PrintAuthorInfo(typeof(ThirdClass));
    }

    private static void PrintAuthorInfo(System.Type t)
    {
        System.Console.WriteLine("Author information for {0}", t);
        System.Attribute[] attrs = System.Attribute.GetCustomAttributes(t);  // reflection

        foreach (System.Attribute attr in attrs)
        {
            if (attr is Author)
            {
                Author a = (Author)attr;
                System.Console.WriteLine("   {0}, version {1:f}", a.GetName(), a.version);
            }
        }
    }
}
```

出力

```
Author information for FirstClass
H. Ackerman, version 1.00
Author information for SecondClass
Author information for ThirdClass
H. Ackerman, version 1.00
M. Knott, version 2.00
```

参照

関連項目

属性の使用 (C# プログラミング ガイド)
属性の対象の明確化 (C# プログラミング ガイド)
カスタム属性の作成 (C# プログラミング ガイド)
System.Reflection
Attribute

概念

C# プログラミング ガイド
リフレクション (C# プログラミング ガイド)
属性 (C# プログラミング ガイド)

方法：属性を使用して C/C++ の共用体を作成する (C# プログラミング ガイド)

属性を使用すると、構造体のメモリ内での配置をカスタマイズできます。たとえば、**StructLayout(LayoutKind.Explicit)** 属性と **FieldOffset** 属性を使用すると、C/C++ の共用体と呼ばれるものを作成できます。

使用例

このコード セグメントでは、`TestUnion` のすべてのフィールドがメモリ内の同じ場所で開始されます。

C#

```
[System.Runtime.InteropServices.StructLayout(LayoutKind.Explicit)]
struct TestUnion
{
    [System.Runtime.InteropServices.FieldOffset(0)]
    public int i;

    [System.Runtime.InteropServices.FieldOffset(0)]
    public double d;

    [System.Runtime.InteropServices.FieldOffset(0)]
    public char c;

    [System.Runtime.InteropServices.FieldOffset(0)]
    public byte b;
}
```

フィールドが別の明示的に設定された場所で開始されるもう 1 つの例を次に示します。

C#

```
[System.Runtime.InteropServices.StructLayout(LayoutKind.Explicit)]
struct TestExplicit
{
    [System.Runtime.InteropServices.FieldOffset(0)]
    public long lg;

    [System.Runtime.InteropServices.FieldOffset(0)]
    public int i1;

    [System.Runtime.InteropServices.FieldOffset(4)]
    public int i2;

    [System.Runtime.InteropServices.FieldOffset(8)]
    public double d;

    [System.Runtime.InteropServices.FieldOffset(12)]
    public char c;

    [System.Runtime.InteropServices.FieldOffset(14)]
    public byte b;
}
```

2 つの **int** フィールドの `i1` と `i2` は、`lg` と同じメモリ位置を共有します。このような構造体レイアウトの制御は、プラットフォーム呼び出しのときに便利です。

参照

関連項目

属性の使用 (C# プログラミング ガイド)
属性の対象の明確化 (C# プログラミング ガイド)
カスタム属性の作成 (C# プログラミング ガイド)

リフレクションによる属性へのアクセス (C# プログラミング ガイド)
System.Reflection
Attribute
概念
C# プログラミング ガイド
リフレクション (C# プログラミング ガイド)
属性 (C# プログラミング ガイド)

共通の属性 (C# プログラミング ガイド)

ここで、C# プログラムで最もよく使用される属性について説明します。

このセクションの内容

- Conditional (C# プログラミング ガイド)
- Obsolete (C# プログラミング ガイド)
- グローバル属性 (C# プログラミング ガイド)
- AttributeUsage (C# プログラミング ガイド)

参照

関連項目

属性の使用 (C# プログラミング ガイド)
リフレクションによる属性へのアクセス (C# プログラミング ガイド)
System.Reflection
Attribute

概念

C# プログラミング ガイド
リフレクション (C# プログラミング ガイド)

Conditional (C# プログラミング ガイド)

プリプロセス ID に応じて、メソッドを実行するようにします。`Conditional` 属性は、ConditionalAttribute のエイリアスであり、メソッドまたは属性クラスに適用できます。

この例で、`Conditional` は、プログラム固有の診断情報について表示/非表示を切り替えるメソッドに適用されます。

```
      #define TRACE_ON
using System;
using System.Diagnostics;

public class Trace
{
    [Conditional("TRACE_ON")]
    public static void Msg(string msg)
    {
        Console.WriteLine(msg);
    }
}

public class ProgramClass
{
    static void Main()
    {
        Trace.Msg("Now in Main...");
        Console.WriteLine("Done.");
    }
}
```

`TRACE_ON` ID が定義されていないと、トレース出力は表示されません。

`Conditional` 属性は、リリース ビルドではなく、次のようにデバッグ ビルドでトレース機能とログ機能を有効にするときに、`DEBUG` ID でよく使用されます。

```
      [Conditional("DEBUG")]
static void DebugMethod()
{
}
```

解説

Conditional とマークされたメソッドが呼び出される場合、指定したプリプロセスのシンボルが存在するかどうかで、呼び出しが含まれるかどうかが決まります。シンボルが定義されている場合は呼び出しが行われ、定義されていない場合は呼び出しは行われません。メソッドを #if (C# リファレンス) や #endif (C# リファレンス) で囲むのではなく、次のように `Conditional` を使用すると、コードがわかりやすく洗練され、ミスが発生する可能性が低くなります。

```
#if DEBUG
void ConditionalMethod()
{
}
#endif
```

条件付きのメソッドは、クラスまたは構造体の宣言にあるメソッドにする必要があります。また、戻り値の型は void (C# リファレンス) にします。

複数 ID の使用

メソッドで複数の **Conditional** 属性が指定されている場合は、条件シンボルの 1 つでも定義されているとメソッドの呼び出しが行われます。つまり、複数のシンボルは論理 OR で結合されます。この例では、`A` または `B` が存在する場合、メソッドが呼び出されます。

C#

```
[Conditional("A"), Conditional("B")]
```

```
static void DoIfAorB()
{
    // ...
}
```

論理 AND 演算でシンボルを結合するには、条件付きメソッドを順番に定義します。たとえば、次の 2 つ目のメソッドは、`A` と `B` の両方が定義されている場合にのみ実行されます。

C#

```
[Conditional("A")]
static void DoIfA()
{
    DoIfAandB();
}

[Conditional("B")]
static void DoIfAandB()
{
    // Code to execute when both A and B are defined...
}
```

属性クラスでの Conditional の使用

Conditional 属性は、属性クラスの定義にも適用できます。この例のカスタム属性 `Documentation` では、DEBUG が定義されている場合にのみ、メタデータに情報が追加されます。

C#

```
[Conditional("DEBUG")]
public class Documentation : System.Attribute
{
    string text;

    public Documentation(string text)
    {
        this.text = text;
    }
}

class SampleClass
{
    // This attribute will only be included if DEBUG is defined.
    [Documentation("This method displays an integer.")]
    static void DoWork(int i)
    {
        System.Console.WriteLine(i.ToString());
    }
}
```

参照

関連項目

属性の対象の明確化 (C# プログラミング ガイド)
カスタム属性の作成 (C# プログラミング ガイド)
リフレクションによる属性へのアクセス (C# プログラミング ガイド)
System.Reflection
Attribute

概念

C# プログラミング ガイド
リフレクション (C# プログラミング ガイド)
属性 (C# プログラミング ガイド)

Obsolete (C# プログラミング ガイド)

Obsolete 属性によって、プログラム要素が、使用を推奨されない要素としてマークされます。要素に Obsolete とマークするたびに、属性の設定内容に応じて、警告やエラーが生成されます。次に例を示します。

```
      [System.Obsolete("use class B")]
class A
{
    public void Method() { }
}
class B
{
    [System.Obsolete("use NewMethod", true)]
    public void OldMethod()  { }
    public void NewMethod()  { }
}
```

この例で、Obsolete 属性は A クラスと B.OldMethod メソッドに適用されています。B.OldMethod に適用されている属性のコンストラクタで、2 つ目の引数が **true** に設定されているため、このメソッドを使用するとコンパイラ エラーになり、A クラスを使用すると単に警告が生成されます。一方で、B.NewMethod を呼び出しても警告やエラーは生成されません。

属性のコンストラクタで、1 つ目の引数として指定された文字列は、警告またはエラーの一部に表示されます。たとえば、次のコードを前の定義と共に使用すると、2 つの警告と 1 つのエラーが生成されます。

```
// Generates 2 warnings:
A a = new A();
// Generate no errors or warnings:
B b = new B();
b.NewMethod();
// Generates an error, terminating compilation:
b.OldMethod();
```

A クラスでは 2 つの警告が生成されます。1 つはクラス参照の宣言、もう 1 つはクラスのコンストラクタで生成されます。

Obsolete 属性は引数なしでも使用できますが、その項目の使用が推奨されない理由と代わりに使用する項目を引数に指定することをお勧めします。

Obsolete 属性は、シングルユースの属性です。属性を使用できる任意の要素に適用できます。Obsolete は、ObsoleteAttribute のエイリアスです。

参照

関連項目

属性の対象の明確化 (C# プログラミング ガイド)
カスタム属性の作成 (C# プログラミング ガイド)
リフレクションによる属性へのアクセス (C# プログラミング ガイド)
Attribute
System.Reflection

概念

C# プログラミング ガイド
リフレクション (C# プログラミング ガイド)
属性 (C# プログラミング ガイド)

グローバル属性 (C# プログラミング ガイド)

ほとんどの属性は、クラスやメソッドなど、特定の言語要素に結び付けられています。ただし、属性の中にはグローバルなものがあり、アセンブリまたはモジュール全体に適用されます。たとえば、AssemblyVersionAttribute 属性は、次のように、バージョン情報をアセンブリに埋め込むときに使用できます。

```
[assembly: AssemblyVersion("1.0.0.0")]
```

ソース コードでは、グローバル属性は、トップレベルの using ディレクティブより後、型または名前空間の宣言より前に指定します。グローバル属性は複数のソース ファイルに指定できますが、指定したファイルは、1 つのコンパイル パスでコンパイルする必要があります。

次に、よく使用される .NET Framework のアセンブリ レベル属性を示します。

AssemblyCompanyAttribute

AssemblyConfigurationAttribute

AssemblyCopyrightAttribute

AssemblyCultureAttribute

AssemblyDescriptionAttribute

AssemblyProductAttribute

AssemblyTitleAttribute

AssemblyTrademarkAttribute

この属性は、Visual Studio の Windows フォーム アプリケーション テンプレートに基づいて、プロジェクトで使用されます。このテンプレートには、`AssemblyInfo.cs` というファイルが含まれます。そのファイルに、この属性のインスタンスが指定されています。

```
[assembly: AssemblyTitle("WindowsApplication1")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Microsoft")]
[assembly: AssemblyProduct("WindowsApplication1")]
[assembly: AssemblyCopyright("Copyright © Microsoft 2005")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
```

メモ :

アセンブリを作成しないと、アセンブリ レベルの属性は無視されます。

アセンブリ署名の属性

以前のバージョンの Visual Studio では、厳密な名前でアセンブリに署名する処理は、このアセンブリ レベルの属性で実行されていました。

- AssemblyKeyFileAttribute
- AssemblyKeyNameAttribute
- AssemblyDelaySignAttribute

この方法もサポートされていますが、プロジェクト デザイナの署名のページを使用することをお勧めします。詳細については、[署名] ページ (プロジェクト デザイナ) および「方法 : アセンブリに署名する (Visual Studio)」を参照してください。

参照

関連項目

カスタム属性の作成 (C# プログラミング ガイド)
リフレクションによる属性へのアクセス (C# プログラミング ガイド)
System.Reflection

概念

C# プログラミング ガイド
属性 (C# プログラミング ガイド)

その他の技術情報

共通の属性 (C# プログラミング ガイド)

AttributeUsage (C# プログラミング ガイド)

カスタムの属性クラスの使用方法を決定します。AttributeUsage は、新しい属性の適用方法を制御するカスタムの属性定義に適用できる属性です。既定の設定を明示的に割り当てると、次のようになります。

```
[System.AttributeUsage(System.AttributeTargets.All,
                   AllowMultiple=false,
                   Inherited=true)]
class NewAttribute : System.Attribute { }
```

この例では、NewAttribute クラスは、属性を使用できるコード要素すべてに適用できますが、各要素に適用できるのは 1 つだけです。基本クラスに適用すると、派生クラスによって継承されます。

AllowMultiple 引数と Inherited 引数はオプションなので、次のコードでも同じ効果があります。

```
[System.AttributeUsage(System.AttributeTargets.All)]
class NewAttribute : System.Attribute { }
```

最初の AttributeUsage 引数は、AttributeTargets 列挙体の 1 つ以上の要素にする必要があります。複数の型を対象にする場合、次のように OR 接続できます。

```
using System;
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
class NewPropertyOrFieldAttribute : Attribute { }
```

次のように AllowMultiple 引数を **true** に設定すると、1 つの要素に結果の属性を複数適用できます。

```
using System;
[AttributeUsage(AttributeTargets.Class, AllowMultiple=true)]
class MultiUseAttr : Attribute { }

[MultiUseAttr]
[MultiUseAttr]
class Class1 { }

[MultiUseAttr, MultiUseAttr]
class Class2 { }
```

この場合、AllowMultiple が **true** に設定されているため、MultiUseAttr を繰り返し適用できます。複数の属性を適用する場合、どちらの形式でも有効です。

Inherited を **false** に設定した場合、属性を指定したクラスから派生するクラスには、属性が継承されません。次に例を示します。

```
using System;
[AttributeUsage(AttributeTargets.Class, Inherited=false)]
class Attr1 : Attribute { }

[Attr1]
class BClass { }

class DClass : BClass { }
```

この場合、Attr1 は 継承によって DClass に適用されません。

解説

AttributeUsage 属性は、シングルユースの属性です。同一のクラスに複数回適用することはできません。AttributeUsage は、AttributeUsageAttribute のエイリアスです。

詳細については、「リフレクションによる属性へのアクセス (C# プログラミング ガイド)」を参照してください。

使用例

次の例は、`Inherited` 引数と `AllowMultiple` 引数を `AttributeUsage` 属性に指定する影響と、クラスに適用されるカスタム属性の列挙方法に関するデモンストレーションです。

```
using System;

// Create some custom attributes:
[AttributeUsage(System.AttributeTargets.Class, Inherited=false)]
class A1 : System.Attribute { }

[AttributeUsage(System.AttributeTargets.Class)]
class A2 : System.Attribute { }

[AttributeUsage(System.AttributeTargets.Class, AllowMultiple=true)]
class A3 : System.Attribute { }

// Apply custom attributes to classes:
[A1,A2]
class BaseClass { }

[A3,A3]
class DerivedClass : BaseClass { }

public class TestAttributeUsage
{
    static void Main()
    {
        BaseClass b = new BaseClass();
        DerivedClass d = new DerivedClass();

        // Display custom attributes for each class.
        Console.WriteLine("Attributes on Base Class:");
        object[] attrs = b.GetType().GetCustomAttributes(true);
        foreach (Attribute attr in attrs)
        {
            Console.WriteLine(attr);
        }

        Console.WriteLine("Attributes on Derived Class:");
        attrs = d.GetType().GetCustomAttributes(true);
        foreach (Attribute attr in attrs)
        {
            Console.WriteLine(attr);
        }
    }
}
```

出力例

```
Attributes on Base Class:
A1
A2
Attributes on Derived Class:
A3
A3
A2
```

参照

関連項目

属性の使用 (C# プログラミング ガイド)
属性の対象の明確化 (C# プログラミング ガイド)
カスタム属性の作成 (C# プログラミング ガイド)
リフレクションによる属性へのアクセス (C# プログラミング ガイド)
Attribute
System.Reflection

概念

C# プログラミング ガイド

リフレクション (C# プログラミング ガイド)
属性 (C# プログラミング ガイド)

コレクション クラス (C# プログラミング ガイド)

.NET Framework では、データの格納と取得に特化されたクラスを使用できます。これらのクラスによって、スタック、キュー、リスト、およびハッシュ テーブルがサポートされます。コレクション クラスはほとんどが同じインターフェイスを実装します。これらのインターフェイスを継承して、より特化されたデータの格納の必要性に見合う新しいコレクション クラスを作成できます。

メモ :

Version 2.0 以降の .NET Framework を対象としたアプリケーションでは、System.Collections.Generic 名前空間内のジェネリック コレクション クラスを使用する必要があります。これらのコレクション クラスは、対応する非ジェネリック コレクション クラスよりもタイプ セーフ性と効率に優れています。

C#

```
ArrayList list = new ArrayList();
list.Add(10);
list.Add(20);
```

コレクション クラスの概要

コレクション クラスには、次の特徴があります。

- コレクション クラスは、System.Collections 名前空間または **System.Collections.Generic** 名前空間の一部として定義されます。
- ほとんどのコレクション クラスは、**ICollection**、**IComparer**、**IEnumerable**、**IList**、**IDictionary**、**IDictionaryEnumerator** の各インターフェイス、およびこれらのジェネリック インターフェイスから派生します。
- ジェネリック コレクション クラスを使用すると、タイプ セーフ性が向上し、場合によっては (特に値型の格納時)、パフォーマンスも向上します。詳細については、「ジェネリックの利点 (C# プログラミング ガイド)」を参照してください。

関連項目

- ジェネリック コレクションを使用する状況
- コレクションとデータ構造体
- コレクション クラスの選択
- コレクション内での比較と並べ替え
- コレクションの作成と操作
- 方法 : foreach を使用してコレクション クラスにアクセスする (C# プログラミング ガイド)
- コレクション クラスのサンプル

参照

概念

C# プログラミング ガイド

配列 (C# プログラミング ガイド)

方法 : foreach を使用してコレクション クラスにアクセスする (C# プログラミング ガイド)

次のコード例では、**foreach** と共に使用できる非ジェネリック コレクション クラスの記述方法を示します。クラスは、C ランタイム関数の **strtok** と同様に、文字列をトークン化します。

メモ :

この例では、ジェネリック コレクション クラスを使用できない場合にのみ推奨される方法を示します。ジェネリックは、Version 2.0 およびそれ以降の C# 言語と .NET Framework でサポートされています。**IEnumerable<T>** をサポートする (つまり、下記の問題を回避する) タイプ セーフなジェネリック コレクション クラスを実装する方法の例については、
「方法 : ジェネリック リストの反復子ブロックを作成する (C# プログラミング ガイド)」を参照してください。

次の例では、`Tokens` で、区切り記号の ' ' と '-' を使って "This is a sample sentence." という文を分割してトークン化し、**foreach** ステートメントによってこれらのトークンを列挙します。

C#

```
Tokens f = new Tokens("This is a sample sentence.", new char[] {' ','-'});

foreach (string item in f)
{
    System.Console.WriteLine(item);
}
```

内部的には、`Tokens` は `IEnumerator` と `IEnumerable` を実装する配列を使用しています。配列の列挙メソッドを使用する方法もありますが、この例では適切ではありません。

C# では、**foreach** と互換性を保つ目的で、コレクション クラスを IEnumerable と IEnumerator から継承することは厳密には必要ありません。要求された `GetEnumerator`、`MoveNext`、`Reset`、`Current` の各メンバがクラスに含まれている限り、コレクション クラスで **foreach** を使用できます。インターフェイスを省略すると、`Current` の戻り値の型を **object** よりも明確に定義できるので、結果としてタイプ セーフになります。

たとえば、上のコードを基にして、次の行を変更します。

```
// No longer inherits from IEnumerable:
public class Tokens
// Doesn't return an IEnumerator:
public TokenEnumerator GetEnumerator()
// No longer inherits from IEnumerator:
public class TokenEnumerator
// Type-safe: returns string, not object:
public string Current
```

この変更により、`Current` によって文字列が返されるため、**foreach** ステートメントで非互換型が使用されたことをコンパイラが検出できるようになります。

```
// Error: cannot convert string to int:
foreach (int item in f)
```

ただし、**IEnumerable** と **IEnumerator** を省略すると、コレクション クラスを他の共通言語ランタイム互換言語の **foreach** ステートメント (または同等のステートメント) と相互運用できなくなるので注意が必要です。

一方、**IEnumerable** と **IEnumerator** の継承、および明示的なインターフェイスの実装を併用することにより、C# のタイプ セーフ性と、他の共通言語ランタイム互換言語との相互運用性の 2 つの利点を享受できます。次に例を示します。

使用例

C#

```
using System.Collections;
```

```
// Declare the Tokens class:
public class Tokens : IEnumerable
{
    private string[] elements;

    Tokens(string source, char[] delimiters)
    {
        // Parse the string into tokens:
        elements = source.Split(delimiters);
    }

    // IEnumerable Interface Implementation:
    //   Declaration of the GetEnumerator() method
    //   required by IEnumerable
    public IEnumerator GetEnumerator()
    {
        return new TokenEnumerator(this);
    }


    // Inner class implements IEnumerator interface:
    private class TokenEnumerator : IEnumerator
    {
        private int position = -1;
        private Tokens t;

        public TokenEnumerator(Tokens t)
        {
            this.t = t;
        }

        // Declare the MoveNext method required by IEnumerator:
        public bool MoveNext()
        {
            if (position < t.elements.Length - 1)
            {
                position++;
                return true;
            }
            else
            {
                return false;
            }
        }

        // Declare the Reset method required by IEnumerator:
        public void Reset()
        {
            position = -1;
        }

        // Declare the Current property required by IEnumerator:
        public object Current
        {
            get
            {
                return t.elements[position];
            }
        }
    }


    // Test Tokens, TokenEnumerator
    static void Main()
    {
        // Testing Tokens by breaking the string into tokens:
        Tokens f = new Tokens("This is a sample sentence.", new char[] {' ','-'});
```

```
            foreach (string item in f)
            {
                System.Console.WriteLine(item);
            }
        }
    }
```

出力

```
This
is
a
sample
sentence.
```

参照

関連項目

System.Collections.Generic

概念

C# プログラミング ガイド

配列 (C# プログラミング ガイド)

コレクション クラス (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

例外と例外処理 (C# プログラミング ガイド)

C# 言語の例外処理機能を使用すると、プログラムの実行中に発生する不測の状況や例外的な状況に対処できます。例外処理では、**try**、**catch**、および **finally** の各キーワードを使用して、成功しない可能性のあるアクションの試行、エラーの処理、およびリソースの後処理を行います。例外は、共通言語ランタイム (CLR: Common Language Runtime)、サードパーティ ライブラリ、または **throw** キーワードを使用するアプリケーション コードによって生成できます。

次の例では、メソッドでゼロ除算をテストしてエラーをキャッチします。例外処理がなければ、このプログラムは、"DivideByZeroException はハンドルされませんでした。" エラーによって終了します。

```
int SafeDivision(int x, int y)
{
    try
    {
        return (x / y);
    }
    catch (System.DivideByZeroException dbz)
    {
        System.Console.WriteLine("Division by zero attempted!");
        return 0;
    }
}
```

例外の概要

例外には、次のような特徴があります。

- ゼロ除算警告や低メモリ警告などの例外的な状況がアプリケーションで発生すると、例外が生成されます。
- 例外をスローする可能性のあるステートメントの周囲に **try** ブロックを使用します。
- **try** ブロックで例外が発生したときに、関連する例外ハンドラが存在する場合、制御フローは直ちにそのハンドラにジャンプします。
- 特定の例外用の例外ハンドラがない場合、プログラムはエラー メッセージを表示して実行を停止します。
- catch ブロックに例外変数が定義されている場合は、これを使用して、発生した例外の詳細な型情報を取得できます。
- 例外を発生させる可能性のあるアクションは、**try** キーワードを使用して実行します。
- 例外ハンドラは、例外が発生したときに実行されるコード ブロックです。C# では、**catch** キーワードを使用して例外ハンドラを定義します。
- 例外は、**throw** キーワードを使用してプログラム側で意図的に生成できます。
- 例外オブジェクトには、呼び出し履歴の状態やエラーのテキスト説明など、エラーに関する詳細情報が含まれています。
- 例外がスローされた場合でも **finally** ブロックのコードは実行されるため、プログラム側でリソースを解放できます。

関連項目

例外と例外処理の詳細については、以下のトピックを参照してください。

- 例外の使用 (C# プログラミング ガイド)
- 例外処理
- 例外の作成とスロー (C# プログラミング ガイド)
- コンパイラにより生成された例外 (C# プログラミング ガイド)
- 方法 : try/catch を使用して例外を処理する (C# プログラミング ガイド)
- 方法 : finally を使用してクリーンアップ コードを実行する (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 8.9.5 throw ステートメント
- 8.10 try ステートメント
- 16 例外

参照

関連項目

C# のキーワード
throw (C# リファレンス)
try-catch (C# リファレンス)
try-finally (C# リファレンス)
try-catch-finally (C# リファレンス)

概念

C# プログラミング ガイド
例外の概要

その他の技術情報

例外のデザインのガイドライン
例外の処理とスロー

例外の使用 (C# プログラミング ガイド)

C# では、プログラムの実行時に発生したエラーは、例外という機能を通じてプログラムに伝えられます。例外は、エラーが発生したコードによってスローされ、エラーを修正できるコードによってキャッチされます。例外は、.NET Framework 共通言語ランタイム (CLR: Common Language Runtime) またはプログラムのコードによってスローできます。スローされた例外は、例外の **catch** ステートメントが見つかるまで呼び出し履歴をさかのぼります。キャッチされない例外は、システムが提供する汎用例外ハンドラによって処理され、ダイアログ ボックスが表示されます。

例外は、Exception から派生したクラスによって表されます。このクラスは、例外の型を識別し、その例外に関する詳細情報を含むプロパティを保持します。例外をスローするには、例外の派生クラスのインスタンスを作成し、必要に応じて例外のプロパティを設定してから、**throw** キーワードを使用してオブジェクトをスローします。次に例を示します。

C#

```
private static void TestThrow()
{
    System.ApplicationException ex =
        new System.ApplicationException("Demonstration exception in TestThrow()");

    throw ex;
}
```

例外がスローされると、現在のステートメントが **try** ブロックに存在するかどうかがランタイムによって確認されます。存在する場合は、**try** ブロックに関連付けられている **catch** ブロックが例外をキャッチできるかどうかを確認します。通常は、この **Catch** ブロックによって例外の型が指定されます。**catch** ブロックの型と、例外または例外の基本クラスの型が一致する場合、**catch** ブロックはメソッドを処理できます。次に例を示します。

C#

```
static void TestCatch()
{
    try
    {
        TestThrow();
    }
    catch (System.ApplicationException ex)
    {
        System.Console.WriteLine(ex.ToString());
    }
}
```

例外をスローするステートメントが **try** ブロックに存在しない場合、またはステートメントを含む **try** ブロックに適合する **catch** ブロックが存在しない場合、ランタイムは、呼び出し側のメソッドで **try** ステートメントと **catch** ブロックを探します。ランタイムは、呼び出し履歴を続けて、対応する **catch** ブロックを検索します。**catch** ブロックが見つかり、実行されると、**catch** ブロックの後の最初のステートメントに制御が渡されます。

try ステートメントには、複数の **catch** ブロックを含めることができます。例外を処理できる最初の **catch** ステートメントが実行され、その後の **catch** ステートメントは、対応していても無視されます。次に例を示します。

C#

```
static void TestCatch2()
{
    try
    {
        TestThrow();
    }
    catch (System.ApplicationException ex)
    {
        System.Console.WriteLine(ex.ToString());  // this block will be executed
    }
    catch (System.Exception ex)
    {
        System.Console.WriteLine(ex.ToString());  // this block will NOT be executed
    }
```

```
        System.Console.WriteLine("Done");  // this statement is executed after the catch block
    }
```

catch ブロックが実行される前に、対応する **catch** ブロックが存在する **try** ブロックを含む、ランタイムによって評価された **try** ブロックに **finally** ブロックがあるかどうかが確認されます。**Finally** ブロックがあると、中止された **try** ブロックによって残されることがあるあいまいな状態をクリーンアップしたり、ランタイムのガベージ コレクタがオブジェクトを終了させるのを待たずに外部リソース (グラフィック ハンドル、データベース接続、またはファイル ストリームなど) を解放したりできます。次に例を示します。

C#

```
  static void TestFinally()
  {
      System.IO.FileStream file = null;
      System.IO.FileInfo fileInfo = new System.IO.FileInfo("C:\\file.txt");

      try
      {
          file = fileInfo.OpenWrite();
          file.WriteByte(0xF);
      }
      finally
      {
          // Closing the file allows you to reopen it immediately - otherwise IOException is
  thrown.
          if (file != null)
          {
              file.Close();
          }
      }

      try
      {
          file = fileInfo.OpenWrite();
          System.Console.WriteLine("OpenWrite() succeeded");
      }
      catch (System.IO.IOException)
      {
          System.Console.WriteLine("OpenWrite() failed");
      }

  }
```

`WriteByte()` が例外をスローした場合は、`file.Close()` を呼び出さないと、ファイルを再度開こうとする 2 番目の **try** ブロックのコードは失敗し、ファイルはロックされたままになります。**finally** ブロックは、例外がスローされなくても実行されるので、上の例の **finally** ブロックにより、ファイルを適切に閉じて、エラーを回避できます。

例外がスローされた後に、対応する **catch** ブロックが呼び出し履歴に見つからない場合は、次のいずれかが生じます。

- 例外がデストラクタの内部で発生した場合、デストラクタは中止され、基本デストラクタ (存在する場合) が呼び出されます。
- 呼び出し履歴に静的コンストラクタまたは静的フィールド初期化子が含まれている場合、TypeInitializationException がスローされ、この新しい例外の InnerException プロパティに元の例外が割り当てられます。
- スレッドの開始位置に到達すると、スレッドは終了します。

参照
関連項目
例外と例外処理 (C# プログラミング ガイド)
概念
C# プログラミング ガイド

例外処理 (C# プログラミング ガイド)

C# では、例外の影響を受ける可能性があるコードを分割するために try ブロックを使用し、スローされた例外を処理するために catch ブロックを使用します。また、finally ブロックを使用すると、例外がスローされたかどうかとは無関係にコードを実行できます。try/catch 構成要素以降のコードは、例外がスローされた場合に実行されないため、このブロックが必要になることがあります。try ブロックは、catch ブロックまたは finally ブロックと一緒に使用する必要があり、複数の catch ブロックを含めることができます。次に例を示します。

C#

```
try
{
    // Code to try here.
}
catch (System.Exception ex)
{
    // Code to handle exception here.
}
```

C#

```
try
{
    // Code to try here.
}
finally
{
    // Code to execute after try here.
}
```

C#

```
try
{
    // Code to try here.
}
catch (System.Exception ex)
{
    // Code to handle exception here.
}
finally
{
    // Code to execute after try (and possibly catch) here.
}
```

catch ブロックも **finally** ブロックも存在しない **try** ステートメントは、コンパイラ エラーになります。

catch ブロック

catch ブロックでは、キャッチする例外の種類を指定できます。この種類 (例外フィルタと呼びます) は、Exception 型またはその派生型にする必要があります。アプリケーション定義の例外は、ApplicationException から派生させる必要があります。

異なる例外フィルタを持つ複数の **catch** ブロックを使用できます。例外がスローされるたびに複数の **catch** ブロックが上から下まで評価されますが、実行される **catch** ブロックは 1 つだけです。スローされた例外の型または基本クラスを正確に指定している最初の **catch** ブロックが実行されます。一致する例外フィルタを指定する catch ブロックが存在しない場合は、フィルタを指定しない **catch** ブロック (存在する場合) が実行されます。最も限定的な (最派生) 例外クラスを持つ **catch** ブロックを最初に配置することが重要です。

例外は、次の場合にキャッチする必要があります。

- 例外がスローされた原因を明確に把握でき、FileNotFoundException オブジェクトをキャッチしたり、ユーザーに新しいファイル名の入力を求めたりするような特定の回復措置を実装できる場合。
- より限定的な新しい例外を生成し、スローできる場合。次に例を示します。

C#

```
int GetInt(int[] array, int index)
{
    try
    {
        return array[index];
    }
    catch(System.IndexOutOfRangeException e)
    {
        throw new System.ArgumentOutOfRangeException(
            "Parameter index is out of range.");
    }
}
```

- 例外を部分的に処理する場合。たとえば、**catch** ブロックを使用すると、エラー ログにエントリを追加できますが、例外を再スローして、その後の例外処理を可能にすることもできます。次に例を示します。

C#

```
try
{
    // try to access a resource
}
catch (System.UnauthorizedAccessException e)
{
    LogError(e);  // call a custom error logging procedure
    throw e;      // re-throw the error
}
```

finally ブロック

finally ブロックでは、**try** ブロックで実行されたアクションをクリーンアップできます。**finally** ブロックが存在する場合、このブロックは、**try** ブロックと **catch** ブロックの後に実行されます。**finally** ブロックは、例外がスローされたかどうかや、例外の型に一致する **catch** ブロックが検出されたかどうかとは無関係に常に実行されます。

finally ブロックを使用すると、ガベージ コレクタが実行時にオブジェクトを終了するのを待たずに、ファイル ストリーム、データベース接続、グラフィックス ハンドルなどのリソースを解放できます。詳細については、「using ステートメント (C# リファレンス)」を参照してください。

次の例では、**finally** ブロックを使用して、**try** ブロックで開かれたファイルを閉じます。ファイルを閉じる前に、ファイル ハンドルの状態がチェックされることに注意してください。**try** ブロックでファイルを開くことができなかった場合、ファイル ハンドルは **null** に設定されます。また、ファイルが正常に開かれ、例外がスローされない場合も、**finally** ブロックは実行され、開いているファイルを閉じます。

C#

```
System.IO.FileStream file = null;
System.IO.FileInfo fileinfo = new System.IO.FileInfo("C:\\file.txt");
try
{
    file = fileinfo.OpenWrite();
    file.WriteByte(0xF);
}
finally
{
    // check for null because OpenWrite
    // might have failed
    if (file != null)
    {
        file.Close();
    }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 16 例外
- 8.9.5 throw ステートメント
- 8.10 try ステートメント

参照

関連項目

例外と例外処理 (C# プログラミング ガイド)
try-catch (C# リファレンス)
try-finally (C# リファレンス)
try-catch-finally (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

例外の作成とスロー (C# プログラミング ガイド)

例外は、プログラムの実行中にエラーが発生したことを示すために使用されます。エラーを説明する例外オブジェクトは、作成後、throw キーワードによりスローされます。ランタイムは、対応する最も近い例外ハンドラを検索します。

例外は、次の場合にスローされる必要があります。

- メソッドが、定義されている機能を完了できない場合。たとえば、メソッドのパラメータの値が無効な場合は、次のような例外をスローします。

  C#

```
static void CopyObject(SampleClass original)
{
    if (original == null)
    {
        throw new System.ArgumentException("Parameter cannot be null", "original");
    }

}
```

- オブジェクトの状態に照らして不適切な呼び出しがオブジェクトに対して行われた場合。たとえば、読み取り専用ファイルへの書き込みを試みた場合は、次のような例外をスローします。オブジェクトの状態により操作が許可されない場合、InvalidOperationException のインスタンスまたはこのクラスの派生に基づくオブジェクトがスローされます。**InvalidOperationException** オブジェクトをスローするメソッドの例を次に示します。

  C#

```
class ProgramLog
{
    System.IO.FileStream logFile = null;
    void OpenLog(System.IO.FileInfo fileName, System.IO.FileMode mode) {}

    void WriteLog()
    {
        if (!this.logFile.CanWrite)
        {
            throw new System.InvalidOperationException("Logfile cannot be read-only");
        }
        // Else write data to the log and return.
    }
}
```

- メソッドの引数が原因で例外が発生した場合。この場合、元の例外をキャッチして、ArgumentException インスタンスを作成する必要があります。元の例外は、InnerException パラメータとして **ArgumentException** のコンストラクタに渡す必要があります。

  C#

```
static int GetValueFromArray(int[] array, int index)
{
    try
    {
        return array[index];
    }
    catch (System.IndexOutOfRangeException ex)
    {
        System.ArgumentException argEx = new System.ArgumentException("Index is out of
range", "index", ex);
```

```
            throw argEx;
        }
    }
```

例外には、StackTrace というプロパティがあります。この文字列には、現在の呼び出し履歴にあるメソッドの名前と、各メソッドについて例外がスローされたファイル名と行番号が含まれます。**StackTrace** オブジェクトは、**throw** ステートメントの位置で CLR によって自動的に作成されるため、例外は、スタック トレースが開始される位置からスローする必要があります。

例外にはいずれも Message というプロパティがあります。この文字列は、例外の原因を説明するように設定する必要があります。セキュリティ関連情報をメッセージ テキストに含めないように注意してください。**Message** の他に、**ArgumentException** には ParamName というプロパティがあります。このプロパティは、例外がスローされる原因になった引数の名前に設定する必要があります。プロパティ Set アクセス操作子の場合は、**ParamName** を `value` に設定する必要があります。

パブリック メソッドとプロテクト メソッドは、意図された機能を完了できない場合に例外をスローする必要があります。スローされる例外クラスは、エラー状態に最も明確に適合する例外である必要があります。これらの例外は、クラス機能の一部として記述する必要があり、派生クラスや元のクラスの更新では、下位互換性を確保するために同じ動作を保持する必要があります。

例外は、正常実行の一部としてプログラムのフローの変更には使用せず、エラー条件の報告および処理にのみ使用してください。そして、戻り値またはパラメータとして返されるのではなく、スローされるようにしてください。また、**System.Exception**、**System.SystemException**、**NullReferenceException**、または **IndexOutOfRangeException** を意図的にスローしないでください。

例外クラスの定義

プログラムでは、**System** 名前空間で定義済みの、上記以外の任意の例外クラスをスローできます。また、ApplicationException から派生させて固有の例外クラスを作成することもできます。派生クラスでは、少なくとも 4 つのコンストラクタを定義する必要があります。1 つ目は既定のコンストラクタ、2 つ目はメッセージ プロパティを設定するコンストラクタ、3 つ目は **Message** と **InnerException** の両方のプロパティを設定するコンストラクタです。そして 4 つ目のコンストラクタは、例外をシリアル化するのに使用します。新しい例外クラスは、シリアル化できるクラスにする必要があります。この例を次に示します。

C#

```
public class InvalidDepartmentException : System.ApplicationException
{
    public InvalidDepartmentException() {}
    public InvalidDepartmentException(string message) {}
    public InvalidDepartmentException(string message, System.Exception inner) {}

    // Constructor needed for serialization
    // when exception propagates from a remoting server to the client.
    protected InvalidDepartmentException(System.Runtime.Serialization.SerializationInfo inf
o,
        System.Runtime.Serialization.StreamingContext context) {}
}
```

新しいプロパティを例外クラスに追加するのは、それらが提供するデータが例外を解決する上で有効な場合に限定する必要があります。新しいプロパティを派生例外クラスに追加した場合は、`ToString()` をオーバーライドして追加情報を返す必要があります。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 8.9.5 throw ステートメント
- 8.10 try ステートメント
- 16 例外

参照

関連項目

例外と例外処理 (C# プログラミング ガイド)

例外処理 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

コンパイラにより生成された例外 (C# プログラミング ガイド)

例外には、基本操作が失敗すると、.NET Framework の共通言語ランタイム (CLR) により自動的にスローされるものがあります。これらの例外とそれぞれのエラー条件を以下に示します。

例外	説明
ArithmeticException	DivideByZeroException や OverflowException など、算術演算中に発生する例外の基本クラスです。
ArrayTypeMismatchException	要素の実際の型が、配列の実際の型と互換性がないために、要素を配列に格納できなかった場合にスローされます。
DivideByZeroException	整数値を 0 で除算しようとしたときにスローされます。
IndexOutOfRangeException	0 未満のインデックス、または配列の範囲外のインデックスを使用して、配列のインデックス付けをしようとしたときにスローされます。
InvalidCastException	基本型からインターフェイスまたは派生型への明示的変換が、実行時に失敗した場合にスローされます。
NullReferenceException	値が null のオブジェクトを参照しようとするとスローされます。
OutOfMemoryException	新しい演算子を使用してメモリを割り当てようとして失敗した場合にスローされます。共通言語ランタイムの使用可能なメモリ容量が不足しています。
OverflowException	`checked` コンテキストで算術演算がオーバーフローしたときにスローされます。
StackOverflowException	保留状態のメソッド呼び出しが多すぎて、実行スタックに空きがなくなったときにスローされます。一般的には、再帰が深いか、または無限再帰の場合に発生します。
TypeInitializationException	静的コンストラクタが例外をスローし、その例外をキャッチする、対応する `catch` 句が存在しない場合にスローされます。

参照

関連項目

例外と例外処理 (C# プログラミング ガイド)
例外処理 (C# プログラミング ガイド)
try-catch (C# リファレンス)
try-finally (C# リファレンス)
try-catch-finally (C# リファレンス)

概念

C# プログラミング ガイド

方法：try/catch を使用して例外を処理する (C# プログラミング ガイド)

try-catch ブロックの目的は、作業コードによって生成された例外をキャッチし、処理することです。例外によっては、**catch** ブロックで処理し、例外を再スローせずに問題を解決できるものもありますが、多くの場合、適切な例外がスローされるようにする必要があります。

使用例

この例の IndexOutOfRangeException は最も適切な例外ではありません。呼び出し元から `index` 引数が渡されてエラーが発生したため、より適切なメソッドは ArgumentOutOfRangeException になります。

C#

```
class TestTryCatch
{
    static int GetInt(int[] array, int index)
    {
        try
        {
            return array[index];
        }
        catch (System.IndexOutOfRangeException e)  // CS0168
        {
            System.Console.WriteLine(e.Message);
            //set IndexOutOfRangeException to the new exception's InnerException
            throw new System.ArgumentOutOfRangeException("index parameter is out of range."
, e);
        }
    }
}
```

説明

例外を発生させるコードが **try** ブロックに囲まれています。そのすぐ後に追加されている **catch** ステートメントが、`IndexOutOfRangeException` が発生した場合に処理します。**catch** ブロックは、`IndexOutOfRangeException` を処理し、代わりにより適切な `ArgumentOutOfRangeException` 例外をスローします。呼び出し元にできるだけ多くの情報を提供するため、元の例外を新しい例外の InnerException として指定することをお勧めします。**InnerException** プロパティは readonly なので、コンストラクタまたは新しい例外で割り当てる必要があります。

参照

関連項目

例外と例外処理 (C# プログラミング ガイド)

例外処理 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

方法：finally を使用してクリーンアップ コードを実行する (C# プログラミング ガイド)

finally ステートメントの目的は、例外がスローされた場合でも、オブジェクト (一般に外部リソースを保持しているオブジェクト) に対して必要なクリーンアップを即座に実行できるようにすることです。次のように、共通言語ランタイムによってオブジェクトがガベージ コレクションされるまで待機する代わりに、オブジェクトを使用した後すぐに FileStream で Close を呼び出すというのも、このようなクリーンアップの一例です。

C#

```
static void CodeWithoutCleanup()
{
    System.IO.FileStream file = null;
    System.IO.FileInfo fileInfo = new System.IO.FileInfo("C:\\file.txt");

    file = fileInfo.OpenWrite();
    file.WriteByte(0xF);

    file.Close();
}
```

使用例

上のコードを **try-catch-finally** ステートメントに変えるには、次のようにクリーンアップ コードを作業コードから分離します。

C#

```
static void CodeWithCleanup()
{
    System.IO.FileStream file = null;
    System.IO.FileInfo fileInfo = null;

    try
    {
        fileInfo = new System.IO.FileInfo("C:\\file.txt");

        file = fileInfo.OpenWrite();
        file.WriteByte(0xF);
    }
    catch(System.Exception e)
    {
        System.Console.WriteLine(e.Message);
    }
    finally
    {
        if (file != null)
        {
            file.Close();
        }
    }
}
```

`OpenWrite()` 呼び出しの前に **try** ブロック内で例外がいつでも発生する可能性があるため、または `OpenWrite()` 呼び出し自体が失敗する可能性があるため、ファイルを閉じようとしたときにそのファイルが開いているという保証はありません。**finally** ブロックは、**Close** メソッドを呼び出す前に、**FileStream** オブジェクトが **null** でないことを確認するチェックを行います。この **null** チェックを行わないと、**finally** ブロックは独自に NullReferenceException をスローする可能性がありますが、finally ブロックにおける例外のスローはできるだけ回避する必要があります。

データベース接続も、**finally** ブロックで閉じられる対象になります。データベース サーバーに許容される接続数は限られている場合があるため、データベース接続はできるだけ早く閉じる必要があります。接続を閉じる前に例外がスローされる場合でも、ガベージ コレクションを待機するより、**finally** ブロックを使用することをお勧めします。

参照

関連項目

例外と例外処理 (C# プログラミング ガイド)
例外処理 (C# プログラミング ガイド)
using ステートメント (C# リファレンス)
try-catch (C# リファレンス)
try-finally (C# リファレンス)
try-catch-finally (C# リファレンス)
概念
C# プログラミング ガイド

方法：CLS 準拠でない例外をキャッチする

C++、CLI を含む一部の .NET 言語では、オブジェクトは、Exception から派生していない例外をスローできます。このような例外をCLS 準拠でない例外、または Exception クラスから派生していない例外と呼びます。Visual C# では、CLS 準拠でない例外をスローすることはできませんが、これらの例外を次の 2 とおりの方法でキャッチできます。

- `catch (Exception e)` ブロック内で RuntimeWrappedException としてキャッチ。

  既定では、Visual C# アセンブリは CLS 準拠でない例外をラップされた例外としてキャッチします。WrappedException プロパティからアクセスできる元の例外にアクセスする必要がある場合は、この方法を使用します。この方法で例外をキャッチする方法については、次の例を参照してください。

- `catch (Exception)` ブロックか `catch (Exception e)` ブロックの後ろに置かれた汎用の catch ブロック (例外の型が指定されていない catch ブロック) 内でキャッチ。

  CLS 準拠でない例外に対して何らかのアクションを実行する必要があり、例外情報にアクセスする必要がない場合は、この方法を使用します。既定では、共通言語ランタイムはすべての例外をラップします。この動作を無効にするには、アセンブリレベルの属性 `[assembly: RuntimeCompatibilityAttribute(WrapNonExceptionThrows = false)]` をコード ファイル (厳密には AssemblyInfo.cs ファイル) に追加します。

CLS 準拠でない例外をキャッチするには

1. `catch(Exception e) block`内で **as** キーワードを使用して、`e` を **RuntimeWrappedException** にキャストできるかどうかをテストします。
2. **WrappedException** プロパティから元の例外にアクセスします。

使用例

次の例では、C++ または CLR で記述されたクラス ライブラリからスローされた CLS 準拠でない例外をキャッチする方法を示します。この例では、Visual C# クライアント コードは、スローされる例外の型が System.String であることを事前に認識しています。コードからこの型にアクセスできる場合は、**WrappedException** プロパティを元の型にキャストできます。

```
// Class library written in C++/CLR
    ThrowNonCLS.Class1 myClass = new ThrowNonCLS.Class1();

    try
    {
     // throws gcnew System::String(
     // "I do not derive from System.Exception!");
     myClass.TestThrow();
    }


    catch (Exception e)
    {
     RuntimeWrappedException rwe = e as RuntimeWrappedException;
     if (rwe != null)
     {
       String s = rwe.WrappedException as String;
       if (s != null)
       {
         Console.WriteLine(s);
       }
     }
     else
     {
        // handle other System.Exception types
     }
    }
```

参照

関連項目

例外と例外処理 (C# プログラミング ガイド)
RuntimeWrappedException

相互運用性 (C# プログラミング ガイド)

相互運用機能によって、アンマネージ コードへの既存の投資を保護し活用できます。共通言語ランタイム (CLR: Common Language Runtime) の制御の下で動作するコードを "マネージ コード"、CLR の外部で動作するコードを "アンマネージ コード" と言います。アンマネージ コードの例としては、COM、COM+、C++ の各コンポーネント、ActiveX コンポーネント、および Win32 API があります。

.NET Framework は、プラットフォーム呼び出しサービス、System.Runtime.InteropServices 名前空間、CLR、および COM 相互運用性 (COM interop) を通じてアンマネージ コードとの相互運用を可能にします。

- マネージ コードからアンマネージ API を使用するには、プラットフォーム呼び出しを使用する方法と、C++ の IJW (It Just Works) を使用する方法の 2 つがあります。プラットフォーム呼び出しを使用すると、マネージ コードで、Win32 API やカスタム DLL などのアンマネージ ダイナミック リンク ライブラリ (DLL) からエクスポートされた関数を呼び出すことができます。CLR は、DLL の読み込みとパラメータのマーシャリングを処理します。パフォーマンスを向上させるには、プラットフォーム呼び出しを使用するよりも、.NET Framework に等価の関数が用意されていないかどうかを確認してください。詳細については、「プラットフォーム呼び出しの詳細」を参照してください。
- COM 相互運用機能を使用すると、マネージ コードは、COM インターフェイスと COM クライアントを通じて COM オブジェクトと対話できます。COM コンポーネントは、次の 2 つの方法でマネージ コードから使用できます。
    - OLE オートメーション互換の COM コンポーネントを呼び出す場合は、COM 相互運用機能または tlbimp.exe を使用します。CLR は、COM コンポーネントのアクティブ化とパラメータのマーシャリングを処理します。
    - IDL ベースの COM コンポーネントの場合は、IJW と C++ を使用します。IUnknown、IDispatch、および他の標準 COM インターフェイスを実装する各パブリック マネージ クラスは、COM 相互運用機能を通じて、アンマネージ コードからすべて呼び出すことができます。詳細については、「Microsoft .NET/COM Migration and Interoperability」を参照してください。

詳細については、「アンマネージ コードとの相互運用」および「Improving Interop Performance」を参照してください。

PInvoke と COM 相互運用機能はいずれもマーシャリングを使用して、整数、文字列、配列、構造体、ポインタなどの引数をマネージ コードとアンマネージ コード間で変換します。詳細については、「相互運用マーシャリングの概要」を参照してください。

関連項目

相互運用性の概要 (C# プログラミング ガイド)

方法 : COM 相互運用機能を使用して Excel スプレッドシートを作成する (C# プログラミング ガイド)

方法 : プラットフォーム呼び出しを使用して Wave ファイルを再生する (C# プログラミング ガイド)

方法 : COM 相互運用機能を使用して Word によるスペル チェックを行う (C# プログラミング ガイド)

COM クラスの例 (C# プログラミング ガイド)

方法 : Excel 用のオートメーション アドインとしてマネージ コードを使用する (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 10.5.7 外部メソッド
- 17.5 相互運用の属性
- 18.8 動的なメモリ割り当て

参照
概念
C# プログラミング ガイド
その他の技術情報
.NET Compact Framework の相互運用性

相互運用性の概要 (C# プログラミング ガイド)

C# マネージ コードとアンマネージ コード間の相互運用性を実現するツールと方法には、プラットフォーム呼び出しサービスと .NET Framework および COM 相互運用性ツールがあります。

プラットフォーム呼び出し

プラットフォーム呼び出しは、アンマネージ コードを、エクスポートされた関数として検索し、呼び出します。また、入出力パラメータ、整数、文字列、配列、構造体などの呼び出しの引数も必要に応じてマーシャリングします。詳細については、「プラットフォーム呼び出しによるデータのマーシャリング」および「アンマネージ DLL 関数の処理」を参照してください。

メモ :

共通言語ランタイム (CLR) は、システム リソースへのアクセスを管理します。CLR 外部のアンマネージ コードの呼び出しは、このセキュリティ機構をバイパスするため、セキュリティ上のリスクが生じます。たとえば、アンマネージ コードは、アンマネージ コード内のリソースを直接呼び出し、CLR のセキュリティ機構をバイパスすることがあります。詳細については、「.NET Framework Security」を参照してください。

.NET Framework と COM 間の相互運用性ツール

- マネージ コードから COM API を呼び出すには、タイプ ライブラリ インポータ (Tlbimp.exe) を使用します。このツールは、型ライブラリを入力として取り込み、.NET Framework アセンブリおよび関連するマネージ メタデータを出力します。出力された .NET Framework アセンブリは、プロジェクト参照として Visual Studio プロジェクトに追加できます。たとえば、`TlbImp comlibrary.dll /out: comlnetibrary.dll` を使用して、プロジェクトに `comlnetibrary.dll` への参照を追加します。詳細については、「タイプ ライブラリのアセンブリとしてのインポート」および「Calling COM Components from .NET Clients」を参照してください。
- COM からマネージ コードを呼び出すには、タイプ ライブラリ エクスポータ (Tlbexp.exe) を使用します。このツールは、マネージ アセンブリを入力として取り込み、そのアセンブリで定義されているすべての **public** 型の COM 定義を含む型ライブラリを生成します。
- **COM** クライアントからマネージ コンポーネントを呼び出すには、アセンブリ登録ツール (Regasm.exe) を使用します。このツールは、.NET Framework アセンブリ内のメタデータを読み取り、レジストリ エントリを追加して、COM クライアントがマネージ クラスを作成できるようにします。
- ActiveX コントロールを呼び出すには、Windows フォーム ActiveX コントロール インポータ (Aximp.exe) を使用します。このツールは、ActiveX コントロールの型ライブラリを入力として取り込み、ActiveX コントロールを Windows フォームでホストできるようにするラッパー コントロールを生成します。たとえば、`Aximp activex.ocx` は、`viz.activex.dl` と `Axactivex.dll` の 2 つのファイルを作成するため、自動生成された `Axactivex.dll` をプロジェクト参照として使用できます。

メモ :

COM interopへのマネージ アセンブリの登録などで上記のツールを使用するには、管理者アクセス許可またはパワー ユーザー セキュリティ アクセス許可が必要です。詳細については、「.NET Framework Security」を参照してください。

相互運用性の例と方法

詳細については、「相互運用マーシャリングの概要」を参照してください。相互運用性を実現する方法の詳細については、以下のトピックを参照してください。

- 方法 : プラットフォーム呼び出しを使用して Wave ファイルを再生する (C# プログラミング ガイド)
- 方法 : Excel 用のオートメーション アドインとしてマネージ コードを使用する (C# プログラミング ガイド)
- 方法 : COM 相互運用機能を使用して Word によるスペル チェックを行う (C# プログラミング ガイド)
- 方法 : COM 相互運用機能を使用して Excel スプレッドシートを作成する (C# プログラミング ガイド)

参照

概念

C# プログラミング ガイド

カスタム マーシャリングの概要

その他の技術情報

COM 相互運用機能によるデータのマーシャリング

相互運用マーシャリング

方法：COM 相互運用機能を使用して Excel スプレッドシートを作成する (C# プログラミング ガイド)

以下のコード例は、**COM interop**を使用して、**Excel** スプレッドシートを作成する方法を示しています。**Excel** の詳細については、「Microsoft Excel のオブジェクト」および「Open メソッド」を参照してください。

この例では、.NET Framework **COM interop**機能を使用して、既存の **Excel** スプレッドシートを C# で開く方法を示しています。**Excel** アセンブリを使用して **Excel** スプレッドシートを開き、セル範囲にデータを入力します。

> **メモ：**
>
> このコードを適切に実行するには、システムに **Excel** をインストールしておく必要があります。

> **メモ：**
>
> 使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

COM 相互運用機能を使用して Excel スプレッドシートを作成するには

1. Visual Studio で新しい C# コンソール アプリケーションを作成し、**CreateExcelWorksheet** という名前を付けます。
2. Excel アセンブリを、プロジェクトへの参照として追加します。プロジェクトを右クリックし、[参照の追加] をクリックします。
3. [参照の追加] ダイアログ ボックスの [COM] タブをクリックし、[Microsoft Excel 11.0 Object Library] を見つけます。
4. [Microsoft Excel 11.0 Object Library] をダブルクリックし、[OK] をクリックします。

   > **メモ：**
   >
   > インストールされている Office のバージョンによって、Excel アセンブリの名前は、Excel 10 Object Library または Excel 11 Object Library になります。

5. 次のコードをコピーし、`Program.cs` ファイルの内容に貼り付けます。

   C#

```
using System;
using System.Reflection;
using Microsoft.Office.Interop.Excel;

public class CreateExcelWorksheet
{
    static void Main()
    {
        Microsoft.Office.Interop.Excel.Application xlApp = new Microsoft.Office.Intero
p.Excel.Application();

        if (xlApp == null)
        {
            Console.WriteLine("EXCEL could not be started. Check that your office inst
allation and project references are correct.");
            return;
        }
        xlApp.Visible = true;

        Workbook wb = xlApp.Workbooks.Add(XlWBATemplate.xlWBATWorksheet);
        Worksheet ws = (Worksheet)wb.Worksheets[1];
```

```
        if (ws == null)
        {
            Console.WriteLine("Worksheet could not be created. Check that your office
installation and project references are correct.");
        }

        // Select the Excel cells, in the range c1 to c7 in the worksheet.
        Range aRange = ws.get_Range("C1", "C7");

        if (aRange == null)
        {
            Console.WriteLine("Could not get a range. Check to be sure you have the co
rrect versions of the office DLLs.");
        }

        // Fill the cells in the C1 to C7 range of the worksheet with the number 6.
        Object[] args = new Object[1];
        args[0] = 6;
        aRange.GetType().InvokeMember("Value", BindingFlags.SetProperty, null, aRange,
args);

        // Change the cells in the C1 to C7 range of the worksheet to the number 8.
        aRange.Value2 = 8;
    }
}
```

セキュリティ

COM interopを使用するには、管理者アクセス許可またはパワー ユーザー セキュリティ アクセス許可が必要です。セキュリティの詳細については、「.NET Framework Security」を参照してください。

参照

処理手順

方法 : COM 相互運用機能を使用して Word によるスペル チェックを行う (C# プログラミング ガイド)

概念

C# プログラミング ガイド

相互運用性の概要 (C# プログラミング ガイド)

その他の技術情報

.NET Compact Framework の相互運用性

COM の相互運用性に関するサンプル

方法：プラットフォーム呼び出しを使用して Wave ファイルを再生する (C# プログラミング ガイド)

次の C# のコード例は、プラットフォーム呼び出しサービスを利用して、Windows プラットフォームで Wave サウンド ファイルを再生する方法を示しています。

使用例

このコード例では、**DllImport** を使用して、**winmm.dll** の **PlaySound** メソッド エントリ ポイントを `Form1 PlaySound()` としてインポートします。この例には、ボタン付きの簡単な Windows フォームがあります。ボタンをクリックすると、標準の Windows OpenFileDialog ダイアログ ボックスが表示され、再生するファイルを開くことができます。Wave ファイルを選択すると、winmm.DLL アセンブリ メソッドの **PlaySound()** メソッドを使ってファイルが再生されます。winmm.dll の **PlaySound** メソッドの詳細については、「Using PlaySound to Play Waveform-Audio Files」を参照してください。.wav 拡張子を持つファイルを参照して選択します。[開く] をクリックし、プラットフォーム呼び出しを使って Wave ファイルを再生します。選択したファイルの完全パスがテキスト ボックスに表示されます。

[開いているファイル] ダイアログ ボックスには、フィルタ設定に従って .wav 拡張子を持つファイルだけが表示されます。

C#

```
dialog1.Filter = "Wav Files (*.wav)|*.wav";
```

C#

```
using System.Windows.Forms;

namespace WinSound
{
    public partial class Form1 : Form
    {
        private TextBox textBox1;
        private Button button1;

        public Form1()  //constructor
        {
            InitializeComponent();
        }

        [System.Runtime.InteropServices.DllImport("winmm.DLL", EntryPoint = "PlaySound", Se
tLastError = true)]
        private static extern bool PlaySound(string szSound, System.IntPtr hMod, PlaySoundF
lags flags);

        [System.Flags]
        public enum PlaySoundFlags : int
        {
            SND_SYNC = 0x0000,
            SND_ASYNC = 0x0001,
            SND_NODEFAULT = 0x0002,
            SND_LOOP = 0x0008,
            SND_NOSTOP = 0x0010,
            SND_NOWAIT = 0x00002000,
            SND_FILENAME = 0x00020000,
            SND_RESOURCE = 0x00040004
        }

        private void button1_Click (object sender, System.EventArgs e)
        {
            OpenFileDialog dialog1 = new OpenFileDialog();

            dialog1.Title = "Browse to find sound file to play";
            dialog1.InitialDirectory = @"c:\";
            dialog1.Filter = "Wav Files (*.wav)|*.wav";
            dialog1.FilterIndex = 2;
            dialog1.RestoreDirectory = true;
```

```
            if(dialog1.ShowDialog() == DialogResult.OK)
            {
                textBox1.Text = dialog1.FileName;
                PlaySound (dialog1.FileName, new System.IntPtr(), PlaySoundFlags.SND_SYNC);
            }
        }
    }
}
```

コードのコンパイル方法

コードをコンパイルするには

1. Visual Studio で、新しい C# Windows アプリケーション プロジェクトを作成し、**WinSound** という名前を付けます。
2. 上記のコードをコピーし、`Form1.cs` ファイルの内容に貼り付けます。
3. 下記のコードをコピーし、`Form1.Designer.cs` ファイル内の `InitializeComponent()` メソッドの既存コードより後に貼り付けます。

   C#

   ```
   this.button1 = new System.Windows.Forms.Button();
   this.textBox1 = new System.Windows.Forms.TextBox();
   this.SuspendLayout();
   //
   // button1
   //
   this.button1.Location = new System.Drawing.Point(192, 40);
   this.button1.Name = "button1";
   this.button1.Size = new System.Drawing.Size(88, 24);
   this.button1.TabIndex = 0;
   this.button1.Text = "Browse";
   this.button1.Click += new System.EventHandler(this.button1_Click);
   //
   // textBox1
   //
   this.textBox1.Location = new System.Drawing.Point(8, 40);
   this.textBox1.Name = "textBox1";
   this.textBox1.Size = new System.Drawing.Size(168, 20);
   this.textBox1.TabIndex = 1;
   this.textBox1.Text = "FIle path";
   //
   // Form1
   //
   this.AutoScaleDimensions = new System.Drawing.SizeF(5, 13);
   this.ClientSize = new System.Drawing.Size(292, 266);
   this.Controls.Add(this.textBox1);
   this.Controls.Add(this.button1);
   this.Name = "Form1";
   this.Text = "Platform Invoke WinSound C#";
   this.ResumeLayout(false);
   this.PerformLayout();
   ```

4. コードをコンパイルして実行します。

セキュリティ

詳細については、「.NET Framework Security」を参照してください。

参照

処理手順

プラットフォーム呼び出しの技術サンプル

概念

C# プログラミング ガイド

相互運用性の概要 (C# プログラミング ガイド)

その他の技術情報

プラットフォーム呼び出しによるデータのマーシャリング

方法：COM 相互運用機能を使用して Word によるスペル チェックを行う (C# プログラミング ガイド)

次のコード例は、COM 相互運用機能を使用して、Visual C# アプリケーションで Word のスペル チェック機能を使用する方法を示しています。詳細については、「ProofreadingErrors コレクション オブジェクト」および「Microsoft Word のオブジェクト」を参照してください。

使用例

次の例は、C# アプリケーションで Word のスペル チェック機能を使用する方法を示しています。まず、COM 相互運用機能を使用して新しい **Word.application** オブジェクトを作成し、次に、**Range** オブジェクトで **ProofreadingErrors** コレクションを使用し、該当する範囲でスペルに間違いのある単語を検索します。

C#

```
using System.Reflection;
using Word = Microsoft.Office.Interop.Word;

namespace WordSpell
{
    public partial class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label1;

        public Form1()  //constructor
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, System.EventArgs e)
        {
            Word.Application app = new Word.Application();

            int errors = 0;
            if (textBox1.Text.Length > 0)
            {
                app.Visible = false;

                // Setting these variables is comparable to passing null to the function.
                // This is necessary because the C# null cannot be passed by reference.
                object template = Missing.Value;
                object newTemplate = Missing.Value;
                object documentType = Missing.Value;
                object visible = true;

                Word._Document doc1 = app.Documents.Add(ref template, ref newTemplate, ref
documentType, ref visible);
                doc1.Words.First.InsertBefore(textBox1.Text);
                Word.ProofreadingErrors spellErrorsColl = doc1.SpellingErrors;
                errors = spellErrorsColl.Count;

                object optional = Missing.Value;

                doc1.CheckSpelling(
                    ref optional, ref optional, ref optional, ref optional, ref optional, r
ef optional,
                    ref optional, ref optional, ref optional, ref optional, ref optional, r
ef optional);

                label1.Text = errors + " errors corrected ";
                object first = 0;
                object last = doc1.Characters.Count - 1;
                textBox1.Text = doc1.Range(ref first, ref last).Text;
            }
```

```
            object saveChanges = false;
            object originalFormat = Missing.Value;
            object routeDocument = Missing.Value;

            app.Quit(ref saveChanges, ref originalFormat, ref routeDocument);
        }
    }
}
```

コードのコンパイル方法

上の例では、Word をシステムにインストールしておく必要があります。インストールしている Office のバージョンに応じて、**Word** アセンブリは、Microsoft office 10 Object Library または Word 11 Object Library という名前になります。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コードをコンパイルするには

1. 新しい C# Windows アプリケーション プロジェクトを Visual Studio で作成し、**WordSpell** という名前を付けます。
2. 上記のコードをコピーし、`Form1.cs` ファイルの内容に貼り付けます。
3. 下記のコードをコピーし、`Form1.Designer.cs` ファイル内の `InitializeComponent()` メソッドの既存コードより後に貼り付けます。

C#

```
this.textBox1 = new System.Windows.Forms.TextBox();
this.button1 = new System.Windows.Forms.Button();
this.label1 = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(40, 40);
this.textBox1.Multiline = true;
this.textBox1.Name = "textBox1";
this.textBox1.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;
this.textBox1.Size = new System.Drawing.Size(344, 136);
this.textBox1.TabIndex = 0;
this.textBox1.Text = "";
//
// button1
//
this.button1.Location = new System.Drawing.Point(392, 40);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(96, 23);
this.button1.TabIndex = 1;
this.button1.Text = "Check Spelling";
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// label1
//
this.label1.Location = new System.Drawing.Point(40, 24);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(336, 16);
this.label1.TabIndex = 2;
```

```
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(5, 13);
this.ClientSize = new System.Drawing.Size(496, 205);
this.Controls.Add(this.label1);
this.Controls.Add(this.button1);
this.Controls.Add(this.textBox1);
this.Name = "Form1";
this.Text = "SpellCheckDemo";
this.ResumeLayout(false);
```

4. 参照として **Word** アセンブリをプロジェクトに追加します。プロジェクトを右クリックし、[参照の追加] をクリックして、[参照の追加] ダイアログ ボックスの [COM] タブをクリックします。[Microsoft Office 11.0 Object Library] をダブルクリックし、[OK] をクリックします。

   使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

セキュリティ

COM 相互運用機能を使用するには、管理者アクセス許可またはパワー ユーザー セキュリティ アクセス許可が必要です。詳細については、「.NET Framework Security」を参照してください。

参照

処理手順

方法 : COM 相互運用機能を使用して Excel スプレッドシートを作成する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

相互運用性の概要 (C# プログラミング ガイド)

その他の技術情報

COM の相互運用性に関するサンプル

COM クラスの例 (C# プログラミング ガイド)

ここでは、COM オブジェクトとして公開されるクラスの例を紹介します。このコードを .cs ファイルに保存して、プロジェクトに追加後、[COM の相互運用機能に登録] プロパティを [True] に設定します。詳細については、「方法 : コンポーネントを COM 相互運用機能に登録する」を参照してください。

Visual C# オブジェクトを COM に公開するには、クラス インターフェイス、イベント インターフェイス (必要な場合)、およびクラス自体を宣言する必要があります。クラス メンバを COM で参照するには、次の規則に従う必要があります。

- クラスがパブリックであること。
- プロパティ、メソッド、およびイベントがパブリックであること。
- プロパティとメソッドがクラス インターフェイスで宣言されていること。
- イベントがイベント インターフェイスで宣言されていること。

これらのインターフェイスで宣言されていない、クラス内のほかのパブリック メンバは、COM から参照されませんが、ほかの .NET Framework オブジェクトからは参照されます。

プロパティとメソッドを COM に公開するには、それらをクラス インターフェイスで宣言し、**DispId** 属性でマークを付けて、クラスに実装する必要があります。メンバをインターフェイスで宣言する順序は、COM の vtable で使用される順序になります。

クラスのイベントを公開するには、それらをイベント インターフェイスで宣言し、**DispId** 属性でマークを付ける必要があります。このクラスではこのインターフェイスを実装しないでください。

クラスによってクラス インターフェイスが実装されます。クラスでは、複数のインターフェイスを実装できますが、最初に実装されるのは既定のクラス インターフェイスです。ここで、COM に対して公開するメソッドとプロパティを実装します。このメソッドとプロパティは、パブリックとしてマークされており、クラス インターフェイスの宣言と同じであることが必要です。また、ここでクラスから発生するイベントを宣言します。このイベントは、パブリックとしてマークされており、イベント インターフェイスの宣言と同じであることが必要です。

詳細については、「COM 相互運用性 (第 1 部) サンプル」、「COM 相互運用性 (第 2 部) サンプル」、および「COM の相互運用性に関するサンプル」を参照してください。

使用例
C#

```
using System.Runtime.InteropServices;

namespace project_name
{
    [Guid("EAA4976A-45C3-4BC5-BC0B-E474F4C3C83F")]
    public interface ComClass1Interface
    {
    }

    [Guid("7BD20046-DF8C-44A6-8F6B-687FAA26FA71"),
        InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]
    public interface ComClass1Events
    {
    }

    [Guid("0D53A3E8-E51A-49C7-944E-E72A2064F938"),
        ClassInterface(ClassInterfaceType.None),
        ComSourceInterfaces(typeof(ComClass1Events))]
    public class ComClass1 : ComClass1Interface
    {
    }
}
```

参照
関連項目
相互運用性 (C# プログラミング ガイド)
[ビルド] ページ (プロジェクト デザイナ) (C#)
概念

方法：Excel 用のオートメーション アドインとしてマネージ コードを使用する (C# プログラミング ガイド)

Excel のオートメーション アドインを使用すると、セル数式として呼び出される、COM ライブラリのパブリック関数を使用できます。次の例は、Excel ワークシートのセルで所得税率を計算する C# アドインを作成する方法を示しています。ComRegisterFunctionAttribute は、アドインを自動的に登録し、これ以外に、マネージ コードを COM アセンブリとして登録するのに必要なツールはありません。詳細については、「相互運用性の概要 (C# プログラミング ガイド)」を参照してください。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

所得税の計算

通常の税率表では、個人の年間所得から課税額を計算できます。仮の個人税率表の例を次に示します。

税率表の例

1. 所得金額が $7,000 未満の場合、税額は所得金額の 10% です。
2. 所得金額が $7,000 以上で $28,400 未満の場合、税額は、$7,000 以上の金額の 15% に $700.00 を加算した金額になります。
3. 所得金額が $28,400 以上で $68,800 未満の場合、税額は、$28,400 以上の金額の 25% に $ 3,910.00 を加算した金額になります。
4. 所得金額が $68,800 以上で $143,500 未満の場合、税額は、$68,800 以上の金額の 28% に $14,010.00 を加算した金額になります。
5. 所得金額が $143,500 以上で $311,950 未満の場合、税額は、$143,500 以上の金額の 33% に $34,926.00 を加算した金額になります。
6. 所得金額が $311,950 以上の場合、税額は、$311,950 以上の金額の 35% に $90,514.50 を加算した金額になります。

Visual Studio とマネージ コードを使用した、Excel 用のオートメーション アドインの作成

1. **ExcelAddIn** という名前の新しい Visual C# クラス ライブラリ プロジェクトを作成します。
2. [プロジェクト] メニューの [プロパティ] をクリックし、[ビルド] ペイン内をクリックします。[COM 相互運用機能の登録] チェック ボックスをオンにします。COM 相互運用機能のため、アセンブリが自動的に登録されます。
3. 次のコードをクラス ファイルに貼り付けます。

C#

```
using System.Runtime.InteropServices;

namespace TaxTables
{
    [ClassInterface(ClassInterfaceType.AutoDual)]
    public class TaxTables
    {
        public static double Tax(double income)
        {
            if (income >      0 && income <=   7000) {return            (.10 * income)
;}
            if (income >   7000 && income <=  28400) {return   700.00 + (.15 * (income
- 7000));}
            if (income >  28400 && income <=  68800) {return  3910.00 + (.25 * (income
- 28400));}
```

```
            if (income >  68800 && income <= 143500) {return 14010.00 + (.28 * (income
- 68800));}
            if (income > 143500 && income <= 311950) {return 34926.00 + (.33 * (income
- 143500));}
            if (income > 311950)                     {return 90514.50 + (.35 * (income
- 311950));}
            return 0;
        }

        [ComRegisterFunctionAttribute]
        public static void RegisterFunction(System.Type t)
        {
            Microsoft.Win32.Registry.ClassesRoot.CreateSubKey
                ("CLSID\\{" + t.GUID.ToString().ToUpper() + "}\\Programmable");
        }

        [ComUnregisterFunctionAttribute]
        public static void UnregisterFunction(System.Type t)
        {
            Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey
                ("CLSID\\{" + t.GUID.ToString().ToUpper() + "}\\Programmable");
        }
    }
}
```

コードの実行

Excel アドインの実行

- **ExcelAddIn** プロジェクトを構築し、F5 キーを押してコンパイルします。
- Excel で新しいブックを開きます。
- [ツール] メニューの [アドイン] をポイントし、[オートメーション] をクリックします。
- [オートメーション サーバー] ダイアログ ボックスのアドインの一覧の [ExcelAddIn] をクリックし、[OK] をクリックします。
- ブックのセルに「`=Tax(23500)`」と入力します。セルに 3175 と表示されます。
- ExcelAddIn.dll を別のディレクトリに移動した後に登録するには、**/codebase** を使用して **regasm** を実行します。アセンブリが署名されていないという警告が表示される場合があります。

セキュリティ

COM 相互運用機能を使用するには、管理者またはパワー ユーザー セキュリティ アクセス許可が必要です。詳細については、「.NET Framework Security」を参照してください。

参照

処理手順

方法 : COM 相互運用機能を使用して Excel スプレッドシートを作成する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

相互運用性の概要 (C# プログラミング ガイド)

スレッド処理 (C# プログラミング ガイド)

スレッド処理を使用すると、C# プログラムが同時処理を実行し、一度に複数の操作を行うことができるようになります。たとえば、スレッド処理を使用すると、ユーザー入力を監視したり、バックグラウンド タスクを実行したり、複数の入力ストリームを同時に処理したりできます。System.Threading 名前空間により、マルチスレッド プログラミングをサポートするクラスとインターフェイスが用意され、スレッドの新規作成および開始、複数スレッドの同期、スレッドの中断、スレッドの中止などのタスクを従来よりも簡単に実行できます。

メイン スレッドの外部で実行する関数を作成し、新しい Thread オブジェクトでそれを指すようにするだけで、C# コードにスレッド処理を組み込むことができます。C# アプリケーションで新しいスレッドを作成する方法を次のコード例に示します。

C#

```
System.Threading.Thread newThread;
newThread = new System.Threading.Thread(anObject.AMethod);
```

次のコード例では、C# アプリケーションで新しいスレッドを開始します。

C#

```
newThread.Start();
```

マルチスレッド処理によって、応答性やマルチタスクに関連する問題が解決されますが、同時に、リソースの共有や同期という問題が発生することもあります。メインのスレッド スケジュール機構に応じて、スレッドが警告なしに中断され、再開されるためです。詳細については、「スレッドの同期 (C# プログラミング ガイド)」を参照してください。概要については、「スレッドの使用とスレッド処理」を参照してください。

概要

スレッドには、次の特徴があります。

- スレッドを使用すると、C# プログラムで同時処理を実行できるようになります。
- .NET Framework の **System.Threading** 名前空間により、以前よりも簡単にスレッドを使用できます。
- スレッドは、アプリケーションのリソースを共有します。詳細については、「スレッドの使用とスレッド処理」を参照してください。

関連項目

詳細については、以下のトピックを参照してください。

- スレッド処理の使用 (C# プログラミング ガイド)
- 方法 : スレッドを作成および終了する (C# プログラミング ガイド)
- 方法 : スレッド プールを使用する (C# プログラミング ガイド)
- 方法 : producer スレッドと consumer スレッドを同期する (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.10 実行順序
- 8.12 lock ステートメント
- 10.4.3 Volatile フィールド
- 10.7.1 フィールドのように使用するイベント

参照
処理手順
Monitorによる同期化の技術サンプル
待機による同期化の技術サンプル
関連項目
Mutex
概念

C# プログラミング ガイド
ミューテックス
Monitor
インタロックされた操作
AutoResetEvent
デリゲート (C# プログラミング ガイド)

その他の技術情報

Thread クラス
[HOWTO] Visual C# .NET を使用してマルチスレッド環境で共有リソースへのアクセスを同期する方法

スレッド処理の使用 (C# プログラミング ガイド)

既定では、C# プログラムは 1 つのスレッドを使用します。このスレッドは、`Main` メソッドで開始および終了するプログラム内のコードを実行します。`Main` によって直接または間接的に実行されるすべてのコマンドは、既定のスレッド、つまりプライマリ スレッドによって実行されます。このスレッドは、`Main` から制御が戻ると終了します。ただし、補助スレッドを作成して使用することで、プライマリ スレッドと並行してコードを実行することもできます。これらスレッドは、一般にワーカー スレッドと呼ばれます。

ワーカー スレッドを使用すると、時間がかかるタスクやタイミングが重要なタスクを、プライマリ スレッドを停止せずに実行できます。たとえば、サーバー アプリケーションでは、前の要求が完了するまで待機せずに着信要求を処理できるように、ワーカー スレッドがよく使用されます。ワーカー スレッドは、デスクトップ アプリケーションで "バックグラウンド" タスクを実行するためにも使用されます。これにより、ユーザー インターフェイス要素を駆動するメイン スレッドがユーザーの操作に応答し続けることができます。

マルチスレッド処理により、スループットと応答性の問題が解決されますが、デッドロックや競合状態など、リソース共有に関する問題の発生原因になることもあります。複数のスレッドは、ファイル ハンドルやネットワーク接続などのさまざまなリソースを必要とするタスクに最適です。複数のスレッドを単一のリソースに割り当てると、同期上の問題が発生する可能性があり、またスレッドを頻繁にブロックしながら他のスレッドを待機するのは、複数のスレッドを使用する趣旨にそぐいません。

ワーカー スレッドは、他のスレッドによって使用されるリソースの多くを必要としない、時間がかかるタスクやタイミングが重要なタスクを実行するために使用するのが一般的です。もちろん、プログラム内の一部のリソースには、複数のスレッドがアクセスする必要があります。そのために、System.Threading 名前空間には、スレッドの同期に必要なクラスが用意されています。これらのクラスは、Mutex、Monitor、Interlocked、AutoResetEvent、および ManualResetEvent です。

これらのクラスの一部またはすべてを使用して、複数のスレッドの動作を同期できますが、マルチスレッド処理は、C# 言語によっても一部サポートされます。たとえば、C# の Lock ステートメントは、**Monitor** を暗黙に使用して同期機能を実現します。

次のトピックでは、一般的なマルチスレッド処理の手法について説明します。

- 方法 : スレッドを作成して開始し、スレッド間で対話する
- 方法 : producer スレッドと consumer スレッドを同期する
- 方法 : スレッド プールを使用する (C# プログラミング ガイド)

関連項目

詳細については、以下のトピックを参照してください。

- [HOWTO] Visual C# .NET を使用してスレッドを作成する方法
- HOW TO:Visual C# .NET を使って、スレッド プールの作業項目を送信します。
- [HOWTO] Visual C# .NET を使用してマルチスレッド環境で共有リソースへのアクセスを同期する方法
- スレッドおよびスレッド処理
- マルチスレッド処理のためのデータの同期

参照

処理手順

Monitorによる同期化の技術サンプル
待機による同期化の技術サンプル
スレッド処理のサンプル

関連項目

Mutex

概念

C# プログラミング ガイド
Monitor
インタロックされた操作
AutoResetEvent

その他の技術情報

マネージ スレッド処理
スレッドの使用とスレッド処理
スレッド処理のサンプル
Thread クラス

スレッドの同期 (C# プログラミング ガイド)

次の各セクションでは、マルチスレッド アプリケーションでリソースへのアクセスを同期するために使用できる機能とクラスについて説明します。

アプリケーションで複数のスレッドを使用する利点の 1 つは、各スレッドを非同期的に実行できる点にあります。Windows アプリケーションでは、これによって時間のかかるタスクをバックグラウンドで実行しながら、アプリケーションのウィンドウやコントロールを応答可能な状態に維持できます。サーバー アプリケーションでは、マルチスレッドを使用することにより、受け取った各要求を別個のスレッドで処理できるようになります。マルチスレッドを使用しない場合、前の要求が完全に満たされるまで、新しい要求はサービスを受けることができません。

ただし、スレッドでの処理が非同期であるために、ファイル ハンドル、ネットワーク接続、およびメモリなどのリソースへのアクセスを調整する必要が生じます。調整が行われないと、互いに別のスレッドの動作が認識できず、複数のスレッドが同時に同じリソースにアクセスしてしまうことになります。その結果、予期しないデータ破損が発生します。

整数のデータ型に対する単純な演算の場合は、Interlocked クラスのメンバを使用することにより、スレッドの同期を実現できます。その他のすべてのデータ型やスレッド セーフではないリソースについては、このトピックで説明する構成要素を使用しない限り、マルチスレッド処理を安全に実行することはできません。

マルチスレッド プログラミングの背景情報については、次を参照してください。

- スレッド処理の使用 (C# プログラミング ガイド)
- マネージ スレッド処理の基本
- スレッドの使用とスレッド処理
- マネージ スレッド処理の実施

lock キーワード

lock キーワードを使用すると、他のスレッドからの割り込みを受けることなくコード ブロックを確実に最後まで実行できます。これは、コード ブロックの実行中に、特定のオブジェクトに対して同時に使用できないロックを取得することで実現されます。

lock ステートメントは、引数としてオブジェクトが渡される **lock** キーワードから始まり、その後に、一度に 1 つのスレッドだけが実行するコード ブロックが続きます。次に例を示します。

C#

```
public void Function()
{
    System.Object lockThis = new System.Object();
    lock(lockThis)
    {
        // Access thread-sensitive resources.
    }
}
```

lock キーワードに渡される引数は、参照型に基づくオブジェクトである必要があり、ロックのスコープを定義するために使用されます。上記の例では、関数の外部にオブジェクトへの参照が存在しないため、ロックのスコープはこの関数に限定されます。厳密には、**lock** に渡されるオブジェクトは、複数のスレッドで共有されるリソースを一意に識別するためだけに使用されるので、任意のクラスのインスタンスを使用できます。ただし実際には、このオブジェクトは、スレッドの同期が必要なリソースを表すのが普通です。たとえば、複数のスレッドがコンテナ オブジェクトを使用する場合、そのコンテナを lock キーワードに渡すと、lock に続く、同期されたコード ブロックがそのコンテナにアクセスできるようになります。他のスレッドが同じコンテナにアクセスする前にコンテナをロックすると、このオブジェクトへのアクセスを安全に同期できます。

一般に、**public** 型や、アプリケーションの制御が及ばないオブジェクト インスタンスはロックしないことをお勧めします。たとえば、インスタンスにパブリックにアクセスできる場合、`lock(this)` は問題となることがあります。ユーザーの制御が及ばないコードによってもこのオブジェクトがロックされる可能性があるからです。この場合、複数のスレッドが同じオブジェクトの解放を待機しているような場合にはデッドロック状態が発生することがあります。オブジェクトではなく、パブリックなデータ型をロックした場合も同じ理由から問題が生じることがあります。リテラル文字列は共通言語ランタイム (CLR: Common Language Runtime) のインターン プールに存在しているため、リテラル文字列をロックすることは特に危険です。つまり、プログラム全体では任意のリテラル文字列のインスタンスは 1 つしか存在しませんが、まったく同じオブジェクトが、実行中のすべてのアプリケーション ドメインのすべてのスレッド上のリテラルを表すためです。この結果、アプリケーション プロセスの任意の場所で同じ内容を持つ文字列をロックした場合、アプリケーション内のその文字列のすべてのインスタンスがロックされてしまいます。したがって、インターン プールに存在しないプライベート メンバまたはプロテクト メンバをロックすることをお勧めします。クラスによっては、ロック専用のメンバを提供するものもあります。たとえば、Array 型では SyncRoot が提供されます。多くのコレクション型でも、`SyncRoot` メンバが提供されます。

lock キーワードの詳細については、次を参照してください。

- lock ステートメント (C# リファレンス)
- 方法 : producer スレッドと consumer スレッドを同期する (C# プログラミング ガイド)

Monitor

lock キーワードと同様、Monitor クラスも複数のスレッドによるコード ブロックの同時実行を防ぎます。Enter メソッドは 1 つのスレッドにのみ後続のステートメントに進むことを許可します。他のすべてのスレッドは、実行中のスレッドが Exit を呼び出すまでブロックされます。これは **lock** キーワードを使用した場合とまったく同じ結果になります。実際、**lock** キーワードは Monitor クラスを使用して実装されます。次に例を示します。

C#

```
lock(x)
{
    DoSomething();
}
```

このようにすると、次の記述と同じ結果が得られます。

C#

```
System.Object obj = (System.Object)x;
System.Threading.Monitor.Enter(obj);
try
{
    DoSomething();
}
finally
{
    System.Threading.Monitor.Exit(obj);
}
```

通常、**Monitor** クラスを直接使用するよりも **lock** キーワードを使用することをお勧めします。この理由としては、**lock** の方が簡潔であるということと、**lock** を使用すると、保護されたコードが例外をスローした場合でも基になる Monitor が確実に解放されるということがあります。Monitor を確実に解放するには、**finally** キーワードを使用します。このキーワードを使用することにより、例外がスローされたかどうかに関係なく関連付けられているコード ブロックが実行されます。

Monitor の詳細については、「Monitorによる同期化の技術サンプル」を参照してください。

同期イベントと待機ハンドル

スレッド依存のコード ブロックが同時に実行されないようにするために lock や Monitor を使用することは有効ですが、このような構成要素だけでは、スレッド間でイベントをやりとりすることはできません。そこで、同期イベントが必要になります。これは、シグナル状態と非シグナル状態という 2 つの状態を持つオブジェクトで、これを使用することによりスレッドをアクティブにしたり中断したりできます。非シグナル状態の同期イベントを待機させることによってスレッドを中断できます。また、イベントの状態をシグナル状態に変更することによりスレッドをアクティブにできます。既にシグナル状態にあるイベントをスレッドが待機している場合、スレッドは遅延なしに実行され続けます。

同期イベントには、AutoResetEvent と ManualResetEvent の 2 種類があります。この 2 つで唯一違うのは、**AutoResetEvent** の場合、1 つのスレッドをアクティブにすると、シグナル状態から非シグナル状態に変化するという点です。逆に **ManualResetEvent** では、そのシグナル状態によって任意の数のスレッドをアクティブにでき、Reset が呼び出された場合のみ非シグナル状態に戻ります。

WaitOne、WaitAny、または WaitAll の待機メソッドのいずれかを呼び出すことによって、スレッドを待機させることができます。System.Threading.WaitHandle.WaitOne は、単一のイベントがシグナル状態になるまでスレッドを待機させます。System.Threading.WaitHandle.WaitAny は、指定した 1 つ以上のイベントがシグナル状態になるまでスレッドをブロックします。また、System.Threading.WaitHandle.WaitAll は、指定したすべてのイベントがシグナル状態になるまでスレッドをブロックします。イベントは、その Set メソッドが呼び出されると、シグナル状態になります。

次の例では、`Main` 関数によってスレッドが作成され開始されます。新しいスレッドは **WaitOne** メソッドを使用してイベントを待機します。このスレッドは、`Main` 関数を実行しているプライマリ スレッドによってイベントがシグナル状態になるまで中断されます。イベントがシグナル状態になると、この補助スレッドに制御が戻ります。この場合は 1 つのスレッドだけをアクティブにするためにイベントを使用しているので、**AutoResetEvent** クラスと **ManualResetEvent** クラスのいずれも使用できます。

C#

```
using System;
using System.Threading;
```

```
class ThreadingExample
{
    static AutoResetEvent autoEvent;

    static void DoWork()
    {
        Console.WriteLine("   worker thread started, now waiting on event...");
        autoEvent.WaitOne();
        Console.WriteLine("   worker thread reactivated, now exiting...");
    }

    static void Main()
    {
        autoEvent = new AutoResetEvent(false);

        Console.WriteLine("main thread starting worker thread...");
        Thread t = new Thread(DoWork);
        t.Start();

        Console.WriteLine("main thrad sleeping for 1 second...");
        Thread.Sleep(1000);

        Console.WriteLine("main thread signaling worker thread...");
        autoEvent.Set();
    }
}
```

スレッドを同期するためのイベントの使用例については、次を参照してください。

- Monitorによる同期化の技術サンプル
- 読み取り/書き込み同期の技術サンプル
- スレッド プールの技術サンプル
- 待機による同期化の技術サンプル

Mutex オブジェクト

ミューテックスは、複数のスレッドによってコード ブロックが同時に実行されるのを防ぐという点で Monitor に似ています。実際、"mutex (ミューテックス)" という名前は、"mutually exclusive (同時に指定できない)" という用語の短縮形です。ただし、Monitor とは違って、ミューテックスを使用するとプロセス間でスレッドを同期できます。ミューテックスは、Mutex クラスによって表されます。

プロセス間での同期を行うために使用されるミューテックスは、名前付きミューテックスと呼ばれます。このようなミューテックスは別のアプリケーションで使用される可能性があるため、グローバル変数や静的変数を使用して共有できないからです。したがって、両方のアプリケーションから同じミューテックス オブジェクトにアクセスできるように、名前を付ける必要があります。

ミューテックスを使用するとプロセス間でスレッドを同期できますが、通常は **Monitor** の使用をお勧めします。その理由は、Monitor が .NET Framework 専用にデザインされているため、より適切にリソースを利用できる点にあります。一方、**Mutex** クラスは Win32 の構成要素のラッパーです。ミューテックスは Monitor よりも強力ですが、**Monitor** クラスよりも相互運用機能の遷移に必要な計算上の負荷が大きくなってしまいます。ミューテックスの使用例については、「ミューテックス」を参照してください。

関連項目

- 方法 : スレッドを作成および終了する (C# プログラミング ガイド)
- 方法 : スレッド プールを使用する (C# プログラミング ガイド)
- [HOWTO] Visual C# .NET を使用してマルチスレッド環境で共有リソースへのアクセスを同期する方法
- [HOWTO] Visual C# .NET を使用してスレッドを作成する方法
- HOW TO: Visual C# .NET を使って、スレッド プールの作業項目を送信します。
- [HOWTO] Visual C# .NET を使用してマルチスレッド環境で共有リソースへのアクセスを同期する方法

参照
関連項目

Thread
WaitOne
WaitAny
WaitAll
Monitor
Mutex
AutoResetEvent
ManualResetEvent
Interlocked
WaitHandle

概念

C# プログラミング ガイド

方法：スレッドを作成および終了する (C# プログラミング ガイド)

以下の例では、補助スレッドつまりワーカー スレッドを作成および使用して、プライマリ スレッドとの並行処理を実行する方法を示します。また、1 つのスレッドに別のスレッドを待機させる方法とスレッドを適切に終了する方法も示します。マルチスレッド処理の背景情報については、「マネージ スレッド処理」および「スレッド処理の使用 (C# プログラミング ガイド)」を参照してください。

この例では、`Worker` という名前のクラスを作成します。このクラスには、ワーカー スレッドが実行する、`DoWork` というメソッドが含まれています。このメソッドは、実質的にワーカー スレッド用の `Main` 関数です。ワーカー スレッドは、このメソッドを呼び出して実行を開始し、このメソッドから制御が戻ると自動的に終了します。`DoWork` メソッドは次のようになります。

C#

```
public void DoWork()
{
    while (!_shouldStop)
    {
        Console.WriteLine("worker thread: working...");
    }
    Console.WriteLine("worker thread: terminating gracefully.");
}
```

このメソッドの他に、`Worker` クラスには、`DoWork` に対し制御を戻すよう指示するためのメソッドもあります。このメソッドは `RequestStop` というメソッドで、次のようになります。

C#

```
public void RequestStop()
{
    _shouldStop = true;
}
```

`RequestStop` メソッドは、`_shouldStop` データ メンバに **true** を設定するだけです。このデータ メンバは、`DoWork` メソッドによってチェックされるため、この処理には、`DoWork` から制御を戻し、それによってワーカー スレッドを終了させるという間接的な効果があります。ただし、`DoWork` と `RequestStop` は別々のスレッドによって実行されることに注意する必要があります。`DoWork` は、ワーカー スレッドによって実行され、`RequestStop` は、プライマリ スレッドによって実行されるため、`_shouldStop` データ メンバは、次のように **volatile** と宣言されます。

C#

```
private volatile bool _shouldStop;
```

volatile キーワードは、複数のスレッドが `_shouldStop` データ メンバにアクセスするため、このメンバの状態についての最適化を想定しないようコンパイラに警告します。詳細については、「volatile (C# リファレンス)」を参照してください。

`_shouldStop` データ メンバで **volatile** を使用することにより、正規のスレッド同期手法を使わずに複数のスレッドからこのメンバに安全にアクセスできますが、これは、`_shouldStop` が **bool** であるからにすぎません。このことは、`_shouldStop` を変更する際に単一の分割不可能な操作しか必要ないことを意味します。ただし、このデータ メンバがクラス、構造体、または配列の場合は、複数のスレッドからアクセスすると、断続的なデータの破損が生じる可能性があります。たとえば、配列内のデータを変更するスレッドがあるとします。Windows は、他のスレッドの実行を可能にするために定期的にスレッドを中断します。そのため、このスレッドは、一部の配列要素を割り当ててから別の要素を割り当てるまで停止されることがあります。このため、配列はプログラマが意図しなかった状態になり、その結果、この配列を読み取る別のスレッドが失敗する可能性があります。

ワーカー スレッドを実際に作成する前に、`Main` 関数は、`Worker` オブジェクトと、`Thread` のインスタンスを作成します。このスレッド オブジェクトが、`Worker.DoWork` メソッドをエントリ ポイントとして使用するように構成するには、このメソッドへの参照を **Thread** コンストラクタに渡します。この例を次に示します。

C#

```
Worker workerObject = new Worker();
Thread workerThread = new Thread(workerObject.DoWork);
```

この時点では、ワーカー スレッド オブジェクトが存在し、構成されていますが、ワーカー スレッドそのものはまだ作成されていません。ワーカー スレッドが作成されるのは、次のように `Main` が Start メソッドを呼び出してからです。

C#

```
workerThread.Start();
```

この時点で、システムは、ワーカー スレッドの実行を開始しますが、プライマリ スレッドに対して非同期的に実行します。これは、`Main` 関数がコードの実行を継続し、それと同時にワーカー スレッドが初期化されることを意味します。ワーカー スレッドが実行する機会を得る前に、`Main` 関数がワーカー スレッドを終了しないようにするために、`Main` 関数は、ワーカー スレッド オブジェクトの IsAlive プロパティに **true** が設定されるまで、次のようにループします。

C#

```
while (!workerThread.IsAlive);
```

次に、プライマリ スレッドは、Sleep への呼び出しによって、短時間停止されます。これにより、ワーカー スレッドの `DoWork` 関数は、`Main` 関数がその他のコマンドを実行する前に、`DoWork` メソッドの内部ループを次のように数回実行します。

C#

```
Thread.Sleep(1);
```

1 ミリ秒経過した後、`Main` は、前に導入した `Worker.RequestStop` メソッドを使って、ワーカー スレッド オブジェクトに対し終了するように通知します。

C#

```
workerObject.RequestStop();
```

スレッドは、Abort への呼び出しを使って別のスレッドから終了することもできますが、その場合、タスクが完了したかどうかに関係なく、対象のスレッドが強制的に終了されるため、リソースをクリーンアップする機会が得られなくなります。そのため、上の例に示した手法をお勧めします。

最後に、`Main` 関数は、ワーカー スレッド オブジェクトで Join メソッドを呼び出します。このメソッドは、オブジェクトが表すスレッドが終了するまで、現在のスレッドをブロック、つまり待機させます。そのため、**Join** は、ワーカー スレッドから制御が戻ってワーカー スレッド自体が終了するまで、制御を戻しません。

C#

```
workerThread.Join();
```

この時点では、`Main` を実行しているプライマリ スレッドのみが存在します。Main 関数は 1 つの最終メッセージを表示した後、制御を戻し、プライマリ スレッドも終了します。

完全な例は次のようになります。

使用例

C#

```
using System;
using System.Threading;

public class Worker
{
    // This method will be called when the thread is started.
    public void DoWork()
    {
        while (!_shouldStop)
        {
            Console.WriteLine("worker thread: working...");
        }
        Console.WriteLine("worker thread: terminating gracefully.");
```

```
        }
        public void RequestStop()
        {
            _shouldStop = true;
        }
        // Volatile is used as hint to the compiler that this data
        // member will be accessed by multiple threads.
        private volatile bool _shouldStop;
    }

    public class WorkerThreadExample
    {
        static void Main()
        {
            // Create the thread object. This does not start the thread.
            Worker workerObject = new Worker();
            Thread workerThread = new Thread(workerObject.DoWork);

            // Start the worker thread.
            workerThread.Start();
            Console.WriteLine("main thread: Starting worker thread...");

            // Loop until worker thread activates.
            while (!workerThread.IsAlive);

            // Put the main thread to sleep for 1 millisecond to
            // allow the worker thread to do some work:
            Thread.Sleep(1);

            // Request that the worker thread stop itself:
            workerObject.RequestStop();

            // Use the Join method to block the current thread
            // until the object's thread terminates.
            workerThread.Join();
            Console.WriteLine("main thread: Worker thread has terminated.");
        }
    }
```

サンプル出力

```
main thread: starting worker thread...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: terminating gracefully...
main thread: worker thread has terminated
```

参照

処理手順

スレッド処理のサンプル

関連項目

スレッド処理 (C# プログラミング ガイド)
スレッド処理の使用 (C# プログラミング ガイド)
volatile (C# リファレンス)
Thread
Mutex
Monitor
Start

IsAlive
Sleep
Join
Abort
概念
C# プログラミング ガイド
その他の技術情報
マネージ スレッド処理
スレッド処理のサンプル

方法：producer スレッドと consumer スレッドを同期する (C# プログラミング ガイド)

lock キーワードと AutoResetEvent クラスおよび ManualResetEvent クラスを使用して、プライマリ スレッドと 2 つのワーカー スレッドの間でスレッドの同期を行う方法を次に示します。詳細については、「lock ステートメント (C# リファレンス)」を参照してください。

この例では、補助的なスレッドであるワーカー スレッドを 2 つ作成します。一方のスレッドでは要素を生成し、スレッド セーフではないジェネリック キューに格納します。詳細については、「Queue」を参照してください。もう一方のスレッドでは、このキューの項目を使用します。また、プライマリ スレッドでは定期的にキューの内容を表示して、3 つのスレッドがキューにアクセスできるようにします。さらに、**lock** キーワードを使用してキューへのアクセスを同期し、キューの状態が破損しないようにします。

lock キーワードを使用して同時アクセスを回避するだけでなく、2 つのイベント オブジェクトによってさらに確実な同期を実現します。一方のイベント オブジェクトはワーカー スレッドに対して終了を通知するために使用され、もう一方は、新しい項目がキューに追加された際に producer スレッドから consumer スレッドに通知するために使用されます。この 2 つのイベント オブジェクトは `SyncEvents` というクラスにカプセル化されます。カプセル化することによって、consumer スレッドを表すオブジェクトおよび producer スレッドを表すオブジェクトにこれらのイベントを容易に渡すことができます。`SyncEvents` クラスは次のように定義されます。

C#

```
public class SyncEvents
{
    public SyncEvents()
    {

        _newItemEvent = new AutoResetEvent(false);
        _exitThreadEvent = new ManualResetEvent(false);
        _eventArray = new WaitHandle[2];
        _eventArray[0] = _newItemEvent;
        _eventArray[1] = _exitThreadEvent;
    }

    public EventWaitHandle ExitThreadEvent
    {
        get { return _exitThreadEvent; }
    }
    public EventWaitHandle NewItemEvent
    {
        get { return _newItemEvent; }
    }
    public WaitHandle[] EventArray
    {
        get { return _eventArray; }
    }

    private EventWaitHandle _newItemEvent;
    private EventWaitHandle _exitThreadEvent;
    private WaitHandle[] _eventArray;
}
```

AutoResetEvent クラスは "新しい項目" イベントに対して使用されます。consumer スレッドがこのイベントに応答するたびに、このイベントを自動的にリセットする必要があるためです。また、**ManualResetEvent** クラスは "終了" イベントに対して使用されます。このイベントがシグナル状態になったときに複数のスレッドが応答するようにする必要があるためです。代わりに **AutoResetEvent** を使用した場合、1 つのスレッドだけがこのイベントに応答した後、このイベントは非シグナル状態に戻ります。他のスレッドは応答せず、この場合は終了しません。

`SyncEvents` クラスは 2 つのイベントを作成し、2 つの異なった形式で格納します。1 つは、**AutoResetEvent** および **ManualResetEvent** 双方の基本クラスである EventWaitHandle として格納し、もう 1 つは WaitHandle に基づく配列内に格納します。後ほど consumer スレッドの説明で示すように、consumer スレッドがいずれのイベントにも応答できるようにするには、この配列が必要です。

consumer スレッドと producer スレッドは、それぞれ `Consumer` クラスと `Producer` クラスによって表され、これらのクラスはいずれも `ThreadRun` というメソッドを定義します。これらのメソッドは、`Main` メソッドが作成するワーカー スレッドのエントリ ポイントとして使用されます。

`Producer` クラスが定義する `ThreadRun` メソッドは次のようになります。

C#

```
// Producer.ThreadRun
public void ThreadRun()
{
    int count = 0;
    Random r = new Random();
    while (!_syncEvents.ExitThreadEvent.WaitOne(0, false))
    {
        lock (((ICollection)_queue).SyncRoot)
        {
            while (_queue.Count < 20)
            {
                _queue.Enqueue(r.Next(0,100));
                _syncEvents.NewItemEvent.Set();
                count++;
            }
        }
    }
    Console.WriteLine("Producer thread: produced {0} items", count);
}
```

このメソッドは、"スレッドの終了" イベントがシグナル状態になるまでループします。このイベントの状態は、`SyncEvents` クラスによって定義される `ExitThreadEvent` プロパティを使用して、WaitOne メソッドによりテストされます。この場合、**WaitOne** によって使用される最初の引数がゼロ (メソッドからすぐに制御が戻ることを示す) であるため、現在のスレッドをブロックせずにイベントの状態がチェックされます。**WaitOne** が **true** を返す場合、当該のイベントは現在シグナル状態になっています。この場合、`ThreadRun` メソッドから制御が戻るため、ワーカー スレッドはこのメソッドの実行を終了します。

"スレッドの終了" イベントがシグナル状態になるまで、`Producer.ThreadStart` メソッドはキュー内に 20 の項目を維持しようとします。項目は 0 から 100 までの整数のいずれかです。consumer スレッドとプライマリ スレッドが同時にコレクションにアクセスしないようにするには、新しい項目を追加する前にコレクションをロックする必要があります。これには、**lock** キーワードを使用します。**lock** に渡される引数は、ICollection インターフェイスを介して公開される SyncRoot フィールドです。このフィールドは、スレッド アクセスを同期するために提供されている専用のフィールドです。**lock** に続くコード ブロックに含まれるすべての命令に対して、コレクションへの排他アクセスが付与されます。producer がキューに新しい項目を追加するたびに、"新しい項目" イベントの Set メソッドが呼び出されます。これは consumer スレッドに対し、中断状態から抜け出して新しい項目を処理するように通知します。

`Consumer` オブジェクトも `ThreadRun` というメソッドを定義します。producer バージョンの `ThreadRun` と同様に、このメソッドは `Main` メソッドが作成するワーカー スレッドによって実行されます。ただし、consumer バージョンの `ThreadStart` は 2 つのイベントに応答する必要があります。`Consumer.ThreadRun` メソッドは次のようになります。

C#

```
// Consumer.ThreadRun
public void ThreadRun()
{
    int count = 0;
    while (WaitHandle.WaitAny(_syncEvents.EventArray) != 1)
    {
        lock (((ICollection)_queue).SyncRoot)
        {
            int item = _queue.Dequeue();
        }
        count++;
    }
    Console.WriteLine("Consumer Thread: consumed {0} items", count);
}
```

このメソッドは WaitAny を使用して、指定した配列内の待機ハンドルのいずれかがシグナル状態になるまで consumer スレッドをブロックします。この場合、配列には 2 つのハンドルがあり、1 つはワーカー スレッドを終了するためのもので、もう 1 つは新しい項目がコレクションに追加されたことを示すためのものです。**WaitAny** は、シグナル状態になったイベントのインデックスを返します。"新しい項目" イベントは配列の最初にあるため、ゼロというインデックスは新しい項目を示します。この場合、1 のインデックスをチェックします。このインデックスは "スレッドの終了" イベントを示し、このメソッドが項目を使用し続けているかどうかを確認するために使用されます。"新しい項目" イベントがシグナル状態になっている場合、**lock** を使用してコレクションへの排他アクセスを取得し、新しい項目を使用できます。この例では、何千という項目を生成したり使用したりするため、使用される各項目は表示されません。その代わりに `Main` を使用してキューの内容を定期的に表示できます。これについては次に説明します。

Main メソッドでは、次に示すように、まず、その内容が生成および使用されるキューと、既に説明した SyncEvents のインスタンスを作成します。

C#

```
Queue<int> queue = new Queue<int>();
SyncEvents syncEvents = new SyncEvents();
```

次に Main は、ワーカー スレッドで使用する Producer オブジェクトと Consumer オブジェクトを構成します。ただし、このステップでは実際のワーカー スレッドを作成したり起動したりすることはありません。

C#

```
Producer producer = new Producer(queue, syncEvents);
Consumer consumer = new Consumer(queue, syncEvents);
Thread producerThread = new Thread(producer.ThreadRun);
Thread consumerThread = new Thread(consumer.ThreadRun);
```

キューおよび同期イベント オブジェクトは、Consumer スレッドと Producer スレッドの両方にコンストラクタ引数として渡されます。これによって、両方のオブジェクトにはそれぞれのタスクを実行するうえで必要な共有リソースが提供されます。次に、各オブジェクトの ThreadRun メソッドを引数として使用して、2 つの新しい Thread オブジェクトが作成されます。各ワーカー スレッドは開始時に、スレッドのエントリ ポイントとしてこの引数を使用します。

次に、Main は、次に示すように Start メソッドを呼び出して、2 つのワーカー スレッドを起動します。

C#

```
producerThread.Start();
consumerThread.Start();
```

この時点で、2 つの新しいワーカー スレッドが作成され、現在 Main メソッドを実行しているプライマリ スレッドとは無関係に、非同期の実行が開始されます。実際、次に Main は、Sleep メソッドを呼び出してプライマリ スレッドを中断します。このメソッドは、指定した時間 (ミリ秒単位) だけ現在実行中のスレッドを中断します。指定した時間が経過すると、Main が再度アクティブになり、この時点でキューの内容を表示します。Main は次に示すように、この処理を 4 回繰り返します。

C#

```
for (int i=0; i<4; i++)
{
    Thread.Sleep(2500);
    ShowQueueContents(queue);
}
```

最後に、Main は、"スレッドの終了" イベントの **Set** メソッドを呼び出して、ワーカー スレッドに対し終了を通知します。次に、各ワーカー スレッドで Join メソッドを呼び出し、各ワーカー スレッドがイベントに応答して終了するまでプライマリ スレッドをブロックします。

スレッドの同期の最後の例として、ShowQueueContents メソッドについて説明します。このメソッドは、consumer スレッドや producer スレッドと同様、**lock** を使用してキューへの排他アクセスを取得します。ただしこの場合、ShowQueueContents はコレクション全体を列挙するので、排他アクセスがとりわけ重要になります。コレクションの列挙ではコレクション全体の内容を走査する必要があるため、非同期操作によるデータ破損が特に生じやすくなるからです。ShowQueueContents メソッドは次のようになります。

C#

```
syncEvents.ExitThreadEvent.Set();

producerThread.Join();
consumerThread.Join();
```

最後に、ShowQueueContents は Main によって呼び出されるため、このメソッドがプライマリ スレッドによって実行されることに注意してください。つまり、このメソッドが項目のキューへの排他アクセスを取得した時点で、実際に producer スレッドと consumer スレッドの両方がキューへのアクセスをブロックされることになります。ShowQueueContents は、次のようにキューをロックして内容を列挙します。

C#

```
private static void ShowQueueContents(Queue<int> q)
{
    lock (((ICollection)q).SyncRoot)
    {
        foreach (int item in q)
        {
            Console.Write("{0} ", item);
        }
    }
    Console.WriteLine();
}
```

完全な例を次に示します。

使用例

C#

```
using System;
using System.Threading;
using System.Collections;
using System.Collections.Generic;

public class SyncEvents
{
    public SyncEvents()
    {

        _newItemEvent = new AutoResetEvent(false);
        _exitThreadEvent = new ManualResetEvent(false);
        _eventArray = new WaitHandle[2];
        _eventArray[0] = _newItemEvent;
        _eventArray[1] = _exitThreadEvent;
    }

    public EventWaitHandle ExitThreadEvent
    {
        get { return _exitThreadEvent; }
    }
    public EventWaitHandle NewItemEvent
    {
        get { return _newItemEvent; }
    }
    public WaitHandle[] EventArray
    {
        get { return _eventArray; }
    }

    private EventWaitHandle _newItemEvent;
    private EventWaitHandle _exitThreadEvent;
    private WaitHandle[] _eventArray;
}
public class Producer
{
    public Producer(Queue<int> q, SyncEvents e)
    {
        _queue = q;
        _syncEvents = e;
    }
    // Producer.ThreadRun
    public void ThreadRun()
    {
        int count = 0;
        Random r = new Random();
        while (!_syncEvents.ExitThreadEvent.WaitOne(0, false))
        {
```

```
            lock (((ICollection)_queue).SyncRoot)
            {
                while (_queue.Count < 20)
                {
                    _queue.Enqueue(r.Next(0,100));
                    _syncEvents.NewItemEvent.Set();
                    count++;
                }
            }
        }
        Console.WriteLine("Producer thread: produced {0} items", count);
    }
    private Queue<int> _queue;
    private SyncEvents _syncEvents;
}

public class Consumer
{
    public Consumer(Queue<int> q, SyncEvents e)
    {
        _queue = q;
        _syncEvents = e;
    }
    // Consumer.ThreadRun
    public void ThreadRun()
    {
        int count = 0;
        while (WaitHandle.WaitAny(_syncEvents.EventArray) != 1)
        {
            lock (((ICollection)_queue).SyncRoot)
            {
                int item = _queue.Dequeue();
            }
            count++;
        }
        Console.WriteLine("Consumer Thread: consumed {0} items", count);
    }
    private Queue<int> _queue;
    private SyncEvents _syncEvents;
}

public class ThreadSyncSample
{
    private static void ShowQueueContents(Queue<int> q)
    {
        lock (((ICollection)q).SyncRoot)
        {
            foreach (int item in q)
            {
                Console.Write("{0} ", item);
            }
        }
        Console.WriteLine();
    }

    static void Main()
    {
        Queue<int> queue = new Queue<int>();
        SyncEvents syncEvents = new SyncEvents();

        Console.WriteLine("Configuring worker threads...");
        Producer producer = new Producer(queue, syncEvents);
        Consumer consumer = new Consumer(queue, syncEvents);
        Thread producerThread = new Thread(producer.ThreadRun);
        Thread consumerThread = new Thread(consumer.ThreadRun);

        Console.WriteLine("Launching producer and consumer threads...");
        producerThread.Start();
```

```
            consumerThread.Start();

            for (int i=0; i<4; i++)
            {
                Thread.Sleep(2500);
                ShowQueueContents(queue);
            }

            Console.WriteLine("Signaling threads to terminate...");
            syncEvents.ExitThreadEvent.Set();

            producerThread.Join();
            consumerThread.Join();
        }

    }
```

サンプル出力

```
Configuring worker threads...
Launching producer and consumer threads...
22 92 64 70 13 59 9 2 43 52 91 98 50 96 46 22 40 94 24 87
79 54 5 39 21 29 77 77 1 68 69 81 4 75 43 70 87 72 59
0 69 98 54 92 16 84 61 30 45 50 17 86 16 59 20 73 43 21
38 46 84 59 11 87 77 5 53 65 7 16 66 26 79 74 26 37 56 92
Signalling threads to terminate...
Consumer Thread: consumed 1053771 items
Producer thread: produced 1053791 items
```

参照

処理手順

Monitorによる同期化の技術サンプル
待機による同期化の技術サンプル

関連項目

スレッドの同期 (C# プログラミング ガイド)
lock ステートメント (C# リファレンス)
AutoResetEvent
ManualResetEvent
Set
Join
WaitOne
WaitAll
Queue
ICollection
Start
Sleep
WaitHandle
EventWaitHandle

概念

C# プログラミング ガイド

その他の技術情報

Thread クラス

方法：スレッド プールを使用する (C# プログラミング ガイド)

スレッド プールとは、複数のタスクをバックグラウンドで実行するときに使用できるスレッドのコレクションです (背景情報については、「スレッド処理の使用 (C# プログラミング ガイド)」を参照してください)。これにより、プライマリ スレッドは自由に他のタスクを非同期的に実行できます。

スレッド プールは、サーバー アプリケーションでよく使用されます。受信した要求がそれぞれスレッド プールのスレッドに割り当てられるため、プライマリ スレッドを占有したり、後続の要求の処理を遅延させたりせずに、要求を非同期的に処理できます。

プール内のスレッドがタスクを完了すると、待機スレッドのキューに戻り、再使用できるようになります。これで、アプリケーションは、タスクごとに新しいスレッドを作成するという負荷を避けることができます。

スレッド プールには、通常、最大数のスレッドがあります。スレッドがすべてビジーである場合、追加のタスクは、スレッドが使用可能になり、処理できるようになるまでキューに配置されます。

スレッド プールは独自に実装できますが、.NET Framework が ThreadPool クラスを通じて提供するスレッド プールを使用する方が簡単です。

次の例では、.NET Framework のスレッド プールを使用して、20 から 40 までの 10 個の数値の `Fibonacci` 結果を計算しています。各 `Fibonacci` 結果は、計算を実行する `ThreadPoolCallback` というメソッドを提供する `Fibonacci` クラスによって表されます。各 Fibonacci 値を表すオブジェクトを作成し、`ThreadPoolCallback` メソッドを QueueUserWorkItem に渡すことにより、プールで使用可能なスレッドを割り当て、メソッドを実行します。

各 `Fibonacci` オブジェクトには、計算するセミランダム値が割り当てられ、また 10 個のスレッドがそれぞれ競ってプロセッサ時間を使用するため、10 個の結果がすべて計算されるまでにかかる時間を事前に知ることはできません。そのため、各 `Fibonacci` オブジェクトには、構築時に ManualResetEvent クラスのインスタンスが渡されます。各オブジェクトの計算が終了すると、提供されたイベント オブジェクトが送出されます。これにより、プライマリ スレッドは、10 個の `Fibonacci` オブジェクトの計算が完了するまで、WaitAll によって実行をブロックできます。計算が完了すると、`Main` メソッドによって、各 `Fibonacci` 結果が表示されます。

使用例

C#

```
using System;
using System.Threading;

public class Fibonacci
{
    public Fibonacci(int n, ManualResetEvent doneEvent)
    {
        _n = n;
        _doneEvent = doneEvent;
    }

    // Wrapper method for use with thread pool.
    public void ThreadPoolCallback(Object threadContext)
    {
        int threadIndex = (int)threadContext;
        Console.WriteLine("thread {0} started...", threadIndex);
        _fibOfN = Calculate(_n);
        Console.WriteLine("thread {0} result calculated...", threadIndex);
        _doneEvent.Set();
    }

    // Recursive method that calculates the Nth Fibonacci number.
    public int Calculate(int n)
    {
        if (n <= 1)
        {
            return n;
        }

        return Calculate(n - 1) + Calculate(n - 2);
    }

    public int N { get { return _n; } }
    private int _n;
```

```
        public int FibOfN { get { return _fibOfN; } }
        private int _fibOfN;

        private ManualResetEvent _doneEvent;
    }

    public class ThreadPoolExample
    {
        static void Main()
        {
            const int FibonacciCalculations = 10;

            // One event is used for each Fibonacci object
            ManualResetEvent[] doneEvents = new ManualResetEvent[FibonacciCalculations];
            Fibonacci[] fibArray = new Fibonacci[FibonacciCalculations];
            Random r = new Random();

            // Configure and launch threads using ThreadPool:
            Console.WriteLine("launching {0} tasks...", FibonacciCalculations);
            for (int i = 0; i < FibonacciCalculations; i++)
            {
                doneEvents[i] = new ManualResetEvent(false);
                Fibonacci f = new Fibonacci(r.Next(20,40), doneEvents[i]);
                fibArray[i] = f;
                ThreadPool.QueueUserWorkItem(f.ThreadPoolCallback, i);
            }

            // Wait for all threads in pool to calculation...
            WaitHandle.WaitAll(doneEvents);
            Console.WriteLine("All calculations are complete.");

            // Display the results...
            for (int i= 0; i<FibonacciCalculations; i++)
            {
                Fibonacci f = fibArray[i];
                Console.WriteLine("Fibonacci({0}) = {1}", f.N, f.FibOfN);
            }
        }
    }
```

サンプル出力

```
launching 10 tasks...
result calculated...
result calculated...
result calculated...
result calculated...
result calculated...
result calculated...
result calculated...
result calculated...
result calculated...
result calculated...
all calculations complete
Fibonacci(22) = 17711
Fibonacci(25) = 75025
Fibonacci(32) = 2178309
Fibonacci(36) = 14930352
Fibonacci(32) = 2178309
Fibonacci(26) = 121393
Fibonacci(35) = 9227465
Fibonacci(23) = 28657
Fibonacci(39) = 63245986
Fibonacci(22) = 17711
```

参照

処理手順

Monitorによる同期化の技術サンプル

待機による同期化の技術サンプル

関連項目

スレッド処理 (C# プログラミング ガイド)
スレッド処理の使用 (C# プログラミング ガイド)
Mutex
WaitAll
ManualResetEvent
Set
ThreadPool
QueueUserWorkItem
ManualResetEvent

概念

C# プログラミング ガイド
Monitor

その他の技術情報

.NET Framework におけるセキュリティ
[HOWTO] Visual C# .NET を使用してマルチスレッド環境で共有リソースへのアクセスを同期する方法

パフォーマンス (C# プログラミング ガイド)

このセクションでは、パフォーマンスに悪影響を与える可能性のある 2 つの問題と、パフォーマンスの問題に関するリソースへのリンクを紹介します。

ボックス化とボックス化解除

ボックス化とボックス化解除は、負荷の大きい処理です。値型をボックス化するときは、完全に新しいオブジェクトを作成する必要があります。これには、代入と比較して最大 20 倍もの時間がかかることがあります。また、ボックス化を解除するときは、キャストのプロセスで代入の 4 倍の時間がかかることがあります。詳細については、「ボックス化とボックス化解除」を参照してください。

デストラクタ

空のデストラクタは使用しないでください。デストラクタがクラスに存在するときは、エントリが終了キューで作成されます。デストラクタを呼び出すと、ガベージ コレクタが呼び出され、このキューを処理します。デストラクタが空の場合は、これによってパフォーマンスが低下します。詳細については、「デストラクタ (C# プログラミング ガイド)」を参照してください。

その他の技術情報

- Writing Faster Managed Code: Know What Things Cost
- Writing High-Performance Managed Applications : A Primer
- ガベージ コレクタの基本とパフォーマンスのヒント
- Performance Tips and Tricks in .NET Applications

参照

関連項目

セキュリティ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

リフレクション (C# プログラミング ガイド)

リフレクションは、アセンブリ、モジュール、および型をカプセル化する、Type 型のオブジェクトを提供します。リフレクションを使用すると、動的に型のインスタンスを作成したり、作成したインスタンスを既存のオブジェクトにバインドしたり、さらに既存のオブジェクトから型を取得してそのオブジェクトのメソッドを呼び出したり、フィールドやプロパティにアクセスしたりできます。コードで属性を使用している場合、リフレクションを使用すると、それらにアクセスできます。詳細については、「属性」を参照してください。

次の例は、すべての型が **Object** 基本クラスから継承した静的メソッド **GetType** を使用して変数の型を取得する簡単なリフレクションを示しています。

C#

```
// Using GetType to obtain type information:
int i = 42;
System.Type type = i.GetType();
System.Console.WriteLine(type);
```

出力は次のとおりです。

```
System.Int32
```

次の例では、リフレクションを使用して、読み込まれたアセンブリの完全名を取得します。

C#

```
// Using Reflection to get information from an Assembly:
System.Reflection.Assembly o = System.Reflection.Assembly.Load("mscorlib.dll");
System.Console.WriteLine(o.GetName());
```

出力は次のとおりです。

```
mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

リフレクションの概要

リフレクションは、次の場合に役立ちます。

- プログラムのメタデータ内の属性にアクセスする必要がある場合。「リフレクションによる属性へのアクセス (C# プログラミング ガイド)」を参照してください。
- アセンブリの型をチェックし、インスタンス化する場合。
- 実行時に新しい型を作成する場合。System.Reflection.Emit でクラスを使用します。
- 遅延バインディングを実行するために、実行時に作成された型でメソッドにアクセスする場合。「型の動的な読み込みおよび使用」を参照してください。

関連項目

詳細については以下を参照してください。

- リフレクションの概要
- 型情報の表示
- リフレクションとジェネリック型
- **System.Reflection.Emit**
- リフレクションによる属性へのアクセス (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.12 属性

- 7.5.11 typeof 演算子

参照

関連項目

アプリケーション ドメイン (C# プログラミング ガイド)

概念

C# プログラミング ガイド

アセンブリとグローバル アセンブリ キャッシュ (C# プログラミング ガイド)

C# の DLL (C# プログラミング ガイド)

Visual C# を使用すると、他の .NET アプリケーションやアンマネージ コードからも呼び出すことのできる DLL を作成できます。

このセクションの内容

方法 : C# DLL を作成して使用する (C# プログラミング ガイド)

クラス ライブラリ プロジェクトを Visual C# で作成する方法について説明します。

関連するセクション

/baseaddress (DLL のベース アドレスの指定) (C# コンパイラ オプション)

DLL 関数を保持するクラスの作成

DLL のブレークポイントが機能しない原因について

DLL プロジェクトのデバッグ

参照

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

方法：C# DLL を作成して使用する (C# プログラミング ガイド)

ダイナミック リンク ライブラリ (DLL) は、実行時にプログラムにリンクされます。DLL のビルド例、使用例として、次に示すシナリオを考えてみます。

- `MathLibrary.DLL`: 実行時に呼び出されるメソッドが収められているライブラリ ファイルです。この例では、DLL には 2 つのメソッド `Add` と `Multiply` が含まれています。
- `Add.cs`: メソッド `Add(long i, long j)` が入っているソース ファイルです。このメソッドは、パラメータの和を返します。メソッド `Add` を含むクラス `AddClass` は、名前空間 `UtilityMethods` のメンバです。
- `Mult.cs`: メソッド `Multiply(long x, long y)` のソース コードです。このメソッドは、パラメータの積を返します。メソッド `Multiply` を含むクラス `MultiplyClass` も、名前空間 `UtilityMethods` のメンバです。
- `TestCode.cs`: `Main` メソッドを含むファイルです。DLL ファイルのメソッドを使って、実行時引数の和と積を計算します。

使用例

C#

```
// File: Add.cs
namespace UtilityMethods
{
    public class AddClass
    {
        public static long Add(long i, long j)
        {
            return (i + j);
        }
    }
}
```

C#

```
// File: Mult.cs
namespace UtilityMethods
{
    public class MultiplyClass
    {
        public static long Multiply(long x, long y)
        {
            return (x * y);
        }
    }
}
```

C#

```
// File: TestCode.cs

using UtilityMethods;

class TestCode
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Calling methods from MathLibrary.DLL:");

        if (args.Length != 2)
        {
            System.Console.WriteLine("Usage: TestCode <num1> <num2>");
            return;
        }
```

```
            long num1 = long.Parse(args[0]);
            long num2 = long.Parse(args[1]);

            long sum = AddClass.Add(num1, num2);
            long product = MultiplyClass.Multiply(num1, num2);

            System.Console.WriteLine("{0} + {1} = {2}", num1, num2, sum);
            System.Console.WriteLine("{0} * {1} = {2}", num1, num2, product);
        }
    }
```

このファイルには、DLL のメソッド `Add` と `Multiply` を使用するアルゴリズムが入っています。最初に、コマンド ラインから入力された引数 `num1` と `num2` を解析します。次に、`AddClass` クラスの `Add` メソッドを使って和を計算し、`MultiplyClass` クラスの `Multiply` メソッドを使って積を計算します。

ファイルの先頭で **using** ディレクティブを指定すると、コンパイル中に非修飾クラス名を使って DLL メソッドを参照できます。次はその例です。

C#

```
  MultiplyClass.Multiply(num1, num2);
```

ディレクティブを指定しない場合は、完全修飾名を使用する必要があります。次はその例です。

C#

```
  UtilityMethods.MultiplyClass.Multiply(num1, num2);
```

実行

プログラムを実行するには、EXE ファイルの名前に続けて 2 つの数値を入力します。次はその例です。

```
TestCode 1234 5678
```

出力

```
Calling methods from MathLibrary.DLL:
1234 + 5678 = 6912
1234 * 5678 = 7006652
```

コードのコンパイル方法

`MathLibrary.DLL` ファイルを作成するには、次のコマンド ラインを使って `Add.cs` と `Mult.cs` をコンパイルします。

```
csc /target:library /out:MathLibrary.DLL Add.cs Mult.cs
```

/target:library コンパイラ オプションを指定すると、コンパイラは EXE ファイルではなく DLL ファイルを出力します。/out コンパイラ オプションで指定されたファイル名が、DLL のファイル名になります。このオプションを指定しないと、コンパイラは最初のファイル名 (`Add.cs`) を DLL の名前として使います。

実行可能ファイル `TestCode.exe` を作成するには、次のコマンド ラインを使用します。

```
csc /out:TestCode.exe /reference:MathLibrary.DLL TestCode.cs
```

ここでの **/out** コンパイラ オプションでは、EXE ファイルを出力するようにコンパイラに指示し、出力ファイルの名前 (`TestCode.exe`) を指定しています。このコンパイラ オプションは省略できます。/reference コンパイラ オプションは、このプログラムが使用する DLL ファイルを指定します。

参照

処理手順

方法 : DLL のベース アドレスを指定する

概念

C# プログラミング ガイド

DLL 関数を保持するクラスの作成

セキュリティ (C# プログラミング ガイド)

セキュリティは、すべての C# アプリケーションにとって不可欠な側面であり、設計や実装が完了したときだけでなく、開発のすべての段階で考慮する必要があります。

セキュリティに関する C# 固有の推奨事項

以下の一覧は、セキュリティ上の問題をすべて網羅したものではありません。この一覧では、C# での開発時に認識しておく必要がある一般的な問題に焦点を絞っています。

- 整数型の算術演算および変換に対してオーバーフロー チェック コンテキストを制御するには、checked キーワードを使用してください。
- パラメータには、必ず最も制限されたデータ型を使用してください。たとえば、データ構造体のサイズを表す値をメソッドに渡すときは、整数ではなく、符号なし整数を使用してください。
- ファイル名に基づいて判断しないでください。ファイル名は多様に表現できるので、ファイルによってはテストが回避される可能性があるからです。
- パスワードやその他の機密情報は、絶対にアプリケーションにハードコーディングしないでください。
- SQL クエリの生成に使用される入力は、必ず検証してください。
- メソッドへの入力はすべて検証してください。System.Text.RegularExpressions 名前空間の正規表現メソッドは、電子メール アドレスなどの入力が正しい書式であることを確認するのに便利です。
- 例外情報は表示しないでください。例外情報を表示すると、潜在的な攻撃者に貴重な手掛かりを与えてしまいます。
- 最小限の特権で実行しているときにアプリケーションが正常に動作することを確認してください。管理者としてログインするようにユーザーに求めるアプリケーションはほとんどありません。
- 独自の暗号化アルゴリズムを使用せずに、System.Security.Cryptography クラスを使用してください。
- アセンブリには、厳密な名前を付けてください。
- XML ファイルやその他の構成ファイルに機密情報を保管しないでください。
- ネイティブ コードをラップするマネージ コードは、慎重にチェックしてください。ネイティブ コードが安全であり、特にバッファ オーバーランに対して保護されることを確認してください。
- アプリケーションの外部から引き渡されたデリゲート型を使用するときは、十分に注意してください。
- FxCop をアセンブリで実行し、Microsoft .NET Framework の設計ガイドラインに準拠していることを確認してください。FxCop は、200 種類以上のコードの問題を検出し、警告を出力することもできます。

このセクションの内容

以下の MSDN トピックでは、安全で信頼性の高いソフトウェアの作成について詳しく説明します。

- Microsoft Security Developer Center.
- Defend Your Apps and Critical User Info with Defensive Coding Techniques.
- .NET Framework におけるセキュリティ概要.
- Defend Your Code with Top Ten Security Tips Every Developer Must Know.
- Security Brief: Beware of Fully Trusted Code.
- 報告された Microsoft ASP.NET の脆弱性に関する情報.
- Windows フォームのセキュリティの概要
- 安全なマネージ コントロールの作成

参照
概念
C# プログラミング ガイド

C# リファレンス

このセクションでは、C# のキーワード、演算子、およびコンパイラのエラーと警告に関するリファレンス情報を紹介します。

このセクションの内容

C# のキーワード

C# のキーワードと構文に関する情報を提供します。

C# の演算子

C# の演算子と構文に関する情報を提供します。

C# プリプロセッサ ディレクティブ

C# ソース コードに埋め込むコンパイラ コマンドに関する情報を提供します。

C# コンパイラ オプション

コンパイラ オプションとその使用法に関する情報を提供します。

C# 用語集

C# 関連の用語集です。

関連するセクション

C# 言語仕様

Microsoft Word 形式で作成された最新バージョンの C# 言語仕様へのリンクを紹介します。

C# FAQ

C# Developer Center 内の、C# についてよく寄せられる質問の一覧を提供します。

C# に関する Microsoft サポート技術情報の文書

MSDN に格納されている、C# に関連したサポート技術情報の文書を検索します。

Visual C#

Visual C# ドキュメントへのポータルを提供します。

Visual C# のサンプル

Visual C# サンプルの一覧およびサンプルへのリンクを提供します。

Visual C# のコード エディタの機能

IDE およびエディタについて説明する概念トピックおよびタスク トピックへのリンクを提供します。

Visual C# によるアプリケーションの作成

一般的なプログラミング タスクの実施方法を説明するトピックへのリンクを提供します。

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C#

C# のキーワード

キーワードは、コンパイラに対して特別な意味を持つ定義済みの識別子であり、既に予約されています。プリフィックスとして `@` を付けない限り、プログラム内で識別子として使うことはできません。たとえば、`@if` は有効な識別子ですが、`if` はキーワードであるため違います。

abstract	event	new	struct
as	explicit	null	switch
base	extern	object	this
bool	false	operator	throw
break	finally	out	true
byte	fixed	override	try
case	float	params	typeof
catch	for	private	uint
char	foreach	protected	ulong
checked	goto	public	unchecked
class	if	readonly	unsafe
const	implicit	ref	ushort
continue	in	return	using
decimal	int	sbyte	virtual
default	interface	sealed	volatile
delegate	internal	short	void
do	is	sizeof	while
double	lock	stackalloc	
else	long	static	
enum	namespace	string	

コンテキスト キーワード

get	partial	set
value	where	yield

参照

概念

C# プログラミング ガイド

その他の技術情報

型 (C# リファレンス)

C# の型指定システムには、次のカテゴリがあります。

- 値型
- 参照型
- ポインタ型

値型の変数はデータを格納し、参照型の変数は実データへの参照を格納します。参照型は、オブジェクトとも呼ばれます。ポインタ型は、unsafe モードだけで使用できます。

ボックス化とボックス化解除を使用して、値型を参照型に変換でき、参照型から値型に変換して戻すことができます。ボックス化された型を除き、参照型を値型に変換することはできません。

ここでは、void についても説明します。

値型は null 許容です。つまり、値以外の状態を格納できます。詳細については、「nullable 型 (C# プログラミング ガイド)」を参照してください。

参照

関連項目

C# のキーワード
キャスト (C# プログラミング ガイド)
データ型 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス
型のリファレンス表 (C# リファレンス)

値型 (C# リファレンス)

値型は、次の 2 つの主要カテゴリで構成されます。

- 構造体
- 列挙体

構造体は、次のカテゴリに分類されます。

- 数値型
    - 整数型
    - 浮動小数点型
    - decimal
- bool
- ユーザー定義の struct 型。

値型の主な機能

値型に基づく変数は、値を直接含みます。ある値型の変数を別の変数に代入すると、含まれている値がコピーされます。これは参照型の変数の代入とは異なります。参照型の変数の代入では、オブジェクト自体ではなく、オブジェクトへの参照がコピーされます。

すべての値型は、System.ValueType から暗黙的に派生します。

参照型とは異なり、値型から新しい型を派生させることはできません。ただし、参照型と同様に、struct 型はインターフェイスを実装できます。

参照型とは異なり、値型に **null** 値を含めることはできません。ただし、Null 許容型の機能では、値型を **null** に代入できます。

それぞれの値型には、その型の既定値を初期化する既定のコンストラクタが暗黙的に存在します。値型の既定値の詳細については、「既定値の一覧表」を参照してください。

単純型の主な機能

C# 言語に不可欠なすべての単純型は、.NET Framework System 型のエイリアスです。たとえば、int は System.Int32 のエイリアスです。すべてのエイリアスの一覧については、「組み込み型の一覧表 (C# リファレンス)」を参照してください。

定数式のオペランドがすべて単純型の定数である場合、定数式はコンパイル時に評価されます。

単純型は、リテラルを使用して初期化できます。たとえば、'A' は **char** 型のリテラルであり、2001 は **int** 型のリテラルです。

値型の初期化

C# のローカル変数は、使用する前に初期化する必要があります。たとえば、次のように、初期化せずにローカル変数を宣言するとします。

```
int myInt;
```

この変数は、初期化した後でないと使用できません。初期化するには、次のステートメントを使用します。

```
myInt = new int();  // Invoke default constructor for int type.
```

このステートメントは、下のステートメントと同等です。

```
myInt = 0;         // Assign an initial value, 0 in this example.
```

宣言と初期化を 1 つのステートメントで行うことができます。次に例を示します。

```
int myInt = new int();
```

または

```
int myInt = 0;
```

new 演算子を使用すると、それぞれの型の既定のコンストラクタが呼び出され、既定値が変数に代入されます。前の例では、既定のコンストラクタが `myInt` に値 `0` を代入します。既定のコンストラクタによる値の代入の詳細については、「既定値の一覧表」を参照してください。

ユーザー定義型の場合は、new を使用して既定のコンストラクタを呼び出します。たとえば、`Point` struct の既定のコンストラクタを呼び出すステートメントは、次のとおりです。

```
Point p = new Point(); // Invoke default constructor for the struct.
```

既定のコンストラクタが呼び出されると、struct は明らかに代入済みと見なされ、すべてのメンバがそれぞれの既定値に初期化されます。

new 演算子の詳細については、「new」を参照してください。

数値型の出力書式の詳細については、「数値結果の書式指定の一覧表」を参照してください。

参照

関連項目

C# のキーワード

参照型 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

型 (C# リファレンス)

型のリファレンス表 (C# リファレンス)

bool (C# リファレンス)

bool キーワードは、System.Boolean のエイリアスです。ブール値 (true および false) を格納する変数を宣言するときに使用します。

メモ：
null の値も受け取ることができるブール変数が必要な場合は、`bool` を使用します。詳細については、「null 許容型 (C# プログラミング ガイド)」を参照してください。

リテラル

bool 変数にはブール値を代入できます。**bool** として評価される式を **bool** 変数に代入することもできます。

```
// keyword_bool.cs
using System;
public class MyClass
{
    static void Main()
    {
        bool i = true;
        char c = '0';
        Console.WriteLine(i);
        i = false;
        Console.WriteLine(i);

        bool Alphabetic = (c > 64 && c < 123);
        Console.WriteLine(Alphabetic);
    }
}
```

出力

```
True
False
False
```

変換

C++ では、**bool** 型の値を **int** 型の値に変換できます。つまり、**false** は 0 と等価であり、**true** は 0 以外の値と等価です。C# では、**bool** 型とその他の型は変換できません。たとえば、次の **if** ステートメントは、C++ では有効ですが、C# では無効です。

```
int x = 123;
if (x)   // Invalid in C#
{
    printf_s("The value of x is nonzero.");
}
```

int 型の変数をテストするには、次のように、明示的に特定の値 (たとえば 0) と比較する必要があります。

```
int x = 123;
if (x != 0)   // The C# way
{
    Console.Write("The value of x is nonzero.");
}
```

使用例

ここでは、キーボードから文字が入力されると、入力文字がアルファベットかどうかをチェックするプログラムの例を示します。アルファベットの場合は、大文字か小文字かをチェックします。これらのチェックは、IsLetter および IsLower を使用して実行され、この両方が **bool** 型を返します。

```
// keyword_bool_2.cs
using System;
public class BoolTest
```

```
{
    static void Main()
    {
        Console.Write("Enter a character: ");
        char c = (char)Console.Read();
        if (Char.IsLetter(c))
        {
            if (Char.IsLower(c))
            {
                Console.WriteLine("The character is lowercase.");
            }
            else
            {
                Console.WriteLine("The character is uppercase.");
            }
        }
        else
        {
            Console.WriteLine("Not an alphabetic character.");
        }
    }
}
```

入力

```
X
```

サンプル出力

```
Enter a character: X
The character is uppercase.
Additional sample runs might look as follow:
Enter a character: x
The character is lowercase.

Enter a character: 2
The character is not an alphabetic character.
```

C# 言語仕様

bool および関連項目の詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 4.1.8 bool 型
- 7.9.4 ブール型等値演算子
- 7.11.1 ブール条件論理演算子

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

byte (C# リファレンス)

byte キーワードは、次の表に示された値を格納する整数型を示します。

型	範囲	サイズ	.NET Framework 型
byte	0 ～ 255	符号なし 8 ビット整数	System.Byte

リテラル

byte 変数の宣言と初期化の例を次に示します。

```
byte myByte = 255;
```

上のように宣言すると、整数リテラル `255` は暗黙的に int から **byte** に変換されます。整数リテラルが **byte** の範囲を超えると、コンパイル エラーになります。

変換

byte から short、ushort、int、uint、long、ulong、float、double、decimal への暗黙の型変換が組み込まれています。

より大きな記憶領域のサイズを持つ、リテラル以外の数値型を暗黙的に **byte** に変換することはできません。整数型の記憶領域サイズの詳細については、「整数型の一覧表 (C# リファレンス)」を参照してください。たとえば、2 つの **byte** 変数 `x` と `y` があるとします。

```
        byte x = 10, y = 20;
```

次の代入ステートメントは、代入演算子の右側にある算術式が既定で **int** に評価されるため、コンパイル エラーになります。

```
// Error: conversion from int to byte:
byte z = x + y;
```

このエラーを修正するには、キャストを使用します。

```
// OK: explicit conversion:
byte z = (byte)(x + y);
```

ただし、次のステートメントは使用できます。このステートメントでは、変換先の変数の記憶領域サイズは元のサイズ以上になります。

```
int x = 10, y = 20;
int m = x + y;
long n = x + y;
```

浮動小数点型から **byte** への暗黙の型変換はありません。たとえば、次のステートメントは、明示的なキャストを使用しない場合、コンパイラ エラーになります。

```
// Error: no implicit conversion from double:
byte x = 3.0;
// OK: explicit conversion:
byte y = (byte)3.0;
```

オーバーロードされたメソッドを呼び出すときは、キャストを使用する必要があります。たとえば、**byte** パラメータと int パラメータを使用したオーバーロードされたメソッドがあるとします。

```
public static void SampleMethod(int i) {}
public static void SampleMethod(byte b) {}
```

byte キャストを使用すると、正しい型が呼び出されます。次に例を示します。

```
// Calling the method with the int parameter:
SampleMethod(5);
// Calling the method with the byte parameter:
SampleMethod((byte)5);
```

浮動小数点型と整数型の混在する算術式の詳細については、「float」と「double」を参照してください。

暗黙の数値変換規則の詳細については、「暗黙的な数値変換の一覧表 (C# リファレンス)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.5 整数型

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
Byte

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

char (C# リファレンス)

char キーワードを使用して、次の表に示された範囲の Unicode 文字を宣言します。Unicode 文字は、世界中の文字言語のほとんどを 16 ビット文字で表します。

型	範囲	サイズ	.NET Framework 型
char	U+0000 ～ U+ffff	Unicode 16 ビット文字	System.Char

リテラル

char 型の定数は、文字リテラル、16 進のエスケープ シーケンス、Unicode 表現として記述できます。また、整数の文字コードをキャストできます。次のステートメントはすべて **char** 変数を宣言し、文字 `x` を使って初期化しています。

```
        char char1 = 'Z';        // Character literal
char char2 = '\x0058';   // Hexadecimal
char char3 = (char)88;   // Cast from integral type
char char4 = '\u0058';   // Unicode
```

変換

char は、ushort、int、uint、long、ulong、float、double、または decimal に暗黙的に変換できます。ただし、他の型から **char** 型への暗黙の型変換はありません。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 2.4.4.4 文字リテラル
- 4.1.5 整数型

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
Char Structure

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

decimal (C# リファレンス)

decimal キーワードは、128 ビットのデータ型を示します。**decimal** 型は、浮動小数点型よりも有効桁数が多く、範囲が狭いので、財務や金融の計算に適しています。**decimal** 型の概算の範囲と有効桁数は、次のとおりです。

型	おおよその範囲	有効桁数	.NET Framework 型
decimal	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$	有効桁数 28 ～ 29	System.Decimal

リテラル

実数値リテラルを **decimal** として扱うには、サフィックス m または M を使用します。次に例を示します。

```
        decimal myMoney = 300.5m;
```

サフィックス m がない場合は double として扱われ、コンパイラ エラーになります。

変換

整数型は、暗黙的に **decimal** に変換され、結果は **decimal** になります。したがって、サフィックスなしで整数リテラルを使用して 10 進変数を初期化できます。次に例を示します。

```
        decimal myMoney = 300;
```

浮動小数点型と **decimal** 型の間に暗黙の型変換はありません。2 つの型の間で変換を実行するには、キャストを使用する必要があります。次に例を示します。

```
        decimal myMoney = 99.9m;
double x = (double)myMoney;
myMoney = (decimal)x;
```

decimal 型と数値の整数型を同じ式に混在させることもできます。ただし、**decimal** 型と浮動小数点型をキャストなしで混在させると、コンパイル エラーになります。

暗黙の数値変換の詳細については、「暗黙的な数値変換の一覧表 (C# リファレンス)」を参照してください。

明示的な数値変換の詳細については、「明示的な数値変換の一覧表 (C# リファレンス)」を参照してください。

10 進出力の書式指定

結果の書式を指定するには、**String.Format** メソッドを使用するか、**String.Format()** を呼び出す System.Console.Write メソッドを使用します。通貨書式を指定するには、例 2 に示すように、標準の通貨の書式指定文字列 "C" または "c" を使用します。**String.Format** メソッドの詳細については、「System.String.Format(System.String,System.Object)」を参照してください。

使用例

ここでは、同じ式に **decimal** と int が混在している例を示します。結果は **decimal** 型になります。

double 変数と **decimal** 変数を追加する場合には、次のようなステートメントを使用します。

```
double x = 9;
Console.WriteLine(d + x); // Error
```

次のエラー メッセージが表示されます。

```
Operator '+' cannot be applied to operands of type 'double' and 'decimal'
```

```
// keyword_decimal.cs
// decimal conversion
using System;
public class TestDecimal
```

```
{
    static void Main ()
    {
        decimal d = 9.1m;
        int y = 3;
        Console.WriteLine(d + y);   // Result converted to decimal
    }
}
```

出力

```
12.1
```

ここでは、通貨の書式指定文字列を使用して、出力の書式を指定する例を示します。小数点以下の桁数が $0.99 を超えるため、`x` を丸めている点に注意してください。最大固定桁数を表す変数 `y` は、適切に書式設定された数値がそのまま表示されます。

```
// keyword_decimal2.cs
// Decimal type formatting
using System;
public class TestDecimalFormat
{
    static void Main ()
    {
        decimal x = 0.999m;
        decimal y = 9999999999999999999999999999m;
        Console.WriteLine("My amount = {0:C}", x);
        Console.WriteLine("Your amount = {0:C}", y);
    }
}
```

出力

```
My amount = $1.00
Your amount = $9,999,999,999,999,999,999,999,999,999.00
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.7 Decimal 型

参照
関連項目
C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
Decimal Structure
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

double (C# リファレンス)

double キーワードは、64 ビットの浮動小数点数を格納する単純型を示します。**double** 型の有効桁数とおおよその範囲は、次のとおりです。

型	おおよその範囲	有効桁数	.NET Framework 型
double	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	15 ～ 16 桁	System.Double

リテラル

既定では、代入演算子の右側にある実数値リテラルは **double** として扱われます。ただし、整数を **double** として扱う必要がある場合は、サフィックス d または D を使用します。次に例を示します。

```
        double x = 3D;
```

変換

数値の整数型と浮動小数点型を 1 つの式に混在させることができます。混在する場合は、整数型が浮動小数点型に変換されます。式の評価は、次の規則に従います。

- 浮動小数点型のうちの 1 つが **double** である場合、式は **double** になります。関係式またはブール式の場合は bool になります。
- 式に **double** 型が 1 つも存在しない場合、式は float になります。関係式またはブール式の場合は bool になります。

浮動小数点の式に含まれる値は、次のとおりです。

- 正および負の 0
- 正および負の無限大
- 非数 (NaN)
- 有限の 0 以外の値

値の詳細については、「IEEE Standard for Binary Floating-Point Arithmetic」(http://www.ieee.org/portal/index.jsp) を参照してください。

使用例

次の例では、int、short、float、および **double** を加算した結果が、**double** 型として出力されます。

```
// keyword_double.cs
// Mixing types in expressions
using System;
class MixedTypes
{
    static void Main()
    {
        int x = 3;
        float y = 4.5f;
        short z = 5;
        double w = 1.7E+3;
        // Result of the 2nd argument is a double:
        Console.WriteLine("The sum is {0}", x + y + z + w);
    }
}
```

出力

```
The sum is 1712.5
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.5 整数型

参照

関連項目

C# のキーワード
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス
既定値の一覧表 (C# リファレンス)
浮動小数点型の一覧表 (C# リファレンス)

enum (C# リファレンス)

enum キーワードを使用して、列挙型を宣言します。列挙型は、列挙子並びと呼ばれる名前付き定数の集まりで構成された固有の型です。すべての列挙型には基になる型があり、基になる型には char 以外の任意の整数型を指定できます。列挙要素の基になる既定の型は int です。既定では、最初の列挙子の値は 0 で、後続の列挙子の値は 1 ずつ増加していきます。次に例を示します。

```
enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

この列挙では、`Sat` は `0`、`Sun` は `1`、`Mon` は `2` のように 1 ずつ増加していきます。列挙子に初期化子を指定すると、既定値をオーバーライドします。次に例を示します。

```
enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
```

この列挙では、要素の並びは `0` ではなく、`1` から開始します。

型 `Days` の変数には、基になる型の範囲内であれば任意の値を割り当てることができます。値は名前付き定数に限定されません。

enum E の既定値は、式 `(E)0` によって算出された値です。

メモ :

列挙子の名前に空白を使用することはできません。

基になる型は、列挙子ごとに割り当てるストレージの大きさを指定します。ただし、**enum** 型を整数型に変換するには、明示的なキャストが必要です。たとえば、次のステートメントでは、キャストを使用して **enum** から int に変換することで、列挙子 `Sun` を int 型の変数に代入します。

```
int x = (int)Days.Sun;
```

ビットごとの OR 演算と組み合わされている要素を含む列挙体に System.FlagsAttribute を適用すると、一部のツールで **enum** を使用するときに、その動作に属性が反映されます。このような変更は、**Console** クラス メソッド、式エバリュエータなどのツールを使用するときに生じます (例 3 を参照してください)。

信頼性の高いプログラミング

新しいバージョンの列挙型に追加の値を割り当てるか、新しいバージョンの列挙型メンバの値を変更すると、依存関係のあるソース コードに問題が発生することがあります。**enum** 値を switch ステートメントで使用し、さらに追加要素が **enum** 型に追加されている場合に、既定値のテストを行うと、予期しない true が返されることがあります。

作成したコードを他の開発者が使用する場合、新規要素を **enum** 型に追加したときのコードの動作を示すガイドラインを記述しておくことが重要です。

使用例

ここでは、列挙 `Days` の宣言例を示します。2 つの列挙子は明示的に整数に変換され、整数変数に代入されます。

```
// keyword_enum.cs
// enum initialization:
using System;
public class EnumTest
{
    enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};

    static void Main()
    {
        int x = (int)Days.Sun;
        int y = (int)Days.Fri;
        Console.WriteLine("Sun = {0}", x);
        Console.WriteLine("Fri = {0}", y);
    }
}
```

出力

```
Sun = 2
Fri = 7
```

この例では、**long** 型のメンバを持つ **enum** を宣言するときに基本型オプションを使用しています。列挙体の基になる型が **long** であっても、キャストを使用して列挙体メンバを **long** 型に明示的に変換する必要があります。

```
// keyword_enum2.cs
// Using long enumerators
using System;
public class EnumTest
{
    enum Range :long {Max = 2147483648L, Min = 255L};
    static void Main()
    {
        long x = (long)Range.Max;
        long y = (long)Range.Min;
        Console.WriteLine("Max = {0}", x);
        Console.WriteLine("Min = {0}", y);
    }
}
```

出力

```
Max = 2147483648
Min = 255
```

次のコード例では、**enum** 宣言での **System.FlagsAttribute** 属性の使用とその効果を示します。

```
// enumFlags.cs
// Using the FlagsAttribute on enumerations.
using System;

[Flags]
public enum CarOptions
{
    SunRoof = 0x01,
    Spoiler = 0x02,
    FogLights = 0x04,
    TintedWindows = 0x08,
}

class FlagTest
{
    static void Main()
    {
        CarOptions options = CarOptions.SunRoof | CarOptions.FogLights;
        Console.WriteLine(options);
        Console.WriteLine((int)options);
    }
}
```

出力

```
SunRoof, FogLights
5
```

説明

`Sat=1` の初期化子を削除した場合の結果は、次のとおりです。

```
Sun = 1
Fri = 6
```

説明

この例では、**FlagsAttribute** を削除すると次のように出力される点に注意してください。

```
5
5
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.10 列挙型
- 6.2.2 明示的な列挙値変換
- 14 列挙型

参照

処理手順

属性のサンプル

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)

概念

C# プログラミング ガイド
列挙体のデザイン

その他の技術情報

C# リファレンス

float (C# リファレンス)

float キーワードは、32 ビットの浮動小数点数を格納する単純型を示します。**float** 型の有効桁数とおおよその範囲は、次のとおりです。

型	おおよその範囲	有効桁数	.NET Framework 型
float	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	7 桁	System.Single

リテラル

既定では、代入演算子の右側にある実数値リテラルは double として扱われます。したがって、float 変数を初期化するには、サフィックス `f` または `F` を使用します。次に例を示します。

```
float x = 3.5F;
```

上の宣言でサフィックスを使用しなかった場合は、double 値を **float** 変数に格納することになるため、コンパイル エラーになります。

変換

数値の整数型と浮動小数点型を 1 つの式に混在させることができます。混在する場合は、整数型が浮動小数点型に変換されます。式の評価は、次の規則に従います。

- 浮動小数点型のうちの 1 つが double である場合、式は double になります。関係式またはブール式の場合は bool になります。
- 式に double 型が 1 つも存在しない場合、式は **float** になります。関係式またはブール式の場合は bool になります。

浮動小数点の式に含まれる値は、次のとおりです。

- 正および負の 0
- 正および負の無限大
- 非数 (NaN)
- 有限の 0 以外の値

値の詳細については、「IEEE Standard for Binary Floating-Point Arithmetic」(http://www.ieee.org/) を参照してください。

使用例

次の例では、int、short、および **float** は数式で使用されているため、**float** 型を結果として返します。この式には double が存在しない点に注意してください。

```
// keyword_float.cs
// Mixing types in expressions
using System;
class MixedTypes
{
    static void Main()
    {
        int x = 3;
        float y = 4.5f;
        short z = 5;
        Console.WriteLine("The result is {0}", x * y / z);
    }
}
```

出力

```
The result is 2.7
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 4.1.6 浮動小数点型

- 6.2.1 明示的な数値変換

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
Single Structure

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

int (C# リファレンス)

int キーワードは、次の表に示されたサイズと範囲に従って値を格納する整数型を示します。

型	範囲	サイズ	.NET Framework 型
int	-2,147,483,648 ～ 2,147,483,647	符号付き 32 ビット整数	System.Int32

リテラル

int 型の変数の宣言と初期化の例を次に示します。

```
        int i = 123;
```

サフィックスがない整数リテラルの型は、**int**、uint、long、ulong のうち、その整数の値を表すことができる最も範囲の狭い型になります。この例では、**int** 型です。

変換

int から long、float、double、decimal への暗黙の型変換が組み込まれています。次に例を示します。

```
// '123' is an int, so an implicit conversion takes place here:
float f = 123;
```

sbyte、byte、short、ushort、または char から **int** への暗黙の型変換が組み込まれています。たとえば、次の代入ステートメントは、キャストを使用しない場合、コンパイル エラーになります。

```
long aLong = 22;
int i1 = aLong;       // Error: no implicit conversion from long.
int i2 = (int)aLong;  // OK: explicit conversion.
```

また、浮動小数点型から **int** への暗黙の型変換が行われないことに注意してください。たとえば、次のステートメントは、明示的なキャストを使用しない場合、コンパイラ エラーになります。

```
        int x = 3.0;         // Error: no implicit conversion from double.
int y = (int)3.0;    // OK: explicit conversion.
```

浮動小数点型と整数型の混在する算術式の詳細については、「float (C# リファレンス)」と「double (C# リファレンス)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.5 整数型

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
Int32 Structure

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

long (C# リファレンス)

long キーワードは、次の表に示されたサイズと範囲に従って値を格納する整数型を示します。

型	範囲	サイズ	.NET Framework 型
long	–9,223,372,036,854,775,808 ～ 9,223,372,036,854,775,807	符号付き 64 ビット整数	System.Int64

リテラル

long 変数の宣言と初期化の例を次に示します。

```
        long long1 = 4294967296;
```

サフィックスがない整数リテラルの型は、int、uint、**long**、ulong のうち、その整数の値を表すことができる最も範囲の狭い型になります。上の例では、uint の範囲を超えているので、**long** 型になります。整数型の記憶サイズについては、「整数型の一覧表 (C# リファレンス)」を参照してください。

また、**long** 型にサフィックス L を付けることもできます。次に例を示します。

```
        long long2 = 4294967296L;
```

サフィックス L を使用した場合、リテラル整数の型は、サイズに応じて **long** または ulong のいずれかに決まります。この例では、ulong の範囲よりも小さいため、**long** になります。

サフィックスは、オーバーロードされたメソッドの呼び出しでよく使われます。たとえば、**long** パラメータと int パラメータを使用したオーバーロードされたメソッドがあるとします。

```
public static void SampleMethod(int i) {}
public static void SampleMethod(long l) {}
```

サフィックス L を使用すると、正しい型が呼び出されます。次に例を示します。

```
SampleMethod(5);    // Calling the method with the int parameter
SampleMethod(5L);   // Calling the method with the long parameter
```

long 型とその他の数値の整数型を同じ式で使用できます。その場合、式は **long** として評価されます。関係式またはブール式の場合は bool として評価されます。**long** として評価される式の例を次に示します。

```
898L + 88
```

メモ :

小文字の "l" もサフィックスとして使用できます。ただし、文字 "l" は数字の "1" と混同しやすいので、コンパイラから警告が出されます。明確にするには "L" を使用します。

浮動小数点型と整数型の混在する算術式の詳細については、「float」と「double」を参照してください。

変換

long から float、double、または decimal への暗黙の型変換が組み込まれています。その他の型の場合は、キャストを使用する必要があります。たとえば、次のステートメントは、明示的なキャストを使用しない場合、コンパイル エラーになります。

```
int x = 8L;        // Error: no implicit conversion from long to int
int x = (int)8L;   // OK: explicit conversion to int
```

sbyte、byte、short、ushort、int、uint、または char から **long** への暗黙の型変換が組み込まれています。

また、浮動小数点型から **long** への暗黙の型変換が行われないことに注意してください。たとえば、次のステートメントは、明示的なキャストを使用しない場合、コンパイラ エラーになります。

```
        long x = 3.0;          // Error: no implicit conversion from double
long y = (long)3.0;   // OK: explicit conversion
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.5 整数型

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
Int64 Structure

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

sbyte (C# リファレンス)

sbyte キーワードは、次の表に示されたサイズと範囲に従って値を格納する整数型を示します。

型	範囲	サイズ	.NET Framework 型
sbyte	-128 ～ 127	符号付き 8 ビット整数	System.SByte

リテラル

sbyte 変数の宣言と初期化の例を次に示します。

```
        sbyte sByte1 = 127;
```

上のように宣言すると、整数リテラル 127 は暗黙的に int から **sbyte** に変換されます。整数リテラルが **sbyte** の範囲を超えると、コンパイル エラーになります。

オーバーロードされたメソッドを呼び出す場合は、キャストを使用する必要があります。たとえば、**sbyte** パラメータと int パラメータを使用したオーバーロードされたメソッドがあるとします。

```
public static void SampleMethod(int i) {}
public static void SampleMethod(sbyte b) {}
```

sbyte キャストを使用すると、正しい型が呼び出されます。次に例を示します。

```
// Calling the method with the int parameter:
SampleMethod(5);
// Calling the method with the sbyte parameter:
SampleMethod((sbyte)5);
```

変換

sbyte から short、int、long、float、double、decimal への暗黙の型変換が組み込まれています。

記憶サイズがより大きいリテラル以外の数値型は、**sbyte** への暗黙の型変換ができません。整数型の記憶サイズについては、「整数型の一覧表 (C# リファレンス)」を参照してください。たとえば、2 つの **sbyte** 変数 x と y があるとします。

```
        sbyte x = 10, y = 20;
```

次の代入ステートメントは、代入演算子の右側にある算術式が既定で int に評価されるため、コンパイル エラーになります。

```
        sbyte z = x + y;   // Error: conversion from int to sbyte
```

この問題を解決するには、次のような式をキャストします。

```
        sbyte z = (sbyte)(x + y);   // OK: explicit conversion
```

ただし、次のステートメントは使用できます。このステートメントでは、変換先の変数の記憶サイズは元のサイズ以上になります。

```
        sbyte x = 10, y = 20;
int m = x + y;
long n = x + y;
```

また、浮動小数点型から **sbyte** への暗黙の型変換が行われないことに注意してください。たとえば、次のステートメントは、明示的なキャストを

使用しない場合、コンパイラ エラーになります。

```
        sbyte x = 3.0;         // Error: no implicit conversion from double
sbyte y = (sbyte)3.0;  // OK: explicit conversion
```

浮動小数点型と整数型の混在する算術式の詳細については、「float」と「double」を参照してください。

暗黙の数値変換規則の詳細については、「暗黙的な数値変換の一覧表 (C# リファレンス)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.5 整数型

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
SByte Structure

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

short (C# リファレンス)

short キーワードは、次の表に示されたサイズと範囲に従って値を格納する整数データ型を示します。

データ型	範囲	サイズ	.NET Framework 型
short	-32,768 ～ 32,767	符号付き 16 ビット整数	System.Int16

リテラル

short 変数の宣言と初期化の例を次に示します。

```
short x = 32767;
```

上のように宣言すると、整数リテラル `32767` は暗黙的に int から **short** に変換されます。整数リテラルを **short** の格納場所に格納できない場合は、コンパイル エラーになります。

オーバーロードされたメソッドを呼び出す場合は、キャストを使用する必要があります。たとえば、**short** パラメータと int パラメータを使用したオーバーロードされたメソッドがあるとします。

```
public static void SampleMethod(int i) {}
public static void SampleMethod(short s) {}
```

short キャストを使用すると、正しい型が呼び出されます。次に例を示します。

```
SampleMethod(5);          // Calling the method with the int parameter
SampleMethod((short)5);   // Calling the method with the short parameter
```

変換

short から int、long、float、double、decimal への暗黙の型変換が組み込まれています。

記憶サイズがより大きいリテラル以外の数値型は、**short** への暗黙の型変換ができません。整数型の記憶サイズについては、「整数型の一覧表 (C# リファレンス)」を参照してください。たとえば、2 つの **short** 変数 `x` と `y` があるとします。

```
short x = 5, y = 12;
```

次の代入ステートメントは、代入演算子の右側にある算術式が既定で int に評価されるため、コンパイル エラーになります。

```
short z = x + y; // Error: no conversion from int to short
```

このエラーを修正するには、キャストを使用します。

```
short z = (short)(x + y); // OK: explicit conversion
```

ただし、次のステートメントは使用できます。このステートメントでは、変換先の変数の記憶サイズは元のサイズ以上になります。

```
int m = x + y;
long n = x + y;
```

浮動小数点型から **short** への暗黙の型変換はありません。たとえば、次のステートメントは、明示的なキャストを使用しない場合、コンパイラ エラーになります。

```
short x = 3.0;          // Error: no implicit conversion from double
short y = (short)3.0;   // OK: explicit conversion
```

浮動小数点型と整数型の混在する算術式の詳細については、「float」と「double」を参照してください。

暗黙の数値変換規則の詳細については、「暗黙的な数値変換の一覧表 (C# リファレンス)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.5 整数型

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
Int16

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

struct (C# リファレンス)

struct 型は、通常、四角形の座標や在庫品目の特性など、関連のある変数の小さなグループをカプセル化するために使用します。次の例は、単純な構造体の宣言を示しています。

```
public struct Book
{
    public decimal price;
    public string title;
    public string author;
}
```

解説

構造体には、コンストラクタ、定数、フィールド、メソッド、プロパティ、インデクサ、演算子、イベント、および入れ子にされた型を含めることもできます。ただし、このようなメンバが複数必要な場合は、代わりに、型をクラスにすることを検討してください。

構造体はインターフェイスを実装できますが、別の構造体を継承できません。このため、構造体のメンバを **protected** と宣言することはできません。

詳細については、「構造体 (C# プログラミング ガイド)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 11 構造体

参照

関連項目

C# のキーワード
組み込み型の一覧表 (C# リファレンス)
値型 (C# リファレンス)
class (C# リファレンス)
インターフェイス (C# リファレンス)
オブジェクト、クラス、および構造体 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス
既定値の一覧表 (C# リファレンス)
型 (C# リファレンス)

uint (C# リファレンス)

uint キーワードは、次の表に示されたサイズと範囲に従って値を格納する整数型を示します。

型	範囲	サイズ	.NET Framework 型
uint	0 ～ 4,294,967,295	符号なし 32 ビット整数	System.UInt32

リテラル

uint 型の変数の宣言と初期化の例を次に示します。

```
        uint myUint = 4294967290;
```

サフィックスがない整数リテラルの型は、int、**uint**、long、ulong のうち、その整数の値を表すことができる最も範囲の狭い型になります。この例では、**uint** です。

```
        uint uInt1 = 123;
```

また、サフィックス u または U を使用することもできます。次に例を示します。

```
        uint uInt2 = 123U;
```

サフィックス `U` または `u` を使用すると、リテラルの型は、リテラルの数値に応じて **uint** または **ulong** のいずれかに決まります。次に例を示します。

```
Console.WriteLine(44U.GetType());
Console.WriteLine(323442434344U.GetType());
```

このコードでは、2 番目のリテラルが **uint** 型で格納するには大きすぎるため、`System.UInt32`、`System.UInt64` の順に表示します。これはそれぞれ、**uint** および **ulong** の基になる型です。

変換

uint から long、ulong、float、double、または decimal への暗黙の型変換が組み込まれています。次に例を示します。

```
float myFloat = 4294967290;   // OK: implicit conversion to float
```

byte、ushort、または char から **uint** への暗黙の型変換が組み込まれています。その他の型の場合は、キャストを使用する必要があります。たとえば、次の代入ステートメントは、キャストを使用しない場合、コンパイル エラーになります。

```
long aLong = 22;
// Error -- no implicit conversion from long:
uint uInt1 = aLong;
// OK -- explicit conversion:
uint uInt2 = (uint)aLong;
```

また、浮動小数点型から **uint** への暗黙の型変換が行われないことに注意してください。たとえば、次のステートメントは、明示的なキャストを使用しない場合、コンパイラ エラーになります。

```
// Error -- no implicit conversion from double:
uint x = 3.0;
// OK -- explicit conversion:
uint y = (uint)3.0;
```

浮動小数点型と整数型の混在する算術式の詳細については、「float」と「double」を参照してください。

暗黙の数値変換規則の詳細については、「暗黙的な数値変換の一覧表 (C# リファレンス)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.5 整数型

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
UInt32 Structure

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

ulong (C# リファレンス)

ulong キーワードは、次の表に示されたサイズと範囲に従って値を格納する整数型を示します。

型	範囲	サイズ	.NET Framework 型
ulong	0 ～ 18,446,744,073,709,551,615	符号なし 64 ビット整数	System.UInt64

リテラル

ulong 変数の宣言と初期化の例を次に示します。

```
ulong uLong = 9223372036854775808;
```

サフィックスがない整数リテラルの型は、int、uint、long、**ulong** のうち、その整数の値を表すことができる最も範囲の狭い型になります。上記の例では、**ulong** 型です。

サフィックスを使用してリテラルの型を指定する場合は、次の規則に従います。

- L または l を使用した場合、リテラル整数の型は、サイズに応じて long または **ulong** のいずれかに決まります。

  メモ：

  小文字の "l" をサフィックスとして使用できます。ただし、文字 "l" は数字の "1" と混同しやすいので、コンパイラから警告が出されます。明確にするには "L" を使用します。

- U または u を使用した場合、リテラル整数の型は、サイズに応じて uint または **ulong** のいずれかに決まります。
- UL、ul、Ul、uL、LU、lu、Lu、lU を使用した場合、リテラル整数の型は **ulong** です。

  たとえば、次の 3 つのステートメントの出力は、エイリアス **ulong** に対応したシステム型 **UInt64** になります。

  ```
  Console.WriteLine(9223372036854775808L.GetType());
  Console.WriteLine(123UL.GetType());
  Console.WriteLine((123UL + 456).GetType());
  ```

サフィックスは、オーバーロードされたメソッドの呼び出しでよく使われます。たとえば、**ulong** パラメータと int パラメータを使用したオーバーロードされたメソッドがあるとします。

```
public static void SampleMethod(int i) {}
public static void SampleMethod(ulong l) {}
```

ulong パラメータでサフィックスを使用すると、正しい型が呼び出されます。次に例を示します。

```
SampleMethod(5);    // Calling the method with the int parameter
SampleMethod(5UL);  // Calling the method with the ulong parameter
```

変換

ulong から float、double、または decimal への暗黙の型変換が組み込まれています。

ulong から整数型への暗黙の型変換はありません。たとえば、次のステートメントは、明示的なキャストを使用しない場合、コンパイル エラーになります。

```
long long1 = 8UL;   // Error: no implicit conversion from ulong
```

byte、ushort、uint、または char から **ulong** への暗黙の型変換が組み込まれています。

浮動小数点型から **ulong** への暗黙の型変換はありません。たとえば、次のステートメントは、明示的なキャストを使用しない場合、コンパイラ

エラーになります。

```
// Error -- no implicit conversion from double:
ulong x = 3.0;
// OK -- explicit conversion:
ulong y = (ulong)3.0;
```

浮動小数点型と整数型の混在する算術式の詳細については、「float」と「double」を参照してください。

暗黙の数値変換規則の詳細については、「暗黙的な数値変換の一覧表 (C# リファレンス)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.5 整数型

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
UInt64 Structure

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

ushort (C# リファレンス)

ushort キーワードは、次の表に示されたサイズと範囲に従って値を格納する整数データ型を示します。

型	範囲	サイズ	.NET Framework 型
ushort	0 ～ 65,535	符号なし 16 ビット整数	System.UInt16

リテラル

ushort 変数の宣言と初期化の例を次に示します。

```
        ushort myShort = 65535;
```

上のように宣言すると、整数リテラル `65535` は暗黙的に int から **ushort** に変換されます。整数リテラルが **ushort** の範囲を超えると、コンパイル エラーになります。

オーバーロードされたメソッドを呼び出す場合は、キャストを使用する必要があります。たとえば、**ushort** パラメータと int パラメータを使用したオーバーロードされたメソッドがあるとします。

```
public static void SampleMethod(int i) {}
public static void SampleMethod(ushort s) {}
```

ushort キャストを使用すると、正しい型が呼び出されます。次に例を示します。

```
// Calls the method with the int parameter:
SampleMethod(5);
// Calls the method with the ushort parameter:
SampleMethod((ushort)5);
```

変換

ushort から int、uint、long、ulong、float、double、decimal への暗黙の型変換が組み込まれています。

byte または char から **ushort** への暗黙の型変換が組み込まれています。その他の型の場合は、キャストを使用して明示的な変換を実行する必要があります。たとえば、2 つの **ushort** 変数 `x` と `y` があるとします。

```
        ushort x = 5, y = 12;
```

次の代入ステートメントは、代入演算子の右側にある算術式が既定で **int** に評価されるため、コンパイル エラーになります。

```
        ushort z = x + y;   // Error: conversion from int to ushort
```

このエラーを修正するには、キャストを使用します。

```
        ushort z = (ushort)(x + y);   // OK: explicit conversion
```

ただし、次のステートメントは使用できます。このステートメントでは、変換先の変数の記憶サイズは元のサイズ以上になります。

```
int m = x + y;
long n = x + y;
```

また、浮動小数点型から **ushort** への暗黙の型変換が行われないことに注意してください。たとえば、次のステートメントは、明示的なキャストを使用しない場合、コンパイラ エラーになります。

```
// Error -- no implicit conversion from double:
ushort x = 3.0;
// OK -- explicit conversion:
ushort y = (ushort)3.0;
```

浮動小数点型と整数型の混在する算術式の詳細については、「float」と「double」を参照してください。

暗黙の数値変換規則の詳細については、「暗黙的な数値変換の一覧表 (C# リファレンス)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.3 型と変数
- 4.1.5 整数型

参照

関連項目

C# のキーワード
整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)
UInt16 Structure

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

参照型 (C# リファレンス)

参照型の変数はオブジェクトと呼ばれ、実データへの参照を格納します。ここでは、参照型の宣言に使用するキーワードについて説明します。キーワードは、次のとおりです。

- class
- interface
- delegate

次の組み込みの参照型も紹介します。

- object
- string

参照
関連項目
C# のキーワード
値型 (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス
型 (C# リファレンス)

class (C# リファレンス)

クラスは、次の例に示すように、**class** キーワードを使用して宣言します。

```
      class TestClass
{
    // Methods, properties, fields, events, delegates
    // and nested classes go here.
}
```

解説

C++ とは異なり、C# で許可される継承は 1 つだけです。つまり、1 つの基本クラスの実装だけを継承できます。ただし、クラスは複数のインターフェイスを実装できます。クラスの継承とインターフェイスの実装の例を次の表に示します。

継承	例
なし。	`class ClassA { }`
1 つ	`class DerivedClass: BaseClass { }`
なし。2 つのインターフェイスを実装。	`class ImplClass: IFace1, IFace2 { }`
1 つ。1 つのインターフェイスを実装。	`class ImplDerivedClass: BaseClass, IFace1 { }`

アクセス レベルの protected と private は、入れ子になったクラスだけで使用できます。

型パラメータを持つジェネリック クラスも宣言できます。詳細については、「ジェネリック クラス (C# プログラミング ガイド)」を参照してください。

クラスは、次のメンバを宣言できます。

- コンストラクタ
- デストラクタ
- 定数
- フィールド
- メソッド
- プロパティ
- インデクサ
- 演算子
- イベント
- デリゲート
- クラス
- インターフェイス
- 構造体

使用例

ここでは、クラスのフィールド、コンストラクタ、メソッドの宣言例を示します。オブジェクト インスタンスの作成、インスタンス データの出力の例も示します。この例では、2 つのクラスを宣言します。`Kid` クラスには、2 つのプライベート フィールド (`name` および `age`) と、2 つのパブリック メソッドがあります。2 番目のクラスである `MainClass` は、`Main` の格納に使用されます。

```
// keyword_class.cs
// class example
```

```
using System;
class Kid
{
    private int age;
    private string name;

    // Default constructor:
    public Kid()
    {
        name = "N/A";
    }

    // Constructor:
    public Kid(string name, int age)
    {
        this.name = name;
        this.age = age;
    }

    // Printing method:
    public void PrintKid()
    {
        Console.WriteLine("{0}, {1} years old.", name, age);
    }
}

class MainClass
{
    static void Main()
    {
        // Create objects
        // Objects must be created using the new operator:
        Kid kid1 = new Kid("Craig", 11);
        Kid kid2 = new Kid("Sally", 10);

        // Create an object using the default constructor:
        Kid kid3 = new Kid();

        // Display results:
        Console.Write("Kid #1: ");
        kid1.PrintKid();
        Console.Write("Kid #2: ");
        kid2.PrintKid();
        Console.Write("Kid #3: ");
        kid3.PrintKid();
    }
}
```

出力

```
Kid #1: Craig, 11 years old.
Kid #2: Sally, 10 years old.
Kid #3: N/A, 0 years old.
```

説明

上記の例で、プライベート フィールド (`name` および `age`) にアクセスできるのは、`Kid` クラスのパブリック メソッドだけであることに注意してください。たとえば、次のステートメントを使用して `Main` メソッドから子供の名前を出力できません。

```
Console.Write(kid1.name);   // Error
```

`Main` から `Kid` のプライベート メンバへのアクセスは、`Main` がそのクラスのメンバである場合にのみ可能です。

アクセス修飾子を指定せずにクラス内で宣言された型は、既定では **private** になるため、この例のデータ メンバは、キーワードが削除されても、**private** のままです。

また、既定のコンストラクタを使って作成したオブジェクト (`kid3`) は、既定では、年齢フィールドが 0 に初期化されることに注意してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6 クラスとオブジェクト
- 3.4.4 クラスのメンバ
- 4.2.1 クラス型
- 10 クラス

参照

関連項目

C# のキーワード

参照型 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

delegate (C# リファレンス)

delegate 型の宣言は、次の形式をとります。

```
public delegate void TestDelegate(string message);
```

delegate キーワードは、指定されたメソッドまたは匿名メソッドをカプセル化するための参照型を宣言する場合に使用します。デリゲートは C++ の関数ポインタに類似していますが、タイプ セーフであり、安全です。デリゲートの適用については、「デリゲート (C# プログラミング ガイド)」および「汎用デリゲート (C# プログラミング ガイド)」を参照してください。

解説

デリゲートは、イベントの基礎になります。

デリゲートをインスタンス化するには、デリゲートに指定されたメソッドまたは匿名メソッドを関連付けます。詳細については、「名前付きメソッド (C# プログラミング ガイド)」および「匿名メソッド (C# プログラミング ガイド)」を参照してください。

名前付きメソッドで使用する場合、許容されるシグネチャを持つメソッドでデリゲートをインスタンス化する必要があります。メソッド シグネチャで許容される範囲の詳細については、「デリゲートの共変性と反変性 (C# プログラミング ガイド)」を参照してください。匿名メソッドで使用する場合、デリゲートおよびデリゲートと関連付けるコードを共に宣言します。デリゲートをインスタンス化するこの 2 つの方法について説明します。

```
using System;
// Declare delegate -- defines required signature:
delegate void SampleDelegate(string message);

class MainClass
{
    // Regular method that matches signature:
    static void SampleDelegateMethod(string message)
    {
        Console.WriteLine(message);
    }

    static void Main()
    {
        // Instantiate delegate with named method:
        SampleDelegate d1 = SampleDelegateMethod;
        // Instantiate delegate with anonymous method:
        SampleDelegate d2 = delegate(string message)
        {
            Console.WriteLine(message);
        };

        // Invoke delegate d1:
        d1("Hello");
        // Invoke delegate d2:
        d2(" World");
    }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.11 デリゲート
- 15 デリゲート

参照

関連項目

C# のキーワード
参照型 (C# リファレンス)
匿名メソッド (C# プログラミング ガイド)

概念

C# プログラミング ガイド

デリゲート (C# プログラミング ガイド)

イベント (C# プログラミング ガイド)

名前付きメソッド (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

インターフェイス (C# リファレンス)

インターフェイスには、メソッド、デリゲート、またはイベントのシグネチャのみが含まれています。メソッドの実装は、次の例に示すように、インターフェイスを実装するクラス内で行われます。

```
      interface ISampleInterface
{
    void SampleMethod();
}

class ImplementationClass : ISampleInterface
{
    // Explicit interface member implementation:
    void ISampleInterface.SampleMethod()
    {
        // Method implementation.
    }

    static void Main()
    {
        // Declare an interface instance.
        ISampleInterface obj = new ImplementationClass();

        // Call the member.
        obj.SampleMethod();
    }
}
```

解説

インターフェイスは、名前空間またはクラスのメンバであり、次のメンバのシグネチャを含むことができます。

- メソッド
- プロパティ
- インデクサ
- イベント

インターフェイスは、1 つ以上の基本インターフェイスから継承できます。

基本型のリストに基本クラスとインターフェイスが含まれる場合は、基本クラスがリストの最初に表示されます。

インターフェイスを実装するクラスは、そのインターフェイスのメンバを明示的に実装できます。明示的に実装されているメンバには、クラス インスタンスではアクセスできません。インターフェイスのインスタンスを使用した場合にのみ、アクセスできます。次に例を示します。

インターフェイスの明示的な実装の詳細とコード例については、「明示的なインターフェイスの実装 (C# プログラミング ガイド)」を参照してください。

使用例

ここでは、インターフェイスの実装例を示します。この例では、`IPoint` インターフェイスに、フィールド値の設定と取得を行うプロパティの宣言が含まれています。`Point` クラスには、プロパティの実装が含まれています。

```
// keyword_interface_2.cs
// Interface implementation
using System;
interface IPoint
{
   // Property signatures:
   int x
   {
      get;
      set;
```

```
    }

    int y
    {
       get;
       set;
    }
 }

 class Point : IPoint
 {
    // Fields:
    private int _x;
    private int _y;

    // Constructor:
    public Point(int x, int y)
    {
       _x = x;
       _y = y;
    }

    // Property implementation:
    public int x
    {
       get
       {
          return _x;
       }

       set
       {
          _x = value;
       }
    }

    public int y
    {
       get
       {
          return _y;
       }
       set
       {
          _y = value;
       }
    }
 }

 class MainClass
 {
    static void PrintPoint(IPoint p)
    {
       Console.WriteLine("x={0}, y={1}", p.x, p.y);
    }

    static void Main()
    {
       Point p = new Point(2, 3);
       Console.Write("My Point: ");
       PrintPoint(p);
    }
 }
```

出力

```
My Point: x=2, y=3
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.9 インターフェイス
- 3.4.5 インターフェイスのメンバ
- 4.2.4 インターフェイス型
- 10.1.2.2 インターフェイスの実装
- 11.2 構造体インターフェイス
- 13 インターフェイス

参照
関連項目
C# のキーワード
参照型 (C# リファレンス)
プロパティの使用 (C# プログラミング ガイド)
インデクサの使用 (C# プログラミング ガイド)
class (C# リファレンス)
struct (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

object (C# リファレンス)

object 型は、.NET Framework の Object のエイリアスです。C# の統一型システムでは、定義済みの型やユーザー定義の型、参照型や値型など、すべての型が、**Object** から直接的または間接的に継承されます。任意の型の値を **object** 型の変数に代入できます。値型の変数をオブジェクトに変換することを "ボックス化" と言います。型オブジェクトの変数を値型に変換することを "ボックス化解除" と言います。詳細については、「ボックス化とボックス化解除」を参照してください。

使用例

次の例では、**object** 型の変数が任意のデータ型の値を受け取る方法、および **object** 型の変数が .NET Framework からの **Object** のメソッドを使用する方法を示しています。

```
// keyword_object.cs
using System;
class SampleClass
{
   public int i = 10;
}

class MainClass
{
   static void Main()
   {
      object a;
      a = 1;   // an example of boxing
      Console.WriteLine(a);
      Console.WriteLine(a.GetType());
      Console.WriteLine(a.ToString());

      a = new SampleClass();
      SampleClass classRef;
      classRef = (SampleClass)a;
      Console.WriteLine(classRef.i);
   }
}
```

出力

```
1
System.Int32
1
10
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1 概要
- 4.2.2 object 型

参照
関連項目
C# のキーワード
参照型 (C# リファレンス)
値型 (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

string (C# リファレンス)

string 型は、Unicode 文字のシーケンスを表します。文字数がゼロの場合もあります。**string** は、.NET Framework の String のエイリアスです。

string は参照型ですが、等値演算子 (**==** および **!=**) は、**string** オブジェクトの参照ではなく、値を比較するように定義されます。値を比較することで、文字列が等しいかを直感的にテストできます。次に例を示します。

```
      string a = "hello";
string b = "h";
// Append to contents of 'b'
b += "ello";
Console.WriteLine(a == b);
Console.WriteLine((object)a == (object)b);
```

文字列の内容が等しいので、"True"、"False" の順に表示されますが、`a` および `b` は同じ文字列インスタンスを参照しません。

+ 演算子は、文字列を連結します。

```
      string a = "good " + "morning";
```

これは、"good morning" を含む文字列オブジェクトを作成します。

文字列は変更不可です。文字列オブジェクトの作成後、そのコンテンツを変更することはできません。構文では変更可能に見えても、変更不可です。たとえば、このコードを作成すると、コンパイラは新しい文字列を格納する新しいシーケンス オブジェクトを生成し、変数 b には引き続き "h" が格納されます。

```
      string b = "h";
b += "ello";
```

[] 演算子は、**string** の各文字へのアクセスに使用できます。

```
      string str = "test";
char x = str[2];  // x = 's';
```

リテラル文字列は **string** 型であり、二重引用符で囲む形式と、@ と二重引用符で囲む形式の 2 とおりがあります。二重引用符で囲む場合は、リテラル文字列の前後に二重引用符 (") を付けます。

```
"good morning"  // a string literal
```

リテラル文字列には、エスケープ シーケンスを含む任意の文字リテラルを含めることができます。

```
      string a = "\\\u0066\n";
```

この文字列には、円記号 (\)、文字 f、および改行文字が含まれています。

メモ :

エスケープ コード \u*dddd* (*dddd* は 4 桁の数字) は、Unicode 文字 U + *dddd* を表します。8 桁の Unicode エスケープ コード \U*dddd*\U*dddd* も認識できます。

@ と二重引用符で囲む場合は、先頭に @ を付け、さらに、リテラル文字列の前後に二重引用符を付けます。次に例を示します。

```
@"good morning"  // a string literal
```

@ と二重引用符を使用した場合の利点は、エスケープ シーケンスが処理されないため、たとえば、完全修飾ファイル名が書きやすくなることです。

```
@"c:\Docs\Source\a.txt"  // rather than "c:\\Docs\\Source\\a.txt"
```

@ と二重引用符で囲まれた文字列中に二重引用符を含めるには、二重引用符を二重にします。

```
@""Ahoy!"" cried the captain." // "Ahoy!" cried the captain.
```

また、参照 (/reference (メタデータのインポート)) 識別子が C# キーワードである場合も、@ 記号を使用します。

使用例

```
// keyword_string.cs
using System;
class TestClass
{
   static void Main()
   {
      string a = "\u0068ello ";
      string b = "world";
      Console.WriteLine( a + b );
      Console.WriteLine( a + b == "hello world" );
   }
}
```

出力

```
hello world
True
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 2.4.2 識別子
- 2.4.4.5 リテラル文字列
- 4.2.3 string 型
- 7.9.7 文字列等値演算子

参照

関連項目

C# のキーワード
参照型 (C# リファレンス)
値型 (C# リファレンス)

概念

C# プログラミング ガイド
C# プログラミング ガイド

その他の技術情報

C# リファレンス
数値結果テーブルの書式設定 (C# リファレンス)

void (C# リファレンス)

メソッドの戻り値の型として **void** を使用する場合は、メソッドが値を戻さないことを示します。

void は、メソッドのパラメータ リストでは指定できません。パラメータや戻り値を持たないメソッドの宣言は、次のとおりです。

```
void SampleMethod();
```

また、**void** は、unsafe コンテキストで不明な型へのポインタを宣言するときにも使用されます。詳細については、「ポインタ型 (C# プログラミング ガイド)」を参照してください。

void は、.NET Framework の System.Void 型のエイリアスです。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.5 メソッド

参照
関連項目
C# のキーワード
参照型 (C# リファレンス)
値型 (C# リファレンス)
メソッド (C# プログラミング ガイド)
概念
C# プログラミング ガイド
unsafe コードとポインタ (C# プログラミング ガイド)
その他の技術情報
C# リファレンス

型のリファレンス表 (C# リファレンス)

次のリファレンス表に、C# の型の要約が示されています。

組み込み型の一覧表

整数型の一覧表

浮動小数点型の一覧表

既定値の一覧表

値型

暗黙の数値変換

明示的な数値変換の一覧表

数値型の出力書式の詳細については、「数値結果の書式指定の一覧表」を参照してください。

参照

関連項目

参照型 (C# リファレンス)
値型 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

組み込み型の一覧表 (C# リファレンス)

C# の組み込み型のキーワードは、次のとおりです。これらは、System 名前空間で定義されている型のエイリアスです。

C# 型	.NET Framework 型
bool	**System.Boolean**
byte	**System.Byte**
sbyte	**System.SByte**
char	**System.Char**
decimal	**System.Decimal**
double	**System.Double**
float	**System.Single**
int	**System.Int32**
uint	**System.UInt32**
long	**System.Int64**
ulong	**System.UInt64**
object	**System.Object**
short	**System.Int16**
ushort	**System.UInt16**
string	**System.String**

解説

表内の **object** と **string** を除くすべての型は、単純型と呼ばれます。

C# 型のキーワードとエイリアスは、相互に交換できます。たとえば、整数の変数を宣言する場合は、次のいずれかの宣言を使用できます。

```
int x = 123;
System.Int32 x = 123;
```

C# 型の実際の型を表示するには、`GetType()` システム メソッドを使用します。たとえば、`myVariable` の型を表すシステム エイリアスを表示するには、次のステートメントを使用します。

```
Console.WriteLine(myVariable.GetType());
```

また、typeof 演算子も使用できます。

参照

関連項目

C# のキーワード

値型 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

既定値の一覧表 (C# リファレンス)

数値結果テーブルの書式設定 (C# リファレンス)

型のリファレンス表 (C# リファレンス)

整数型の一覧表 (C# リファレンス)

単純型のサブセットである整数型のサイズと範囲は、次のとおりです。

型	範囲	サイズ
sbyte	-128 ～ 127	符号付き 8 ビット整数
byte	0 ～ 255	符号なし 8 ビット整数
char	U+0000 ～ U+ffff	Unicode 16 ビット文字
short	-32,768 ～ 32,767	符号付き 16 ビット整数
ushort	0 ～ 65,535	符号なし 16 ビット整数
int	-2,147,483,648 ～ 2,147,483,647	符号付き 32 ビット整数
uint	0 ～ 4,294,967,295	符号なし 32 ビット整数
long	-9,223,372,036,854,775,808 ～ 9,223,372,036,854,775,807	符号付き 64 ビット整数
ulong	0 ～ 18,446,744,073,709,551,615	符号なし 64 ビット整数

整数リテラルの値が **ulong** の範囲を超えると、コンパイル エラーになります。

参照

関連項目

C# のキーワード

組み込み型の一覧表 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

浮動小数点型の一覧表 (C# リファレンス)

既定値の一覧表 (C# リファレンス)

数値結果テーブルの書式設定 (C# リファレンス)

型のリファレンス表 (C# リファレンス)

浮動小数点型の一覧表 (C# リファレンス)

浮動小数点型の有効桁数とおおよその範囲は、次のとおりです。

型	おおよその範囲	有効桁数
float	±1.5e − 45 〜 ±3.4e38	7 桁
double	±5.0e − 324 〜 ±1.7e308	15 〜 16 桁

参照

関連項目

組み込み型の一覧表 (C# リファレンス)
整数型の一覧表 (C# リファレンス)
decimal (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス
既定値の一覧表 (C# リファレンス)
数値結果テーブルの書式設定 (C# リファレンス)
型のリファレンス表 (C# リファレンス)

既定値の一覧表 (C# リファレンス)

既定のコンストラクタから返される値型の既定値は、次のとおりです。既定のコンストラクタを呼び出すには、**new** 演算子を次のように使用します。

```
int myInt = new int();
```

次のステートメントは、上のステートメントと同じ結果になります。

```
int myInt = 0;
```

C# では、初期化する前の変数は使用できないことに注意してください。

値型	既定値
bool	**false**
byte	0
char	'\0'
decimal	0.0M
double	0.0D
enum	式 (E)0 によって算出された値。E は、列挙の識別子です。
float	0.0F
int	0
long	0L
sbyte	0
short	0
struct	すべての値型フィールドを既定値に設定し、すべての参照型フィールドを **null** に設定して算出された値。
uint	0
ulong	0
ushort	0

参照

関連項目

値型 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス
型のリファレンス表 (C# リファレンス)

値型の一覧表 (C# リファレンス)

次の表に、C# の値型をカテゴリ別に示します。

値型	カテゴリ
bool	ブール値
byte	整数値 (符号なし)
char	整数値 (符号なし)
decimal	10 進数値
double	浮動小数点数値
enum	列挙値
float	浮動小数点数値
int	整数値 (符号付き)
long	整数値 (符号付き)
sbyte	整数値 (符号付き)
short	整数値 (符号付き)
struct	ユーザー定義の構造体
uint	整数値 (符号なし)
ulong	整数値 (符号なし)
ushort	整数値 (符号なし)

参照

関連項目

値型 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

数値結果テーブルの書式設定 (C# リファレンス)

型のリファレンス表 (C# リファレンス)

暗黙的な数値変換の一覧表 (C# リファレンス)

組み込まれた暗黙の数値変換を次に示します。暗黙の変換は、メソッドの呼び出しや代入ステートメントなど、多くの状況で発生することがあります。

変換前	変換後
sbyte	**short**、**int**、**long**、**float**、**double**、または **decimal**
byte	**short**、**ushort**、**int**、**uint**、**long**、**ulong**、**float**、**double**、または **decimal**
short	**int**、**long**、**float**、**double**、または **decimal**
ushort	**int**、**uint**、**long**、**ulong**、**float**、**double**、または **decimal**
int	**long**、**float**、**double**、または **decimal**
uint	**long**、**ulong**、**float**、**double**、または **decimal**
long	**float**、**double**、または **decimal**
char	**ushort**、**int**、**uint**、**long**、**ulong**、**float**、**double**、または **decimal**
float	**double**
ulong	**float**、**double**、または **decimal**

解説

- **int**、**uint**、または **long** から **float**への変換、および **long** から **double** への変換では、有効桁数が桁落ちすることがありますが、絶対値は損なわれません。
- **char** 型への暗黙の型変換はありません。
- 浮動小数点型と **decimal** 型の間には、暗黙の型変換はありません。
- **int** 型の定数式は、定数式の値が変換後の型の範囲内である場合、**sbyte**、**byte**、**short**、**ushort**、**uint**、または **ulong** に変換できます。

C# 言語仕様

詳細については、C# 言語の仕様 (C# 言語仕様) を参照してください。

- 6.1 暗黙の変換
- 7.15 定数式

参照

関連項目

整数型の一覧表 (C# リファレンス)
組み込み型の一覧表 (C# リファレンス)
明示的な数値変換の一覧表 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

明示的な数値変換の一覧表 (C# リファレンス)

暗黙の型変換が行われない場合に、明示的な数値の変換を行います。キャスト式を使用して任意の数値型を他の数値型に変換します。これらの変換を次に示します。

変換前	変換後
sbyte	**byte**、**ushort**、**uint**、**ulong**、または **char**
byte	**Sbyte** または **char**
short	**sbyte**、**byte**、**ushort**、**uint**、**ulong**、または **char**
ushort	**sbyte**、**byte**、**short**、または **char**
int	**sbyte**、**byte**、**short**、**ushort**、**uint**、**ulong**、、または **char**
uint	**sbyte**、**byte**、**short**、**ushort**、**int**、または **char**
long	**sbyte**、**byte**、**short**、**ushort**、**int**、**uint**、**ulong**、または **char**
ulong	**sbyte**、**byte**、**short**、**ushort**、**int**、**uint**、**long**、または **char**
char	**sbyte**、**byte**、または **short**
float	**sbyte**、**byte**、**short**、**ushort**、**int**、**uint**、**long**、**ulong**、**char**、、または **decimal**
double	**sbyte**、**byte**、**short**、**ushort**、**int**、**uint**、**long**、**ulong**、**char**、**float**、、または **decimal**
decimal	**sbyte**、**byte**、**short**、**ushort**、**int**、**uint**、**long**、**ulong**、**char**、**float**、または **double**

解説

- 明示的な数値変換によって、有効桁数が桁落ちしたり、例外がスローされる場合があります。
- **decimal** 値を整数型に変換すると、値は一番近い整数値に丸められます。結果の整数値が変換先の型の範囲を超えた場合は、OverflowException がスローされます。
- **double** 値または **float** 値を整数型に変換すると、値が切り捨てられます。整数値が、変換後の値の範囲を超えた場合は、オーバーフロー チェック コンテキストに従います。checked コンテキストでは、**OverflowException** がスローされます。これに対して、unchecked コンテキストでは、変換後の型の未指定値に変換されます。
- **double** から **float** への変換では、**double** 値は最も近い **float** 値に丸められます。**double** 値が、変換後の型の範囲外であるため格納できない場合は、0 または無限大になります。
- **float** または **double** から **decimal** への変換では、変換前の値が **decimal** 表記に変換され、必要に応じて小数第 28 位までの近似値に丸められます。変換元の値に応じて、次のいずれかが生じる場合があります。
    - 変換元の値が小さすぎて **decimal** で表すことができない場合、結果は 0 になります。
    - 変換元の値が非数 (NaN)、無限大、または大きすぎて **decimal** で表すことができない場合は、**OverflowException** がスローされます。
- **decimal** から **float** または **double** への変換では、**decimal** 値は最も近い **double** 値または **float** 値に丸められます。

明示的な変換の詳細については、『C# 言語仕様』の「6.2 明示的な変換」を参照してください。仕様にアクセスする方法の詳細については、「C# 言語仕様」を参照してください。

参照

関連項目

整数型の一覧表 (C# リファレンス)

組み込み型の一覧表 (C# リファレンス)
暗黙的な数値変換の一覧表 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

数値結果テーブルの書式設定 (C# リファレンス)

数値結果の書式を指定するには、**String.Format** メソッドを使用するか、**String.Format** を呼び出す **Console.Write** メソッドを使用します。書式を指定するには、書式指定文字列を使用します。サポートされる標準の書式指定文字列を次の表に示します。書式指定文字列は *Axx* という形式になります。ここで、*A* は書式指定子、*xx* は精度指定子です。書式指定子は、数値に適用する書式の種類を制御し、有効桁数指定子は、書式付き出力の有効桁数または小数点以下の桁数を制御します。

標準およびカスタムの書式指定文字列の詳細については、「書式設定の概要」を参照してください。**String.Format** メソッドの詳細については、「System.String.Format」を参照してください。

文字	説明	例	出力
C または c	通貨	Console.Write("{0:C}", 2.5); Console.Write("{0:C}", -2.5);	$2.50 ($2.50)
D または d	10 進数	Console.Write("{0:D5}", 25);	00025
E または e	指数	Console.Write("{0:E}", 250000);	2.500000E+005
F または f	固定小数点	Console.Write("{0:F2}", 25); Console.Write("{0:F0}", 25);	25.00 25
G または g	汎用	Console.Write("{0:G}", 2.5);	2.5
N または n	番号	Console.Write("{0:N}", 2500000);	2,500,000.00
X または x	16 進数	Console.Write("{0:X}", 250); Console.Write("{0:X}", 0xffff);	FA FFFF

参照

関連項目

string (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

型のリファレンス表 (C# リファレンス)

修飾子 (C# リファレンス)

修飾子は、型および型メンバの宣言を修飾するために使用されます。ここでは、C# の修飾子について説明します。

修飾子	目的
アクセス修飾子 • public • private • internal • protected	型および型メンバの宣言されたアクセシビリティを指定します。
abstract	クラスが、他のクラスの基本クラスになるためだけのものであることを示します。
const	フィールドまたはローカル変数の値が変更されないことを指定します。
event	イベントを宣言します。
extern	メソッドが外部で実装されることを示します。
• new	基本クラス メンバから継承メンバを隠ぺいします。
override	基本クラスから継承された仮想メンバの新しい実装を提供します。
partial	同一アセンブリに部分クラスまたは部分構造体を定義します。
readonly	フィールドを宣言します。このフィールドは、宣言の一部として、または同じクラスのコンストラクタ内でだけ、値の代入ができます。
sealed	クラスの継承ができないことを指定します。
static	特定のオブジェクトではなく、型自体に所属するメンバを宣言します。
unsafe	unsafe コンテキストを宣言します。
virtual	メソッドまたはアクセサを宣言します。これらの実装は、派生クラスでオーバーライドするメンバによって変更できます。
volatile	オペレーティング システム、ハードウェア、現在実行中のスレッドなどによって、フィールドがプログラム中で変更される場合があることを示します。

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

アクセス修飾子 (C# リファレンス)

アクセス修飾子は、メンバまたは型の宣言されたアクセシビリティを指定するために使用されるキーワードです。ここでは、4 つのアクセス修飾子について説明します。

- public
- protected
- internal
- private

アクセス修飾子を使用して、以下の 5 つのアクセシビリティ レベルを指定できます。

public: アクセスの制限はありません。

protected: アクセスは、コンテナ クラス、またはコンテナ クラスから派生した型に制限されます。

Internal: アクセスは現在のアセンブリに制限されます。

protected internal: アクセスは、現在のアセンブリ、またはコンテナ クラスから派生した型に制限されます。

private: アクセスはコンテナ型に制限されます。

ここでは、以下についても説明します。

- アクセシビリティ レベル (C# リファレンス) : 4 つのアクセス修飾子を使用して 5 つのアクセシビリティ レベルを宣言します。
- アクセシビリティ ドメイン (C# リファレンス) : プログラムのセクション内で、メンバを参照できる位置を指定します。
- アクセシビリティ レベルの使用に関する制限事項 (C# リファレンス) : 宣言されたアクセシビリティ レベルの使用に関する制限事項をまとめたものです。

参照
関連項目
C# のキーワード
修飾子 (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

アクセシビリティ レベル (C# リファレンス)

アクセス修飾子 public、protected、internal、または private を使用して、メンバに対する宣言済みのアクセシビリティとして以下のいずれか 1 つを指定できます。

宣言されたアクセシビリティ	説明
public	アクセスの制限はありません。
protected	アクセスは、コンテナ クラス、またはコンテナ クラスから派生した型に制限されます。
internal	アクセスは現在のアセンブリに制限されます。
protected internal	アクセスは、現在のアセンブリ、またはコンテナ クラスから派生した型に制限されます。
private	アクセスはコンテナ型に制限されます。

1 つのメンバまたは型に対してアクセス修飾子を 1 つだけ指定できますが、**protected internal** の組み合わせは例外です。

アクセス修飾子は、名前空間では使用できません。名前空間には、アクセス制限はありません。

メンバの宣言が行われるコンテキストに応じて、特定の宣言されたアクセシビリティだけが許可されます。メンバ宣言にアクセス修飾子の指定がない場合には、既定のアクセシビリティが使用されます。

他の型の中の入れ子になっていないトップ レベルの型は、**internal** または **public** のアクセシビリティだけを持つことができます。このような型の既定のアクセシビリティは **internal** です。

他の型のメンバである入れ子にされた型は、次の表に示すように、宣言されたアクセシビリティを持つことができます。

メンバとして属する型	既定のメンバ アクセシビリティ	メンバの許可される宣言されたアクセシビリティ
enum	**public**	なし
class	**private**	**public** **protected** **internal** **private** **protected internal**
interface	**public**	なし
struct	**private**	**public** **internal** **private**

入れ子にされた型のアクセシビリティは、アクセシビリティ ドメインに依存します。アクセシビリティ ドメインは、メンバの宣言されたアクセシビリティと、すぐ上位のコンテナ型のアクセシビリティ ドメインによって決定されます。ただし、入れ子にされた型のアクセシビリティ ドメインが、その型を含んでいる型のアクセシビリティ ドメインを上回ることはできません。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.5.1 宣言されたアクセシビリティ
- 3.5.3 インスタンス メンバへのプロテクト アクセス
- 3.5.4 アクセシビリティの制約

- 10.2.3 アクセス修飾子
- 10.2.6.2 宣言されたアクセシビリティ

参照

関連項目

C# のキーワード
アクセス修飾子 (C# リファレンス)
アクセシビリティ ドメイン (C# リファレンス)
アクセシビリティ レベルの使用に関する制限事項 (C# リファレンス)
アクセス修飾子 (C# プログラミング ガイド)
public (C# リファレンス)
private (C# リファレンス)
protected (C# リファレンス)
internal (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

アクセシビリティ ドメイン (C# リファレンス)

メンバのアクセシビリティ ドメインは、プログラムのセクション内で、メンバを参照できる場所を指定します。メンバが他の型の入れ子になっている場合、そのメンバのアクセシビリティ ドメインは、メンバのアクセシビリティ レベルと、それを含む型のアクセシビリティ ドメインによって決定されます。

トップ レベルの型のアクセシビリティ ドメインは、少なくとも、トップ レベルの型が宣言されているプロジェクトのプログラム テキストです。つまり、プロジェクトのソース ファイル全体です。入れ子にされた型のアクセシビリティ ドメインは、少なくとも、入れ子にされた型が宣言されている型のプログラム テキストです。つまり、入れ子にされたすべての型を含む、型の本体です。入れ子にされた型のアクセシビリティ ドメインは、含んでいる側の型のアクセシビリティ ドメインの外には決して出ません。これらの概念を次の例で説明します。

使用例

この例には、トップ レベルの型 `T1` と、2 つの入れ子にされたクラス `M1` および `M2` があります。クラスには、異なる種類の宣言されたアクセシビリティを持つフィールドがあります。`Main` メソッドでは、各ステートメントに続くコメントが、各メンバのアクセシビリティ ドメインを示しています。アクセス不可のメンバを参照しようとするステートメントがコメントになっている点に注意してください。アクセス不可のメンバを参照したために発生したコンパイラ エラーを見るには、コメントを一度に 1 つずつ解除してください。

```
// cs_Accessibility_Domain.cs
using System;
namespace AccessibilityDomainNamespace
{
    public class T1
    {
        public static int publicInt;
        internal static int internalInt;
        private static int privateInt = 0; // CS0414

        public class M1
        {
            public static int publicInt;
            internal static int internalInt;
            private static int privateInt = 0; // CS0414
        }

        private class M2
        {
            public static int publicInt = 0;
            internal static int internalInt = 0;
            private static int privateInt = 0; // CS0414
        }
    }

    class MainClass
    {
        static void Main()
        {
            // Access is unlimited:
            T1.publicInt = 1;
            // Accessible only in current assembly:
            T1.internalInt = 2;
            // Error: inaccessible outside T1:
            //    T1.myPrivateInt = 3;

            // Access is unlimited:
            T1.M1.publicInt = 1;
            // Accessible only in current assembly:
            T1.M1.internalInt = 2;
            // Error: inaccessible outside M1:
            //    T1.M1.myPrivateInt = 3;

            // Error: inaccessible outside T1:
            //    T1.M2.myPublicInt = 1;
            // Error: inaccessible outside T1:
            //    T1.M2.myInternalInt = 2;
            // Error: inaccessible outside M2:
```

```
            //    T1.M2.myPrivateInt = 3;
        }
    }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.5.1 宣言されたアクセシビリティ
- 3.5.2 アクセシビリティ ドメイン
- 3.5.4 アクセシビリティの制約
- 10.2.3 アクセス修飾子
- 10.2.6.2 宣言されたアクセシビリティ

参照

関連項目

C# のキーワード
アクセス修飾子 (C# リファレンス)
アクセシビリティ レベル (C# リファレンス)
アクセシビリティ レベルの使用に関する制限事項 (C# リファレンス)
アクセス修飾子 (C# プログラミング ガイド)
public (C# リファレンス)
private (C# リファレンス)
protected (C# リファレンス)
internal (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

アクセシビリティ レベルの使用に関する制限事項 (C# リファレンス)

型を宣言する場合、宣言する型が、少なくとも他のメンバまたは型と同程度にアクセス可能であるようにする必要があるかどうかを考慮する必要があります。たとえば、直接基本クラスは、少なくともその派生クラスと同程度にアクセス可能である必要があります。次の宣言はコンパイラ エラーになりますが、それは `BaseClass` クラスのアクセシビリティが `MyClass` のアクセシビリティよりも低いためです。

```
class BaseClass {...}
public class MyClass: BaseClass {...} // Error
```

宣言されたアクセシビリティ レベルを使用する時の制限を次の表にまとめて示します。

コンテキスト	解説
クラス	クラスの型の直接基本クラスは、少なくとも、クラスの型自体と同程度にアクセス可能である必要があります。
インターフェイス	インターフェイスの型の明示的な基本インターフェイスは、少なくとも、インターフェイスの型自体と同程度にアクセス可能である必要があります。
デリゲート	デリゲート型の戻り値の型およびパラメータの型は、少なくとも、デリゲート型自体と同程度にアクセス可能である必要があります。
定数	定数の型は、少なくとも定数自体と同程度にアクセス可能である必要があります。
フィールド	フィールドの型は、少なくともフィールド自体と同程度にアクセス可能である必要があります。
メソッド	メソッドの戻り値の型およびパラメータの型は、少なくとも、メソッド自体と同程度にアクセス可能である必要があります。
プロパティ	プロパティの型は、少なくともプロパティ自体と同程度にアクセス可能である必要があります。
イベント	イベントの型は、少なくともイベント自体と同程度にアクセス可能である必要があります。
インデクサ	インデクサの型およびパラメータの型は、少なくとも、インデクサ自体と同程度にアクセス可能である必要があります。
演算子	演算子の戻り値の型およびパラメータの型は、少なくとも、演算子自体と同程度にアクセス可能である必要があります。
コンストラクタ	コンストラクタのパラメータの型は、少なくともコンストラクタ自体と同程度にアクセス可能である必要があります。

使用例

各種の型の不適切な宣言の例を次に示します。各宣言へのコメントに、予期されるコンパイラ エラーを示します。

```
// Restrictions_on_Using_Accessibility_Levels.cs
// CS0052 expected as well as CS0053, CS0056, and CS0057
// To make the program work, change access level of both class B
// and MyPrivateMethod() to public.

using System;

// A delegate:
delegate int MyDelegate();

class B
{
    // A private method:
    static int MyPrivateMethod()
    {
        return 0;
    }
}
```

```
public class A
{
    // Error: The type B is less accessible than the field A.myField.
    public B myField = new B();

    // Error: The type B is less accessible
    // than the constant A.myConst.
    public readonly B myConst = new B();

    public B MyMethod()
    {
        // Error: The type B is less accessible
        // than the method A.MyMethod.
        return new B();
    }

    // Error: The type B is less accessible than the property A.MyProp
    public B MyProp
    {
        set
        {
        }
    }

    MyDelegate d = new MyDelegate(B.MyPrivateMethod);
    // Even when B is declared public, you still get the error:
    // "The parameter B.MyPrivateMethod is not accessible due to
    // protection level."

    public static B operator +(A m1, B m2)
    {
        // Error: The type B is less accessible
        // than the operator A.operator +(A,B)
        return new B();
    }

    static void Main()
    {
        Console.Write("Compiled successfully");
    }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.5.1 宣言されたアクセシビリティ
- 3.5.4 アクセシビリティの制約
- 10.2.3 アクセス修飾子
- 10.2.6.2 宣言されたアクセシビリティ
- 10.2.6.5 包含する型の private メンバおよび protected メンバへのアクセス

参照

関連項目

C# のキーワード
アクセス修飾子 (C# リファレンス)
アクセシビリティ ドメイン (C# リファレンス)
アクセシビリティ レベル (C# リファレンス)
アクセス修飾子 (C# プログラミング ガイド)
public (C# リファレンス)
private (C# リファレンス)
protected (C# リファレンス)
internal (C# リファレンス)

概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

internal (C# リファレンス)

internal キーワードは、型および型メンバのためのアクセス修飾子です。internal 型またはメンバは、この例に示すように同じアセンブリのファイル内でのみアクセスできます。

```
public class BaseClass
{
    // Only accessible within the same assembly
    internal static int x = 0;
}
```

internal と他のアクセス修飾子の比較については、「アクセシビリティ レベル」および「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

アセンブリの詳細については、「アセンブリとグローバル アセンブリ キャッシュ (C# プログラミング ガイド)」を参照してください。

内部アクセスは、コンポーネントのグループがアプリケーション コードの残りの部分には公開されないプライベートな手法で協調動作できるので、一般的にはコンポーネント ベースの開発に使用されます。たとえば、グラフィカル ユーザー インターフェイスを構築するためのフレームワークでは、内部アクセスでメンバを使用して協調動作する Control クラスと Form クラスを提供できます。これらのメンバは内部メンバなので、フレームワークを使用するコードに対しては公開されません。

内部アクセスのメンバまたは型を、メンバまたは型が定義されているアセンブリの外側で参照するとエラーになります。

メモ :

internal virtual メソッドは、C# ではオーバーライドできませんが、言語によってはオーバーライドできる場合もあります。たとえば、Microsoft Intermediate Language (MSIL) でオーバーライドされる場合があります。

使用例

この例には、`Assembly1.cs` および `Assembly2.cs` という 2 つのファイルがあります。1 つ目のファイルには、内部基本クラス `BaseClass` があります。2 つ目のファイルでは、`BaseClass` のインスタンス化が試行されますがエラーになります。

```
// Assembly1.cs
// compile with: /target:library
internal class BaseClass
{
   public static int intM = 0;
}
```

```
// Assembly1_a.cs
// compile with: /reference:Assembly1.dll
class TestAccess
{
   static void Main()
   {
      BaseClass myBase = new BaseClass();   // CS0122
   }
}
```

この例では、例 1 で使用したのと同じファイルを使用します。ただし、`BaseClass` のアクセシビリティ レベルを **public** に変更します。また、`IntM` メンバのアクセシビリティ レベルを **internal** に変更します。この場合、クラスをインスタンス化できますが、internal メンバにはアクセスできません。

```
// Assembly2.cs
// compile with: /target:library
public class BaseClass
{
   internal static int intM = 0;
}
```

```
// Assembly2_a.cs
```

```
// compile with: /reference:Assembly1.dll
public class TestAccess
{
   static void Main()
   {
      BaseClass myBase = new BaseClass();   // Ok.
      BaseClass.intM = 444;    // CS0117
   }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.5.1 宣言されたアクセシビリティ
- 3.5.4 アクセシビリティの制約
- 10.2.3 アクセス修飾子
- 10.2.6.2 宣言されたアクセシビリティ

参照

関連項目

C# のキーワード
アクセス修飾子 (C# リファレンス)
アクセシビリティ レベル (C# リファレンス)
修飾子 (C# リファレンス)
public (C# リファレンス)
private (C# リファレンス)
protected (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

private (C# リファレンス)

private キーワードは、メンバ アクセス修飾子です。プライベートなアクセスは、許容度が最も低いアクセス レベルです。この例に示すように、プライベートなメンバには、クラスの本体内か、メンバが宣言されている構造体の内部でだけアクセス可能です。

```
class Employee
{
    private int i;
    double d;   // private access by default
}
```

同じ本体にある入れ子になったクラスも、プライベートなメンバにアクセスできます。

プライベートなメンバへの参照を、クラスの外側や、メンバが宣言されているクラスの外側から行った場合は、コンパイル エラーになります。

private と他のアクセス修飾子の比較については、「アクセシビリティ レベル」および「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

使用例

この例では、`Employee` クラスに `name` および `salary` という 2 つのプライベート データ メンバが含まれています。これらのメンバは、プライベート メンバであり、メンバ メソッド以外からはアクセスできないため、`GetName` および `Salary` というパブリック メソッドが追加され、プライベート メンバへの制御されたアクセスを許可します。`name` メンバはパブリック メソッドを介してアクセスされ、`salary` メンバはパブリックな読み取り専用プロパティを介してアクセスされます。詳細については、「プロパティ (C# プログラミング ガイド)」を参照してください。

```
// private_keyword.cs
using System;
class Employee
{
    private string name = "FirstName, LastName";
    private double salary = 100.0;

    public string GetName()
    {
        return name;
    }

    public double Salary
    {
        get { return salary; }
    }
}

class MainClass
{
    static void Main()
    {
        Employee e = new Employee();

        // The data members are inaccessible (private), so
        // then can't be accessed like this:
        //    string n = e.name;
        //    double s = e.salary;

        // 'name' is indirectly accessed via method:
        string n = e.GetName();

        // 'salary' is indirectly accessed via property
        double s = e.Salary;
    }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.5.1 宣言されたアクセシビリティ
- 3.5.4 アクセシビリティの制約
- 10.2.3 アクセス修飾子
- 10.2.6.2 宣言されたアクセシビリティ
- 10.2.6.5 包含する型の private メンバおよび protected メンバへのアクセス

参照

関連項目

C# のキーワード
アクセス修飾子 (C# リファレンス)
アクセシビリティ レベル (C# リファレンス)
修飾子 (C# リファレンス)
public (C# リファレンス)
protected (C# リファレンス)
internal (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

protected (C# リファレンス)

protected キーワードは、メンバ アクセス修飾子です。protected メンバには、そのクラス内で派生クラスからアクセスできます。**protected** と他のアクセス修飾子の比較については、「アクセシビリティ レベル」を参照してください。

基本クラスのプロテクト メンバに派生クラスでアクセス可能なのは、派生したクラス型を使ってアクセスが行われる場合だけです。たとえば、次に示すコード セグメントを検討してみます。

```
// protected_keyword.cs
using System;
class A
{
    protected int x = 123;
}

class B : A
{
    static void Main()
    {
        A a = new A();
        B b = new B();

        // Error CS1540, because x can only be accessed by
        // classes derived from A.
        // a.x = 10;

        // OK, because this class derives from A.
        b.x = 10;
    }
}
```

ステートメント `a.x =10` は、エラーになります。これは、A が B から派生していないためです。

構造体のメンバは保護されませんが、これは構造体の継承ができないためです。

使用例

次の例では、`DerivedPoint` クラスが `Point` の派生クラスです。このため、基本クラスのプロテクト メンバに、派生クラスから直接アクセスできます。

```
// protected_keyword_2.cs
using System;
class Point
{
    protected int x;
    protected int y;
}

class DerivedPoint: Point
{
    static void Main()
    {
        DerivedPoint dp = new DerivedPoint();

        // Direct access to protected members:
        dp.x = 10;
        dp.y = 15;
        Console.WriteLine("x = {0}, y = {1}", dp.x, dp.y);
    }
}
```

出力

```
x = 10, y = 15
```

説明

x および y のアクセス レベルを private に変更すると、コンパイラがエラー メッセージを発行します。

```
'Point.y' is inaccessible due to its protection level.

'Point.x' is inaccessible due to its protection level.
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.5.1 宣言されたアクセシビリティ
- 3.5.3 インスタンス メンバへのプロテクト アクセス
- 3.5.4 アクセシビリティの制約
- 10.2.3 アクセス修飾子

参照

関連項目

C# のキーワード
アクセス修飾子 (C# リファレンス)
アクセシビリティ レベル (C# リファレンス)
修飾子 (C# リファレンス)
public (C# リファレンス)
private (C# リファレンス)
internal (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

public (C# リファレンス)

public キーワードは、型および型メンバのためのアクセス修飾子です。パブリックなアクセスは、許容度が最も高いアクセス レベルです。この例に示すように、パブリック メンバへのアクセスに関する制限はありません。

```
class SampleClass
{
    public int x; // No access restrictions.
}
```

詳細については、「アクセス修飾子 (C# プログラミング ガイド)」および「アクセシビリティ レベル (C# リファレンス)」を参照してください。

使用例

次の例では、`Point` および `MainClass` という 2 つのクラスが宣言されています。`Point` のパブリック メンバ x と y は、`MainClass` から直接アクセスされます。

```
// protected_public.cs
// Public access
using System;
class Point
{
    public int x;
    public int y;
}

class MainClass
{
    static void Main()
    {
        Point p = new Point();
        // Direct access to public members:
        p.x = 10;
        p.y = 15;
        Console.WriteLine("x = {0}, y = {1}", p.x, p.y);
    }
}
```

出力

```
x = 10, y = 15
```

public アクセス レベルを private または protected に変更すると、次のエラー メッセージが表示されることになります。

```
'Point.y' is inaccessible due to its protection level.
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.5.1 宣言されたアクセシビリティ
- 3.5.4 アクセシビリティの制約
- 10.2.3 アクセス修飾子
- 10.2.6.2 宣言されたアクセシビリティ

参照

関連項目

アクセス修飾子 (C# プログラミング ガイド)
C# のキーワード
アクセス修飾子 (C# リファレンス)
アクセシビリティ レベル (C# リファレンス)
修飾子 (C# リファレンス)

private (C# リファレンス)
protected (C# リファレンス)
internal (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

abstract (C# リファレンス)

abstract 修飾子は、クラス、メソッド、プロパティ、インデクサ、およびイベントで使用できます。クラスの宣言に **abstract** 修飾子を使用した場合、クラスは他のクラスの基本クラスとなることだけを目的とします。抽象としてマークされているメンバ、または抽象クラスのメンバは、抽象クラスから派生したクラスを使用して実装する必要があります。

この例では、`Square` クラスは `ShapesClass` から派生したクラスであるので、このクラスにより `Area` が実装されます。

```
      abstract class ShapesClass
{
    abstract public int Area();
}
class Square : ShapesClass
{
    int x, y;
    // Not providing an Area method results
    // in a compile-time error.
    public override int Area()
    {
        return x * y;
    }
}
```

抽象クラスの詳細については、「抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)」を参照してください。

解説

抽象クラスには、以下に示す特徴があります。

- 抽象クラスはインスタンス化できません。
- 抽象クラスは、抽象メソッドおよび抽象アクセサを含む可能性があります。
- 抽象クラスを sealed (C# リファレンス) 修飾子で修飾することはできません。つまり、クラスは継承されません。
- 抽象クラスから派生される非抽象クラスは、継承される抽象メソッドおよび抽象アクセサの実際の実装をすべて含んでいる必要があります。

abstract 修飾子をメソッド宣言またはプロパティ宣言に使用すると、メソッドまたはプロパティに実装を含まないことを示します。

抽象メソッドには、以下に示す特徴があります。

- 抽象メソッドは、暗黙的に virtual なメソッド (仮想メソッド) です。
- 抽象メソッドの宣言は、抽象クラス内でだけ許可されます。
- 抽象メソッドの宣言では実際の実装は用意されないので、メソッドの本体はなく、メソッドの宣言は単にセミコロンで終わり、シグネチャに続く中かっこ ({ }) はありません。次に例を示します。

  ```
  public abstract void MyMethod();
  ```

- この実装はオーバーライドするメソッドによって用意されoverride (C# リファレンス)、非抽象クラスのメンバとなります。
- 抽象メソッドの宣言で static 修飾子または virtual 修飾子を使用するとエラーになります。

抽象プロパティは抽象メソッドと同様に動作しますが、宣言の構文および呼び出しの構文に相違があります。

- 静的プロパティで **abstract** 修飾子を使用するのはエラーです。
- 継承された抽象プロパティを派生クラス内でオーバーライドできます。オーバーライドするには、override 修飾子を使用するプロパティ宣言を含めます。

抽象クラスは、すべてのインターフェイス メンバの実装を用意する必要があります。

インターフェイスを実装する抽象クラスは、抽象メソッドにインターフェイス メソッドを割り当てることもあります。次に例を示します。

```
interface I
{
    void M();
}
abstract class C: I
{
    public abstract void M();
}
```

使用例

この例では、DerivedClass クラスは抽象クラス BaseClass の派生クラスです。この抽象クラスには AbstractMethod という抽象メソッドと、X および Y という 2 つの抽象プロパティがあります。

```
// abstract_keyword.cs
// Abstract Classes
using System;
abstract class BaseClass   // Abstract class
{
    protected int _x = 100;
    protected int _y = 150;
    public abstract void AbstractMethod();   // Abstract method
    public abstract int X    { get; }
    public abstract int Y    { get; }
}

class DerivedClass : BaseClass
{
    public override void AbstractMethod()
    {
        _x++;
        _y++;
    }

    public override int X   // overriding property
    {
        get
        {
            return _x + 10;
        }
    }

    public override int Y   // overriding property
    {
        get
        {
            return _y + 10;
        }
    }

    static void Main()
    {
        DerivedClass o = new DerivedClass();
        o.AbstractMethod();
        Console.WriteLine("x = {0}, y = {1}", o.X, o.Y);
    }
}
```

出力

```
x = 111, y = 161
```

説明

上の例で、次に示すようなステートメントを使用して抽象クラスのインスタンスの作成を試みるとします。

```
BaseClass bc = new BaseClass();   // Error
```

コンパイラが 'BaseClass' 抽象クラスのインスタンスを作成できないことを知らせるエラーが表示されます。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.5.4 仮想メソッド、オーバーライド メソッド、抽象メソッド
- 10.1.1.1 抽象クラス

参照

関連項目

修飾子 (C# リファレンス)
virtual (C# リファレンス)
override (C# リファレンス)
C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

const (C# リファレンス)

const キーワードは、フィールドまたはローカル変数の宣言の修飾に使用されます。このキーワードを使用すると、フィールドまたはローカル変数の値が定数であること、つまりフィールドまたはローカル変数の値は変更できないことを指定できます。次に例を示します。

```
      const int x = 0;
public const double gravitationalConstant = 6.673e-11;
private const string productName = "Visual C#";
```

解説

定数宣言の型は、宣言で導入されるメンバの型を指定します。定数式は、結果となる値が対象の型、または対象の型に暗黙に変換できる型である必要があります。

定数式は、コンパイル時にすべて評価されます。このため、参照型の定数になりうる値は、**string** と **null** に限られます。

定数宣言は、複数の定数を宣言できます。たとえば、次のように宣言できます。

```
public const double x = 1.0, y = 2.0, z = 3.0;
```

static 修飾子は、定数宣言では使用できません。

定数は、次に示すように、定数式の一部になることができます。

```
public const int c1 = 5;
public const int c2 = c1 + 100;
```

メモ：

readonly (C# リファレンス) キーワードは、**const** キーワードとは異なります。**const** フィールドは、フィールドの宣言でしか初期化できません。**readonly** フィールドは、宣言またはコンストラクタのどちらかで初期化できます。このため、**readonly** フィールドは、使用するコンストラクタに応じて異なる値を持つことができます。また、**const** フィールドがコンパイル時定数であるのに対し、**readonly** フィールドは実行時定数として使用できます。`public static readonly uint l1 = (uint)DateTime.Now.Ticks;` の行に例を示します。

使用例

```
// const_keyword.cs
using System;
public class ConstTest
{
    class SampleClass
    {
        public int x;
        public int y;
        public const int c1 = 5;
        public const int c2 = c1 + 5;

        public SampleClass(int p1, int p2)
        {
            x = p1;
            y = p2;
        }
    }

    static void Main()
    {
        SampleClass mC = new SampleClass(11, 22);
        Console.WriteLine("x = {0}, y = {1}", mC.x, mC.y);
        Console.WriteLine("c1 = {0}, c2 = {1}",
                          SampleClass.c1, SampleClass.c2 );
    }
}
```

出力

```
x = 11, y = 22
c1 = 5, c2 = 10
```

この例では、定数をローカル変数として使用する方法を示しています。

```
// const_keyword_2.cs
using System;
public class MainClass
{
    static void Main()
    {
        const int c = 707;
        Console.WriteLine("My local constant = {0}", c);
    }
}
```

出力

```
My local constant = 707
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 6.1.6 暗黙の定数式変換
- 8.5.2 ローカル定数の宣言

参照
関連項目
C# のキーワード
修飾子 (C# リファレンス)
readonly (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

event (C# リファレンス)

event キーワードを使用して、パブリッシャ クラス内にイベントを宣言します。

解説

次の例では、基になるデリゲート型として EventHandler を使用するイベントを宣言し、発生させる方法について説明します。完全なコード例については、「方法 : .NET Framework ガイドラインに準拠したイベントを発行する (C# プログラミング ガイド)」を参照してください。完全なコード例では、ジェネリック EventHandler<T> デリゲート型の使用方法、イベント サブスクリプションの実行方法、およびイベント ハンドラ メソッドの作成方法も確認できます。

```
public class Publisher
{
    // Declare the delegate (if using non-generic pattern).
    public delegate void SampleEventHandler(object sender, SampleEventArgs e);

    // Declare the event.
    public event SampleEventHandler SampleEvent;

    // Wrap the event in a protected virtual method
    // to enable derived classes to raise the event.
    protected virtual void RaiseSampleEvent()
    {
        // Raise the event by using the () operator.
        SampleEvent(this, new SampleEventArgs("Hello"));
    }
}
```

イベントは、宣言元 (パブリッシャ クラス) のクラスまたは構造体内でしか呼び出せない特殊なマルチキャスト デリゲートです。その他のクラスまたは構造体のイベントをサブスクライブすると、パブリッシャ クラスがイベントを発生させるときにイベント ハンドラ メソッドが呼び出されます。詳細およびコード例については、「イベント (C# プログラミング ガイド)」および「デリゲート (C# プログラミング ガイド)」を参照してください。

イベントは、public、private、protected、internal、または **protected internal** とマークできます。これらのアクセス修飾子により、クラスのユーザーがイベントにどのようにアクセスできるかが定義されます。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

キーワードとイベント

イベントには次のキーワードが適用されます。

キーワード	説明	詳細情報
static	このように宣言すると、クラスのインスタンスが存在しない場合でも、呼び出し元がいつでもイベントを使用できるようになります。	静的クラスと静的クラス メンバ (C# プログラミング ガイド)
virtual	この場合、派生クラスでは、override キーワードを使用して、イベントの動作をオーバーライドできます。	継承 (C# プログラミング ガイド)
sealed	派生クラスが virtual でなくなったことを示します。	
abstract	コンパイラはイベント アクセサ ブロック **add** および **remove** を生成しません。したがって、派生クラスは固有の実装を提供する必要があります。	

イベントは、static キーワードを使用して静的イベントと宣言することもできます。このように宣言すると、クラスのインスタンスが存在しない場合でも、呼び出し元がいつでもイベントを使用できるようになります。詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」を参照してください。

また、イベントは、virtual キーワードを使用して仮想イベントとマークできます。この場合、派生クラスでは、override キーワードを使用して、イベントの動作をオーバーライドできます。詳細については、「継承 (C# プログラミング ガイド)」を参照してください。仮想イベントをオーバーライドするイベントは、シールすることもできます。この場合、派生クラスでは、イベントが仮想でなくなります。イベントを abstract と宣言した場合、コンパイラはイベント アクセサ ブロック **add** および **remove** を生成しません。したがって、派生クラスは固有の実装を提供する必要があります。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.4 イベント
- 7.13.3 イベント代入
- 10.7 イベント
- 13.2.3 インターフェイスのイベント

参照

処理手順

方法 : デリゲートを結合する (マルチキャスト デリゲート) (C# プログラミング ガイド)

関連項目

C# のキーワード

修飾子 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

extern (C# リファレンス)

extern 修飾子は、外部で実装されるメソッドを宣言するために使用します。**extern** 修飾子は一般に、相互運用サービスを使用してアンマネージ コードを呼び出すときに **DllImport** 属性と共に使用します。メソッドはこの場合、次の例に示すように、**static** と宣言される必要もあります。

```
[DllImport("avifil32.dll")]
private static extern void AVIFileInit();
```

メモ :

extern キーワードでは、外部アセンブリのエイリアスも定義できます。これにより、単一アセンブリ内から 1 つのコンポーネントの複数バージョンを参照できます。詳細については、「extern エイリアス (C# リファレンス)」を参照してください。

abstract (C# リファレンス) 修飾子および **extern** 修飾子を一緒に使用して同一のメンバを修飾するのは、エラーです。**extern** 修飾子を使用して、メソッドが C# コードの外部で実装されていることを意味します。一方、**abstract** 修飾子を使用すると、メソッドの実装がクラスには用意されていないことを意味します。

メモ :

extern キーワードの用法は、C++ の場合よりも制限されています。C++ のキーワードとの比較については、『C++ Language Reference』の「Using extern to Specify Linkage」を参照してください。

使用例

この例では、ユーザーの入力したメッセージがプログラムに受け取られ、メッセージ ボックスに表示されます。このプログラムは、`User32.dll` ライブラリからインポートされた `MessageBox` メソッドを使用します。

```
using System;
using System.Runtime.InteropServices;
class MainClass
{
   [DllImport("User32.dll")]
   public static extern int MessageBox(int h, string m, string c, int type);

   static int Main()
   {
      string myString;
      Console.Write("Enter your message: ");
      myString = Console.ReadLine();
      return MessageBox(0, myString, "My Message Box", 0);
   }
}
```

この例では、次の例で C# プログラム内から起動する C プログラムから DLL を作成します。

```
// cmdll.c
// compile with: /LD
int __declspec(dllexport) SampleMethod(int i)
{
   return i*10;
}
```

この例では、**extern** を例示するために、`CM.cs` および `Cmdll.c` という 2 つのファイルを使用します。この C ファイルは、例 2 で作成された外部 DLL です。これは C# プログラムで呼び出されます。

```
// cm.cs
using System;
using System.Runtime.InteropServices;
public class MainClass
```

```
{
   [DllImport("Cmdll.dll")]
   public static extern int SampleMethod(int x);

   static void Main()
   {
      Console.WriteLine("SampleMethod() returns {0}.", SampleMethod(5));
   }
}
```

出力

```
SampleMethod() returns 50.
```

解説

プロジェクトをビルドするには

- Visual C++ のコマンド ラインを使用して、`Cmdll.c` を DLL にコンパイルします。

  ```
  cl /LD Cmdll.c
  ```

- コマンド ラインを使用して `CM.cs` をコンパイルします。

  ```
  csc CM.cs
  ```

これで、実行可能ファイル `CM.exe` が作成されます。このプログラムが実行されると、`SampleMethod` が値 5 を DLL ファイルに渡し、DLL は渡された値に 10 を乗算した値を返します。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 10.5.7 外部メソッド

参照

関連項目

C# のキーワード
修飾子 (C# リファレンス)
System.Runtime.InteropServices.DllImportAttribute

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

override (C# リファレンス)

override 修飾子は、継承したメソッド、プロパティ、インデクサ、またはイベントの抽象実装や仮想実装を拡張したり修飾したりする際に必要です。

この例では、`Square` クラスが `Area` のオーバーライド実装を指定する必要があります。これは、`Area` が抽象 `ShapesClass` から継承されているためです。

```
abstract class ShapesClass
{
    abstract public int Area();
}

class Square : ShapesClass
{
    int x, y;
    // Because ShapesClass.Area is abstract, failing to override
    // the Area method would result in a compilation error.
    public override int Area()
    {
        return x * y;
    }
}
```

override キーワードの使い方の詳細については、「Override キーワードと New キーワードによるバージョン管理 (C# プログラミング ガイド)」および「Override キーワードと New キーワードを使用する場合について (C# プログラミング ガイド)」を参照してください。

解説

override メソッドは、基本クラスから継承されたメンバの新しい実装を用意します。**override** 宣言によってオーバーライドされるメソッドのことをオーバーライドされる基本メソッドと言います。オーバーライドされた基本メソッドは、**override** メソッドと同じシグネチャを持つ必要があります。継承の詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

非仮想メソッドまたは静的メソッドのオーバーライドはできません。オーバーライドされる基本メソッドは、**virtual**、**abstract**、または **override** のいずれかである必要があります。

override 宣言は、**virtual** メソッドのアクセシビリティを変更できません。**override** メソッドと **virtual** メソッドは、同じアクセス レベル修飾子を持つ必要があります。

override メソッドの修飾に、修飾子 **new**、**static**、**virtual**、または **abstract** は使用できません。

プロパティ宣言のオーバーライドで指定する、アクセス修飾子、型、および名前は、継承されるプロパティのものと正確に同一である必要があります。オーバーライドされるプロパティは、**virtual**、**abstract**、または **override** である必要があります。

使用例

この例では、`Employee` という基本クラスと、`SalesEmployee` という派生クラスを定義します。`SalesEmployee` クラスには追加のプロパティ `salesbonus` があり、このプロパティを考慮するため、`CalculatePay` メソッドがオーバーライドされます。

```
using System;
class TestOverride
{
    public class Employee
    {
        public string name;

        // Basepay is defined as protected, so that it may be
        // accessed only by this class and derrived classes.
        protected decimal basepay;

        // Constructor to set the name and basepay values.
        public Employee(string name, decimal basepay)
        {
            this.name = name;
            this.basepay = basepay;
        }
```

```
        // Declared virtual so it can be overridden.
        public virtual decimal CalculatePay()
        {
            return basepay;
        }
    }

    // Derive a new class from Employee.
    public class SalesEmployee : Employee
    {
        // New field that will affect the base pay.
        private decimal salesbonus;

        // The constructor calls the base-class version, and
        // initializes the salesbonus field.
        public SalesEmployee(string name, decimal basepay,
                  decimal salesbonus) : base(name, basepay)
        {
            this.salesbonus = salesbonus;
        }

        // Override the CalculatePay method
        // to take bonus into account.
        public override decimal CalculatePay()
        {
            return basepay + salesbonus;
        }
    }

    static void Main()
    {
        // Create some new employees.
        SalesEmployee employee1 = new SalesEmployee("Alice",
                      1000, 500);
        Employee employee2 = new Employee("Bob", 1200);

        Console.WriteLine("Employee " + employee1.name +
                  " earned: " + employee1.CalculatePay());
        Console.WriteLine("Employee " + employee2.name +
                  " earned: " + employee2.CalculatePay());
    }
}
```

出力

```
Employee Alice earned: 1500
Employee Bob earned: 1200
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.5.4 仮想メソッド、オーバーライド メソッド、抽象メソッド
- 10.5.4 オーバーライド メソッド

参照

関連項目

継承 (C# プログラミング ガイド)
C# のキーワード
修飾子 (C# リファレンス)
abstract (C# リファレンス)
virtual (C# リファレンス)
new (C# リファレンス)

概念

C# プログラミング ガイド
ポリモーフィズム (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

readonly (C# リファレンス)

readonly キーワードは、フィールドに使用できる修飾子です。フィールド宣言が **readonly** 修飾子を含む場合、宣言によって導入されるフィールドへの代入は、宣言の一部としてだけ、または同じクラスのコンストラクタ内でだけ発生できます。この例では、クラス コンストラクタに値を割り当てる場合でも、`ChangeYear` メソッドでは `year` フィールドの値を変更できません。

```
class Age
{
    readonly int _year;
    Age(int year)
    {
        _year = year;
    }
    void ChangeYear()
    {
        _year = 1967; // Will not compile.
    }
}
```

解説

readonly のフィールドに値の代入ができるのは、次のコンテキスト内に限られます。

- 値が宣言で初期化される場合。たとえば、次のようになります。

  ```
  public readonly int y = 5;
  ```

- インスタンス フィールドの場合は、フィールド宣言を含んでいるクラスのインスタンス コンストラクタ内。静的フィールドの場合は、フィールド宣言を含んでいるクラスの静的コンストラクタ内。また、これらのコンテキスト内でだけ、`readonly` フィールドを out パラメータまたは ref パラメータとして渡すことができます。

メモ :

readonly キーワードは const キーワードとは異なります。**const** フィールドは、フィールドの宣言でしか初期化できません。**readonly** フィールドは、宣言またはコンストラクタのどちらかで初期化できます。このため、**readonly** フィールドは、使用するコンストラクタに応じて異なる値を持つことができます。また、**const** フィールドがコンパイル時定数であるのに対し、**readonly** フィールドは実行時定数として使用できます。次に例を示します。

メモ :

```
public static readonly uint l1 = (uint)DateTime.Now.Ticks;
```

使用例

```
// cs_readonly_keyword.cs
// Readonly fields
using System;
public class ReadOnlyTest
{
   class SampleClass
   {
      public int x;
      // Initialize a readonly field
      public readonly int y = 25;
      public readonly int z;

      public SampleClass()
      {
         // Initialize a readonly instance field
         z = 24;
      }
```

```
        public SampleClass(int p1, int p2, int p3)
        {
            x = p1;
            y = p2;
            z = p3;
        }
    }

    static void Main()
    {
        SampleClass p1 = new SampleClass(11, 21, 32);   // OK
        Console.WriteLine("p1: x={0}, y={1}, z={2}", p1.x, p1.y, p1.z);
        SampleClass p2 = new SampleClass();
        p2.x = 55;   // OK
        Console.WriteLine("p2: x={0}, y={1}, z={2}", p2.x, p2.y, p2.z);
    }
}
```

出力

```
p1: x=11, y=21, z=32
p2: x=55, y=25, z=24
```

説明

上の例で、次のようにステートメントを使用するとします。

```
p2.y = 66; // Error
```

次のコンパイル エラー メッセージが表示されることになります。

```
The left-hand side of an assignment must be an l-value
```

これは、定数に値を代入しようとしたときのエラーと同じです。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 10.4.2 readonly フィールド

参照

関連項目

C# のキーワード
修飾子 (C# リファレンス)
const (C# リファレンス)
フィールド (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

sealed (C# リファレンス)

sealed 修飾子は、クラス、インスタンス メソッド、およびプロパティに適用できます。シール クラスは継承できません。シール メソッドは基本クラスのメソッドをオーバーライドしますが、それ自体はどの派生クラスでもオーバーライドされません。メソッドまたはプロパティに適用する場合、**sealed** 修飾子は常に override (C# リファレンス) と共に使用される必要があります。

このようなクラスを継承しないためには、クラス宣言で **sealed** 修飾子を使用します。次に例を示します。

```
       sealed class SealedClass
{
    public int x;
    public int y;
}
```

シール クラスを基本クラスとして使用する、または abstract 修飾子をシール クラスで使用するとエラーになります。

構造体は暗黙にシールされます。このため、構造体の継承はできません。

継承の詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

使用例

```
// cs_sealed_keyword.cs
using System;
sealed class SealedClass
{
    public int x;
    public int y;
}

class MainClass
{
    static void Main()
    {
        SealedClass sc = new SealedClass();
        sc.x = 110;
        sc.y = 150;
        Console.WriteLine("x = {0}, y = {1}", sc.x, sc.y);
    }
}
```

出力

```
x = 110, y = 150
```

上の例で、次に示すようなステートメントを使用して、シール クラスの継承を試みます。

```
class MyDerivedC: SealedClass {} // Error
```

次のエラー メッセージが表示されることになります。

```
'MyDerivedC' cannot inherit from sealed class 'SealedClass'.
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 10.1.1.2 シール クラス
- 10.5.5 シール メソッド

参照

関連項目

C# のキーワード
静的クラスと静的クラス メンバ (C# プログラミング ガイド)
抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)

アクセス修飾子 (C# プログラミング ガイド)
修飾子 (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

static (C# リファレンス)

static 修飾子は、静的メンバの宣言に使用します。静的メンバは、特定のオブジェクトではなく、型自体に属するメンバです。**static** 修飾子は、クラス、フィールド、メソッド、プロパティ、演算子、イベント、およびコンストラクタと組み合わせて使用できますが、インデクサ、デストラクタ、またはクラス以外の型で使用することはできません。たとえば、次のクラスは **static** として宣言され、**static** メソッドのみが含まれます。

```
      static class CompanyEmployee
{
    public static string GetCompanyName(string name) { ... }
    public static string GetCompanyAddress(string address) { ... }
}
```

詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」を参照してください。

解説

- 定数宣言や型宣言は、暗黙に静的メンバです。
- 静的メンバは、インスタンスを使って参照できません。代わりに、型の名前を使って参照します。たとえば、次のクラスを考えます。

  ```
  public class MyBaseC
  {
      public struct MyStruct
      {
          public static int x = 100;
      }
  }
  ```

  静的メンバ x を参照するには、同じスコープで静的メンバにアクセス可能でない限り、完全名を使用します。

  ```
  MyBaseC.MyStruct.x
  ```

- クラスのインスタンスには同クラスのすべてのインスタンス フィールドのコピーが別に含まれますが、各静的フィールドのコピーは 1 つだけです。
- 静的メソッドまたは静的プロパティ アクセサへの参照には、this は使用できません。
- **static** キーワードをクラスに適用する場合、そのクラスのすべてのメンバが静的であることが必要です。
- 静的クラスを含め、クラスには静的コンストラクタを含めることができます。静的コンストラクタは、プログラム開始時点からクラスがインスタンス化される時点までの間に呼び出されます。

  メモ :

  static キーワードの用法は、C++ の場合よりも制限されています。C++ キーワードと比較するには、「static」を参照してください。

静的メンバを例示するために、ある企業の従業員を表すクラスを考えてみます。このクラスには、従業員の数を数えるメソッドと、従業員の数を格納するフィールドがあると仮定します。メソッドおよびフィールドは共に、従業員のどのインスタンスにも属していません。代わりに、それらは企業のクラスに属しています。このため、クラスの静的なメンバとして宣言する必要があります。

使用例

この例では、新しい従業員の名前と ID を読み取り、従業員数のカウンタを 1 インクリメントして、新しい従業員の情報および新しい従業員総数を表示します。簡略化のために、この例では現在の従業員数をキーボード入力から読み取っています。実際のアプリケーションでは、ファイルから情報を読み取るようにしてください。

```
// cs_static_keyword.cs
using System;
```

```
public class Employee
{
    public string id;
    public string name;

    public Employee()
    {
    }

    public Employee(string name, string id)
    {
        this.name = name;
        this.id = id;
    }

    public static int employeeCounter;

    public static int AddEmployee()
    {
        return ++employeeCounter;
    }
}

class MainClass : Employee
{
    static void Main()
    {
        Console.Write("Enter the employee's name: ");
        string name = Console.ReadLine();
        Console.Write("Enter the employee's ID: ");
        string id = Console.ReadLine();

        // Create and configure the employee object:
        Employee e = new Employee(name, id);
        Console.Write("Enter the current number of employees: ");
        string n = Console.ReadLine();
        Employee.employeeCounter = Int32.Parse(n);
        Employee.AddEmployee();

        // Display the new information:
        Console.WriteLine("Name: {0}", e.name);
        Console.WriteLine("ID:   {0}", e.id);
        Console.WriteLine("New Number of Employees: {0}",
                    Employee.employeeCounter);
    }
}
```

入力

```
Tara Strahan
AF643G
15
```

サンプル出力

```
Enter the employee's name: Tara Strahan
Enter the employee's ID: AF643G
Enter the current number of employees: 15
Name: Tara Strahan
ID:   AF643G
New Number of Employees: 16
```

この例では、宣言されていない別の静的フィールドを使用して静的フィールドを初期化できるけれども、静的フィールドに値を明示的に割り当てない限り、結果が未定義になることを示します。

```
// cs_static_keyword_2.cs
using System;
class Test
```

```
{
   static int x = y;
   static int y = 5;

   static void Main()
   {
      Console.WriteLine(Test.x);
      Console.WriteLine(Test.y);

      Test.x = 99;
      Console.WriteLine(Test.x);
   }
}
```

出力

```
0
5
99
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.5.3 静的メソッドとインスタンス メソッド
- 5.1.1 静的変数
- 10.2.5 静的メンバとインスタンス メンバ
- 10.4.1 静的フィールドとインスタンス フィールド
- 10.4.5.1 静的フィールドの初期化
- 10.5.2 静的メソッドとインスタンス メソッド
- 10.6.1 静的プロパティとインスタンス プロパティ
- 10.7.3 静的イベントとインスタンス イベント
- 10.11 静的コンストラクタ

参照

関連項目

C# のキーワード
修飾子 (C# リファレンス)
静的クラスと静的クラス メンバ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

unsafe (C# リファレンス)

unsafe キーワードは、unsafe コンテキストを示し、ポインタが関係するすべての操作に必要です。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

unsafe 修飾子は、型またはメンバの宣言で使用できます。このため、型またはメンバの宣言の範囲全体が、unsafe コンテキストと見なされます。**unsafe** 修飾子を使用して宣言されたメソッドの例を次に示します。

```
      unsafe static void FastCopy(byte[] src, byte[] dst, int count)
{
    // Unsafe context: can use pointers here.
}
```

unsafe コンテキストのスコープは、パラメータ リストからメソッドの末尾にまで拡張されます。このため、パラメータ リストではポインタも使用できます。

```
      unsafe static void FastCopy ( byte* ps, byte* pd, int count ) {...}
```

また、unsafe ブロックを使用すると、アンセーフ コードをそのブロック内で使用できるようになります。次に例を示します。

```
      unsafe
{
    // Unsafe context: can use pointers here.
}
```

アンセーフ コードをコンパイルするには、/unsafe コンパイラ オプションを指定する必要があります。アンセーフ コードは、共通言語ランタイムでは検証できません。

使用例

```
// cs_unsafe_keyword.cs
// compile with: /unsafe
using System;
class UnsafeTest
{
   // Unsafe method: takes pointer to int:
   unsafe static void SquarePtrParam(int* p)
   {
      *p *= *p;
   }

   unsafe static void Main()
   {
      int i = 5;
      // Unsafe method: uses address-of operator (&):
      SquarePtrParam(&i);
      Console.WriteLine(i);
   }
}
```

出力

```
25
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 18 アンセーフ コード

参照

関連項目

C# のキーワード

fixed ステートメント (C# リファレンス)

固定サイズ バッファ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

unsafe コードとポインタ (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

virtual (C# リファレンス)

virtual キーワードを使用して、メソッド、プロパティ、インデクサ、またはイベント宣言を変更し、派生クラスでオーバーライドできるようにします。たとえば、このメソッドは、このメソッドを継承するクラスでオーバーライドできます。

```
public virtual double Area()
{
    return x * y;
}
```

仮想メンバの実装は、オーバーライドするメンバによって派生クラスで変更できます。**virtual** キーワードの使い方の詳細については、「Override キーワードと New キーワードによるバージョン管理 (C# プログラミング ガイド)」および「Override キーワードと New キーワードを使用する場合について (C# プログラミング ガイド)」を参照してください。

解説

仮想メンバが呼び出されるときには、オブジェクトの実行時の型が、オーバーライドするメンバで確認されます。メンバをオーバーライドしている派生クラスがない場合には、おそらくはオリジナルのメンバである、最派生クラスでオーバーライドするメンバが呼び出されます。

既定では、これらのメソッドは非仮想です。非仮想メソッドのオーバーライドはできません。

virtual 修飾子は、**static**、**abstract, private**、または **override** の各修飾子と一緒には使用できません。

仮想プロパティは抽象メソッドと同様に動作しますが、宣言の構文および呼び出しの構文に相違があります。

- 静的プロパティで **virtual** 修飾子を使用するのはエラーです。
- 継承された仮想プロパティを派生クラス内でオーバーライドできます。オーバーライドするには、**override** 修飾子を使用しているプロパティ宣言をインクルードします。

使用例

この例では、`Dimensions` クラスに 2 本の座標軸 `x`、`y`、および `Area()` 仮想メソッドがあります。図形のクラス、たとえば `Circle`、`Cylinder`、および `Sphere` が `Dimensions` クラスを継承し、各図形について、表面積が計算されます。各派生クラスには、`Area()` の独自のオーバーライド実装があります。このプログラムは、メソッドと関連付けられているオブジェクトに従って、`Area()` の適切な実装を呼び出すことによって、各図形の適切な表面積の計算および表示を行います。

継承された `Circle`、`Sphere`、および `Cylinder` のすべてのクラスが、基本クラスを初期化するコンストラクタを使用することに注意してください。たとえば、次のようなコードです。

```
public Cylinder(double r, double h): base(r, h) {}
```

これは、C++ の初期化リストに似ています。

```
// cs_virtual_keyword.cs
using System;
class TestClass
{
    public class Dimensions
    {
        public const double PI = Math.PI;
        protected double x, y;
        public Dimensions()
        {
        }
        public Dimensions(double x, double y)
        {
            this.x = x;
            this.y = y;
        }

        public virtual double Area()
        {
            return x * y;
```

```
        }
    }

    public class Circle : Dimensions
    {
        public Circle(double r) : base(r, 0)
        {
        }

        public override double Area()
        {
            return PI * x * x;
        }
    }

    class Sphere : Dimensions
    {
        public Sphere(double r) : base(r, 0)
        {
        }

        public override double Area()
        {
            return 4 * PI * x * x;
        }
    }

    class Cylinder : Dimensions
    {
        public Cylinder(double r, double h) : base(r, h)
        {
        }

        public override double Area()
        {
            return 2 * PI * x * x + 2 * PI * x * y;
        }
    }

    static void Main()
    {
        double r = 3.0, h = 5.0;
        Dimensions c = new Circle(r);
        Dimensions s = new Sphere(r);
        Dimensions l = new Cylinder(r, h);
        // Display results:
        Console.WriteLine("Area of Circle   = {0:F2}", c.Area());
        Console.WriteLine("Area of Sphere   = {0:F2}", s.Area());
        Console.WriteLine("Area of Cylinder = {0:F2}", l.Area());
    }
}
```

出力

```
Area of Circle   = 28.27
Area of Sphere   = 113.10
Area of Cylinder = 150.80
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.5.4 仮想メソッド、オーバーライド メソッド、抽象メソッド
- 10.5.3 仮想メソッド
- 10.6.3 仮想、シール、オーバーライド、抽象の各アクセサ

参照

関連項目
修飾子 (C# リファレンス)
C# のキーワード
abstract (C# リファレンス)
override (C# リファレンス)
new (C# リファレンス)
概念
C# プログラミング ガイド
ポリモーフィズム (C# プログラミング ガイド)
その他の技術情報
C# リファレンス

volatile (C# リファレンス)

volatile キーワードは、同時に実行中の複数のスレッドによってフィールドが変更される可能性があることを示します。**volatile** と宣言されているフィールドは、シングル スレッドによるアクセスを前提とする、コンパイラの最適化の対象にはなりません。このため、フィールドには常に最新の値が含まれます。

volatile 修飾子は、通常、アクセスをシリアル化する lock ステートメント (C# リファレンス) ステートメントが使用されない場合に複数のスレッドによりアクセスされるフィールドに対して使用します。マルチスレッド シナリオにおける **volatile** の例については、「方法 : スレッドを作成および終了する (C# プログラミング ガイド)」を参照してください。

volatile キーワードは次の型のフィールドに使用できます。

- 参照型
- ポインタ型 (unsafe コンテキスト内)ポインタ自体は volatile にすることができますが、ポインタが指しているオブジェクトは volatile にすることができません。つまり、"volatile を指すポインタ" は宣言できません。
- sbyte、byte、short、ushort、int、uint、char、float、bool などの整数型
- 整数ベースの型の列挙型
- 参照型であることが判明しているジェネリック型パラメータ
- IntPtr 型および UIntPtr 型

volatile キーワードは、クラスまたは構造体のフィールドにのみ適用できます。ローカル変数を **volatile** で宣言することはできません。

使用例

次の例では、public のフィールド変数を **volatile** として宣言する方法を示します。

```
// csharp_volatile.cs
// compile with: /target:library
class Test
{
   public volatile int i;

   Test(int _i)
   {
      i = _i;
   }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.10 実行順序
- 10.4.3 Volatile フィールド

参照

関連項目

C# のキーワード

修飾子 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

ステートメントの種類 (C# リファレンス)

ステートメントとは、プログラムの命令のことです。特に記述される場合を除いて、ステートメントは順番に実行されます。C# には、次のカテゴリのステートメントがあります。

カテゴリ	C# のキーワード
選択ステートメント	if、else、switch、case
繰り返しステートメント	do、for、foreach、in、while
ジャンプ ステートメント	break、continue、default、goto、return、yield (C# リファレンス)
例外処理ステートメント	throw、try-catch、try-finally、try-catch-finally
Checked と Unchecked	checked、unchecked
fixed ステートメント	fixed
lock ステートメント	lock

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

選択ステートメント (C# リファレンス)

選択ステートメントは、指定された条件が **true** か false かに基づいて、プログラムの制御を特定の流れに移動させます。

選択ステートメントでは、次のキーワードを使用します。

- if
- else
- switch
- case
- default

参照
関連項目
C# のキーワード
ステートメントの種類 (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

if-else (C# リファレンス)

if ステートメントは、**Boolean** 型の式の値に基づいて、実行するステートメントを選択します。次の例では、**Boolean** フラグ `flagCheck` が **true** に設定され、**if** ステートメントで検証されます。出力は `The flag is set to true` になります。

```
bool flagCheck = true;
if (flagCheck == true)
{
    Console.WriteLine("The flag is set to true.");
}
else
{
    Console.WriteLine("The flag is set to false.");
}
```

解説

かっこで囲まれた式が true と評価されると、`Console.WriteLine("The boolean flag is set to ture.");` ステートメントが実行されます。**if** ステートメントを実行した後、制御は次のステートメントに移動します。この例では else は実行されません。

複数のステートメントを実行する場合には、上の例で示すように、**{}** を使用してこれらのステートメントをブロックに組み込むと、条件付きで実行できます。

条件の評価に従って実行されるステートメントは、元の **if** ステートメントに入れ子になった他の **if** ステートメントなど、どのような種類のステートメントでもかまいません。**if** ステートメントが入れ子にされている場合、**else** 句は、対応する **else** を持たない最後の **if** に従属します。次に例を示します。

```
        if (x > 10)
    if (y > 20)
        Console.Write("Statement_1");
    else
        Console.Write("Statement_2");
```

この例では、`Statement_2` は、条件 `(y > 20)` が **false** の場合に表示されます。ただし、`Statement_2` と条件 `(x >10)` を関連付ける場合は次のように中かっこを使用します。

```
        if (x > 10)
{
    if (y > 20)
        Console.Write("Statement_1");
}
else
    Console.Write("Statement_2");
```

この場合は、`Statement_2` は、条件 `(x > 10)` が **false** の場合に表示されます。

例 1

キーボードから文字を入力して、入力文字がアルファベットかどうかをチェックする例を次に示します。アルファベットの場合は、大文字か小文字かをチェックします。いずれの場合にも、それぞれに応じたメッセージが表示されます。

```
// statements_if_else.cs
// if-else example
using System;
class IfTest
{
    static void Main()
    {
        Console.Write("Enter a character: ");
        char c = (char)Console.Read();
        if (Char.IsLetter(c))
```

```
        {
            if (Char.IsLower(c))
            {
                Console.WriteLine("The character is lowercase.");
            }
            else
            {
                Console.WriteLine("The character is uppercase.");
            }
        }
        else
        {
            Console.WriteLine("Not an alphabetic character.");
        }
    }
}
```

入力

```
2
```

出力例

```
Enter a character: 2
The character is not an alphabetic character.
```

他の実行例は次のようになります。

実行サンプル 2:

```
Enter a character: A
The character is uppercase.
```

実行サンプル 3 :

```
Enter a character: h
The character is lowercase.
```

また、次の else-if 配置によって **if** ステートメントを拡張し、複数の条件を処理することもできます。

```
            if (Condition_1)
{
    // Statement_1;
}
else if (Condition_2)
{
    // Statement_2;
}
else if (Condition_3)
{
    // Statement_3;
}
else
{
    // Statement_n;
}
```

例 2

入力文字が小文字、大文字、数字のいずれであるかをチェックする例を次に示します。それ以外の場合、入力文字は英数字ではありません。このプログラムでは、else-if による条件分岐を使用します。

```
// statements_if_else2.cs
// else-if
```

```
using System;
public class IfTest
{
    static void Main()
    {
        Console.Write("Enter a character: ");
        char c = (char)Console.Read();

        if (Char.IsUpper(c))
        {
            Console.WriteLine("Character is uppercase.");
        }
        else if (Char.IsLower(c))
        {
            Console.WriteLine("Character is lowercase.");
        }
        else if (Char.IsDigit(c))
        {
            Console.WriteLine("Character is a number.");
        }
        else
        {
            Console.WriteLine("Character is not alphanumeric.");
        }
    }
}
```

入力

```
E
```

出力例

```
Enter a character: E
The character is uppercase.
```

他の実行例は次のようになります。

実行サンプル 2 :

```
Enter a character: e
The character is lowercase.
```

実行サンプル 3 :

```
Enter a character: 4
The character is a number.
```

実行サンプル 4:

```
Enter a character: $
The character is not alphanumeric.
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.5 if ステートメント
- 8.7.1 if ステートメント

参照

関連項目

C# のキーワード

?: 演算子 (C# リファレンス)

The if-else Statement
switch (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

switch (C# リファレンス)

switch ステートメントは、次の例に示すように、ステートメントの本体にある **case** ステートメントの 1 つに制御と列挙体を渡すことで複数選択を処理する制御ステートメントです。

```
int caseSwitch = 1;
switch (caseSwitch)
{
    case 1:
        Console.WriteLine("Case 1");
        break;
    case 2:
        Console.WriteLine("Case 2");
        break;
    default:
        Console.WriteLine("Default case");
        break;
}
```

解説

制御は、switch の値と一致する **case** ステートメントに移ります。**switch** ステートメントには、**case** インスタンスを任意の数だけ指定できますが、2 つの case ステートメントに同じ値を指定することはできません。ステートメント本体の実行は指定されたステートメントから始まり、**break** ステートメントによって制御が **case** 本体の外部に移動するまで実行されます。**break** などのジャンプ ステートメントは、**case** ステートメントまたは **default** ステートメントのいずれであっても、各 **case** ブロックの後ろに記述する必要があります。1 つの例外を除き (C++ の **switch** ステートメントとは異なり)、C# では、ある case ラベルからその下の case ラベルへの暗黙的な落下 (フォール スルー) をサポートしていません。この例外とは、**case** ステートメントにコードがない場合をいいます。

case 式が switch 値と一致しない場合、制御はオプションの **default** ラベルの後ろにあるステートメントに移動します。**default** ラベルがない場合、制御は **switch** の外部に移動します。

使用例

```
// statements_switch.cs
using System;
class SwitchTest
{
    static void Main()
    {
        Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
        Console.Write("Please enter your selection: ");
        string s = Console.ReadLine();
        int n = int.Parse(s);
        int cost = 0;
        switch(n)
        {
        case 1:
            cost += 25;
            break;
        case 2:
            cost += 25;
            goto case 1;
        case 3:
            cost += 50;
            goto case 1;
        default:
            Console.WriteLine("Invalid selection. Please select 1, 2, or 3.");
            break;
        }
        if (cost != 0)
        {
            Console.WriteLine("Please insert {0} cents.", cost);
        }
        Console.WriteLine("Thank you for your business.");
    }
```

```
}
```

入力

```
2
```

サンプル出力

```
Coffee sizes: 1=Small 2=Medium 3=Large
Please enter your selection: 2
Please insert 50 cents.
Thank you for your business.
```

次のサンプルでは、空の case ラベルについて、ある case ラベルからその下の case ラベルへの落下 (フォール スルー) を許可する例を示します。

```
// statements_switch2.cs
using System;
class SwitchTest
{
    static void Main()
    {
        int n = 2;
        switch(n)
        {
            case 1:
            case 2:
            case 3:
                Console.WriteLine("It's 1, 2, or 3.");
                break;
            default:
                Console.WriteLine("Not sure what it is.");
                break;
        }
    }
}
```

出力

```
It's 1, 2, or 3.
```

コードの説明

- 前述の例では、整数型の変数 `n` が switch-case に使用されています。文字列変数 `s` を直接使用することもできます。その場合は、switch-case を次のように使用します。

```

        switch(s)
{
    case "1":
        // ...
    case "2":
        // ...
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.6 switch ステートメント
- 8.7.2 switch ステートメント

参照
関連項目
C# のキーワード
The switch Statement
if-else (C# リファレンス)

概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

繰り返しステートメント (C# リファレンス)

繰り返しステートメントを使用すると、ループを作成できます。繰り返しステートメントは、ループ終了条件に応じて、埋め込みステートメントを複数回繰り返して実行します。このステートメントは、ジャンプ ステートメントを検出した場合を除いて、順序どおりに実行されます。

繰り返しステートメントでは、次のキーワードを使用します。

- do
- for
- foreach
- in
- while

参照

関連項目

C# のキーワード
ステートメントの種類 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

do (C# リファレンス)

do ステートメントは、指定した式が **false** になるまでステートメントまたは**{}** で囲まれたステートメントのブロックを繰り返し実行します。次の例では、y が 5 未満である限り、**do-while** ループ ステートメントが実行されます。

使用例

```
// statements_do.cs
using System;
public class TestDoWhile
{
    public static void Main ()
    {
        int x = 0;
        do
        {
            Console.WriteLine(x);
            x++;
        } while (x < 5);
    }
}
```

出力

```
0
1
2
3
4
```

解説

while ステートメントとは異なり、**do-while** ループは条件式の評価前に 1 回実行されます。

do-while ブロック内の任意の位置で、break ステートメントを使用してループを抜けることができます。continue ステートメントを使用して、**while** 式の評価ステートメントに直接ステップできます。式の評価が true の場合、ループ内の最初のステートメントで実行が続行されます。式の評価が false の場合、**do-while** ループの後の最初のステートメントで実行が続行されます。

goto ステートメント、return ステートメント、または throw ステートメントを使用しても、**do-while** ループを抜けることができます。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.8 do ステートメント
- 8.8.2 do ステートメント

参照

関連項目

C# のキーワード
The do-while Statement (C++)
繰り返しステートメント (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

for (C# リファレンス)

for ループは、指定した式が **false** になるまでステートメントまたはステートメントのブロックを繰り返し実行します。**for** ループは、配列を繰り返し処理する場合や、順次処理を実行する場合に便利です。次の例では、`int i` の値がコンソールに出力され、ループを実行するたびに `i` が 1 ずつインクリメントされます。

使用例

```
// statements_for.cs
// for loop
using System;
class ForLoopTest
{
    static void Main()
    {
        for (int i = 1; i <= 5; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

出力

```
1
2
3
4
5
```

解説

for ステートメントは、かっこで囲まれたステートメントを次のように繰り返し実行します。

- 最初に、変数 `i` の初期値が評価されます。
- 次に、`i` の値が 5 以下である間は条件が **true** になり、`Console.WriteLine` ステートメントが実行されて `i` が再計算されます。
- `i` が 5 よりも大きい場合には、条件が **false** になり、制御がループの外に移ります。

条件式をテストした後にループが実行されるので、**for** ステートメントは 0 回以上実行されます。

for ステートメントの式はすべてオプションなので、たとえば次のステートメントでは無限ループを記述できます。

```
        for (;;)
{
    // ...
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.9 for ステートメント
- 8.8.3 for ステートメント

参照

関連項目

C# のキーワード
foreach、in (C# リファレンス)
The for Statement
繰り返しステートメント (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

foreach、in (C# リファレンス)

foreach ステートメントは、配列またはオブジェクト コレクションのそれぞれの要素に対して埋め込みステートメントを繰り返します。**foreach** ステートメントは、コレクションを繰り返し処理して目的の情報を取得するのに使用しますが、予期しない動作を防ぐため、コレクション内容の変更には使用しないでください。

解説

埋め込みステートメントは、配列またはコレクション内の各要素に対して繰り返し実行されます。コレクション内の全要素に対する繰り返しが完了すると、制御は、**foreach** ブロックに続く次のステートメントに移動します。

foreach ブロック内の任意の位置で、break キーワードを使用してループを出ることができます。または、continue キーワードを使用して、ループ内の次の反復処理に直接ステップできます。

goto ステートメント、return ステートメント、または throw ステートメントを使用しても、**foreach** ループを抜けることができます。

foreach キーワードとコード例の詳細については、以下のトピックを参照してください。

配列での foreach の使用 (C# プログラミング ガイド)

方法 : foreach を使用してコレクション クラスにアクセスする (C# プログラミング ガイド)

使用例

この例では、**foreach** を使用して整数配列の内容が表示されます。

```
// cs_foreach.cs
class ForEachTest
{
    static void Main(string[] args)
    {
        int[] fibarray = new int[] { 0, 1, 2, 3, 5, 8, 13 };
        foreach (int i in fibarray)
        {
            System.Console.WriteLine(i);
        }
    }
}
```

出力

```
0
1
2
3
5
8
13
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.16 foreach ステートメント
- 8.8.4 foreach ステートメント

参照

関連項目

C# のキーワード

繰り返しステートメント (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

while (C# リファレンス)

while ステートメントは、指定した式が **false** になるまでステートメントまたはステートメントのブロックを実行します。

使用例

```
// statements_while.cs
using System;
class WhileTest
{
    static void Main()
    {
        int n = 1;
        while (n < 6)
        {
            Console.WriteLine("Current value of n is {0}", n);
            n++;
        }
    }
}
```

出力

```
Current value of n is 1
Current value of n is 2
Current value of n is 3
Current value of n is 4
Current value of n is 5
```

```
// statements_while_2.cs
using System;
class WhileTest
{
    static void Main()
    {
        int n = 1;
        while (n++ < 6)
        {
            Console.WriteLine("Current value of n is {0}", n);
        }
    }
}
```

出力

```
Current value of n is 2
Current value of n is 3
Current value of n is 4
Current value of n is 5
Current value of n is 6
```

while 式が評価されてからループが実行されるので、**while** ループは 0 回以上実行されます。do ループは、これと異なり、1 回以上実行されます。

while ループは、break、goto、return、または throw ステートメントがループの外部に制御を移動すると終了できます。ループを終了せずに制御を次の繰り返しに移動させるには、continue ステートメントを使用します。上記の 3 つの例での出力は、`int n` がインクリメントされる位置によって異なる点に注意してください。次の例では出力は生成されません。

```
// statements_while_3.cs
// no output is generated
using System;
class WhileTest
{
    static void Main()
    {
```

```
            int n = 5;
            while (++n < 6)
            {
                Console.WriteLine("Current value of n is {0}", n);
            }
        }
    }
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.7 while ステートメント
- 8.8.1 while ステートメント

参照

関連項目

C# のキーワード

The while Statement

繰り返しステートメント (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

ジャンプ ステートメント (C# リファレンス)

分岐はジャンプ ステートメントで実行されます。ジャンプ ステートメントは、プログラムの制御をすぐに移動できます。ジャンプ ステートメントでは、次のキーワードを使用します。

- break
- continue
- goto
- return
- throw

参照

関連項目

C# のキーワード

ステートメントの種類 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

break (C# リファレンス)

break ステートメントは、これを囲むループまたは switch ステートメントのうち、最も内側のものを終了させます。終了したステートメントの次にステートメントがある場合は、そこに制御が移動します。

使用例

次の例には、条件付きステートメントに 1 から 100 までをカウントするカウンタがあります。ただし、**break** ステートメントによってループは 4 回で終了します。

```
// statements_break.cs
using System;
class BreakTest
{
    static void Main()
    {
        for (int i = 1; i <= 100; i++)
        {
            if (i == 5)
            {
                break;
            }
            Console.WriteLine(i);
        }
    }
}
```

出力

```
1
2
3
4
```

switch ステートメント内での **break** の使用例を次に示します。

```
// statements_break2.cs
// break and switch
using System;
class Switch
{
    static void Main()
    {
        Console.Write("Enter your selection (1, 2, or 3): ");
        string s = Console.ReadLine();
        int n = Int32.Parse(s);

        switch (n)
        {
            case 1:
                Console.WriteLine("Current value is {0}", 1);
                break;
            case 2:
                Console.WriteLine("Current value is {0}", 2);
                break;
            case 3:
                Console.WriteLine("Current value is {0}", 3);
                break;
            default:
                Console.WriteLine("Sorry, invalid selection.");
                break;
        }
    }
}
```

入力

```
1
```

サンプル出力

```
Enter your selection (1, 2, or 3): 1
Current value is 1
```

説明

4 を入力すると、出力は次のようになります。

```
Enter your selection (1, 2, or 3): 4
Sorry, invalid selection.
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.10 break ステートメント、continue ステートメント、および goto ステートメント
- 8.9.1 break ステートメント

参照

関連項目

C# のキーワード
The break Statement
switch (C# リファレンス)
ジャンプ ステートメント (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

continue (C# リファレンス)

continue ステートメントは、continue を含む繰り返しステートメントの次にある繰り返しに制御を移動します。

使用例

次の例では、カウンタが初期化され、1 から 10 までがカウントされます。式 `(i < 9)` と **continue** ステートメントを組み合わせることで、**continue** から for 本体の終わりまでのステートメントが読み飛ばされます。

```
// statements_continue.cs
using System;
class ContinueTest
{
    static void Main()
    {
        for (int i = 1; i <= 10; i++)
        {
            if (i < 9)
            {
                continue;
            }
            Console.WriteLine(i);
        }
    }
}
```

出力

```
9
10
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.10 break ステートメント、continue ステートメント、および goto ステートメント
- 8.9.2 continue ステートメント

参照

関連項目

C# のキーワード
The break Statement
ジャンプ ステートメント (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

goto (C# リファレンス)

goto ステートメントは、プログラムの制御をラベル付きステートメントに直接移動します。

解説

通常、**goto** は **switch** ステートメントの特定の switch-case ラベルまたは default ラベルに制御を移動するのに使用します。

goto ステートメントは、階層の深い入れ子のループから抜ける際にも便利です。

使用例

次の例は、switch ステートメントでの **goto** の使用方法を示しています。

```
// statements_goto_switch.cs
using System;
class SwitchTest
{
    static void Main()
    {
        Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
        Console.Write("Please enter your selection: ");
        string s = Console.ReadLine();
        int n = int.Parse(s);
        int cost = 0;
        switch (n)
        {
            case 1:
                cost += 25;
                break;
            case 2:
                cost += 25;
                goto case 1;
            case 3:
                cost += 50;
                goto case 1;
            default:
                Console.WriteLine("Invalid selection.");
                break;
        }
        if (cost != 0)
        {
            Console.WriteLine("Please insert {0} cents.", cost);
        }
        Console.WriteLine("Thank you for your business.");
    }
}
```

入力

```
2
```

サンプル出力

```
Coffee sizes: 1=Small 2=Medium 3=Large
Please enter your selection: 2
Please insert 50 cents.
Thank you for your business.
```

入れ子のループから **goto** で抜け出す例を次に示します。

```
// statements_goto.cs
// Nested search loops
using System;
public class GotoTest1
{
```

```
    static void Main()
    {
        int x = 200, y = 4;
        int count = 0;
        string[,] array = new string[x, y];

        // Initialize the array:
        for (int i = 0; i < x; i++)

            for (int j = 0; j < y; j++)
                array[i, j] = (++count).ToString();

        // Read input:
        Console.Write("Enter the number to search for: ");

        // Input a string:
        string myNumber = Console.ReadLine();

        // Search:
        for (int i = 0; i < x; i++)
        {
            for (int j = 0; j < y; j++)
            {
                if (array[i, j].Equals(myNumber))
                {
                    goto Found;
                }
            }
        }

        Console.WriteLine("The number {0} was not found.", myNumber);
        goto Finish;

    Found:
        Console.WriteLine("The number {0} is found.", myNumber);

    Finish:
        Console.WriteLine("End of search.");
    }
}
```

入力

```
44
```

サンプル出力

```
Enter the number to search for: 44
The number 44 is found.
End of search.
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.10 break ステートメント、continue ステートメント、および goto ステートメント
- 8.9.3 goto ステートメント

参照

関連項目

C# のキーワード

The goto Statement

ジャンプ ステートメント (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

return (C# リファレンス)

return ステートメントは、メソッドの実行を終了し、呼び出し側のメソッドに制御を戻します。省略可能な値を返すこともできます。メソッドの型が **void** 型の場合、**return** ステートメントは省略できます。

使用例

次の例では、メソッド `A()` が変数 `Area` を double 値として返します。

```
// statements_return.cs
using System;
class ReturnTest
{
    static double CalculateArea(int r)
    {
        double area = r * r * Math.PI;
        return area;
    }

    static void Main()
    {
        int radius = 5;
        Console.WriteLine("The area is {0:0.00}", CalculateArea(radius));
    }
}
```

出力

```
The area is 78.54
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.12 return ステートメント
- 8.9.4 return ステートメント

参照

関連項目

C# のキーワード
The return Statement
ジャンプ ステートメント (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

例外処理ステートメント (C# リファレンス)

C# は、プログラムの実行中に発生する例外という異常事態を処理する機構を組み込んでいます。このような例外は、通常の制御の流れの外で処理されます。

ここでは、例外処理に関する次のトピックについて説明します。

- throw
- try-catch
- try-finally
- try-catch-finally

参照

関連項目

C# のキーワード
ステートメントの種類 (C# リファレンス)
例外と例外処理 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

throw (C# リファレンス)

throw ステートメントは、プログラムの実行中に例外という異常事態が発生したことを通知するために使用します。

解説

スローされる例外は、クラスが System.Exception から派生したオブジェクトです。例を次に示します。

```
class MyException : System.Exception {}
// ...
throw new MyException();
```

通常、**throw** ステートメントは、try-catch または try-finally ステートメントで使用されます。例外がスローされると、プログラムは、この例外を処理する **catch** ステートメントを検索します。

また、**throw** ステートメントを使用して、キャッチした例外を再びスローすることもできます。詳細および例については、「try-catch」および「例外のスロー」を参照してください。

使用例

throw ステートメントで例外をスローする例を次に示します。

```
// throw example
using System;
public class ThrowTest
{
    static void Main()
    {
        string s = null;

        if (s == null)
        {
            throw new ArgumentNullException();
        }

        Console.Write("The string s is null"); // not executed
    }
}
```

出力

ArgumentNullException 例外が発生します。

コード例

try-catch、try-finally、および try-catch-finally の例を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.11 throw ステートメント
- 8.9.5 throw ステートメント

参照

処理手順

方法 : 例外を明示的にスローする

関連項目

The try, catch, and throw Statements
C# のキーワード
例外処理ステートメント (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

try-catch (C# リファレンス)

try-catch ステートメントは、**try** ブロックと、それに続く **catch** 句で構成されます。この句にはさまざまな例外のハンドラを指定します。

解説

try ブロックには、例外を発生させる可能性がある保護されたコードが含まれます。ブロックは、例外がスローされるか、ブロックが正常に終了するまで実行されます。たとえば、次の例では **null** オブジェクトをキャストしようとすると、NullReferenceException 例外が発生します。

```
object o2 = null;
try
{
    int i2 = (int)o2;   // Error
}
```

catch 句は、引数なしで使用できます。引数を指定せずに使用するとあらゆる例外がキャッチされ、汎用的な catch 句と見なされます。また、**System.Exception** から派生したオブジェクト引数を使用することもできます。その場合は、特定の例外が処理されます。次に例を示します。

```
catch (InvalidCastException e)
{
}
```

特定の **catch** 句は、同一の try-catch ステートメントで複数使用できます。この場合、**catch** 句は順序どおりにチェックされるため、**catch** 句の順序が重要になります。例外は、特定性の高い順にキャッチしてください。

throw ステートメントは、**catch** ステートメントでキャッチされた例外を再びスローするために **catch** ブロックで使用できます。次に例を示します。

```
catch (InvalidCastException e)
{
    throw (e);    // Rethrowing exception e
}
```

パラメータのない **catch** 句で処理された例外を再スローする場合、引数のない **throw** ステートメントを使用します。次に例を示します。

```
catch
{
    throw;
}
```

try ブロック内では、そこで宣言されている変数だけを初期化します。外部で宣言された変数を初期化すると、ブロックの実行が完了する前に例外が発生する可能性があるためです。たとえば、以下のコード例では、変数 `x` が **try** ブロック内で初期化されます。この変数を **try** ブロックの外側にある `Write(x)` ステートメントで使おうとすると、コンパイラ エラー "未割り当てのローカル変数 'var' が使用されました。" が発生します。

```
static void Main()
{
    int x;
    try
    {
        // Don't initialize this variable here.
        x = 123;
    }
    catch
    {
    }
    // Error: Use of unassigned local variable 'x'.
    Console.Write(x);
}
```

catch の詳細については、「try-catch-finally」を参照してください。

使用例

例外を発生させる可能性がある `MyMethod()` メソッドへの呼び出しを含む **try** ブロックの例を次に示します。**catch** 句には、メッセージを画面に表示するだけの例外ハンドラがあります。**throw** ステートメントが `MyMethod` の内側から呼び出されると、システムは **catch** ステートメントを検索し、メッセージ "`Exception caught`" を表示します。

```
// try_catch_example.cs
using System;
class MainClass
{
    static void ProcessString(string s)
    {
        if (s == null)
        {
            throw new ArgumentNullException();
        }
    }

    static void Main()
    {
        try
        {
            string s = null;
            ProcessString(s);
        }
        catch (Exception e)
        {
            Console.WriteLine("{0} Exception caught.", e);
        }
    }
}
```

サンプル出力

```
System.ArgumentNullException: Value cannot be null.
   at MainClass.Main() Exception caught.
```

2 つの catch ステートメントを使用する例を次に示します。最初にある、特定性の最も高い例外がキャッチされます。

```
// try_catch_ordering_catch_clauses.cs
using System;
class MainClass
{
    static void ProcessString(string s)
    {
        if (s == null)
        {
            throw new ArgumentNullException();
        }
    }

    static void Main()
    {
        try
        {
            string s = null;
            ProcessString(s);
        }
        // Most specific:
        catch (ArgumentNullException e)
        {
            Console.WriteLine("{0} First exception caught.", e);
        }
        // Least specific:
        catch (Exception e)
        {
            Console.WriteLine("{0} Second exception caught.", e);
```

```
            }
        }
    }
```

サンプル出力

```
System.ArgumentNullException: Value cannot be null.
   at MainClass.Main() First exception caught.
```

説明

前述の例で、特定性の最も低い catch 句から始めると、エラー メッセージ "`A previous catch clause already catches all exceptions of this or a super type ('System.Exception')`" が表示されます。

ただし、特定性の最も低い例外をキャッチするには、throw ステートメントを次のものと置き換えます。

```
throw new Exception();
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.13 try-catch ステートメント
- 8.10 try ステートメント
- 16 例外

参照

処理手順

方法 : 例外を明示的にスローする

関連項目

C# のキーワード
The try, catch, and throw Statements
例外処理ステートメント (C# リファレンス)
throw (C# リファレンス)
try-finally (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

try-finally (C# リファレンス)

finally ブロックは、try-catch (C# リファレンス) ブロックで割り当てられているリソースをクリーンアップする場合、および例外が発生しても実行する必要があるコードを実行する場合に便利です。制御は、try ブロックがどのように終了したかに関係なく、常に finally ブロックに移動します。

解説

catch がステートメント ブロックで発生した例外を処理するのに対して、**finally** はその前にある **try** ブロックがどのように終了したかに関係なくコードのステートメント ブロックを実行することを保証します。

使用例

例外を発生させる無効な変換ステートメントの例を次に示します。プログラムを実行するとランタイム エラー メッセージが表示されますが、**finally** 句はそのまま実行を続け、出力を表示します。

```
// try-finally
using System;
public class MainClass
{
    static void Main()
    {
        int i = 123;
        string s = "Some string";
        object o = s;

        try
        {
            // Invalid conversion; o contains a string not an int
            i = (int)o;
        }
        finally
        {
            Console.Write("i = {0}", i);
        }
    }
}
```

説明

上の例では、`System.InvalidCastException` がスローされます。

例外はキャッチされますが、**finally** ブロックに含まれる出力ステートメントはそのまま実行され、次のようになります。

```
i = 123
```

finally の詳細については、「try-catch-finally」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.14 try-finally ステートメント
- 8.11 try ステートメント
- 16 例外

参照

処理手順

方法 : 例外を明示的にスローする

関連項目

C# のキーワード

The try, catch, and throw Statements

例外処理ステートメント (C# リファレンス)

throw (C# リファレンス)

try-catch (C# リファレンス)

概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

try-catch-finally (C# リファレンス)

通常、**catch** および **finally** は、**try** ブロックのリソースを取得して使用する場合に、対で記述されます。**catch** ブロックで例外的な状況を処理し、**finally** ブロックでリソースを解放します。

例外の再スローの詳細および例については、「try-catch」および「例外のスロー」を参照してください。

使用例

```
// try_catch_finally.cs
using System;
public class EHClass
{
    static void Main()
    {
        try
        {
            Console.WriteLine("Executing the try statement.");
            throw new NullReferenceException();
        }
        catch (NullReferenceException e)
        {
            Console.WriteLine("{0} Caught exception #1.", e);
        }
        catch
        {
            Console.WriteLine("Caught exception #2.");
        }
        finally
        {
            Console.WriteLine("Executing finally block.");
        }
    }
}
```

サンプル出力

```
Executing the try statement.
System.NullReferenceException: Object reference not set to an instance of an object.
   at EHClass.Main() Caught exception #1.
Executing finally block.
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.15 try-catch-finally ステートメント
- 8.10 try ステートメント
- 16 例外

参照

処理手順

方法 : 例外を明示的にスローする

関連項目

C# のキーワード

The try, catch, and throw Statements

例外処理ステートメント (C# リファレンス)

throw (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

チェックありとチェックなし (C# リファレンス)

C# ステートメントは、checked または unchecked のいずれかのコンテキストで実行されます。checked コンテキストでは、算術オーバーフローの例外が発生します。unchecked コンテキストでは、算術オーバーフローは無視され、結果は切り捨てられます。

- checked　checked コンテキストを指定します。
- unchecked　unchecked コンテキストを指定します。

checked も **unchecked** も指定されない場合、既定のコンテキストはコンパイラ オプションなどの外部要因によって決まります。

次の演算は、オーバーフロー チェックの影響を受けます。

- 次の組み込み演算子を整数型に使用した式。

  ++　—　**- (単項)　**+　-　*　/

- 整数型の間での明示的な数値変換。

コンパイラ オプション /checked を使用すると、**checked** キーワードまたは **unchecked** キーワードのスコープに明示的には存在しないすべての整数算術ステートメントに対して、checked または unchecked のコンテキストを指定できます。

参照
関連項目
C# のキーワード
ステートメントの種類 (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

checked (C# リファレンス)

checked キーワードは、整数型の算術演算および変換に対してオーバーフロー チェックを明示的に有効にするために使用します。

解説

既定では、式が変換先の型の範囲外にある値を生成した場合、定数式はコンパイル時エラーを発生させ、非定数式は実行時に評価されて例外を発生させます。ただし、checked キーワードを使用すると、コンパイラ オプションまたは環境設定によりチェックがグローバルに抑制されている場合に、チェックを有効にできます。

unchecked キーワードの使用方法については、「unchecked (C# リファレンス)」の例を参照してください。

使用例

次の例では、非定数式に対する **checked** の使い方を示します。オーバーフローは実行時に報告されます。

```
// statements_checked.cs
using System;
class OverFlowTest
{
    static short x = 32767;   // Max short value
    static short y = 32767;

    // Using a checked expression
    static int CheckedMethod()
    {
        int z = 0;
        try
        {
            z = checked((short)(x + y));
        }
        catch (System.OverflowException e)
        {
            Console.WriteLine(e.ToString());
        }
        return z;
    }

    static void Main()
    {
        Console.WriteLine("Checked output value is: {0}",
                    CheckedMethod());
    }
}
```

サンプル出力

```
System.OverflowException: Arithmetic operation resulted in an overflow.
   at OverFlowTest.CheckedMethod()
Checked output value is: 0
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.2 ブロック ステートメント、checked ステートメント、および unchecked ステートメント
- 7.5.12 checked 演算子と unchecked 演算子
- 8.11 checked ステートメントと unchecked ステートメント

参照

関連項目

C# のキーワード
チェックありとチェックなし (C# リファレンス)
unchecked (C# リファレンス)

概念

unchecked (C# リファレンス)

unchecked キーワードは、整数型の算術演算および変換に対してオーバーフロー チェックを抑制するのに使用します。

解説

unchecked コンテキストでは、式がチェック先の型の範囲外にある値を生成した場合、結果の値は切り捨てられます。次に例を示します。

```
        unchecked
{
    int val = 2147483647 * 2;
}
```

上の計算は **unchecked** ブロックで実行されるため、結果が整数に対して大きすぎることは無視され、`val` に値 `-2` が代入されます。既定では、オーバーフローの検出は有効であり、これは checked を使用する場合と同じ効果があります。

上の例では、**unchecked** を省略した場合、式で定数が使用され、結果がコンパイル時に既知となってしまうため、コンパイル エラーが発生します。**unchecked** キーワードは、非定数の式に対するオーバーフロー検出も抑制します。この抑制がないと、実行時に OverflowException が発生します。

unchecked キーワードは、次のように、演算子としても使用できます。

```
public int UncheckedAdd(int a, int b)
{
    return unchecked(a + b);
}
```

使用例

この例では、定数式を指定した **unchecked** を使用して、**unchecked** ステートメントを使用する方法を示しています。

```
// statements_unchecked.cs
using System;

class TestClass
{
    const int x = 2147483647;   // Max int
    const int y = 2;

    static void Main()
    {
        int z;
        unchecked
        {
            z = x * y;
        }
        Console.WriteLine("Unchecked output value: {0}", z);
    }
}
```

出力

```
Unchecked output value: -2
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.2 ブロック ステートメント、checked ステートメント、および unchecked ステートメント
- 7.5.12 checked 演算子と unchecked 演算子
- 8.11 checked ステートメントと unchecked ステートメント

参照

関連項目

C# のキーワード

チェックありとチェックなし (C# リファレンス)

checked (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

fixed ステートメント (C# リファレンス)

fixed ステートメントは、移動可能な変数がガベージ コレクタにより再配置されることを防ぎます。**fixed** ステートメントは、unsafe コンテキストでのみ使用できます。**Fixed** を使用して、固定サイズ バッファを作成することもできます。

解説

fixed ステートメントは、マネージ変数へのポインタを設定し、*statement* の実行時にマネージ変数を "固定" します。**fixed** がない場合、ガベージ コレクションが予期できないかたちで移動可能なマネージ変数を再配置するため、マネージ変数へのポインタはほとんど役に立ちません。C# コンパイラの場合、**fixed** ステートメントでは、マネージ変数にポインタを割り当てることだけができます。

```
// assume class Point { public int x, y; }
// pt is a managed variable, subject to garbage collection.
Point pt = new Point();
// Using fixed allows the address of pt members to be
// taken, and "pins" pt so it isn't relocated.
fixed ( int* p = &pt.x )
{
    *p = 1;
}
```

ポインタは、配列または文字列のアドレスで初期化できます。

```
        fixed (int* p = arr) ...  // equivalent to p = &arr[0]
fixed (char* p = str) ... // equivalent to p = &str[0]
```

同じ型の場合は、複数のポインタを初期化できます。

```
fixed (byte* ps = srcarray, pd = dstarray) {...}
```

異なる型のポインタを初期化するには、**fixed** ステートメントを入れ子にします。

```
        fixed (int* p1 = &p.x)
{
    fixed (double* p2 = &array[5])
    {
        // Do something with p1 and p2.
    }
}
```

ステートメントのコードを実行すると、固定された変数の固定が解除され、ガベージ コレクションの対象になります。そのため、**fixed** ステートメントの外部にある変数へのポインタは指定しないでください。

メモ :
fixed ステートメントで初期化されたポインタは変更できません。

unsafe モードでは、スタックにメモリを割り当てることができます。スタックは、ガベージ コレクションの対象にはならないので、固定は必要ありません。詳細については、「stackalloc (C# リファレンス)」を参照してください。

使用例

```
// statements_fixed.cs
// compile with: /unsafe
using System;

class Point
{
    public int x, y;
```

```
}

class FixedTest
{
    // Unsafe method: takes a pointer to an int.
    unsafe static void SquarePtrParam (int* p)
    {
        *p *= *p;
    }

    unsafe static void Main()
    {
        Point pt = new Point();
        pt.x = 5;
        pt.y = 6;
        // Pin pt in place:
        fixed (int* p = &pt.x)
        {
            SquarePtrParam (p);
        }
        // pt now unpinned
        Console.WriteLine ("{0} {1}", pt.x, pt.y);
    }
}
```

出力

```
25 6
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 18.3 固定変数と移動可能変数
- 18.6 fixed ステートメント

参照

関連項目

C# のキーワード
unsafe (C# リファレンス)
固定サイズ バッファ (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

lock ステートメント (C# リファレンス)

lock キーワードは、指定のオブジェクトに対する相互排他ロックを取得し、ステートメントを実行し、ロックを解放するステートメント ブロックをクリティカル セクションとしてマークします。このステートメントの形式は、次のとおりです。

```
Object thisLock = new Object();
lock (thisLock)
{
    // Critical code section
}
```

詳細については、「スレッドの同期 (C# プログラミング ガイド)」を参照してください。

解説

lock によって、あるスレッドがクリティカル セクションになっているときは、別のスレッドはコードのクリティカル セクションにはなりません。ロックされたコードを別のスレッドが使おうとすると、オブジェクトが解放されるまで待機 (ブロック) します。

スレッドについては、「スレッド処理 (C# プログラミング ガイド)」で説明します。

lock は、ブロックの最初に Enter を呼び出し、ブロックの最後に Exit を呼び出します。

一般に、**public** 型またはコードの制御範囲外にあるインスタンスに対してロックしないようにしてください。`lock (this)`、`lock (typeof (MyType))`、および `lock ("myLock")` などの一般的な構造は、このガイドラインに反しています。

- インスタンスに対してパブリックにアクセスできる場合には、`lock (this)` が問題になります。
- `MyType` に対してパブリックにアクセスできる場合には、`lock (typeof (MyType))` が問題になります。
- プロセス内で同じ文字列を使用する他のコードは同じロックを共有するので、`lock("myLock")` が問題になります。

ロックする **private** オブジェクトを定義するか、すべてのインスタンスに共通するデータを保護するために **private static** オブジェクト変数を定義することをお勧めします。

使用例

C# でのスレッドの簡単な使用例を次に示します。

```
// statements_lock.cs
using System;
using System.Threading;

class ThreadTest
{
    public void RunMe()
    {
        Console.WriteLine("RunMe called");
    }

    static void Main()
    {
        ThreadTest b = new ThreadTest();
        Thread t = new Thread(b.RunMe);
        t.Start();
    }
}
```

出力

```
RunMe called
```

次の例では、スレッドおよび **lock** が使用されています。**lock** ステートメントが存在する限り、ステートメント ブロックはクリティカル セクションであり、`balance` は負の数にはなりません。

```
// statements_lock2.cs
```

```
using System;
using System.Threading;

class Account
{
    private Object thisLock = new Object();
    int balance;

    Random r = new Random();

    public Account(int initial)
    {
        balance = initial;
    }

    int Withdraw(int amount)
    {

        // This condition will never be true unless the lock statement
        // is commented out:
        if (balance < 0)
        {
            throw new Exception("Negative Balance");
        }

        // Comment out the next line to see the effect of leaving out
        // the lock keyword:
        lock(thisLock)
        {
            if (balance >= amount)
            {
                Console.WriteLine("Balance before Withdrawal :  " + balance);
                Console.WriteLine("Amount to Withdraw        : -" + amount);
                balance = balance - amount;
                Console.WriteLine("Balance after Withdrawal  :  " + balance);
                return amount;
            }
            else
            {
                return 0; // transaction rejected
            }
        }
    }

    public void DoTransactions()
    {
        for (int i = 0; i < 100; i++)
        {
            Withdraw(r.Next(1, 100));
        }
    }
}

class Test
{
    static void Main()
    {
        Thread[] threads = new Thread[10];
        Account acc = new Account(1000);
        for (int i = 0; i < 10; i++)
        {
            Thread t = new Thread(new ThreadStart(acc.DoTransactions));
            threads[i] = t;
        }
        for (int i = 0; i < 10; i++)
        {
            threads[i].Start();
        }
```

```
        }
    }
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.18 lock ステートメント
- 8.12 lock ステートメント

参照

処理手順

Monitorによる同期化の技術サンプル

待機による同期化の技術サンプル

関連項目

スレッド処理 (C# プログラミング ガイド)

C# のキーワード

ステートメントの種類 (C# リファレンス)

MethodImplAttributes Enumeration

スレッドの同期 (C# プログラミング ガイド)

Mutex

概念

C# プログラミング ガイド

Monitor

インタロックされた操作

AutoResetEvent

その他の技術情報

C# リファレンス

メソッドのパラメータ (C# リファレンス)

ref または out を指定せずにメソッドのパラメータを宣言した場合、このパラメータには値を関連付けることができます。この値はメソッド内で変更できますが、変更された値は、制御が呼び出しプロシージャに戻されたときには保持されません。メソッド パラメータのキーワードを使用すると、この動作を変更できます。

ここでは、メソッド パラメータの宣言に使用できるキーワードについて説明します。

- params
- ref
- out

参照
関連項目
C# のキーワード
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

params (C# リファレンス)

params キーワードを使用して、可変個引数リストを引数にとるメソッド パラメータを指定できます。

1 つのメソッド宣言内では、**params** キーワード以後にパラメータを追加できないため、1 つの **params** キーワードだけを使用できます。

使用例

```
// cs_params.cs
using System;
public class MyClass
{

    public static void UseParams(params int[] list)
    {
        for (int i = 0 ; i < list.Length; i++)
        {
            Console.WriteLine(list[i]);
        }
        Console.WriteLine();
    }

    public static void UseParams2(params object[] list)
    {
        for (int i = 0 ; i < list.Length; i++)
        {
            Console.WriteLine(list[i]);
        }
        Console.WriteLine();
    }

    static void Main()
    {
        UseParams(1, 2, 3);
        UseParams2(1, 'a', "test");

        // An array of objects can also be passed, as long as
        // the array type matches the method being called.
        int[] myarray = new int[3] {10,11,12};
        UseParams(myarray);
    }
}
```

出力

```
1
2
3

1
a
test

10
11
12
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 10.5.1.4 パラメータ配列

参照

関連項目

C# のキーワード

メソッドのパラメータ (C# リファレンス)

概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

ref (C# リファレンス)

ref キーワードを使用すると、引数が参照渡しされます。その結果、メソッドでパラメータに加えられた変更は、制御が呼び出し元のメソッドに戻された時点で変数に反映されます。**ref** パラメータを使用するには、メソッド定義と呼び出し元のメソッドの両方で **ref** キーワードを明示的に使用する必要があります。次に例を示します。

```
class RefExample
{
    static void Method(ref int i)
    {
        i = 44;
    }
    static void Main()
    {
        int val = 0;
        Method(ref val);
        // val is now 44
    }
}
```

ref パラメータに渡される引数は、事前に初期化されている必要があります。これは out とは異なります。out の引数は、渡される前に明示的に初期化される必要はありません。「out (C# リファレンス)」を参照してください。

ref と **out** は、実行時の取り扱いは異なりますが、コンパイル時の取り扱いは同じです。そのため、2 つのメソッドのうち一方が **ref** 引数を受け取り、もう一方が out 引数を受け取る場合、これらのメソッドはオーバーロードできません。これら 2 つのメソッドは、たとえば、コンパイルの観点からは同じメソッドになるので、次のコードはコンパイルされません。

```
class CS0663_Example
{
    // compiler error CS0663: "cannot define overloaded
    // methods that differ only on ref and out"
    public void SampleMethod(ref int i) {  }
    public void SampleMethod(out int i) {  }
}
```

ただし、一方のメソッドが **ref** 引数または out 引数を受け取り、もう一方のメソッドがどちらの引数も使用しない場合は、オーバーロードを実行できます。この例を次に示します。

```
class RefOutOverloadExample
{
    public void SampleMethod(int i) {  }
    public void SampleMethod(ref int i) {  }
}
```

解説

プロパティは変数ではないので、**ref** パラメータとして渡すことができません。

配列を渡す方法については、「ref と out を使用した配列の受け渡し」を参照してください。

使用例

値型の参照渡しは、上記のように便利ですが、参照型を渡す場合は **ref** も役立ちます。この方法では、参照自体が参照渡しされるため、呼び出されたメソッドは、参照が参照しているオブジェクトを変更できます。次の例は、参照型を **ref** パラメータとして渡すときにオブジェクト自体を変更できることを示します。

```
class RefRefExample
{
    static void Method(ref string s)
    {
        s = "changed";
    }
    static void Main()
```

```
    {
        string str = "original";
        Method(ref str);
        // str is now "changed"
    }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.1.5 参照パラメータ
- 10.5.1.2 参照パラメータ

参照

関連項目

パラメータの引き渡し (C# プログラミング ガイド)

メソッドのパラメータ (C# リファレンス)

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

out (C# リファレンス)

out キーワードを使用すると、引数が参照渡しされます。このキーワードは ref キーワードに似ていますが、ref の場合は、変数を初期化してから渡す必要があります。**out** パラメータを使用するには、メソッド定義と呼び出し元のメソッドの両方で **out** キーワードを明示的に使用する必要があります。次に例を示します。

```
class OutExample
{
    static void Method(out int i)
    {
        i = 44;
    }
    static void Main()
    {
        int value;
        Method(out value);
        // value is now 44
    }
}
```

out 引数として渡す変数は、渡す前に初期化する必要がありませんが、呼び出し元のメソッドでは、メソッドから制御が戻る前に値を代入する必要があります。

ref キーワードと **out** キーワードの取り扱い方は、実行時は異なりますが、コンパイル時は同じです。そのため、2 つのメソッドのうち一方が **ref** 引数を受け取り、もう一方が **out** 引数を受け取る場合、これらのメソッドはオーバーロードできません。たとえば、これら 2 つのメソッドは、コンパイルの観点からは同じメソッドになるので、次のコードはコンパイルされません。

```
class CS0663_Example
{
    // compiler error CS0663: "cannot define overloaded
    // methods that differ only on ref and out"
    public void SampleMethod(out int i) {  }
    public void SampleMethod(ref int i) {  }
}
```

ただし、一方のメソッドが **ref** 引数または **out** 引数を受け取り、もう一方のメソッドがどちらの引数も使用しない場合は、オーバーロードを実行できます。この例を次に示します。

```
class RefOutOverloadExample
{
    public void SampleMethod(int i) {  }
    public void SampleMethod(out int i) {  }
}
```

解説

プロパティは変数ではないので、**out** パラメータとして渡すことができません。

配列を渡す方法については、「ref と out を使用した配列の受け渡し」を参照してください。

使用例

out メソッドを宣言すると、メソッドが複数の値を返すようにする場合に便利です。**out** パラメータを使用するメソッドは、戻り値の型 (「return (C# リファレンス)」を参照) として変数を使用できますが、呼び出し元のメソッドに 1 つ以上のオブジェクトを **out** パラメータとして返すこともできます。**out** を使用して、1 回のメソッド呼び出しで 3 つの変数を返す例を次に示します。この例では、3 番目の引数に null が代入されます。これにより、メソッドは値をオプションで返すことができます。

```
class OutReturnExample
{
    static void Method(out int i, out string s1, out string s2)
    {
        i = 44;
        s1 = "I've been returned";
```

```
            s2 = null;
        }
        static void Main()
        {
            int value;
            string str1, str2;
            Method(out value, out str1, out str2);
            // value is now 44
            // str1 is now "I've been returned"
            // str2 is (still) null;
        }
    }
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.1.6 出力パラメータ
- 10.5.1.3 出力パラメータ

参照

関連項目

C# のキーワード

メソッドのパラメータ (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

名前空間キーワード (C# リファレンス)

ここでは、名前空間の使用に関連するキーワードと演算子について説明します。

- namespace
- using
- . 演算子
- :: 演算子
- extern エイリアス

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

名前空間 (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

namespace (C# リファレンス)

namespace キーワードは、スコープの宣言に使用します。名前空間のスコープを宣言すると、コードを組織化したり、グローバルに一意な型を作成したりできます。

```
namespace SampleNamespace
{
    class SampleClass{}
    interface SampleInterface{}
    struct SampleStruct{}
    enum SampleEnum{a,b}
    delegate void SampleDelegate(int i);
    namespace SampleNamespace.Nested
    {
       class SampleClass2{}
    }
}
```

解説

名前空間内では、以下の型を 1 つ以上宣言できます。

- 別の名前空間
- class
- interface
- struct
- enum
- delegate

C# ソース ファイル内に名前空間を明示的に宣言しているかどうかに関係なく、コンパイラは既定の名前空間を追加します。作成される無名の名前空間は、グローバル名前空間とも呼ばれ、すべてのファイルに存在します。グローバル名前空間内にある識別子は、名前付き名前空間で利用できます。

名前空間は、暗黙的にパブリックにアクセスされます。この属性は変更できません。名前空間内の要素に割り当てることができるアクセス修飾子については、「アクセス修飾子 (C# リファレンス)」を参照してください。

名前空間は、2 つ以上の宣言で定義できます。たとえば、次の例では、`MyCompany` 名前空間の一部として 2 つのクラスを定義しています。

```
// cs_namespace_keyword.cs
// compile with: /target:library
namespace MyCompany.Proj1
{
    class MyClass
    {
    }
}

namespace MyCompany.Proj1
{
    class MyClass1
    {
    }
}
```

使用例

入れ子になった名前空間で静的なメソッドを呼び出す方法の例を次に示します。

```
// cs_namespace_keyword_2.cs
using System;
```

```
namespace SomeNameSpace
{
    public class MyClass
    {
        static void Main()
        {
            Nested.NestedNameSpaceClass.SayHello();
        }
    }

    // a nested namespace
    namespace Nested
    {
        public class NestedNameSpaceClass
        {
            public static void SayHello()
            {
                Console.WriteLine("Hello");
            }
        }
    }
}
```

出力

```
Hello
```

参照項目

名前空間の使い方の詳細については、次のトピックを参照してください。

- 名前空間 (C# プログラミング ガイド)
- 名前空間の使用 (C# プログラミング ガイド)
- 方法 : 名前空間エイリアス修飾子を使用する (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 3.4.1 名前空間のメンバ
- 3.8 名前空間と型の名前
- 9 名前空間

参照

関連項目

C# のキーワード
名前空間キーワード (C# リファレンス)
using (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

using (C# リファレンス)

using キーワードは、主に次の 2 つの場合に使用します。

- 名前空間のエイリアスを作成する場合、または他の名前空間で定義されている型をインポートする場合にディレクティブとして使用する。「using ディレクティブ (C# リファレンス)」を参照してください。
- 最後にオブジェクトが破棄されるスコープを定義する場合にステートメントとして使用する。「using ステートメント (C# リファレンス)」を参照してください。

参照

関連項目

C# のキーワード
名前空間キーワード (C# リファレンス)
extern (C# リファレンス)

概念

C# プログラミング ガイド
名前空間 (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

using ディレクティブ (C# リファレンス)

using ディレクティブは、次の 2 つの場合に使用します。

- 名前空間で型の使用を許可する場合。これにより、その名前空間内では、型を修飾しないで使用できます。

```
using System.Text;
```

- 名前空間または型のエイリアスを作成する場合。

```
using Project = PC.MyCompany.Project;
```

using キーワードは、using ステートメント (オブジェクトがいつ破棄されるかを定義するステートメント) の作成にも使用します。詳細については、「using ステートメント (C# リファレンス)」を参照してください。

解説

using ディレクティブのスコープは、ディレクティブが記述されているファイルに限定されます。

using エイリアスを定義すると、名前空間または型の識別子を簡単に修飾できます。

using ディレクティブを定義すると、名前空間を指定しないで、名前空間で型を使用できます。**using** ディレクティブでは、指定した名前空間内に入れ子になった別の名前空間へのアクセスは許可されません。

名前空間には、ユーザー定義の名前空間とシステム定義の名前空間の 2 種類があります。ユーザー定義の名前空間とは、ユーザーが記述するコードで定義される名前空間です。システム定義の名前空間の一覧については、「.NET Framework クラス ライブラリ リファレンス」を参照してください。

他のアセンブリのメソッドの参照例については、「C# の DLL の作成と使用」を参照してください。

例 1

名前空間の **using** エイリアスを定義して使用する例を次に示します。

```
namespace PC
{
    // Define an alias for the nested namespace.
    using Project = PC.MyCompany.Project;
    class A
    {
        void M()
        {
            // Use the alias
            Project.MyClass mc = new Project.MyClass();
        }
    }
    namespace MyCompany
    {
        namespace Project
        {
            public class MyClass{}
        }
    }
}
```

例 2

クラスの **using** ディレクティブと **using** エイリアスを定義する例を次に示します。

```
// cs_using_directive2.cs
// Using directive.
using System;
// Using alias for a class.
using AliasToMyClass = NameSpace1.MyClass;
```

```
namespace NameSpace1
{
    public class MyClass
    {
        public override string ToString()
        {
            return "You are in NameSpace1.MyClass";
        }
    }
}

namespace NameSpace2
{
    class MyClass
    {
    }
}

namespace NameSpace3
{
    // Using directive:
    using NameSpace1;
    // Using directive:
    using NameSpace2;

    class MainClass
    {
        static void Main()
        {
            AliasToMyClass somevar = new AliasToMyClass();
            Console.WriteLine(somevar);
        }
    }
}
```

出力

```
You are in NameSpace1.MyClass
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 9.3 using ディレクティブ

参照

関連項目

C# のキーワード
名前空間キーワード (C# リファレンス)
using ステートメント (C# リファレンス)

概念

C# プログラミング ガイド
名前空間 (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

using ステートメント (C# リファレンス)

スコープを定義します。スコープの外部ではオブジェクトが破棄されます。

構文

```
        using (Font font1 = new Font("Arial", 10.0f))
{
}
```

解説

C# では、.NET Framework 共通言語ランタイム (CLR) によって、不要になったオブジェクトの格納用のメモリが自動的に解放されます。メモリは常に解放されるわけではありません。CLR がガベージ コレクションの実行を決定したときにメモリが解放されます。ただし、ファイル ハンドルやネットワーク接続などの制限されたリソースは、通常、できるだけ速やかに解放する必要があります。

using ステートメントを使用すると、リソースを使用するオブジェクトがいつリソースを解放するかを指定できます。**using** ステートメントに設定されたオブジェクトは、IDisposable インターフェイスを実装する必要があります。このインターフェイスは、オブジェクトのリソースを解放する Dispose メソッドを提供します。

using ステートメントは、**using** ステートメントの末尾に到達したり、例外がスローされて、ステートメントの末尾に到達する前に制御がステートメント ブロックを離れたりしたときに終了できます。

オブジェクトは、上の例のように **using** ステートメントの中で宣言できますが、次のように using ステートメントの前で宣言することもできます。

```
Font font2 = new Font("Arial", 10.0f);
using (font2)
{
    // use font2
}
```

using ステートメントでは、複数のオブジェクトを使用できますが、その場合は、次のように **using** ステートメントの中で宣言する必要があります。

```
        using (Font font3 = new Font("Arial", 10.0f),
           font4 = new Font("Arial", 10.0f))
{
    // Use font3 and font4.
}
```

使用例

ユーザー定義クラスがクラス自体の破棄動作を実装する方法を、次の例に示します。使用する型は **IDisposable** から継承した型でなければならない点に注意してください。

```
using System;

class C : IDisposable
{
    public void UseLimitedResource()
    {
        Console.WriteLine("Using limited resource...");
    }

    void IDisposable.Dispose()
    {
        Console.WriteLine("Disposing limited resource.");
    }
}

class Program
{
```

```
        static void Main()
        {
            using (C c = new C())
            {
                c.UseLimitedResource();
            }
            Console.WriteLine("Now outside using statement.");
            Console.ReadLine();
        }
    }
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 5.3.3.17 using ステートメント
- 8.13 using ステートメント

参照

関連項目

C# のキーワード
extern (C# リファレンス)
名前空間キーワード (C# リファレンス)
using ディレクティブ (C# リファレンス)
アンマネージ リソースをクリーンアップするための Finalize および Dispose の実装

概念

C# プログラミング ガイド
名前空間 (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

extern エイリアス (C# リファレンス)

場合によっては、同じ完全修飾型名を持つアセンブリの 2 つのバージョンを参照する必要があります。たとえば、1 つのアプリケーションでアセンブリの複数のバージョンを使用する必要がある場合などが挙げられます。外部アセンブリのエイリアスを使用すると、エイリアスで指定されているルートレベルの名前空間内に各アセンブリの名前空間をラップできます。これにより、これらの名前空間を 1 つのファイルで使用できます。

メモ：
extern キーワードは、メソッドがアンマネージ コードで作成されていることを宣言する場合に、メソッドの修飾子として使用されます。

同じ完全修飾型名を持つ 2 つのアセンブリを参照するには、コマンド ラインで次のようにエイリアスを指定します。

```
/r:GridV1=grid.dll
```

```
/r:GridV2=grid20.dll
```

外部エイリアス `GridV1` と `GridV2` が作成されます。プログラム内からこれらのエイリアスを使用するには、**extern** キーワードを使用してエイリアスを参照します。次に例を示します。

```
extern alias GridV1;
```

```
extern alias GridV2;
```

それぞれの extern エイリアス宣言により追加されるルートレベルの名前空間は、グローバル名前空間と並列になりますが、グローバル名前空間内には位置しません。したがって、各アセンブリの型を参照するときに、適切な名前空間エイリアスをルートとする型の完全修飾名を使用することで、あいまいさのない状態で参照できます。

上記の例では、`GridV1::Grid` は `grid.dll` のグリッド コントロールであり、`GridV2::Grid` は `grid20.dll` のグリッド コントロールです。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 25.4 extern エイリアス

参照

関連項目

C# のキーワード
名前空間キーワード (C# リファレンス)
:: 演算子 (C# リファレンス)
/reference (メタデータのインポート) (C# コンパイラ オプション)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

演算子キーワード (C# リファレンス)

オブジェクトの作成、オブジェクトのランタイム型の検証、型のサイズの取得など、その他の操作を実行するために使用します。ここでは、以下のキーワードについて説明します。

- as　オブジェクトの型を互換性のある型に変換します。
- is　オブジェクトのランタイム型を検証します。
- new
    - new 演算子 (C# リファレンス)　オブジェクトを作成します。
    - new 修飾子　継承されたメンバを隠ぺいします。
    - new 制約 (C# リファレンス)　型パラメータを修飾します。
- sizeof　型のサイズを取得します。
- typeof　型の **System.Type** オブジェクトを取得します。
- true
    - true 演算子 (C# リファレンス)　true を示す場合にはブール値 true を返し、それ以外の場合は false を返します。
    - true リテラル (C# リファレンス)　ブール値 true を表します。
- false
    - false 演算子 (C# リファレンス)　false を示す場合にはブール値 true を返し、それ以外の場合は false を返します。
    - false リテラル (C# リファレンス)　ブール値 false を表します。
- stackalloc　スタックにメモリ ブロックを割り当てます。

次のキーワードは、演算子やステートメントとして使用できます。それぞれの詳細については、「ステートメント」を参照してください。

- checked　checked コンテキストを指定します。
- unchecked　unchecked コンテキストを指定します。

参照
関連項目
C# のキーワード
C# の演算子
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

as (C# リファレンス)

互換性のある参照型の間での変換を実行するために使用されます。次に例を示します。

```
string s = someObject as string;
if (s != null)
{
    // someObject is a string.
}
```

解説

as 演算子はキャスト操作とよく似ています。ただし、変換可能でない場合、**as** は例外を発行する代わりに **null** を返します。次に式の形式を示します。

```
expression as type
```

これは、次と同じ意味になります。

```
expression is type ? (type)expression : (type)null
```

ただし `expression` は一度しか評価されません。

as 演算子は参照変換とボックス化変換だけを実行します。**as** 演算子は、ユーザー定義変換などの他の変換を実行できません。ユーザー定義変換は、キャスト式を使用して実行します。

使用例

```
// cs_keyword_as.cs
// The as operator.
using System;
class Class1
{
}

class Class2
{
}

class MainClass
{
    static void Main()
    {
        object[] objArray = new object[6];
        objArray[0] = new Class1();
        objArray[1] = new Class2();
        objArray[2] = "hello";
        objArray[3] = 123;
        objArray[4] = 123.4;
        objArray[5] = null;

        for (int i = 0; i < objArray.Length; ++i)
        {
            string s = objArray[i] as string;
            Console.Write("{0}:", i);
            if (s != null)
            {
                Console.WriteLine("'" + s + "'");
            }
            else
            {
                Console.WriteLine("not a string");
            }
```

```
            }
        }
    }
```

出力

```
0:not a string
1:not a string
2:'hello'
3:not a string
4:not a string
5:not a string
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 6 変換
- 7.9.10 as 演算子

参照

関連項目

C# のキーワード
is (C# リファレンス)
?: 演算子 (C# リファレンス)
演算子キーワード (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

is (C# リファレンス)

オブジェクトと、指定した型との間に互換性があるかどうかをチェックします。たとえば、オブジェクトが **string** 型と互換性かあるかどうかは次のようにして判断できます。

```
if (obj is string)
{
}
```

解説

is 式は、指定した式が null 以外であり、指定したオブジェクトを指定した型に例外がスローされることなくキャストできる場合に、**true** と評価されます。詳細については、「7.6.6 キャスト式」を参照してください。

式が常に **true** または **false** であることがわかっている場合に **is** キーワードを使用すると、コンパイル時に警告が出力されますが、通常は、実行時の型の互換性が評価されます。

is 演算子はオーバーロードできません。

is 演算子では、参照変換、ボックス化変換、またはボックス化解除変換だけが考慮されます。ユーザー定義変換など、他の変換は考慮されません。

使用例

```
// cs_keyword_is.cs
// The is operator.
using System;
class Class1
{
}
class Class2
{
}

class IsTest
{
    static void Test(object o)
    {
        Class1 a;
        Class2 b;

        if (o is Class1)
        {
            Console.WriteLine("o is Class1");
            a = (Class1)o;
            // Do something with "a."
        }
        else if (o is Class2)
        {
            Console.WriteLine("o is Class2");
            b = (Class2)o;
            // Do something with "b."
        }
        else
        {
            Console.WriteLine("o is neither Class1 nor Class2.");
        }
    }
    static void Main()
    {
        Class1 c1 = new Class1();
        Class2 c2 = new Class2();
        Test(c1);
        Test(c2);
        Test("a string");
    }
```

```
    }
```

出力

```
o is Class1
o is Class2
o is neither Class1 nor Class2.
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 7.9.9 is 演算子

参照
関連項目
C# のキーワード
typeof (C# リファレンス)
as (C# リファレンス)
演算子キーワード (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

new (C# リファレンス)

C# では、**new** キーワードを演算子、修飾子、または制約として使用できます。

new 演算子

オブジェクトを作成し、コンストラクタを呼び出します。

new 修飾子

継承されたメンバを基本クラスのメンバから隠ぺいします。

new 制約

ジェネリック宣言の型パラメータの引数として使用できる型を制限します。

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

new 演算子 (C# リファレンス)

オブジェクトを作成し、コンストラクタを呼び出します。次に例を示します。

```
Class1 o  = new Class1();
```

new 演算子を使用して、値型の既定コンストラクタを呼び出すこともできます。次に例を示します。

```
int i = new int();
```

このステートメントでは、`i` は **int** 型の既定値 0 に初期化されます。このステートメントは次のステートメントと同じ効果があります。

```
int i = 0;
```

既定値の一覧については、「既定値の一覧表」を参照してください。

構造体の既定のコンストラクタを宣言すると、エラーが発生します。これは、すべての値型には、暗黙的にパブリックな既定のコンストラクタがあるためです。パラメータ付きのコンストラクタを struct 型で宣言してその初期値を設定することは可能ですが、この手順が必要になるのは、既定以外の値が必要な場合のみです。

構造体のような値型オブジェクトはスタック領域に作成され、クラスのような参照型オブジェクトはヒープ領域に作成されます。どちらの型のオブジェクトも自動的に破棄されますが、値型に基づくオブジェクトは、スコープの適用範囲の外になると破棄され、参照型に基づくオブジェクトは、そのオブジェクトへの最後の参照が削除された後に随時破棄されます。大量のメモリ、ファイル ハンドル、ネットワーク接続などの固定リソースを消費する参照型では、オブジェクトができるだけすぐに破棄されるように、確定的な終了処理を採用することが望ましい場合があります。詳細については、「using ステートメント (C# リファレンス)」を参照してください。

new 演算子はオーバーロードできません。

new 演算子によるメモリ割り当てが失敗した場合は、OutOfMemoryException 例外がスローされます。

使用例

次の例では、**struct** オブジェクトとクラス オブジェクトは、**new** 演算子で作成および初期化されてから、値が代入されます。既定値と代入値が表示されます。

```
// cs_operator_new.cs
// The new operator.
using System;
struct SampleStruct
{
   public int x;
   public int y;

   public SampleStruct(int x, int y)
   {
      this.x = x;
      this.y = y;
   }
}

class SampleClass
{
   public string name;
   public int id;

   public SampleClass() {}

   public SampleClass(int id, string name)
   {
      this.id = id;
      this.name = name;
   }
}
```

```
class MainClass
{
   static void Main()
   {
      // Create objects using default constructors:
      SampleStruct Location1 = new SampleStruct();
      SampleClass Employee1 = new SampleClass();

      // Display values:
      Console.WriteLine("Default values:");
      Console.WriteLine("   Struct members: {0}, {1}",
            Location1.x, Location1.y);
      Console.WriteLine("   Class members: {0}, {1}",
            Employee1.name, Employee1.id);

      // Create objects using parameterized constructors:
      SampleStruct Location2 = new SampleStruct(10, 20);
      SampleClass Employee2 = new SampleClass(1234, "John Martin Smith");

      // Display values:
      Console.WriteLine("Assigned values:");
      Console.WriteLine("   Struct members: {0}, {1}",
            Location2.x, Location2.y);
      Console.WriteLine("   Class members: {0}, {1}",
            Employee2.name, Employee2.id);
   }
}
```

出力

```
Default values:
   Struct members: 0, 0
   Class members: , 0
Assigned values:
   Struct members: 10, 20
   Class members: John Martin Smith, 1234
```

説明

この例では、文字列の既定値が **null** です。このため、文字列の既定値は出力されません。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 7.5.10 new 演算子

参照
関連項目
C# のキーワード
演算子キーワード (C# リファレンス)
new (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

new 修飾子 (C# リファレンス)

new キーワードを修飾子として使用すると、基本クラスから継承されたメンバを明示的に隠ぺいできます。継承されたメンバの隠ぺいとは、派生バージョンのメンバで基本クラスのバージョンを置換することです。**new** 修飾子を使用せずにメンバを隠ぺいすることは可能ですが、警告が生成されます。メンバを明示的に隠ぺいするために **new** を使用することで、この警告を抑制し、また、この派生バージョンは置換目的で宣言されている、という事実の記録にもなります。

継承されたメンバを隠ぺいするには、同じ名前を使用して派生クラスでもメンバを宣言してから、**new** 修飾子で修飾します。次に例を示します。

```
public class BaseC
{
    public int x;
    public void Invoke() {}
}
public class DerivedC : BaseC
{
    new public void Invoke() {}
}
```

この例では、`BaseC.Invoke` は `DerivedC.Invoke` で隠ぺいされます。`x` フィールドは、似た名前によって隠ぺいされないため、影響を受けません。

継承による名前の隠ぺいは、次のいずれかの形式で行われます。

- 定数、フィールド、プロパティ、型をクラスまたは構造体で使用すると、同じ名前を持つすべての基本クラス メンバが隠ぺいされます。
- メソッドをクラスまたは構造体で使用すると、基本クラスで同じ名前を持つプロパティ、フィールド、型が隠ぺいされます。また、同じシグネチャを持つすべての基本クラス メソッドも隠ぺいされます。
- インデクサをクラスまたは構造体で使用すると、同じシグネチャを持つすべての基本クラス インデクサが隠ぺいされます。

new と override には相反する意味があるため、同じメンバにこの 2 つの修飾子を使用するとエラーになります。**new** を使用すると、同じ名前で新しいメンバが作成され、元のメンバが隠ぺいされます。**override** を使用すると、継承されたメンバの実装が拡張されます。

宣言で、継承されたメンバを隠ぺいしない **new** 修飾子を使用すると、警告が出力されます。

使用例

この例では、基本クラス `BaseC` と派生クラス `DerivedC` が同じフィールド名 `x` を使用するため、継承されるフィールドの値が隠ぺいされます。この例では、**new** 修飾子の使い方を示します。また、基本クラスの隠ぺいされたメンバに完全修飾名を使ってアクセスする方法も示します。

```
// cs_modifier_new.cs
// The new modifier.
using System;
public class BaseC
{
    public static int x = 55;
    public static int y = 22;
}

public class DerivedC : BaseC
{
    // Hide field 'x'
    new public static int x = 100;

    static void Main()
    {
        // Display the new value of x:
        Console.WriteLine(x);
        // Display the hidden value of x:
        Console.WriteLine(BaseC.x);
        // Display the unhidden member y:
        Console.WriteLine(y);
    }
```

```
}
```

出力

```
100
55
22
```

この例では、入れ子になったクラスが、基本クラスにある同名のクラスを隠ぺいします。この例では、**new** 修飾子を使って警告メッセージが表示されないようにする方法に加えて、完全修飾名を使用してクラスの隠ぺいされたメンバにアクセスする方法も示します。

```
// cs_modifer_new_nested.cs
// Using the new modifier with nested types.
using System;
public class BaseC
{
    public class NestedC
    {
        public int x = 200;
        public int y;
    }
}

public class DerivedC : BaseC
{
    // Nested type hiding the base type members.
    new public class NestedC
    {
        public int x = 100;
        public int y;
        public int z;
    }

    static void Main()
    {
        // Creating an object from the overlapping class:
        NestedC c1  = new NestedC();

        // Creating an object from the hidden class:
        BaseC.NestedC c2 = new BaseC.NestedC();

        Console.WriteLine(c1.x);
        Console.WriteLine(c2.x);
    }
}
```

出力

```
100
200
```

説明

new 修飾子を削除すると、プログラムのコンパイルおよび実行は行われますが、次の警告が出力されます。

```
The keyword new is required on 'MyDerivedC.x' because it hides inherited member 'MyBaseC.x'
.
```

また、入れ子にされた型が他の型を隠ぺいしている場合は、次の例で示されているように、入れ子にされた型を **new** 修飾子で修飾することもできます。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 10.2.2 new 修飾子

参照

関連項目

C# のキーワード

演算子キーワード (C# リファレンス)

修飾子 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

new 制約 (C# リファレンス)

new 制約は、ジェネリック クラス宣言内のすべての型引数にパブリックなパラメータなしのコンストラクタが必要であることを示します。この制約は、次の例に示すように、ジェネリック クラスで型の新しいインスタンスを作成する場合に型パラメータに適用されます。

```
class ItemFactory<T> where T : new()
{
    public T GetNewItem()
    {
        return new T();
    }
}
```

new() 制約を別の制約と併用する場合、この制約を最後に指定する必要があります。

```
using System;
public class ItemFactory<T>
    where T : IComparable, new()
{
}
```

詳細については、「型パラメータの制約 (C# プログラミング ガイド)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 19.1.4 制約

参照

関連項目

C# のキーワード
演算子キーワード (C# リファレンス)
System.Collections.Generic

概念

C# プログラミング ガイド
ジェネリック (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

sizeof (C# リファレンス)

値型 (C# リファレンス)のサイズ (バイト単位) を取得します。たとえば、**int** 型のサイズを次のように取得できます。

```
int intSize = sizeof(int);
```

解説

sizeof 演算子を適用できるのは値型だけです。参照型には適用できません。

メモ：
C# バージョン 2.0 以降、定義済みの型に **sizeof** を適用する場合に、unsafe (C# リファレンス) モードを使用する必要はありません。

sizeof 演算子はオーバーロードできません。**sizeof** 演算子により返される値の型は **int** です。次の表に、定義済みの型のサイズを表す定数値を一覧します。

式	結果
sizeof(sbyte)	1
sizeof(byte)	1
sizeof(short)	2
sizeof(ushort)	2
sizeof(int)	4
sizeof(uint)	4
sizeof(long)	8
sizeof(ulong)	8
sizeof(char)	2 (Unicode)
sizeof(float)	4
sizeof(double)	8
sizeof(bool)	1

struct などその他のすべての型については、**sizeof** 演算子はアンセーフ コード ブロックでのみ使用できます。SizeOf メソッドを使用できますが、このメソッドで返される値が、**sizeof** メソッドで返される値と同じでないことがあります。**Marshal.SizeOf** は値のマーシャリング後にサイズを返します。一方、**sizeof** は、共通言語ランタイムによる割り当ての後に埋め込みを含めたサイズを返します。

使用例

```
// cs_operator_sizeof.cs
// compile with: /unsafe
using System;
class MainClass
{
    unsafe static void Main()
    {
        Console.WriteLine("The size of short is {0}.", sizeof(short));
```

```
        Console.WriteLine("The size of int is {0}.", sizeof(int));
        Console.WriteLine("The size of long is {0}.", sizeof(long));
    }
}
```

出力

```
The size of short is 2.
The size of int is 4.
The size of long is 8.
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 18.5.8 sizeof 演算子

参照

関連項目

C# のキーワード

演算子キーワード (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

typeof (C# リファレンス)

型の **System.Type** オブジェクトを取得します。**typeof** 式は次の形式をとります。

```
System.Type type = typeof(int);
```

解説

式の実行時の型を取得するには、次のように、.NET Framework のメソッド GetType を使用できます。

```
int i = 0;
System.Type type = i.GetType();
```

typeof 演算子は、オープン ジェネリック型に使用することもできます。複数の型パラメータが指定されている型では、仕様上、適切な数のコンマが使用されている必要があります。**typeof** 演算子はオーバーロードできません。

使用例

```
// cs_operator_typeof.cs
using System;
using System.Reflection;

public class SampleClass
{
   public int sampleMember;
   public void SampleMethod() {}

   static void Main()
   {
      Type t = typeof(SampleClass);
      // Alternatively, you could use
      // SampleClass obj = new SampleClass();
      // Type t = obj.GetType();

      Console.WriteLine("Methods:");
      MethodInfo[] methodInfo = t.GetMethods();

      foreach (MethodInfo mInfo in methodInfo)
         Console.WriteLine(mInfo.ToString());

      Console.WriteLine("Members:");
      MemberInfo[] memberInfo = t.GetMembers();

      foreach (MemberInfo mInfo in memberInfo)
         Console.WriteLine(mInfo.ToString());
   }
}
```

出力

```
Methods:
Void SampleMethod()
System.Type GetType()
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
Members:
Void SampleMethod()
System.Type GetType()
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
Void .ctor()
Int32 sampleMember
```

次の例では、**GetType** メソッドを使用して、数値計算の結果の格納に使用する型が決定されます。これは、結果として導かれた数値のストレージ要件によって異なります。

```
// cs_operator_typeof2.cs
using System;
class GetTypeTest
{
   static void Main()
   {
      int radius = 3;
      Console.WriteLine("Area = {0}", radius * radius * Math.PI);
      Console.WriteLine("The type is {0}",
                        (radius * radius * Math.PI).GetType()
      );
   }
}
```

出力

```
Area = 28.2743338823081
The type is System.Double
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 7.5.11 typeof 演算子

参照

関連項目

C# のキーワード
is (C# リファレンス)
演算子キーワード (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

true (C# リファレンス)

オーバーロードされた演算子またはリテラルとして使用されます。

true 演算子

true リテラル

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

true 演算子 (C# リファレンス)

true を示す場合には bool (C# リファレンス) 値 true を返し、それ以外のユーザー定義型により定義されている場合は false を返します。これは、データベースで使われているように、true、false、および null (true でも false でもない) を表す型に使用すると便利です。

このような型は、if ステートメント、do ステートメント、while ステートメント、および for ステートメント内の制御式や、条件式で使用できます。

型で **true** 演算子を定義する場合は、false (C# リファレンス) 演算子も定義する必要があります。

型は条件論理演算子 (&& 演算子 (C# リファレンス) および|| 演算子 (C# リファレンス)) を直接オーバーロードすることはできません。条件論理演算子を直接オーバーロードする場合と同様の効果を得るには、通常の論理演算子と **true** および **false** の演算子をオーバーロードします。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 10.9.1 単項演算子
- 7.11.2 ユーザー定義の条件論理演算子
- 7.16 ブール式

参照
関連項目
C# のキーワード
C# の演算子
false (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

true リテラル (C# リファレンス)

ブール値 true を表します。

使用例

```
// cs_keyword_true.cs
using System;
class TestClass
{
    static void Main()
    {
        bool a = true;
        Console.WriteLine( a ? "yes" : "no" );
    }
}
```

出力

```
yes
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 2.4.4.1 ブール型リテラル

参照

関連項目

C# のキーワード

false (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

false (C# リファレンス)

オーバーロードされた演算子またはリテラルとして使用されます。

- false 演算子
- false リテラル

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

false 演算子 (C# リファレンス)

false を示す場合には bool (C# リファレンス) 値 true を返し、それ以外の場合は false を返します。これは、データベースで使われているように、true、false、および null (true でも false でもない) を表す型に使用すると便利です。

このような型は、if ステートメント、do ステートメント、while ステートメント、および for ステートメント内の制御式や、条件式で使用できます。

型で **false** 演算子を定義する場合は、true (C# リファレンス) 演算子も定義する必要があります。

型は条件論理演算子 (& 演算子 (C# リファレンス) および || 演算子 (C# リファレンス)) を直接オーバーロードすることはできません。条件論理演算子を直接オーバーロードする場合と同様の効果を得るには、通常の論理演算子と **true** および **false** の演算子をオーバーロードします。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 10.9.1 単項演算子
- 7.11.2 ユーザー定義の条件論理演算子
- 7.16 ブール式

参照
関連項目
C# のキーワード
C# の演算子
true (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

false リテラル (C# リファレンス)

ブール値 false を表します。

使用例

```
// cs_keyword_false.cs
using System;
class TestClass
{
    static void Main()
    {
        bool a = false;
        Console.WriteLine( a ? "yes" : "no" );
    }
}
```

出力

```
no
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 2.4.4.1 ブール型リテラル

参照

関連項目

C# のキーワード

true (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

stackalloc (C# リファレンス)

unsafe コード コンテキストで、スタックにメモリ ブロックを割り当てるために使用されます。

```
int* fib = stackalloc int[100];
```

解説

この例では、**int** 型の要素を 100 個保持するのに十分なサイズを持つメモリ ブロックが、ヒープではなくスタックに割り当てられます。割り当てられたブロックのアドレスは、`fib` ポインタに格納されます。このメモリは、ガベージ コレクションの対象外であるため、fixed で固定する必要はありません。メモリ ブロックの有効期間は、このブロックが定義されているメソッドの有効期間に限定されます。メソッドから制御が戻る前にメモリを解放することはできません。

stackalloc は、ローカル変数初期化子でだけ有効です。

ポインタ型が使用されるので、**stackalloc** は unsafe (C# リファレンス) コンテキストを必要とします。
「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

stackalloc は、C ランタイム ライブラリの _alloca に似ています。

セキュリティ

アンセーフ コードは、本質的に、非アンセーフ コードほど安全ではありません。ただし、**stackalloc** を使用すると、共通言語ランタイム (CLR: Common Language Runtime) のバッファ オーバーラン検出機能が自動的に有効になります。バッファ オーバーランが検出されると、悪意のあるコードが実行される可能性を最小限に抑えるため、プロセスはできる限り迅速に終了されます。

使用例

```
// cs_keyword_stackalloc.cs
// compile with: /unsafe
using System;
class Test
{
    static unsafe void Main()
    {
        int* fib = stackalloc int[100];
        int* p = fib;
        *p++ = *p++ = 1;
        for (int i = 2; i < 100; ++i, ++p)
        {
            *p = p[-1] + p[-2];
        }
        for (int i = 0; i < 10; ++i)
        {
            Console.WriteLine(fib[i]);
        }
    }
}
```

出力

```
1
1
2
3
5
8
13
21
34
55
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 18.7 スタック割り当て

参照

関連項目

C# のキーワード

演算子キーワード (C# リファレンス)

概念

C# プログラミング ガイド

unsafe コードとポインタ (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

変換キーワード (C# リファレンス)

ここでは、型変換で使用されるキーワードについて説明します。

- explicit
- implicit
- operator

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

explicit (C# リファレンス)

explicit キーワードは、キャストを使って呼び出す必要があるユーザー定義型の変換演算子を宣言します。たとえば、この演算子は Fahrenheit というクラスを Celsius というクラスに変換します。

```
// Must be defined inside a class called Farenheit:
public static explicit operator Celsius(Farenheit f)
{
    return new Celsius((5.0f/9.0f)*(f.degrees-32));
}
```

この変換演算子は次のように呼び出すことができます。

```
Farenheit f = new Farenheit(100.0f);
Celsius c = (Celsius)f;
```

解説

変換演算子は、変換元の型を変換先の型に変換します。変換元の型が変換演算子を提供します。暗黙の型変換の場合とは異なり、明示的な型変換演算子はキャストを使って呼び出す必要があります。変換の演算によって例外が発生したり情報が失われたりする可能性がある場合、その変換には **explicit** キーワードを使用する必要があります。これにより、予想外の結果を招く可能性がある変換演算がコンパイラによって暗黙的に呼び出されることがなくなります。

キャストを省略すると、コンパイル時のエラー (コンパイラ エラー CS0266) の原因になります。

詳細については、「変換演算子の使用 (C# プログラミング ガイド)」を参照してください。

使用例

次の例に示す `Fahrenheit` と `Celsius` の各クラスは、他方のクラスへの変換を行うための明示的な変換演算子を提供します。

```
// cs_keyword_explicit_temp.cs
using System;
class Celsius
{
    public Celsius(float temp)
    {
        degrees = temp;
    }
    public static explicit operator Fahrenheit(Celsius c)
    {
        return new Fahrenheit((9.0f / 5.0f) * c.degrees + 32);
    }
    public float Degrees
    {
        get { return degrees; }
    }
    private float degrees;
}

class Fahrenheit
{
    public Fahrenheit(float temp)
    {
        degrees = temp;
    }
    public static explicit operator Celsius(Fahrenheit f)
    {
        return new Celsius((5.0f / 9.0f) * (f.degrees - 32));
    }
    public float Degrees
    {
        get { return degrees; }
    }
```

```
    private float degrees;
}

class MainClass
{
    static void Main()
    {
        Fahrenheit f = new Fahrenheit(100.0f);
        Console.Write("{0} fahrenheit", f.Degrees);
        Celsius c = (Celsius)f;
        Console.Write(" = {0} celsius", c.Degrees);
        Fahrenheit f2 = (Fahrenheit)c;
        Console.WriteLine(" = {0} fahrenheit", f2.Degrees);
    }
}
```

出力

```
100 fahrenheit = 37.77778 celsius = 100 fahrenheit
```

1 桁の 10 進値を表す構造体 `Digit` を定義する例を次に示します。`byte` 型から `Digit` 型へ変換するための演算子が定義されていますが、すべての `byte` 型を `Digit` 型に変換できるとは限らないため、この変換は明示的に行うように指定されています。

```
// cs_keyword_explicit_2.cs
using System;
struct Digit
{
    byte value;
    public Digit(byte value)
    {
        if (value > 9)
        {
            throw new ArgumentException();
        }
        this.value = value;
    }

    // Define explicit byte-to-Digit conversion operator:
    public static explicit operator Digit(byte b)
    {
        Digit d = new Digit(b);
        Console.WriteLine("conversion occurred");
        return d;
    }
}

class MainClass
{
    static void Main()
    {
        try
        {
            byte b = 3;
            Digit d = (Digit)b; // explicit conversion
        }
        catch (Exception e)
        {
            Console.WriteLine("{0} Exception caught.", e);
        }
    }
}
```

出力

```
conversion occurred
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 6.2 明示的な変換

参照
処理手順
方法 : 構造体間にユーザー定義の変換を実装する (C# プログラミング ガイド)
関連項目
C# のキーワード
implicit (C# リファレンス)
operator (C# リファレンス)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

implicit (C# リファレンス)

implicit キーワードを使用して、暗黙のユーザー定義型変換演算子を宣言します。変換を実行してもデータ損失が発生する心配がない場合に、ユーザー定義型からその他の型、またはその他の型からユーザー型への暗黙の型変換を有効にするために使用します。

```
class Digit
{
    public Digit(double d) { val = d; }
    public double val;
    // ...other members

    // User-defined conversion from Digit to double
    public static implicit operator double(Digit d)
    {
        return d.val;
    }
    //  User-defined conversion from double to Digit
    public static implicit operator Digit(double d)
    {
        return new Digit(d);
    }
}
class Program
{
    static void Main(string[] args)
    {
        Digit dig = new Digit(7);
        //This call invokes the implicit "double" operator
        double num = dig;
        //This call invokes the implicit "Digit" operator
        Digit dig2 = 12;
        Console.WriteLine("num = {0} dig2 = {1}", num, dig2.val);
        Console.ReadLine();
    }
}
```

解説

不要なキャストを省けるため、暗黙の型変換を行う場合はソース コードが読みやすくなります。ただし、暗黙の型変換では、一方の型からもう一方の型へ明示的にキャストする必要がないため、予期しない結果が発生しないように注意する必要があります。暗黙の演算子は、通常、プログラマが変換を認識していなくても安全に使用できるように、変換中に例外をスローしたり情報を失ったりしないことが必要です。このような条件を満たせない変換演算子には、**explicit** キーワードを使用する必要があります。詳細については、
「変換演算子の使用 (C# プログラミング ガイド)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 6.1 暗黙の変換
- 10.9.3 変換演算子

参照

処理手順

方法 : 構造体間にユーザー定義の変換を実装する (C# プログラミング ガイド)

関連項目

C# のキーワード
explicit (C# リファレンス)
operator (C# リファレンス)

概念

operator (C# リファレンス)

operator キーワードを使用して、組み込みの演算子をオーバーロードしたり、クラスまたは構造体宣言内でユーザー定義の変換を行うことができます。

使用例

小数を処理する簡単なクラスを次に示します。このクラスは、小数の加算および乗算を実行するために、+ 演算子および * 演算子をオーバーロードします。また、小数型を倍精度浮動小数点数型に変換する演算子も提供します。

```
// cs_keyword_operator.cs
using System;
class Fraction
{
    int num, den;
    public Fraction(int num, int den)
    {
        this.num = num;
        this.den = den;
    }

    // overload operator +
    public static Fraction operator +(Fraction a, Fraction b)
    {
        return new Fraction(a.num * b.den + b.num * a.den,
           a.den * b.den);
    }

    // overload operator *
    public static Fraction operator *(Fraction a, Fraction b)
    {
        return new Fraction(a.num * b.num, a.den * b.den);
    }

    // user-defined conversion from Fraction to double
    public static implicit operator double(Fraction f)
    {
        return (double)f.num / f.den;
    }
}

class Test
{
    static void Main()
    {
        Fraction a = new Fraction(1, 2);
        Fraction b = new Fraction(3, 7);
        Fraction c = new Fraction(2, 3);
        Console.WriteLine((double)(a * b + c));
    }
}
```

出力

```
0.880952380952381
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 7.2.2 演算子のオーバーロード
- 7.2.3 単項演算子のオーバーロードの解決
- 7.2.4 二項演算子のオーバーロードの解決

参照

処理手順

方法 : 構造体間にユーザー定義の変換を実装する (C# プログラミング ガイド)

関連項目

C# のキーワード

implicit (C# リファレンス)

explicit (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

アクセス キーワード (C# リファレンス)

ここでは、以下のアクセス キーワードについて説明します。

- base

  基本クラスのメンバにアクセスします。

- this

  クラスの現在のインスタンスを参照します。

参照
関連項目
C# のキーワード
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

base (C# リファレンス)

base キーワードは、派生クラス内で基本クラスのメンバにアクセスするために使用されます。

- 他のメソッドによってオーバーライドされた基本クラスのメソッドを呼び出します。
- 派生クラスのインスタンスを作成するときに呼び出される基本クラスのコンストラクタを指定します。

基本クラスにアクセスできるのは、コンストラクタ内、インスタンスのメソッド内、またはインスタンスのプロパティ アクセサ内だけです。

静的メソッド内で **base** キーワードを使用するのはエラーです。

使用例

この例では、基本クラス `Person` および派生クラス `Employee` の両方に、`Getinfo` という名前のメソッドがあります。**base** キーワードを使用すると、基本クラスの `Getinfo` メソッドを派生クラス内で呼び出すことができます。

```
// keywords_base.cs
// Accessing base class members
using System;
public class Person
{
    protected string ssn = "444-55-6666";
    protected string name = "John L. Malgraine";

    public virtual void GetInfo()
    {
        Console.WriteLine("Name: {0}", name);
        Console.WriteLine("SSN: {0}", ssn);
    }
}
class Employee : Person
{
    public string id = "ABC567EFG";
    public override void GetInfo()
    {
        // Calling the base class GetInfo method:
        base.GetInfo();
        Console.WriteLine("Employee ID: {0}", id);
    }
}

class TestClass
{
    static void Main()
    {
        Employee E = new Employee();
        E.GetInfo();
    }
}
```

この例では、派生クラスのインスタンスを作成するときに呼び出される、基本クラスのコンストラクタを指定する方法を示します。

```
// keywords_base2.cs
using System;
public class BaseClass
{
    int num;

    public BaseClass()
    {
        Console.WriteLine("in BaseClass()");
    }

    public BaseClass(int i)
```

```
        {
            num = i;
            Console.WriteLine("in BaseClass(int i)");
        }

        public int GetNum()
        {
            return num;
        }
    }

    public class DerivedClass : BaseClass
    {
        // This constructor will call BaseClass.BaseClass()
        public DerivedClass() : base()
        {
        }

        // This constructor will call BaseClass.BaseClass(int i)
        public DerivedClass(int i) : base(i)
        {
        }

        static void Main()
        {
            DerivedClass md = new DerivedClass();
            DerivedClass md1 = new DerivedClass(1);
        }
    }
```

出力

```
Name: John L. Malgraine
SSN: 444-55-6666
Employee ID: ABC567EFG
```

その他の例については、「new」、「virtual」、および「override」を参照してください。

出力

```
in BaseClass()
in BaseClass(int i)
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.3 基本クラス
- 7.5.8 base-access

参照

関連項目

C# のキーワード

this (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

this (C# リファレンス)

this キーワードは、クラスの現在のインスタンスを参照します。

this の一般的な使い方を次に示します。

- 似た名前によって隠ぺいされるメンバを修飾します。たとえば、次のように使います。

```
public Employee(string name, string alias)
{
    this.name = name;
    this.alias = alias;
}
```

- オブジェクトを他のメソッドにパラメータとして渡します。たとえば、次のように使います。

```
CalcTax(this);
```

- インデクサを宣言します。たとえば、次のように使います。

```
public int this [int param]
{
    get { return array[param];  }
    set { array[param] = value; }
}
```

静的メンバ関数は、クラス レベルで存在し、オブジェクトの一部ではないため、**this** ポインタを持っていません。静的メソッドで **this** を参照するとエラーになります。

使用例

この例では、似た名前によって隠ぺいされている `Employee` クラスのメンバ `name` と `alias` を修飾するために **this** が使用されています。また、別のクラスに属するメソッド `CalcTax` にオブジェクトを渡すためにも使用されています。

```
// keywords_this.cs
// this example
using System;
class Employee
{
    private string name;
    private string alias;
    private decimal salary = 3000.00m;

    // Constructor:
    public Employee(string name, string alias)
    {
        // Use this to qualify the fields, name and alias:
        this.name = name;
        this.alias = alias;
    }

    // Printing method:
    public void printEmployee()
    {
        Console.WriteLine("Name: {0}\nAlias: {1}", name, alias);
        // Passing the object to the CalcTax method by using this:
        Console.WriteLine("Taxes: {0:C}", Tax.CalcTax(this));
    }

    public decimal Salary
    {
```

```
            get { return salary; }
        }
    }
    class Tax
    {
        public static decimal CalcTax(Employee E)
        {
            return 0.08m * E.Salary;
        }
    }

    class MainClass
    {
        static void Main()
        {
            // Create objects:
            Employee E1 = new Employee("John M. Trainer", "jtrainer");

            // Display results:
            E1.printEmployee();
        }
    }
```

出力

```
Name: John M. Trainer
Alias: jtrainer
Taxes: $240.00
```

その他の例については、「class」、および「struct」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 7.5.7 this-access
- 10.2.6.4 this アクセス

参照

関連項目

C# のキーワード
base (C# リファレンス)
メソッド (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

リテラル キーワード (C# リファレンス)

C# には、以下のリテラル キーワードがあります。

- null
- true
- false
- default

参照
関連項目
C# のキーワード
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

null (C# リファレンス)

null キーワードは、null 参照を表すリテラル キーワードです。null 参照はオブジェクトを一切参照しません。**null** は参照型変数の既定値です。

C# 2.0 では、null 許容型が導入されました。これは、値を未定義に設定できるデータ型です。null 許容型 (C# プログラミング ガイド) を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 2.4.4.6 null リテラル

参照

関連項目

C# のキーワード

リテラル キーワード (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

既定値の一覧表 (C# リファレンス)

true (C# リファレンス)

オーバーロードされた演算子またはリテラルとして使用されます。

true 演算子

true リテラル

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

false (C# リファレンス)

オーバーロードされた演算子またはリテラルとして使用されます。

- false 演算子
- false リテラル

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

default (C# リファレンス)

default キーワードです。

default キーワードは **switch** ステートメントまたはジェネリック コードで使用できます。

- switch (C# リファレンス) : 既定のラベルを指定します。
- ジェネリック コードの default キーワード (C# プログラミング ガイド) : 型パラメータの既定値を指定します。参照型の場合は null、値型の場合はゼロになります。

参照
関連項目
C# のキーワード
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

コンテキスト キーワード (C# リファレンス)

コンテキスト キーワードを使用して、コード内で特定の意味を与えることができます。ただし C# ではコンテキスト キーワードは予約語ではありません。このセクションでは、次のコンテキスト キーワードについて説明します。

get　プロパティまたはインデクサのアクセサ メソッドを定義します。

partial　同じコンパイル単位での部分クラス、部分構造体、部分インターフェイスを定義します。

set　プロパティまたはインデクサのアクセサ メソッドを定義します。

where　ジェネリック宣言に制約を追加します。

yield　反復子ブロックで使用され、列挙子オブジェクトに値を返すか、反復処理終了を通知します。

value　アクセサを設定し、イベント ハンドラを追加または削除します。

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

get (C# リファレンス)

プロパティまたはインデクサで、プロパティ値またはインデクサの要素値を取得するアクセサ メソッドを定義します。詳細については、「プロパティ (C# プログラミング ガイド)」および「インデクサ (C# プログラミング ガイド)」を参照してください。

これは、`Seconds` というプロパティの **get** アクセサの例です。

```
class TimePeriod
{
    private double _seconds;
    public double Seconds
    {
        get { return _seconds;  }
        set { _seconds = value; }
    }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.2 プロパティ
- 10.6.2 アクセサ

参照
関連項目
C# のキーワード
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

partial (C# リファレンス)

部分型定義では、クラス、構造体、またはインターフェイスを複数のファイルに分割することを定義できます。

次に File1.cs の部分型定義を示します。

```
namespace PC
{
    partial class A { }
}
```

次に File2.cs の部分型定義を示します。

```
namespace PC
{
    partial class A { }
}
```

解説

クラス型、構造体 型、またはインターフェイス型を複数のファイルに分割する操作は、大規模なプロジェクトや、Windows フォーム デザイナで自動生成されるコードを処理する場合に役立ちます。詳細については、「部分クラス定義 (C# プログラミング ガイド)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 19.4 部分型
- 23 部分型

参照

関連項目

修飾子 (C# リファレンス)
ジェネリックの概要 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

set (C# リファレンス)

プロパティまたはインデクサで、プロパティ値またはインデクサの要素値を設定するアクセサ メソッドを定義します。詳細については、「プロパティ (C# プログラミング ガイド)」および「インデクサ (C# プログラミング ガイド)」を参照してください。

次に示すのは、`Seconds` というプロパティの **set** アクセサの例です。

```
class TimePeriod
{
    private double _seconds;
    public double Seconds
    {
        get { return _seconds;  }
        set { _seconds = value; }
    }
}
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.2 プロパティ
- 10.6.2 アクセサ

参照
関連項目
C# のキーワード
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

value (C# リファレンス)

暗黙の **value** パラメータは、アクセサの設定およびイベント ハンドラの追加と削除に使用されます。

value の使い方の詳細については、「event」および「非対称アクセサのアクセシビリティ (C# プログラミング ガイド)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.2 プロパティ
- 10.6.2 アクセサ

参照

関連項目

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

where (C# リファレンス)

where 句は、ジェネリック宣言で定義されている型パラメータの引数として使用できる型に対する制約を指定します。たとえば、型パラメータ `T` が **IComparable<T>** インターフェイスを実装するように `MyGenericClass` ジェネリック クラスを宣言できます。

```
public class MyGenericClass<T> where T:IComparable { }
```

where 句には、インターフェイス制約だけでなく基本クラス制約も指定できます。基本クラス制約は、ジェネリック型の型引数として使用する型には、基本クラスとして指定されているクラスまたは基本クラス自体が含まれている必要があることを指定します。このような制約を使用する場合は、型パラメータに関する制約よりも前に制約を記述する必要があります。

```
// cs_where.cs
// compile with: /target:library
using System;

class MyClassy<T, U>
    where T : class
    where U : struct
{
}
```

where 句には、コンストラクタ制約も指定できます。新しい演算子を使用して型パラメータのインスタンスを作成できますが、このようにインスタンスを作成するには、コンストラクタ制約 **new()** によって型パラメータに制約を指定する必要があります。new() 制約に基づいて、コンパイラは、指定されている型引数には、アクセス可能なパラメータなしの (または既定の) コンストラクタが必要であることを認識します。次に例を示します。

```
// cs_where_2.cs
// compile with: /target:library
using System;
public class MyGenericClass <T> where T: IComparable, new()
{
    // The following line is not possible without new() constraint:
    T item = new T();
}
```

new() 制約は、**where** 句の最後に記述します。

複数の型パラメータがある場合には、型パラメータごとに **where** 句を 1 つずつ使用します。次に例を示します。

```
// cs_where_3.cs
// compile with: /target:library
using System;
using System.Collections;

interface MyI
{
}

class Dictionary<TKey,TVal>
    where TKey: IComparable, IEnumerable
    where TVal: MyI
{
    public void Add(TKey key, TVal val)
    {
    }
}
```

次に示すように、ジェネリック メソッドの型パラメータにも制約を適用できます。

```
public bool MyMethod<T>(T t) where T : IMyInterface { }
```

デリゲートに対する型パラメータ制約の構文は、メソッドの構文と同じである点に注意してください。

```
delegate T MyDelegate<T>() where T : new()
```

ジェネリック デリゲートについては、「汎用デリゲート (C# プログラミング ガイド)」を参照してください。

制約の構文と使い方の詳細については、「型パラメータの制約 (C# プログラミング ガイド)」を参照してください。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 19.1.4 制約

参照
関連項目
ジェネリックの概要 (C# プログラミング ガイド)
new 制約 (C# リファレンス)
型パラメータの制約 (C# プログラミング ガイド)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

yield (C# リファレンス)

列挙子オブジェクトに値を挿入する場合、または反復処理の終了を通知する場合に、iterator ブロック内で使用されます。形式は次のいずれかになります。

```
yield return <expression>;
yield break;
```

解説

expression は、列挙子オブジェクトの値として評価され、返されます。*expression* は、反復子の yield 型に暗黙に変換できる必要があります。

yield ステートメントは、メソッド、演算子、またはアクセサの本体として使用される **iterator** ブロック内でのみ使用できます。このようなメソッド、演算子、またはアクセサの本体は、次の制約によって制御されます。

- unsafe ブロックは使用できません。
- メソッド、演算子、またはアクセサのパラメータとして、ref または out は使用できません。

匿名メソッドでは、**yield** ステートメントを使用できません。詳細については、「匿名メソッド (C# プログラミング ガイド)」を参照してください。

expression と共に **yield return** ステートメントを使用する場合、**catch** ブロックまたは **catch** 句が 1 つ以上含まれている **try** ブロック内ではこのステートメントを使用できません。詳細については、「例外処理ステートメント (C# リファレンス)」を参照してください。

使用例

次に示す例では、**yield** ステートメントは反復子ブロック (`Power(int number, int power)` メソッド) 内で使用されています。**Power** メソッドが呼び出されると、数値の累乗を含む列挙可能なオブジェクトが返されます。**Power** メソッドの戻り値の型が **IEnumerable** (反復子インターフェイス型) である点に注意してください。

```
// yield-example.cs
using System;
using System.Collections;
public class List
{
    public static IEnumerable Power(int number, int exponent)
    {
        int counter = 0;
        int result = 1;
        while (counter++ < exponent)
        {
            result = result * number;
            yield return result;
        }
    }

    static void Main()
    {
        // Display powers of 2 up to the exponent 8:
        foreach (int i in Power(2, 8))
        {
            Console.Write("{0} ", i);
        }
    }
}
```

出力

```
2 4 8 16 32 64 128 256
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 19.3 反復子
- 22 反復子

参照
関連項目
foreach、in (C# リファレンス)
反復子の使用 (C# プログラミング ガイド)
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

C# の演算子

C# には、多くの演算子が用意されています。演算子とは、式で実行する演算を指定する記号のことです。整数型に対する **==**、**!=**、**<**、**>**、**<=**、**>=**、**binary +**、**binary -**、**^**、**&**、**|**、**~**、**++**、**--**、**sizeof()** などの演算は、通常、列挙体で使用できます。また、演算子の多くはユーザーがオーバーロードできるため、ユーザー定義型に適用された演算子は、意味が変わります。

次の表では、C# 演算子を優先順位別にグループにまとめます。同じグループの演算子の優先順位に差はありません。

演算子のカテゴリ	演算子
1 次式	x.y f(x) a[x] x++ x-- new typeof checked unchecked ->
単項式	+ - ! ~ ++x --x (T)x true false & sizeof
乗算	* / %
加算	+ -
シフト	<< >>

関係式と型検査	< > <= >= is as
等値	== !=
論理 AND	&
論理 XOR	^
論理 OR	|
条件 AND	&&
条件 OR	||
条件	?:
代入	= += -= *= /= %= &= |= ^= <<= >>= ??

算術オーバーフロー

算術演算子 (+、-、*、/) を実行すると、結果が数値型の有効な値の範囲を超えることがあります。詳細については、各演算子に関するセクションを参照してください。概要は、以下のとおりです。

- 整数の算術オーバーフローでは、OverflowException がスローされるか、または結果の最上位ビットが破棄されます。0 による整数除算では、常に **DivideByZeroException** がスローされます。
- 浮動小数点数の算術オーバーフローまたは 0 による浮動小数点除算では、例外はスローされません。これは、浮動小数点型が IEEE 754 に基づいており、無限大および NaN (Not a Number) を表現できるためです。
- 小数の算術オーバーフローでは、常に **OverflowException** がスローされます。0 による小数除算では、常に DivideByZeroException がスローされます。

整数のオーバーフローが発生したときの対処方法は、実行コンテキスト (checked または unchecked) によって異なります。checked コンテキストの場合は、**OverflowException** がスローされます。unchecked コンテキストの場合は、結果の最上位ビットが破棄され、実行が続行され

ます。このように、C# ではオーバーフローを処理するのか、それとも無視するのかをユーザーが選択できます。

算術演算子の場合だけでなく、整数型から整数型へのキャスト (long から int へのキャストなど) でもオーバーフローは発生し、その場合も実行が checked または unchecked のいずれかによって対処が異なります。ただし、ビット処理演算子とシフト演算子ではオーバーフローは発生しません。

参照

処理手順

演算子のオーバーロードのサンプル

関連項目

オーバーロードされた演算子 (C# プログラミング ガイド)

C# のキーワード

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

Visual C#

[] 演算子 (C# リファレンス)

角かっこ (**[]**) は、配列、インデクサ、および属性で使用します。角かっこは、ポインタでも使用できます。

解説

配列型は、型名の後に **[]** が続きます。

```
int[] fib; // fib is of type int[], "array of int"
fib = new int[100]; // create a 100-element int array
```

配列の要素にアクセスするには、目的の要素の添字を角かっこで囲みます。

```
fib[0] = fib[1] = 1;
for( int i=2; i<100; ++i ) fib[i] = fib[i-1] + fib[i-2];
```

配列の添字が範囲外の場合は、例外がスローされます。

配列の添字演算子は、オーバーロードできません。ただし、型ではインデクサおよび 1 つ以上のパラメータをとるプロパティを定義できます。インデクサのパラメータは配列の添字と同じように角かっこで囲みますが、整数でなければならない配列の添字とは異なり、インデクサのパラメータは任意の型として宣言できます。

たとえば、.NET Framework では任意の型のキーと値を関連付ける **Hashtable** 型を定義しています。

```
Collections.Hashtable h =  new Collections.Hashtable();
h["a"] = 123; // note: using a string as the index
```

角かっこは、属性 (C# プログラミング ガイド)を指定するためにも使用します。

```
[attribute(AllowMultiple=true)]
public class Attr
{
}
```

角かっこを使用して、ポインタにインデックスを作成できます。

```
unsafe fixed ( int* p = fib )   // p points to fib from earlier example
{
    p[0] = p[1] = 1;
    for( int i=2; i<100; ++i ) p[i] = p[i-1] + p[i-2];
}
```

添字の範囲チェックは行われません。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.5 演算子
- 7.2 演算子

参照

関連項目

C# の演算子
インデクサ (C# プログラミング ガイド)
unsafe (C# リファレンス)
fixed ステートメント (C# リファレンス)

概念

C# プログラミング ガイド
配列 (C# プログラミング ガイド)

その他の技術情報

() 演算子 (C# リファレンス)

かっこは、式の演算順序を指定するだけでなく、キャストまたは型変換を指定するためにも使用します。

```
double x = 1234.7;
int a;
a = (int)x; // cast double to int
```

解説

キャストでは、型変換演算子が明示的に呼び出されます。型変換演算子が定義されていない場合、キャストは失敗します。型変換演算子の定義については、「explicit」および「implicit」を参照してください。

() 演算子はオーバーロードできません。

詳細については、「キャスト (C# プログラミング ガイド)」を参照してください。

キャスト式が原因で構文があいまいになることがあります。たとえば、(x)-y という式は、キャスト式 (型 x に対する -y のキャスト) またはかっこで囲んだ式と組み合わされた加算式 (値 x - y を計算する) のいずれにも解釈できます。

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.5 演算子
- 7.2 演算子

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

. 演算子 (C# リファレンス)

ドット演算子 (.) は、メンバ アクセスに使用します。ドット演算子は、型または名前空間のメンバを指定します。たとえば、ドット演算子を使用して、.NET Framework クラス ライブラリ内の特定のメソッドにアクセスします。

```
// The class Console in namespace System:
System.Console.WriteLine("hello");
```

解説

たとえば、次のクラスを考えます。

```
class Simple
{
    public int a;
    public void b()
    {
    }
}
Simple s = new Simple();
```

変数 `s` には、`a` と `b` という 2 つのメンバがあります。それらのメンバにアクセスするためにドット演算子を使用します。

```
s.a = 6;   // assign to field a;
s.b();     // invoke member function b;
```

ドットは修飾名にも使用します。修飾名とは、属している名前空間やインターフェイスなどを示す名前のことです。

```
// The class Console in namespace System:
System.Console.WriteLine("hello");
```

using ディレクティブを使用すると、名前の修飾を省略できます。

```
using System;
// ...
System.Console.WriteLine("hello");
Console.WriteLine("hello");   // same thing
```

ただし、識別子があいまいな場合は、修飾する必要があります。

```
using System;
// A namespace containing another Console class:
using OtherSystem;
// ...
// Must qualify Console:
System.Console.WriteLine( "hello" );
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 7.5.4 メンバ アクセス

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

:: 演算子 (C# リファレンス)

名前空間エイリアス修飾子演算子です。

名前空間エイリアス修飾子 (::) を使用して識別子を検索できます。この例に示すように、常に 2 つの識別子の間に配置します。

```
global::System.Console.WriteLine("Hello World");
```

解説

名前空間エイリアス修飾子として `global` を指定できます。これにより、エイリアスを使用した名前空間ではなく、グローバル名前空間で検索が実行されます。

参照項目

:: 演算子の使用例については、以下のセクションを参照してください。

- 方法 : 名前空間エイリアス修飾子を使用する (C# プログラミング ガイド)

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 20.9.5 簡易名
- 25.3 名前空間エイリアス修飾子

参照

関連項目

C# の演算子
名前空間キーワード (C# リファレンス)
. 演算子 (C# リファレンス)
extern エイリアス (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

+ 演算子 (C# リファレンス)

+ 演算子は、単項演算子または二項演算子として機能します。

解説

単項 + 演算子は、すべての数値型に対してあらかじめ定義されています。数値型での単項 + 演算の結果は、単にオペランドの値になります。

二項 + 演算子は、数値型と文字列型に対してあらかじめ定義されています。数値型の場合、+ は 2 つのオペランドの合計を計算します。オペランドの片方または両方が文字列型の場合は、オペランドの文字列表現が連結されます。

デリゲート型でも、デリゲートを連結する二項 + 演算子が用意されています。

単項 + 演算子と二項 + 演算子は、ユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。通常、整数型に対する演算は、列挙に対して適用されます。

使用例

```
// cs_operator_plus.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(+5);        // unary plus
        Console.WriteLine(5 + 5);     // addition
        Console.WriteLine(5 + .5);    // addition
        Console.WriteLine("5" + "5"); // string concatenation
        Console.WriteLine(5.0 + "5"); // string concatenation
        // note automatic conversion from double to string
    }
}
```

出力

```
5
10
5.5
55
55
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.5 演算子
- 7.2 演算子

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

- 演算子 (C# リファレンス)

- 演算子は、単項演算子または二項演算子として機能します。

解説

単項 - 演算子は、すべての数値型に対してあらかじめ定義されています。数値型での単項 - 演算子の結果は、オペランドの符号を反転した数値になります。

二項 - 演算子は、すべての数値型と列挙型に対して組み込まれています。二項 - 演算子では、最初のオペランドから 2 番目のオペランドが減算されます。

デリゲート型でも、デリゲートを削除する二項 - 演算子が用意されています。

単項 - 演算子と二項 - 演算子は、ユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。

使用例

```
// cs_operator_minus.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        Console.WriteLine(-a);
        Console.WriteLine(a - 1);
        Console.WriteLine(a - .5);
    }
}
```

出力

```
-5
4
4.5
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 1.6.6.5 演算子
- 7.2 演算子

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

* 演算子 (C# リファレンス)

乗算演算子 (*) です。オペランドの積を計算します。また、ポインタの読み取りと書き込みを有効にする逆参照演算子でもあります。

解説

すべての数値型には定義済みの乗算演算子があります。

* 演算子は、ポインタ型の宣言やポインタの逆参照にも使用します。この演算子は、unsafe (C# リファレンス) キーワードにより示される unsafe コンテキストでのみ使用できます。この場合、/unsafe (unsafe モードの有効化) (C# コンパイラ オプション) コンパイラ オプションが必要です。逆参照演算子は、間接演算子とも呼ばれます。

* 二項演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。二項演算子をオーバーロードすると、対応する代入演算子がある場合には、この演算子も暗黙でオーバーロードされます。

使用例

```
// cs_operator_mult.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(5 * 2);
        Console.WriteLine(-.5 * .2);
        Console.WriteLine(-.5m * .2m); // decimal type
    }
}
```

出力

```
10
-0.1
-0.10
```

```
// cs_operator_ptr.cs
// compile with: /unsafe
public class MainClass
{
    unsafe static void Main()
    {
        int i = 5;
        int* j = &i;
        System.Console.WriteLine(*j);
    }
}
```

出力

```
5
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

unsafe コードとポインタ (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

/ 演算子 (C# リファレンス)

除算演算子 (/) では、最初のオペランドが 2 番目のオペランドで除算されます。すべての数値型には定義済みの除算演算子があります。

解説

/ 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。/ 演算子をオーバーロードすると、/= 演算子が暗黙にオーバーロードされます。

使用例

```
// cs_operator_division.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(-5/2);
        Console.WriteLine(-5.0/2);
    }
}
```

出力

```
-2
-2.5
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

% 演算子 (C# リファレンス)

剰余演算子 (%) では、最初のオペランドが 2 番目のオペランドで除算された後の剰余が計算されます。すべての数値型には定義済みの剰余演算子があります。

解説

% 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。二項演算子をオーバーロードすると、対応する代入演算子がある場合には、この演算子も暗黙でオーバーロードされます。

使用例

```
// cs_operator_modulus.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(5 % 2);       // int
        Console.WriteLine(-5 % 2);      // int
        Console.WriteLine(5.0 % 2.2);   // double
        Console.WriteLine(5.0m % 2.2m); // decimal
        Console.WriteLine(-5.2 % 2.0);  // double
    }
}
```

出力

```
1
-1
0.6
0.6
-1.2
```

説明

double 型では丸め誤差が発生することに注意してください。

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

& 演算子 (C# リファレンス)

& 演算子は、単項演算子または二項演算子として機能します。

解説

単項 & 演算子では、オペランドのアドレスが返されます (unsafe コンテキストが必要)。

二項 & 演算子は、整数型と **bool** に対してあらかじめ定義されています。整数型の場合、& はオペランドのビットごとの論理 AND を計算します。**bool** オペランドの場合は、オペランドの論理 AND が計算されます。つまり、両方のオペランドが **true** の場合だけ結果が **true** になります。

& 演算子は、1 番目の演算子の値に関係なく、両方の演算子を評価します。次に例を示します。

```
int i = 0;
if (false & ++i == 1)
{
    // i is incremented, but the conditional
    // expression evaluates to false, so
    // this block does not execute.
}
```

二項 **&** 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。通常、整数型に対する演算は、列挙に対して適用されます。二項演算子をオーバーロードすると、対応する代入演算子がある場合には、この演算子も暗黙でオーバーロードされます。

使用例

```
// cs_operator_ampersand.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(true & false); // logical and
        Console.WriteLine(true & true);  // logical and
        Console.WriteLine("0x{0:x}", 0xf8 & 0x3f);  // bitwise and
    }
}
```

出力

```
False
True
0x38
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

| 演算子 (C# リファレンス)

二項 **|** 演算子は、整数型と **bool** に対してあらかじめ定義されています。整数型の場合、**|** ではオペランドのビットごとの OR が計算されます。**bool** オペランドの場合は、**|** によりオペランドの論理 OR が計算されます。つまり、両方のオペランドが **false** の場合だけ結果が **false** になります。

解説

| 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。

使用例

```
// cs_operator_OR.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(true | false);  // logical or
        Console.WriteLine(false | false); // logical or
        Console.WriteLine("0x{0:x}", 0xf8 | 0x3f);   // bitwise or
    }
}
```

出力

```
True
False
0xff
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

^ 演算子 (C# リファレンス)

二項 ^ 演算子は、整数型と **bool** に対してあらかじめ定義されています。整数型の場合、^ ではオペランドのビットごとの排他的 OR が計算されます。**bool** オペランドの場合は、^ によりオペランドの排他的論理和が計算されます。つまり、片方のオペランドが **true** の場合だけ結果が **true** になります。

解説

^ 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。通常、整数型に対する演算は、列挙に対して適用されます。

使用例

```
// cs_operator_bitwise_OR.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(true ^ false);  // logical exclusive-or
        Console.WriteLine(false ^ false); // logical exclusive-or
        // Bitwise exclusive-or:
        Console.WriteLine("0x{0:x}", 0xf8 ^ 0x3f);
    }
}
```

出力

```
True
False
0xc7
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

! 演算子 (C# リファレンス)

論理否定演算子 (**!**) は、オペランドを否定する単項演算子です。この演算子は **bool** に対して定義されており、オペランドが **false** の場合だけ **true** が返されます。

解説

! 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。

使用例

```
// cs_operator_negation.cs
using System;
class MainClass
{
    static void Main()
{
        Console.WriteLine(!true);
        Console.WriteLine(!false);
    }
}
```

出力

```
False
True
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

~ 演算子 (C# リファレンス)

~ 演算子は、そのオペランドにビットごとの補数演算を実行し、各ビットを反転させます。ビットごとの補数演算子は、int、uint、long、および ulong に対して組み込まれています。

解説

~ 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。通常、整数型に対する演算は、列挙に対して適用されます。

使用例

```
// cs_operator_bitwise_compl.cs
using System;
class MainClass
{
    static void Main()
    {
        int[] values = { 0, 0x111, 0xfffff, 0x8888, 0x22000022};
        foreach (int v in values)
        {
            Console.WriteLine("~0x{0:x8} = 0x{1:x8}", v, ~v);
        }
    }
}
```

出力

```
~0x00000000 = 0xffffffff
~0x00000111 = 0xfffffeee
~0x000fffff = 0xfff00000
~0x00008888 = 0xffff7777
~0x22000022 = 0xddffffdd
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

= 演算子 (C# リファレンス)

代入演算子 (=) では、右辺のオペランドの値が左辺のオペランドで示された格納場所、プロパティ、またはインデクサに格納され、その値が結果として返されます。両側のオペランドは、同じ型である必要があります。同じ型でない場合、右辺のオペランドは、左辺のオペランドの型に暗黙に変換できる必要があります。

解説

代入演算子は、オーバーロードできません。

使用例

```
// cs_operator_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        double x;
        int i;
        i = 5; // int to int assignment
        x = i; // implicit conversion from int to double
        i = (int)x; // needs cast
        Console.WriteLine("i is {0}, x is {1}", i, x);
        object obj = i;
        Console.WriteLine("boxed value = {0}, type is {1}",
                  obj, obj.GetType());
        i = (int)obj;
        Console.WriteLine("unboxed: {0}", i);
    }
}
```

出力

```
i is 5, x is 5
boxed value = 5, type is System.Int32
unboxed: 5
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

< 演算子 (C# リファレンス)

すべての数値型と列挙型では、"より小さい" 関係演算子 (<) が定義されています。この演算子では、最初のオペランドが 2 番目のオペランドより小さい場合に **true** が返され、それ以外の場合は **false** が返されます。

解説

< 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。< をオーバーロードする場合は、> もオーバーロードする必要があります。二項演算子をオーバーロードすると、対応する代入演算子がある場合には、この演算子も暗黙でオーバーロードされます。

使用例

```
// cs_operator_less_than.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(1 < 1.1);
        Console.WriteLine(1.1 < 1.1);
    }
}
```

出力

```
True
False
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

> 演算子 (C# リファレンス)

すべての数値型および列挙型では、"より大きい" 関係演算子 (>) が定義されています。この演算子では、最初のオペランドが 2 番目のオペランドより大きい場合に **true** が返され、それ以外の場合は **false** が返されます。

解説

> 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。**>** をオーバーロードする場合は、**<** もオーバーロードする必要があります。二項演算子をオーバーロードすると、対応する代入演算子がある場合には、この演算子も暗黙でオーバーロードされます。

使用例

```
// cs_operator_greater_than.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(1.1 > 1);
        Console.WriteLine(1.1 > 1.1);
    }
}
```

出力

```
True
False
```

参照

関連項目

C# の演算子

explicit (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

?: 演算子 (C# リファレンス)

条件演算子 (**?:**) では、ブール式の値に応じて 2 つの値のいずれかが返されます。次に条件演算子の形式を示します。

```
condition ? first_expression : second_expression;
```

解説

条件が **true** の場合、1 番目の式が評価され、これが結果となります。条件が **false** の場合、2 番目の式が評価され、これが結果となります。常に 2 つの式のいずれか 1 つだけが評価されます。

if-else の構造が必要になる計算は、条件演算子を使用すると、簡潔で明快に表現できます。たとえば、**sin** 関数の計算で 0 による除算を避けるには、次のいずれかで記述できます。

```
if(x != 0.0) s = Math.Sin(x)/x; else s = 1.0;
```

条件演算子を使用すると、次のように記述できます。

```
s = x != 0.0 ? Math.Sin(x)/x : 1.0;
```

条件演算子は結合規則が右から左です。

```
a ? b : c ? d : e
```

この式は、次のように評価されます。

```
a ? b : (c ? d : e)
```

not

```
(a ? b : c) ? d : e
```

条件演算子は、オーバーロードできません。

使用例

```
// cs_operator_conditional.cs
using System;
class MainClass
{
    static double sinc(double x)
    {
        return x != 0.0 ? Math.Sin(x)/x : 1.0;
    }

    static void Main()
    {
        Console.WriteLine(sinc(0.2));
        Console.WriteLine(sinc(0.1));
        Console.WriteLine(sinc(0.0));
    }
}
```

出力

```
0.993346653975306
0.998334166468282
1
```

参照

関連項目

C# の演算子

if-else (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

++ 演算子 (C# リファレンス)

インクリメント演算子 (++) では、オペランドが 1 ずつインクリメントされます。インクリメント演算子は、オペランドの前または後に指定できます。

解説

最初の形式は、前置インクリメント演算です。この演算の結果は、インクリメントが行われた後のオペランドの値になります。

2 番目の形式は、後置インクリメント演算です。この演算の結果は、インクリメントが行われる前のオペランドの値になります。

数値型と列挙型には組み込みのインクリメント演算子があります。++ 演算子はユーザー定義型でオーバーロードできます。通常、整数型に対する演算は、列挙に対して適用されます。

使用例

```
// cs_operator_increment.cs
using System;
class MainClass
{
    static void Main()
    {
        double x;
        x = 1.5;
        Console.WriteLine(++x);
        x = 1.5;
        Console.WriteLine(x++);
        Console.WriteLine(x);
    }
}
```

出力

```
2.5
1.5
2.5
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

-- 演算子 (C# リファレンス)

デクリメント演算子 (--) では、オペランドが 1 ずつデクリメントされます。デクリメント演算子は、`--variable` や `variable--` のように、オペランドの前または後に指定できます。最初の形式は、前置デクリメント演算です。この演算の結果は、"デクリメントが行われた後" のオペランドの値になります。2 番目の形式は、後置デクリメント演算です。この演算の結果は、"デクリメントが行われる前" のオペランドの値になります。

解説

数値型と列挙型には組み込みのデクリメント演算子があります。

-- 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。通常、整数型に対する演算は、列挙に対して適用されます。

使用例

```
// cs_operator_decrement.cs
using System;
class MainClass
{
    static void Main()
    {
        double x;
        x = 1.5;
        Console.WriteLine(--x);
        x = 1.5;
        Console.WriteLine(x--);
        Console.WriteLine(x);
    }
}
```

出力

```
0.5
1.5
0.5
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

&& 演算子 (C# リファレンス)

条件 AND 演算子 (**&&**) では **bool** オペランドの論理 AND が実行されますが、必要な場合のみ、2 番目のオペランドが評価されます。

解説

```
x && y
```

この演算は次の演算に相当します。

```
x & y
```

ただし、`x` が **false** の場合、`y` は評価されません。この場合、AND 演算の結果は `y` の値にかかわらず `false` になるためです。これは、"ショートサーキット" 評価と呼ばれます。

条件 AND 演算子はオーバーロードできませんが、通常の論理演算子および true 演算子と false 演算子のオーバーロードは、条件論理演算子の制約付きのオーバーロードとも見なされます。

使用例

最初のオペランドだけが評価される **&&** を使用した式の例は、次のとおりです。

```
// cs_operator_logical_and.cs
using System;
class MainClass
{
   static bool Method1()
   {
      Console.WriteLine("Method1 called");
      return false;
   }

   static bool Method2()
   {
      Console.WriteLine("Method2 called");
      return true;
   }

   static void Main()
   {
      Console.WriteLine("regular AND:");
      Console.WriteLine("result is {0}", Method1() & Method2());
      Console.WriteLine("short-circuit AND:");
      Console.WriteLine("result is {0}", Method1() && Method2());
   }
}
```

出力

```
regular AND:
Method1 called
Method2 called
result is False
short-circuit AND:
Method1 called
result is False
```

C# 言語仕様

詳細については、「C# 言語仕様」の次のセクションを参照してください。

- 7.11.2 ユーザー定義の条件論理演算子

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

|| 演算子 (C# リファレンス)

条件 OR 演算子 (||) では **bool** オペランドの論理 OR が実行されますが、必要な場合だけ、2 番目のオペランドが評価されます。

解説

```
x || y
```

この演算は次の演算に相当します。

```
x | y
```

ただし、x が **true** の場合、y は評価されません。この場合、OR 演算の結果は y の値にかかわらず **true** になるためです。これは、"ショート サーキット" 評価と呼ばれます。

条件 OR 演算子はオーバーロードできませんが、通常の論理演算子および true 演算子と false 演算子のオーバーロードは、条件論理演算子の制約付きのオーバーロードとも見なされます。

使用例

最初のオペランドだけが評価される **||** を使用した式の例は、次のとおりです。

```
// cs_operator_short_circuit_OR.cs
using System;
class MainClass
{
    static bool Method1()
    {
        Console.WriteLine("Method1 called");
        return true;
    }

    static bool Method2()
    {
        Console.WriteLine("Method2 called");
        return false;
    }

    static void Main()
    {
        Console.WriteLine("regular OR:");
        Console.WriteLine("result is {0}", Method1() | Method2());
        Console.WriteLine("short-circuit OR:");
        Console.WriteLine("result is {0}", Method1() || Method2());
    }
}
```

出力

```
regular OR:
Method1 called
Method2 called
result is True
short-circuit OR:
Method1 called
result is True
```

参照
関連項目
C# の演算子
概念
C# プログラミング ガイド
その他の技術情報
C# リファレンス

<< 演算子 (C# リファレンス)

左シフト演算子 (<<) では、2 番目のオペランドで指定されたビット数だけ最初のオペランドが左にシフトされます。2 番目のオペランドの型は int です。

解説

1 番目のオペランドが int または uint (32 ビット値) の場合、シフト数は 2 番目のオペランドの下位 5 ビットで指定されます。

1 番目のオペランドが long または ulong (64 ビット値) の場合、シフト数は 2 番目のオペランドの下位 6 ビットで指定されます。

1 番目のオペランドの上位ビットは破棄され、シフトの結果空になる下位ビットには 0 が入ります。シフト演算では、オーバーフローは発生しません。

<< 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。オーバーロードでは、最初のオペランドの型はユーザー定義型、2 番目のオペランドの型は **int** である必要があります。二項演算子をオーバーロードすると、対応する代入演算子がある場合には、この演算子も暗黙でオーバーロードされます。

使用例

```
// cs_operator_left_shift.cs
using System;
class MainClass
{
    static void Main()
    {
        int i = 1;
        long lg = 1;
        Console.WriteLine("0x{0:x}", i << 1);
        Console.WriteLine("0x{0:x}", i << 33);
        Console.WriteLine("0x{0:x}", lg << 33);
    }
}
```

出力

```
0x2
0x2
0x200000000
```

説明

1 と 33 では下位 5 ビットが同じであるため、`i<<1` と `i<<33` の結果は同じになります。

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

>> 演算子 (C# リファレンス)

右シフト演算子 (>>) では、2 番目のオペランドで指定されたビット数だけ最初のオペランドが右にシフトされます。

解説

最初のオペランドが int または uint (32 ビット値) の場合、シフト数は 2 番目のオペランドの下位 5 ビット (2 番目のオペランド & 0x1f) で指定されます。

最初のオペランドが long または ulong (64 ビット値) の場合、シフト数は 2 番目のオペランドの下位 6 ビット (2 番目のオペランド & 0x3f) で指定されます。

最初のオペランドが int または long の場合、右シフトは算術シフトです。つまり、シフトの結果空になる上位ビットに符号ビットが設定されます。最初のオペランドが uint 型または ulong 型の場合、右シフトは論理シフトです。つまり、上位ビットには 0 が入ります。

>> 演算子はユーザー定義型でオーバーロードできます。オーバーロードでは、最初のオペランドの型はユーザー定義型、2 番目のオペランドの型は int である必要があります。詳細については、「operator (C# リファレンス)」を参照してください。二項演算子をオーバーロードすると、対応する代入演算子がある場合には、この演算子も暗黙でオーバーロードされます。

使用例

```
// cs_operator_right_shift.cs
using System;
class MainClass
{
    static void Main()
    {
        int i = -1000;
        Console.WriteLine(i >> 3);
    }
}
```

出力

```
-125
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

== 演算子 (C# リファレンス)

組み込みの値型の場合、等値演算子 (==) ではオペランドの値が等しい場合に true が返され、それ以外の場合は **false** が返されます。string 以外の参照型の場合、== では 2 つのオペランドが同じオブジェクトを参照する場合に **true** が返されます。**string** 型の場合は、== は文字列の値を比較します。

解説

== 演算子はユーザー定義の値型でオーバーロードできます。詳細については、「operator」を参照してください。ユーザー定義の参照型でオーバーロードはできますが、既定では、組み込み参照型とユーザー定義参照型のいずれに対しても == は前述のとおりに機能します。== をオーバーロードする場合は、!= もオーバーロードする必要があります。通常、整数型に対する演算は、列挙に対して適用されます。

使用例

```
// cs_operator_equality.cs
using System;
class MainClass
{
    static void Main()
    {
        // Numeric equality: True
        Console.WriteLine((2 + 2) == 4);

        // Reference equality: different objects,
        // same boxed value: False.
        object s = 1;
        object t = 1;
        Console.WriteLine(s == t);

        // Define some strings:
        string a = "hello";
        string b = String.Copy(a);
        string c = "hello";

        // Compare string values of a constant and an instance: True
        Console.WriteLine(a == b);

        // Compare string references;
        // a is a constant but b is an instance: False.
        Console.WriteLine((object)a == (object)b);

        // Compare string references, both constants
        // have the same value, so string interning
        // points to same reference: True.
        Console.WriteLine((object)a == (object)c);
    }
}
```

出力

```
True
False
True
False
True
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

!= 演算子 (C# リファレンス)

非等値演算子 (**!=**) では、オペランドが等しい場合に false が返され、それ以外の場合は true が返されます。非等値演算子は、文字列とオブジェクトを含むすべての型に対して組み込まれています。**!=** 演算子はユーザー定義型でオーバーロードできます。

解説

組み込みの値型の場合、非等値演算子 (**!=**) ではオペランドの値が異なる場合に true が返され、それ以外の場合は false が返されます。**string** 以外の参照型の場合、**!=** では 2 つのオペランドが異なるオブジェクトを参照する場合に true が返されます。**string** 型の場合は、**!=** は文字列の値を比較します。

!= 演算子はユーザー定義の値型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。ユーザー定義の参照型でオーバーロードはできますが、既定では、組み込み参照型とユーザー定義参照型のいずれに対しても **!=** は前述のとおりに機能します。**!=** をオーバーロードする場合は、== もオーバーロードする必要があります。通常、整数型に対する演算は、列挙に対して適用されます。

使用例

```
// cs_operator_inequality.cs
using System;
class MainClass
{
    static void Main()
    {
        // Numeric inequality:
        Console.WriteLine((2 + 2) != 4);

        // Reference equality: two objects, same boxed value
        object s = 1;
        object t = 1;
        Console.WriteLine(s != t);

        // String equality: same string value, same string objects
        string a = "hello";
        string b = "hello";

        // compare string values
        Console.WriteLine(a != b);

        // compare string references
        Console.WriteLine((object)a != (object)b);
    }
}
```

出力

```
False
True
False
False
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

<= 演算子 (C# リファレンス)

すべての数値型と列挙型では、"より小さいか等しい" 関係演算子 (<=) が定義されています。この演算子では、最初のオペランドが 2 番目のオペランドより小さいか等しい場合に **true** が返され、それ以外の場合は **false** が返されます。

解説

<= 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。<= をオーバーロードする場合は、>= もオーバーロードする必要があります。通常、整数型に対する演算は、列挙に対して適用されます。

使用例

```
// cs_operator_less_than_or_equal.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(1 <= 1.1);
        Console.WriteLine(1.1 <= 1.1);
    }
}
```

出力

```
True
True
```

参照

関連項目

C# の演算子

explicit (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

>= 演算子 (C# リファレンス)

すべての数値型および列挙型では、"より大きいか等しい" 関係演算子 (>=) が定義されています。この演算子では、最初のオペランドが 2 番目のオペランドより大きいか等しい場合に **true** が返され、それ以外の場合は **false** が返されます。

解説

>= 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。>= をオーバーロードする場合は、<= もオーバーロードする必要があります。通常、整数型に対する演算は、列挙に対して適用されます。

使用例

```
// cs_operator_greater_than_or_equal.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(1.1 >= 1);
        Console.WriteLine(1.1 >= 1.1);
    }
}
```

出力

```
True
True
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

+= 演算子 (C# リファレンス)

加算代入演算子です。

解説

次のような += 代入演算子を使用する式があるとします。

```
x += y
```

上記のコードは、次のコードと同じです。

```
x = x + y
```

ただし、x が評価されるのは 1 回だけです。+ 演算子の意味は、x および y の型に依存します。たとえば、数値オペランドの場合は加算、文字列オペランドの場合は連結になります。

+= 演算子は直接オーバーロードできませんが、+ 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。

加算代入演算子 (+=) を使用して、イベントに対する応答として呼び出されるメソッドを指定できます。これらのメソッドをイベント ハンドラと呼びます。イベント ハンドラはデリゲート型にカプセル化されているので、このコンテキストで加算代入演算子 (+=) を使用することをデリゲート連結と呼びます。詳細については、「event (C# リファレンス)」および「デリゲート (C# プログラミング ガイド)」を参照してください。

使用例

```
// cs_operator_addition_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        //addition
        int a = 5;
        a += 6;
        Console.WriteLine(a);

        //string concatenation
        string s = "Micro";
        s += "soft";
        Console.WriteLine(s);
    }
}
```

出力

```
11
Microsoft
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

-= 演算子 (C# プログラマーズ リファレンス)

減算代入演算子です。

解説

次のような -= 代入演算子を使用する式があるとします。

```
x -= y
```

これは、次と同じ意味になります。

```
x = x - y
```

ただし、x が評価されるのは 1 回だけです。- 演算子の意味は、x および y の型に依存します。たとえば、数値オペランドの場合は減算、デリゲート オペランドの場合はデリゲートの削除になります。

-= 演算子は直接オーバーロードできませんが、- 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。

使用例

```
// cs_operator_subtraction_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        a -= 6;
        Console.WriteLine(a);
    }
}
```

出力

```
-1
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

*= 演算子 (C# リファレンス)

二項乗算代入演算子です。

解説

次のような *= 代入演算子を使用する式があるとします。

```
x *= y
```

上記のコードは、次のコードと同じです。

```
x = x * y
```

ただし、x が評価されるのは 1 回だけです。* 演算子は、乗算のために数値型に対して組み込まれています。

= 演算子は直接オーバーロードできませんが、 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。

使用例

```
// cs_operator_multiplication_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        a *= 6;
        Console.WriteLine(a);
    }
}
```

出力

```
30
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

/= 演算子 (C# リファレンス)

除算代入演算子です。

解説

次のような **/=** 代入演算子を使用する式があるとします。

```
x /= y
```

上記のコードは、次のコードと同じです。

```
x = x / y
```

ただし、x が評価されるのは 1 回だけです。/ 演算子は、除算のために数値型に対して組み込まれています。

/= 演算子は直接オーバーロードできませんが、/ 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。すべての複合代入演算子において、二項演算子をオーバーロードすると、同等の複合代入が暗黙的にオーバーロードされます。

使用例

```
// cs_operator_division_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        a /= 6;
        Console.WriteLine(a);
        double b = 5;
        b /= 6;
        Console.WriteLine(b);
    }
}
```

出力

```
0
0.833333333333333
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

%= 演算子 (C# リファレンス)

剰余代入演算子です。

解説

次のような **%=** 代入演算子を使用する式があるとします。

```
x %= y
```

上記のコードは、次のコードと同じです。

```
x = x % y
```

ただし、x が評価されるのは 1 回だけです。% 演算子は、除算後の剰余を計算するために数値型に対して組み込まれています。

%= 演算子は直接オーバーロードできませんが、% 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。

使用例

```
// cs_operator_modulus_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        a %= 3;
        Console.WriteLine(a);
    }
}
```

出力

```
2
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

&= 演算子 (C# リファレンス)

AND 代入演算子です。

解説

次のような **&=** 代入演算子を使用する式があるとします。

```
x &= y
```

上記のコードは、次のコードと同じです。

```
x = x & y
```

ただし、x が評価されるのは 1 回だけです。& 演算子では、整数のオペランドではビットごとの論理 AND 演算、**bool** オペランドでは論理 AND 演算が実行されます。

&= 演算子は直接オーバーロードできませんが、二項 & 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。

使用例

```
// cs_operator_and_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 0x0c;
        a &= 0x06;
        Console.WriteLine("0x{0:x8}", a);
        bool b = true;
        b &= false;
        Console.WriteLine(b);
    }
}
```

出力

```
0x00000004
False
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

|= 演算子 (C# リファレンス)

OR 代入演算子です。

解説

次のような |= 代入演算子を使用する式があるとします。

```
x |= y
```

上記のコードは、次のコードと同じです。

```
x = x | y
```

ただし、x が評価されるのは 1 回だけです。| 演算子では、整数のオペランドではビットごとの論理 OR 演算、bool オペランドでは論理 OR 演算が実行されます。

|= 演算子は直接オーバーロードできませんが、| 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator (C# リファレンス)」を参照してください。

使用例

```
// cs_operator_or_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 0x0c;
        a |= 0x06;
        Console.WriteLine("0x{0:x8}", a);
        bool b = true;
        b |= false;
        Console.WriteLine(b);
    }
}
```

出力

```
0x0000000e
True
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

^= 演算子 (C# リファレンス)

排他的 OR 代入演算子です。

解説

次のような形式の式があるとします。

```
x ^= y
```

この式は、次のように評価されます。

```
x = x ^ y
```

ただし、x が評価されるのは 1 回だけです。^ 演算子では、整数のオペランドではビットごとの排他的 OR 演算、bool オペランドでは排他的論理和が実行されます。

^= 演算子は直接オーバーロードできませんが、! 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。

使用例

```
// cs_operator_xor_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 0x0c;
        a ^= 0x06;
        Console.WriteLine("0x{0:x8}", a);
        bool b = true;
        b ^= false;
        Console.WriteLine(b);
    }
}
```

出力

```
0x0000000a
True
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

<<= 演算子 (C# リファレンス)

左シフト代入演算子です。

解説

次のような形式の式があるとします。

```
x <<= y
```

この式は、次のように評価されます。

```
x = x << y
```

ただし、x が評価されるのは 1 回だけです。<< 演算子では、y で指定されたビット数だけ x が左にシフトされます。

<<= 演算子は直接オーバーロードできませんが、<< 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。

使用例

```
// cs_operator_left_shift_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 1000;
        a <<= 4;
        Console.WriteLine(a);
    }
}
```

出力

```
16000
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

>>= 演算子 (C# リファレンス)

右シフト代入演算子です。

解説

次のような形式の式があるとします。

```
x >>= y
```

この式は、次のように評価されます。

```
x = x >> y
```

ただし、x が評価されるのは 1 回だけです。>> 演算子では、y で指定された分だけ x が右にシフトされます。

>>= 演算子は直接オーバーロードできませんが、>> 演算子はユーザー定義型でオーバーロードできます。詳細については、「operator」を参照してください。

使用例

```
// cs_operator_right_shift_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 1000;
        a >>= 4;
        Console.WriteLine(a);
    }
}
```

出力

```
62
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

-> 演算子 (C# リファレンス)

-> 演算子は、ポインタの逆参照とメンバ アクセスを組み合わせます。

解説

次のような形式の式があるとします。

```
x->y
```

この式は次の式と同じです (`x` は `T*` 型のポインタ、`y` は `T` のメンバ)。

```
(*x).y
```

-> 演算子は、アンマネージ コードだけで使用できます。

-> 演算子はオーバーロードできません。

使用例

```
// cs_operator_dereferencing.cs
// compile with: /unsafe
using System;
struct Point
{
    public int x, y;
}

class MainClass
{
    unsafe static void Main()
    {
        Point pt = new Point();
        Point* pp = &pt;
        pp->x = 123;
        pp->y = 456;
        Console.WriteLine ( "{0} {1}", pt.x, pt.y );
    }
}
```

出力

```
123 456
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

?? 演算子 (C# リファレンス)

?? 演算子は、左側のオペランドが null 値でない場合にはこのオペランドを返し、null 値である場合には右側のオペランドを返します。

解説

null 許容型は、値を指定するか、未定義にしておくことができます。**??** 演算子は、null 非許容型に対して null 許容型が割り当てられているときに返す既定値を定義します。**??** 演算子を使用せずに、null 非許容型に対して null 許容型を割り当てると、コンパイル時のエラーが発生します。null 許容型が定義されていない場合にキャストを使用すると、**InvalidOperationException** 例外がスローされます。

詳細については、「null 許容型 (C# プログラミング ガイド)」を参照してください。

使用例

```
// nullable_type_operator.cs
using System;
class MainClass
{
    static int? GetNullableInt()
    {
        return null;
    }

    static string GetStringValue()
    {
        return null;
    }

    static void Main()
    {
        // ?? operator example.
        int? x = null;

        // y = x, unless x is null, in which case y = -1.
        int y = x ?? -1;

        // Assign i to return value of method, unless
        // return value is null, in which case assign
        // default value of int to i.
        int i = GetNullableInt() ?? default(int);

        string s = GetStringValue();
        // ?? also works with reference types.
        // Display contents of s, unless s is null,
        // in which case display "Unspecified".
        Console.WriteLine(s ?? "Unspecified");
    }
}
```

参照

関連項目

C# の演算子

概念

C# プログラミング ガイド

null 許容型 (C# プログラミング ガイド)

その他の技術情報

C# リファレンス

C# プリプロセッサ ディレクティブ

ここでは、C# 言語のプリプロセッサ ディレクティブについて説明します。

#if

#else

#elif

#endif

#define

#undef

#warning

#error

#line

#region

#endregion

#pragma

#pragma warning

#pragma checksum

C# のコンパイラには、独立したプリプロセッサはありませんが、ここで説明するディレクティブは、独立したプリプロセッサがある場合と同様に処理されます。これらのディレクティブは、条件付きコンパイルに使用します。ただし、C や C++ のディレクティブとは異なり、上記のディレクティブではマクロは作成できません。

プリプロセッサ ディレクティブは、1 行に 1 つだけ指定します。

参照

処理手順

条件付きメソッドのサンプル

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#if (C# リファレンス)

#if を使用すると、条件付ディレクティブが実行され、1 つ以上のシンボルについて **true** かどうかが評価されます。シンボルが **true** と評価された場合は、**#if** とこれに最も近い #endif ディレクティブの間にあるすべてのコードが、コンパイラによって評価されます。次に例を示します。

```
#define DEBUG
// ...
#if DEBUG
    Console.WriteLine("Debug version");
#endif
```

複数のシンボルを評価するときには、== (等値)、!= (非等値)、&& (AND)、および || (OR) の演算子を使用できます。シンボルと演算子は、かっこを使用してグループ化できます。

解説

#if を、#else、#elif、#endif、#define、および #undef ディレクティブと組み合わせて使用すると、1 つ以上のシンボルの条件に従って、コードを処理対象にしたり、処理対象から外したりできます。#if は、デバッグ ビルド用にコンパイルするときや、特定の構成でコンパイルするときに使用すると便利です。

#if で始まる条件付きディレクティブは、**#endif** ディレクティブで明示的に終了する必要があります。

#define を使用すると、シンボルを定義できます。定義したシンボルを式として `#if` ディレクティブに渡すと、式は **true** と評価されます。

また、シンボルは /define コンパイラ オプションでも定義できます。#undef を使うと、シンボルを未定義状態にできます。

/define または **#define** で定義されたシンボルは、同じ名前の変数とは競合しません。変数名をプリプロセッサ ディレクティブに渡すことはできません。シンボルはプリプロセッサ ディレクティブだけで評価されます。

#define で定義されたシンボルのスコープは、シンボルが定義されたファイル内だけです。

使用例

```
// preprocessor_if.cs
#define DEBUG
#define VC_V7
using System;
public class MyClass
{
    static void Main()
    {
#if (DEBUG && !VC_V7)
        Console.WriteLine("DEBUG is defined");
#elif (!DEBUG && VC_V7)
        Console.WriteLine("VC_V7 is defined");
#elif (DEBUG && VC_V7)
        Console.WriteLine("DEBUG and VC_V7 are defined");
#else
        Console.WriteLine("DEBUG and VC_V7 are not defined");
#endif
    }
}
```

出力

```
DEBUG and VC_V7 are defined
```

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#else (C# リファレンス)

#else を使用すると、複合条件付きディレクティブを作成できます。つまり、先行する #if ディレクティブ、または #elif ディレクティブ (省略可能) の式が **true** と評価されなかった場合に、コンパイラによって、**#else** とそれに続く **#endif** の間にあるすべてのコードが評価されます。

解説

#else と、その次の #endif の間には、他のプリプロセッサ ディレクティブは指定できません。**#else** の使用例については、「#if」を参照してください。

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#elif (C# リファレンス)

#elif を使用すると、複合条件付きディレクティブを作成できます。**#elif** 式が評価されるのは、先行するディレクティブ式 #if または #elif (省略可能) が **true** と評価されなかった場合です。**#elif** 式が **true** と評価された場合は、**#elif** と次の条件付きディレクティブの間にあるすべてのコードが、コンパイラによって評価されます。次に例を示します。

```
#define VC7
//...
#if debug
    Console.Writeline("Debug build");
#elif VC7
    Console.Writeline("Visual Studio 7");
#endif
```

複数のシンボルを評価するときには、**==** (等値)、**!=** (非等値)、**&&** (AND)、および **||** (OR) の演算子を使用できます。シンボルと演算子は、かっこを使用してグループ化できます。

解説

#elif では、次のように記述した場合と同じ結果が得られます。

```
#else
#if
```

#elif を使用する方が簡単です。**#if** には対になる #endif が必要ですが、**#elif** では不要なためです。

#elif の使用例については、「#if」を参照してください。

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#endif (C# リファレンス)

#endif は、#if ディレクティブで始まる条件付きディレクティブの終了を示します。次に例を示します。

```
      #define DEBUG
// ...
#if DEBUG
    Console.WriteLine("Debug version");
#endif
```

解説

#if で始まる条件付きディレクティブは、**#endif** ディレクティブで明示的に終了する必要があります。**#endif** の使用例については、「#if (C# リファレンス)」を参照してください。

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#define (C# リファレンス)

#define を使用すると、シンボルを定義できます。定義したシンボルを式として #if ディレクティブに渡すと、式は **true** と評価されます。次に例を示します。

```
# define DEBUG
```

解説

シンボルを使用して、コンパイル条件を指定できます。シンボルは、#if または #elif で評価できます。また、**conditional** 属性を使用して、条件付きコンパイルを実行することもできます。

シンボルを定義することはできますが、シンボルに値は代入できません。**#define** ディレクティブは、ファイルの先頭で、ディレクティブやそれ以外の命令よりも前に記述する必要があります。

また、シンボルは /define コンパイラ オプションでも定義できます。#undef を使うと、シンボルを未定義状態にできます。

/define または **#define** で定義されたシンボルは、同じ名前の変数とは競合しません。変数名をプリプロセッサ ディレクティブに渡すことはできません。シンボルはプリプロセッサ ディレクティブだけで評価されます。

#define で定義されたシンボルのスコープは、シンボルが定義されたファイル内だけです。

#define の使用例については、「#if」を参照してください。

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#undef (C# リファレンス)

#undef を使用すると、シンボルを未定義状態にできます。たとえば、未定義状態のシンボルを #iff ディレクティブで式として使用すると、その式は **false** と評価されます。

シンボルは、#define ディレクティブまたは /define コンパイラ オプションで定義できます。**#undef** ディレクティブをファイルに記述する場合は、ディレクティブやそれ以外のステートメントよりも前に記述する必要があります。

使用例

```
// preprocessor_undef.cs
// compile with: /d:DEBUG
#undef DEBUG
using System;
class MyClass
{
    static void Main()
    {
#if DEBUG
        Console.WriteLine("DEBUG is defined");
#else
        Console.WriteLine("DEBUG is not defined");
#endif
    }
}
```

出力

```
DEBUG is not defined
```

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#warning (C# リファレンス)

#warning を使用すると、コードの特定の位置でレベル 1 警告を表示できます。次に例を示します。

```
#warning Deprecated code in this method.
```

解説

#warning は、一般に、条件付きディレクティブ内で使用します。また、#error (C# リファレンス) でユーザー定義のエラーを生成することもできます。

使用例

```
// preprocessor_warning.cs
// CS1030 expected
#define DEBUG
class MainClass
{
    static void Main()
    {
#if DEBUG
#warning DEBUG is defined
#endif
    }
}
```

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#error (C# リファレンス)

#error を使用すると、コードの特定の位置でエラーを表示できます。次に例を示します。

```
#error Deprecated code in this method.
```

解説

#error は一般的に、条件付きディレクティブ内で使用します。

また、#warning (C# リファレンス) でユーザー定義の警告を生成することもできます。

使用例

```
// preprocessor_error.cs
// CS1029 expected
#define DEBUG
class MainClass
{
    static void Main()
    {
#if DEBUG
#error DEBUG is defined
#endif
    }
}
```

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#line (C# リファレンス)

#line を使用すると、エラーや警告で出力するコンパイラの行番号とファイル名 (省略可能) を変更できます。この例は、2 つの警告の行番号がどのように報告されるかを示しています。`#line 200` ディレクティブは、行番号を強制的に 200 に設定します。既定の行番号は 7 です。既定の `#line` ディレクティブの結果として、他の行 (#9) は通常の行番号になります。

```
class MainClass
{
    static void Main()
    {
#line 200
        int i;    // CS0168 on line 200
#line default
        char c;   // CS0168 on line 9
    }
}
```

解説

#line ディレクティブは、ビルド プロセスの、自動化された中間ステップで使われる場合があります。たとえば、元のソース コード ファイルから行を削除した場合でも、コンパイラがファイル内での削除前の行番号のままで出力を生成できるように、行を削除してから **#line** を指定して、削除前の行番号指定をシミュレートします。

#line hidden ディレクティブは、後続の行をデバッガから隠します。これで、開発者がコードをステップ実行するときに、**#line hidden** から次の **#line** ディレクティブ (もう 1 つの **#line hidden** ディレクティブでないことが前提) までのすべての行がステップ オーバーされます。このオプションを ASP.NET で使用すると、ユーザー定義のコードとコンピュータが生成したコードを区別できます。この機能は主に ASP.NET で使用されますが、より多くのソース ジェネレータで利用される可能性があります。

#line hidden ディレクティブは、エラー報告のファイル名や行番号には影響しません。このため、隠ぺいされたブロック内でエラーが検出された場合、コンパイラは現在のファイル名とエラーの行番号を報告します。

#line filename ディレクティブにより、コンパイラ出力に表示するファイル名が指定されます。既定では、ソース コード ファイルの実際の名前が使われます。ファイル名は、二重引用符 ("") で囲みます。

ソース コード ファイルには、任意の数の **#line** ディレクティブを指定できます。

例 1

次の例は、デバッガがどのようにコード内の隠ぺいされた行を無視するかを示しています。この例を実行すると、3 行のテキストが表示されます。ただし、この例で示すようにブレークポイントを設定し、**F10** キーを押してコードをステップ実行すると、デバッガは隠ぺいされた行を無視します。また、隠ぺいされた行にブレークポイントを設定しても、デバッガはその行を無視します。

```
// preprocessor_linehidden.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine("Normal line #1."); // Set break point here.
#line hidden
        Console.WriteLine("Hidden line.");
#line default
        Console.WriteLine("Normal line #2.");
    }
}
```

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#region (C# リファレンス)

#region を使用すると、コードのブロックを指定できます。このブロックは、Visual Studio コード エディタのアウトライン機能を使用して、展開や折りたたみができます。次に例を示します。

```
      #region MyClass definition
public class MyClass
{
    static void Main()
    {
    }
}
#endregion
```

解説

#region ブロックは、#endregion ディレクティブで終了させる必要があります。

#region ブロックは、#if ブロックとオーバーラップすることはできません。ただし、**#region** ブロックを **#if** ブロック内に入れ子にしたり、**#if** ブロックを **#region** ブロック内に入れ子にしたりすることはできます。

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#endregion (C# リファレンス)

#endregion は、#region ブロックの終了を示します。次に例を示します。

```
      #region MyClass definition
class MyClass
{
    static void Main()
    {
    }
}
#endregion
```

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#pragma (C# リファレンス)

#pragma は、このキーワードが含まれているファイルに関するコンパイルについて、特殊な命令をコンパイラに指示します。

```
#pragma pragma-name pragma-arguments
```

パラメータ

pragma-name

認識されるプラグマの名前。

pragma-arguments

プラグマ固有の引数。

参照

関連項目

C# プリプロセッサ ディレクティブ
#pragma 警告 (C# リファレンス)
#pragma checksum (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#pragma 警告 (C# リファレンス)

#pragma warning を使用して特定の警告を有効または無効にします。

```
#pragma warning disable warning-list
#pragma warning restore warning-list
```

パラメータ

warning-list

コンマで区切られた警告番号の一覧。"CS" プレフィックスを付けず、番号だけを入力します。

警告番号を指定しないと、**disable** ではすべての警告が無効になり、**restore** ではすべての警告が有効になります。

使用例

```
// pragma_warning.cs
using System;

#pragma warning disable 414, 3021
[CLSCompliant(false)]
public class C
{
    int i = 1;
    static void Main()
    {
    }
}
#pragma warning restore 3021
[CLSCompliant(false)]  // CS3021
public class D
{
    int i = 1;
    public static void F()
    {
    }
}
```

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

#pragma checksum (C# リファレンス)

ASP.NET ページのデバッグに使用するソース ファイルのチェックサムを生成します。

```
#pragma checksum "filename" "{guid}" "checksum bytes"
```

パラメータ

"filename"

変更または更新を監視する必要があるファイルの名前。

"{guid}"

ファイルのグローバルー意識別子 (GUID: Globally Unique Identifier)。

"*checksum_bytes*"

チェックサムのバイトを表す 16 進形式の文字列。偶数の 16 進数であることが必要です。奇数を使用すると、コンパイル時に警告が出力され、ディレクティブが無視されます。

解説

Visual Studio デバッガは、常に正しいソースを検出することを目的としてチェックサムを使用します。コンパイラはソース ファイルのチェックサムを計算し、プログラム データベース (PDB) ファイルに出力します。デバッガは、その PDB を使用して、ソース ファイルについて算出されたチェックサムとの比較を行います。

この方法は、ASP.NET プロジェクトには使用できません。なぜなら、算出されたチェックサムは .aspx ファイルではなく生成されたソース ファイルを対象としているからです。この問題を解決するため、**#pragma checksum** により ASP.NET ページのチェックサムがサポートされています。

Visual C# に ASP.NET プロジェクトを作成すると、生成されるソース ファイルには .aspx ファイルのチェックサムが含まれています。ソースは、この .aspx ファイルから生成されます。次に、コンパイラがこの情報を PDB ファイルに書き込みます。

ファイルに **#pragma checksum** ディレクティブが見つからない場合、コンパイラはチェックサムを計算し、その値を PDB ファイルに書き込みます。

使用例

```
class TestClass
{
    static int Main()
    {
        #pragma checksum "file.cs" "{3673e4ca-6098-4ec1-890f-8fceb2a794a2}" "{012345678AB}"
// New checksum
    }
}
```

参照

関連項目

C# プリプロセッサ ディレクティブ

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

C# コンパイラ オプション

コンパイラでは、実行可能 (.exe) ファイル、ダイナミック リンク ライブラリ (.dll)、またはコード モジュール (.netmodule) を作成します。

各コンパイラ オプションには、それぞれ **-option** と **/option** という 2 つの形式があります。ドキュメントでは **/option** の形式だけを示しています。

このセクションの内容

コマンド ラインからのビルド

コマンド ラインからの Visual C# アプリケーション ビルドに関する情報。

方法 : コマンド ラインからビルドする

vsvars32.bat を実行して、コマンド ライン ビルドを有効にする手順について説明します。

C# アプリケーションの配置

C# アプリケーションの配置オプションについて説明します。

カテゴリ別の C# コンパイラ オプションの一覧

コンパイラ オプションのカテゴリ別の一覧です。

アルファベット順の C# コンパイラ オプションの一覧

コンパイラ オプションのアルファベット順の一覧です。

コンパイラ エラー CS0001 ～ CS9999

C# コンパイラによって生成されるエラーのリファレンス。

関連するセクション

方法 : プロジェクトのプロパティを設定する (C#、J#)

プロジェクトをコンパイル、ビルド、およびデバッグする方法を制御するプロパティの設定について説明します。Visual C# プロジェクトでのカスタム ビルド ステップに関する情報を含みます。

既定のビルドとカスタム ビルド

ビルドの種類および構成に関する情報。

ビルドの準備と管理

Visual Studio 開発環境内でのビルド手順。

コマンド ラインの指定

C# コンパイラは、その実行可能ファイルの名前 (csc.exe) をコマンド ラインに入力することによって呼び出します。Visual Studio コマンド プロンプトを使用すると、必要な環境変数がすべて自動的に設定されます。Visual Studio のコマンド プロンプトには、[スタート] メニューのショートカットからアクセスできます ([Visual Studio Tools])。それ以外の場合は、手動でパスを調整し、任意のサブディレクトリから csc.exe を呼び出せるようにしてください。Visual Studio コマンド プロンプトを使用しない場合は、vsvars32.bat を実行し、コマンド ラインからのビルドに必要な環境変数を設定する必要があります。vsvars32.bat の詳細については、「方法 : コマンド ラインからのビルド」を参照してください。

.NET Framework SDK のみがインストールされているコンピュータでは、[SDK コマンド プロンプト] ([Microsoft .NET Framework SDK] メニュー オプションから選択可能) を使用すると、C# コンパイラをコマンド ラインで使用できます。

開発環境からビルドする場合は、「ビルドの準備と管理」を参照してください。

通常、実行可能ファイル csc.exe は、システム ディレクトリの Microsoft.NET\Framework\<version> フォルダに格納されています。ただし、この格納場所は、コンピュータの構成によって異なる場合があります。同じコンピュータに異なるバージョンの .NET Framework がインストールされている場合は、複数バージョンの csc.exe が存在していることになります。このような環境の詳細については、
「.NET Framework の複数のバージョンのインストール」を参照してください。

ここでは、次の内容について説明します。

コマンド ライン構文の規則

C# コンパイラは、オペレーティング システムのコマンド ラインで指定された引数を次の規則に従って解釈します。

- 引数は、空白 (スペースまたはタブ) で区切ります。
- キャレット (^) は、エスケープ文字やデリミタとしては認識されません。カレットは、オペレーティング システムのコマンド ライン パーサーによって完全に処理されてからプログラムの argv 配列に渡されます。
- 二重引用符で囲まれた文字列 ("string") は、空白を含む場合でも、単一の引数と見なされます。二重引用符で囲んだ文字列を引数に埋め込むこともできます。
- 円記号を前に付けた二重引用符 (\") は、リテラル二重引用符文字 (") として解釈されます。
- 二重引用符の直前にある円記号以外は、円記号 (\) として解釈されます。
- 二重引用符の直前に円記号が偶数個 (0 個は含まない) あると、円記号のペアごとに 1 個の円記号が argv 配列に格納されます。この場合、二重引用符は文字列のデリミタとして解釈されます。
- 二重引用符の直前に円記号が奇数個 (3 個以上) あると、円記号のペアごとに 1 個の円記号が argv 配列に格納されます。最後の円記号によって二重引用符がエスケープ シーケンスになるため、二重引用符文字 (") がそのまま argv に格納されます。

コマンド ラインの例

- File.cs をコンパイルして File.exe を作成します。

```
csc File.cs
```

- File.cs をコンパイルして File.dll を作成します。

```
csc /target:library File.cs
```

- File.cs をコンパイルして My.exe を作成します。

```
csc /out:My.exe File.cs
```

- 最適化を有効にし、DEBUG シンボルを定義して、現在のディレクトリにあるすべての C# ファイルをコンパイルします。File2.exe が出力されます。

```
csc /define:DEBUG /optimize /out:File2.exe *.cs
```

- 現在のディレクトリにあるすべての C# ファイルをコンパイルして、デバッグ バージョンの File2.dll を作成します。ロゴや警告は表示されません。

```
csc /target:library /out:File2.dll /warn:0 /nologo /debug *.cs
```

- 現在のディレクトリにあるすべての C# ファイルをコンパイルして、Something.xyz (DLL) に出力します。

```
csc /target:library /out:Something.xyz *.cs
```

C# コンパイラと C++ コンパイラの出力の相違点

C# コンパイラを起動してもオブジェクト (.obj) ファイルは作成されず、出力ファイルが直接作成されます。このため、C# コンパイラにはリンカが不要です。

参照

関連項目

アルファベット順の C# コンパイラ オプションの一覧
カテゴリ別の C# コンパイラ オプションの一覧

その他の技術情報

C# コンパイラ オプション

方法：環境変数を設定する

vcvars32.bat ファイルは、適切な環境変数を設定してコマンド ライン ビルドを有効にします。vcvars32.bat の詳細については、次のサポート技術情報の文書を参照してください。

- 「PRB: Vcvars32.bat Generates Out of Environment Message (Q248802)」

以前のバージョンの Visual Studio と最新バージョンの Visual Studio の両方がコンピュータにインストールされている場合は、同じコマンド ウィンドウから異なるバージョンの vsvars32.bat または vcvars32.bat を実行しないでください。

VSVARS32.BAT を実行するには

1. コマンド プロンプトで、Visual Studio をインストールしたディレクトリの Common7\Tools サブディレクトリに移動します。
2. 「`VSVARS32`」と入力して VSVARS32.bat を実行します。

注意
VSVARS32.bat はコンピュータごとに異なる可能性があります。VSVARS32.bat ファイルが見つからない場合や破損している場合でも、別のコンピュータの VSVARS32.bat ファイルと置き換えないでください。その場合は、セットアップ プログラムを再実行してファイルを置き換えてください。

参照

概念

コマンド ラインの指定

C# アプリケーションの配置

C# アプリケーションをビルドしたら、いよいよ、アプリケーションを配布することになります。C# は .NET 言語であるため、C# の実行可能ファイルを他のコンピュータに配布するには、配布先の各コンピュータに .NET Framework が (場合によっては、アプリケーションに固有の依存ファイルも) インストールされている必要があります。.NET Framework の配布方法としては、さまざまな選択肢が用意されています。概要については、「.NET Framework の再頒布」を参照してください。

一般に、完成したアプリケーションを他のコンピュータに移動することを、"配置" と呼びます。Microsoft の開発環境では、配置に必要なしくみが用意されています。詳細については、「アプリケーションとコンポーネントの配置」を参照してください。

ビルドおよび配布作業をコマンド ラインから行う機会が多い場合、アプリケーションの配置と、依存関係を持つファイルの再配布には、これ以外の方法を検討する必要があります。

参照

概念

コマンド ラインの指定

カテゴリ別の C# コンパイラ オプションの一覧

次のコンパイラ オプションは、カテゴリ別に並んでいます。アルファベット順の一覧については、「アルファベット順の C# コンパイラ オプションの一覧」を参照してください。

最適化

オプション	目的
/filealign	出力ファイル内のセクションのサイズを指定します。
/optimize	最適化を有効または無効にします。

出力ファイル

オプション	目的
/doc	処理後のドキュメント コメントが出力される XML ファイルを指定します。
/out	出力ファイルを指定します。
/pdb	.pdb ファイルの名前と場所を指定します。
/platform	出力プラットフォームを指定します。
/target	/target:exe、/target:library、/target:module、/target:winexe の 4 つのオプションのうち、いずれかを使用して出力ファイルの形式を指定します。

.NET Framework アセンブリ

オプション	目的
/addmodule	このアセンブリの一部となる 1 つ以上のモジュールを指定します。
/delaysign	公開キーのみ追加し、アセンブリの署名は保留するようコンパイラに指示します。
/keycontainer	暗号化キー コンテナの名前を指定します。
/keyfile	暗号化キーの格納されたファイル名を指定します。
/lib	/reference を使って参照されるアセンブリの場所を指定します。
/nostdlib	標準ライブラリ (mscorlib.dll) をインポートしないようコンパイラに指示します。
/reference	アセンブリを含むファイルからメタデータをインポートします。

デバッグ/エラーのチェック

オプション	目的
/bugreport	バグを簡単に報告するための情報を含むファイルを作成します。
/checked	データ型の境界をオーバーフローする整数算術演算の実行時に例外を発生するかどうかを指定します。
/debug	デバッグ情報を出力するようコンパイラに指示します。
/errorreport	エラー レポートの動作を設定します。
/fullpaths	コンパイラ出力時のファイルへの絶対パスを指定します。
/nowarn	指定された警告を生成しないようコンパイラに指示します。

/warn	警告レベルを設定します。
/warnaserror	警告をエラーとして扱います。

プリプロセッサ

オプション	目的
/define	プリプロセッサ シンボルを定義します。

リソース

オプション	目的
/linkresource	マネージ リソースへのリンクを作成します。
/resource	.NET Framework リソースを出力ファイルに埋め込みます。
/win32icon	出力ファイルに挿入する .ico ファイルを指定します。
/win32res	出力ファイルに挿入する Win32 リソースを指定します。

その他

オプション	目的
@	応答ファイルを指定します。
/?	標準出力にコンパイラ オプションの一覧を表示します。
/baseaddress	DLL を読み込むベース アドレスを指定します。
/codepage	コンパイルですべてのソース コード ファイルに使用するコード ページを指定します。
/help	標準出力にコンパイラ オプションの一覧を表示します。
/langversion	使用する言語のバージョンを指定します。
/main	**Main** メソッドの場所を指定します。
/noconfig	コンパイルに csc.rsp を使用しないようコンパイラに指示します。
/nologo	コンパイラの著作権情報が表示されないようにします。
/recurse	コンパイルするソース ファイルをサブディレクトリで検索します。
/unsafe	unsafe キーワードが使用されたコードのコンパイルを有効にします。
/utf8output	UTF-8 エンコーディングを使用してコンパイラ出力を表示します。

旧式のオプション

/incremental	インクリメンタル コンパイルを有効にします。

参照

処理手順

方法 : 環境変数を設定する

関連項目

アルファベット順の C# コンパイラ オプションの一覧

その他の技術情報

C# コンパイラ オプション

アルファベット順の C# コンパイラ オプションの一覧

次のコンパイラ オプションは、アルファベット順に並んでいます。カテゴリ別の一覧については、「カテゴリ別の C# コンパイラ オプションの一覧」を参照してください。

オプション	目的
@	応答ファイルを読み込んで、オプションを追加します。
/?	使用法に関する説明を標準出力に表示します。
/addmodule	指定されたモジュールをアセンブリにリンクさせます。
/baseaddress	ビルドするライブラリのベース アドレスを指定します。
/bugreport	'障害報告' ファイルを作成します。/errorreport:prompt または /errorreport:send と組み合わせて指定した場合、このファイルがクラッシュ情報と共に送信されます。
/checked	オーバーフローの例外を生成するようコンパイラに指示します。
/codepage	ソース ファイルを開くときに使用するコードページを指定します。
/debug	デバッグ情報を生成します。
/define	条件付きコンパイルのシンボルを定義します。
/delaysign	厳密名キーのうち、公開キーのみを使用した遅延署名をアセンブリに適用します。
/doc	生成する XML ドキュメント ファイルを指定します。
/errorreport	内部コンパイル エラーの扱い (prompt、send、または none) を指定します。既定は none です。
/filealign	出力ファイルにおけるセクションの配置を指定します。
/fullpaths	エラーが発生したファイルの絶対パスを生成するようコンパイラに指示します。
/help	使用法に関する説明を標準出力に表示します。
/incremental	インクリメンタル コンパイルを有効にします。旧式のオプションであり、互換性のために残されています。
/keycontainer	厳密名キー コンテナを指定します。
/keyfile	厳密名キー ファイルを指定します。
/langversion	言語バージョンのモード (ISO-1 または Default) を指定します。
/lib	参照を検索する追加のディレクトリを指定します。
/linkresource	指定されたリソースをアセンブリにリンクさせます。
/main	エントリ ポイントの存在する型を指定します (その他のエントリ ポイントはすべて無視されます)。

/noconfig	CSC.RSP ファイルを自動で追加しないようコンパイラに指示します。
/nologo	コンパイル時の著作権メッセージが表示されないようにします。
/nostdlib	標準ライブラリ (mscorlib.dll) を参照しないようコンパイラに指示します。
/nowarn	特定の警告メッセージを無効にします。
/optimize	最適化を有効または無効にします。
/out	出力ファイル名を指定します (既定では、メイン クラスまたは最初のファイルの基本名)。
/pdb	.pdb ファイルの名前と場所を指定します。
/platform	コードを実行できるプラットフォーム (x86、Itanium、x64、または anycpu) を制限します。既定値は anycpu です。
/recurse	ワイルドカードの指定に基づき、現在のディレクトリおよびサブディレクトリに格納されたすべてのファイルをインクルードします。
/reference	指定されたアセンブリ ファイルからメタデータを参照します。
/resource	指定されたリソースを埋め込みます。
/target	/target:exe/target:library/target:module/target:winexe の 4 つのオプションのうち、いずれかを使用して出力ファイルの形式を指定します。
/unsafe	アンセーフ コードの使用を許可します。
/utf8output	コンパイラのメッセージを UTF-8 エンコーディングで出力します。
/warn	警告レベル (0 ～ 4) を設定します。
/warnaserror	警告をエラーとして報告します。
/win32icon	出力に使用するアイコンを指定します。
/win32res	Win32 リソース ファイル (..res) を指定します。

参照

処理手順

方法 : 環境変数を設定する

関連項目

カテゴリ別の C# コンパイラ オプションの一覧

その他の技術情報

C# コンパイラ オプション

@ (応答ファイルの指定) (C# コンパイラ オプション)

@ オプションを使用すると、コンパイラ オプションおよびコンパイルするソース コード ファイルを含むファイルを指定できます。

```
@response_file
```

引数
response_file

コンパイラ オプションやコンパイルするソース コード ファイルの一覧を含むファイル。

解説

コンパイラ オプションとソース コード ファイルは、コマンド ラインで指定した場合と同じように、コンパイラによって処理されます。

コンパイル時に複数の応答ファイルを指定するには、複数の応答ファイル オプションを指定します。次に例を示します。

```
@file1.rsp @file2.rsp
```

応答ファイルでは、複数のコンパイラ オプションとソース コード ファイルを 1 行に記述できます。1 つのコンパイラ オプションは 1 行に指定する必要があり、複数行にわたって指定できません。応答ファイルには、シャープ記号 (#) で始まるコメントを記述できます。

応答ファイルでのコンパイラ オプションの指定方法は、コマンド ラインでのコンパイラ オプションの指定方法と同じです。詳細については、「コマンド ラインからのビルド」を参照してください。

コンパイラでは、検出順にコマンド オプションを処理します。このため、コマンド ライン引数によって、応答ファイルで先に指定したオプションをオーバーライドできます。逆に、応答ファイルのオプションが、コマンド ラインや他の応答ファイルで先に指定したオプションをオーバーライドすることもあります。

C# では、csc.exe ファイルと同じディレクトリに csc.rsp ファイルがあります。csc.rsp の詳細については、「/noconfig (csc.rsp の無視)」を参照してください。

このコンパイラ オプションは、Visual Studio 開発環境で設定することも、プログラムから変更することもできません。

使用例

サンプルの応答ファイルの一部を次に示します。

```
# build the first output file
/target:exe /out:MyExe.exe source1.cs source2.cs
```

参照
その他の技術情報
C# コンパイラ オプション

/addmodule (メタデータのインポート) (C# コンパイラ オプション)

このオプションを指定すると、target:module スイッチで生成されたモジュールが現在のコンパイルに追加されます。

```
/addmodule:file[;file2]
```

引数
file, file2

メタデータを含む出力ファイル。このファイルにアセンブリ マニフェストを含めることはできません。複数のファイルをインポートする場合は、コンマまたはセミコロンでファイル名を区切ります。

解説

/addmodule を使用して追加されたすべてのモジュールは、実行時に出力ファイルと同じディレクトリに配置されている必要があります。つまり、コンパイル時には任意のディレクトリのモジュールを指定できますが、実行時に指定するモジュールはアプリケーション ディレクトリ内に配置する必要があります。実行時にモジュールがアプリケーション ディレクトリ内に存在しない場合は、TypeLoadException になります。

file にアセンブリを含めることはできません。たとえば、/target:module を使用して出力ファイルが作成された場合、そのメタデータは **/addmodule** を使用してインポートできます。

出力ファイルが **/target:module** 以外の **/target** オプションを使用して作成された場合、そのメタデータは **/addmodule** ではインポートできませんが、/reference を使用してインポートできます。

このコンパイラ オプションは Visual Studio では使用できないため、プロジェクトではモジュールを参照できません。また、このコンパイラ オプションは、コードからは変更できません。

使用例

ソース ファイル `input.cs` をコンパイルし、`metad1.netmodule` および `metad2.netmodule` からメタデータを追加して `out.exe` を作成するには、次のコードを使用します。

```
csc /addmodule:metad1.netmodule;metad2.netmodule /out:out.exe input.cs
```

参照
その他の技術情報
C# コンパイラ オプション

/baseaddress (DLL のベース アドレスの指定) (C# コンパイラ オプション)

/baseaddress オプションを使用すると、DLL を読み込むベース アドレスを指定できます。

```
/baseaddress:address
```

引数
address

DLL のベース アドレス。このアドレスは、10 進数、16 進数、または 8 進数で指定できます。

解説

DLL の既定のベース アドレスは、.NET Framework 共通言語ランタイムによって設定されます。

このアドレスの下位ワードは丸められることに注意してください。たとえば、0x11110001 と指定すると、丸められて 0x11110000 になります。

DLL の署名プロセスを完了するには、SN.EXE の –R を使用します。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [詳細] をクリックします。
4. DLL のベース アドレス プロパティを変更します。

   このコンパイラ オプションをコードから設定するには、「BaseAddress」を参照してください。

参照
その他の技術情報
C# コンパイラ オプション

/bugreport (問題点のレポート)

後で分析を行うことができるように、デバッグ情報をファイルに出力するように指定します。

```
/bugreport:file
```

引数
file

バグ レポートを作成するファイルの名前。

解説

/bugreport オプションを指定すると、*file* には次の情報が出力されます。

- コンパイル時のすべてのソース コード ファイルのコピー。
- コンパイルで使用されたコンパイラ オプションの一覧。
- コンパイラ、ランタイム、およびオペレーティング システムのバージョン情報。
- .NET Framework (SDK を含む) に付属のアセンブリを除く参照アセンブリおよびモジュール (16 進数として保存)。
- コンパイラの出力 (指定されている場合)。
- 問題点の説明。プロンプトが表示されます。
- 問題点の修正方法の説明。プロンプトが表示されます。

このオプションに **/errorreport:prompt** または **/errorreport:send** を指定すると、ファイルに保存される情報が Microsoft Corporation に送信されます。

すべてのソース コード ファイルのコピーが *file* に組み込まれてしまうため、問題があると思われるコードをできるだけ小さなプログラムとして再生成することがあります。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

生成されたファイルの内容により、ソース コードが公開され、情報が誤って暴露される場合があることに注意してください。

参照
関連項目
/errorreport (エラー報告動作の設定) (C# コンパイラ オプション)
その他の技術情報
C# コンパイラ オプション

/checked (整数算術演算のチェック) (C# コンパイラ オプション)

/checked オプションは、checked キーワードまたは unchecked キーワードのスコープ外にある整数算術演算ステートメントの結果がデータ型の範囲外の値になった場合に、実行時に例外を発生させるかどうかを指定します。

```
/checked[+ | <U>-</U>]
```

解説

checked キーワードまたは **unchecked** キーワードのスコープ内にある整数算術演算ステートメントは、**/checked** オプションの影響を受けません。

checked キーワードまたは **unchecked** キーワードのスコープ外にある整数算術演算ステートメントの結果がデータ型の範囲外の値になる場合、コンパイル時に **/checked+ (/checked)** オプションを指定すると、そのステートメントは実行時に例外を発生します。コンパイル時に **/checked-** が使用された場合、そのステートメントは実行時に例外を発生しません。

このオプションの既定値は **/checked-** なので、オプションを付けなかった場合も同じ動作になります。**/checked-** を使用する場面としては、大規模なアプリケーションのビルドがあります。このようなアプリケーションのビルドでは、自動化ツールが使用されることがあります。このツールで、**/checked** が自動的に + に設定されるような場合に、**/checked-** を指定することによって、グローバルな既定値をオーバーライドできます。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [詳細] をクリックします。
4. [演算のオーバーフローおよびアンダーフローのチェック] プロパティを変更します。

プログラムによってこのコンパイラ オプションにアクセスする方法については、「CheckForOverflowUnderflow」を参照してください。

使用例

`t2.cs` をコンパイルするとき、**checked** キーワードまたは **unchecked** キーワードのスコープ外にある整数算術演算ステートメントの結果がデータ型の範囲外の値になった場合に実行時例外を発生させるには、次のコードを使用します。

```
csc t2.cs /checked
```

参照

その他の技術情報

C# コンパイラ オプション

/codepage (ソース コード ファイルのコード ページの指定) (C# コンパイラ オプション)

このオプションは、コンパイル時に使用するコード ページを指定します。システムで現在使用されている既定のコードページ以外のコードページを使用する場合に使用します。

```
/codepage:id
```

引数
id

コンパイル時にすべてのソース コード ファイルで使うコード ページの ID。

解説

コンピュータの既定のコード ページを使わずに作成されたソース コード ファイルをコンパイルする場合は、**/codepage** オプションで使用するコード ページを指定できます。**/codepage** は、コンパイルするすべてのソース コード ファイルに適用されます。

ソース コードの作成時に使用されたコード ページがコンピュータで有効なコード ページと同じ場合、または UNICODE か UTF-8 の場合は、**/codepage** を使う必要はありません。

システムでサポートされているコード ページを探す方法については、「GetCPInfo」を参照してください。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

参照
その他の技術情報
C# コンパイラ オプション

/debug (デバッグ情報の生成) (C# コンパイラ オプション)

/debug オプションを指定すると、コンパイラによってデバッグ情報が生成され、出力ファイルに格納されます。

```
/debug[+ | <U>-</U>]
/debug:{<U>full</U> | pdbonly}
```

引数

+ | -

+ を指定するか、または **/debug** だけを指定すると、コンパイラによってデバッグ情報が生成され、プログラム データベース (.pdb) ファイルにその情報が出力されます。**-** を指定すると、デバッグ情報は作成されません。**/debug** を指定しない場合は、**-** が有効になります。

full | pdbonly

コンパイラによって生成されるデバッグ情報の種類を指定します。**/debug:pdbonly** を指定しない場合、つまり full 引数を使用すると、実行中のプログラムにデバッガをアタッチできます。pdbonly を指定すると、プログラムがデバッガで開始されたときにはソース コードをデバッグできますが、実行中のプログラムをデバッガにアタッチしたときはアセンブラしか表示されません。

解説

このオプションを使用してデバッグ ビルドを作成します。**/debug**、**/debug+**、**/debug:full** のいずれも指定しなかった場合、プログラムの出力ファイルをデバッグすることはできません。

/debug:full を使用する場合は、JIT によって最適化されるコードの速度とサイズに若干影響が生じる点に注意してください。また、**/debug:full** でデバッグした場合、わずかではありますが、コードの品質にも影響が生じます。リリース バージョンのコードには、**/debug:pdbonly** を使用するか、PDB を一切使用しないことをお勧めします。

メモ :

/debug:pdbonly と **/debug:full** の唯一の違いは、**/debug:full** でコンパイルした場合、デバッグ情報が利用可能であることを JIT コンパイラに通知するための DebuggableAttribute が生成される点です。したがって、**/debug:full** を使用する場合に、コード内で **DebuggableAttribute** が false に設定されていると、エラーが生成されます。

アプリケーションのデバッグ パフォーマンスを構成する方法の詳細については、「イメージのデバッグの簡略化」を参照してください。

.pdb ファイルの場所を変更する方法については、「/pdb (デバッグ シンボル ファイルの指定) (C# コンパイラ オプション)」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [詳細] をクリックします。
4. [デバッグ情報] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「DebugSymbols」を参照してください。

使用例

デバッグ情報をファイル `app.pdb` に出力する例を次に示します。

```
csc /debug /out:app.pdb test.cs
```

参照

その他の技術情報

C# コンパイラ オプション

/define (プリプロセッサ定義) (C# コンパイラ オプション)

/define オプションは、*name* をプログラム内のシンボルとして定義します。

```
/define:name[;name2]
```

引数
name, name2

定義する 1 つ以上のシンボルの名前。

解説

/define オプションには、ソース ファイルで #define プリプロセッサ ディレクティブを使用する場合と同じ効果があります。ソース ファイルの undef ディレクティブがこの定義を削除するか、またはコンパイラがファイルの終端に達するまで、シンボルは削除されません。

このオプションで作成されるシンボルを #if、#else、#elif、および #endif で使用すると、ソース ファイルを条件付きでコンパイルできます。

/d は **/define** の省略形です。

/define では、シンボル名をセミコロンまたはコンマで区切ることにより、複数のシンボルを定義できます。次に例を示します。

```
/define:DEBUG;TUESDAY
```

C# コンパイラ自体は、ソース コードで使用できるシンボルやマクロを定義しません。すべてのシンボル定義はユーザーが定義する必要があります。

メモ :

C++ などの言語とは異なり、C# の **#define** では、シンボルに値を割り当てることはできません。たとえば、**#define** を使用して、マクロを作成したり、定数を定義したりすることはできません。定数を定義する必要がある場合は、**enum** 変数を使用します。C++ のようなマクロを作成する場合は、ジェネリックなどで代用してください。マクロはエラーを招きやすいため、C# では、マクロの使用を禁止し、代わりに、より安全な方法を提供しています。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [条件付きコンパイル シンボル] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「DefineConstants」を参照してください。

使用例

```
// preprocessor_define.cs
// compile with: /define:xx
// or uncomment the next line
// #define xx
using System;
public class Test
{
    public static void Main()
    {
        #if (xx)
            Console.WriteLine("xx defined");
        #else
            Console.WriteLine("xx not defined");
        #endif
    }
}
```

参照

その他の技術情報

C# コンパイラ オプション

/delaysign (アセンブリの遅延署名) (C# コンパイラ オプション)

このオプションを使用すると、デジタル署名を後で追加できるように、コンパイラは出力ファイルに署名用のスペースを予約します。

```
/delaysign[ + | - ]
```

引数

+ | -

完全署名されたアセンブリを作成する場合は、**/delaysign-** を使用します。アセンブリに公開キーだけを含める場合は、**/delaysign+** を使います。既定値は **/delaysign-** です。

解説

/delaysign オプションは、/keyfile または /keycontainer と共に使用した場合にだけ有効です。

完全署名されたアセンブリを要求すると、コンパイラはマニフェスト (アセンブリ メタデータ) を含むファイルをハッシュし、そのハッシュに秘密キーで署名します。結果として得られるデジタル署名は、マニフェストを含むファイルに格納されます。アセンブリを遅延署名に設定すると、コンパイラは署名の計算も格納も行いませんが、後で署名を追加できるようにファイルに領域を確保します。

たとえば、**/delaysign+** を指定すると、テスト時にはアセンブリをグローバル キャッシュに格納できます。テスト後に、アセンブリ リンカ ユーティリティを使用してアセンブリに秘密キーを追加することにより、そのアセンブリに完全署名できます。

詳細については、「厳密な名前付きアセンブリの作成と使用」および「アセンブリへの遅延署名」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [署名] プロパティ ページをクリックします。
3. [遅延署名のみ] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「DelaySign」を参照してください。

参照

その他の技術情報

C# コンパイラ オプション

/doc (ドキュメント コメントの処理) (C# コンパイラ オプション)

/doc オプションを使用すると、XML ファイル内にドキュメント コメントを含めることができます。

```
/doc:file
```

引数
file

コンパイルするソース コード ファイル内のコメントが出力される XML 形式の出力ファイル。

解説

ソース コード ファイルでは、次の項目の前にあるドキュメント コメントを処理して、XML ファイルに追加できます。

- クラス、デリゲート、インターフェイスなどのユーザー定義型
- フィールド、イベント、プロパティ、メソッドなどのメンバ

Main を含むソース コード ファイルが最初に XML に出力されます。

生成された .xml ファイルで IntelliSense 機能を使用するには、サポートするアセンブリの名前と .xml ファイル名を同じにして、そのファイルをアセンブリと同じディレクトリに置いてください。これで、アセンブリが Visual Studio プロジェクトで参照されると、.xml ファイルも同様に検出されます。詳細については、「コード コメントの追加」を参照してください。

/target:module を使用してコンパイルしない限り、*file* は、コンパイルの出力ファイルのアセンブリ マニフェストを含んでいるファイルの名前を指定する <assembly> </assembly> タグを含みます。

メモ :

/doc オプションは、すべての入力ファイル (プロジェクトの設定で行った場合は、そのプロジェクト内のすべてのファイル) に適用されます。特定のファイルまたはコードの特定セクションについて、ドキュメントのコメントに関する警告を無効にするには、#pragma warning を使用します。

コードのコメントからドキュメントを生成する方法については、「ドキュメント コメントとして推奨されるタグ」を参照してください。

具体的な例については、「XML ドキュメントのサンプル」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [XML ドキュメント ファイル] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「DocumentationFile」を参照してください。

参照
処理手順
XML ドキュメントのサンプル
その他の技術情報
C# コンパイラ オプション

/errorreport (エラー報告動作の設定) (C# コンパイラ オプション)

このオプションは、C# 内部コンパイル エラーを Microsoft に報告するときの便利な方法として機能します。

```
/errorreport:{ none | prompt | queue | send }
```

引数

none

内部コンパイラ エラーを収集しない、またはマイクロソフトに送信しないことを報告します。

prompt

内部コンパイラ エラーを受け取ったときに、レポートを送信するためのダイアログ ボックスを表示します。開発環境でアプリケーションをコンパイルするときは、**prompt** が既定の値です。

queue

エラー レポートをキューに配置します。管理者権限を使ってログインするとポップアップ ウィンドウが表示され、前回のログイン以降に発生したエラーを報告できます。エラー レポートを送信するためのダイアログ ボックスは、3 日に 1 度表示されます。アプリケーションをコマンド ラインからコンパイルするときは、**queue** が既定の値です。

send

内部コンパイラ エラーのレポートをマイクロソフトに自動的に送信します。このオプションを有効にするには、まずマイクロソフトのデータ コレクション ポリシーに同意する必要があります。コンピュータで **/errorreport:send** を初めて指定すると、マイクロソフトのデータ コレクション ポリシーが記載されている Web サイトが表示されます。

解説

コンパイラがソース コード ファイルを処理できないと、内部コンパイラ エラー (ICE: Internal Compiler Error) が発生します。ICE が発生した場合は、コードの修正に利用できる出力ファイルや診断は生成されません。

以前のリリースでは、ICE が発生した場合はマイクロソフトのテクニカル サポートに連絡して問題を報告することをお勧めしていました。**/errorreport** を使用すると、ICE 情報を Visual C# チームに直接提供できます。エラー レポートは、今後リリースされるコンパイラの機能向上に役立ちます。

ユーザーが使用できるレポート送信機能は、使用しているコンピュータやユーザー ポリシーのアクセス許可に応じて異なります。

エラー デバッガの詳細については、「Description of the Dr. Watson for Windows (Drwtsn32.exe) Tool」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [詳細] をクリックします。
4. [内部コンパイル エラー報告] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「ErrorReport」を参照してください。

参照

その他の技術情報

C# コンパイラ オプション

/filealign (セクション配置の指定) (C# コンパイラ オプション)

/filealign オプションを使用すると、出力ファイル内のセクションのサイズを指定できます。

```
/filealign:number
```

引数
number

出力ファイル内のセクションのサイズを指定する値。有効な値は 512、1024、2048、4096、および 8192 です (単位はバイト)。

解説

各セクションは、**/filealign** 値の倍数となる境界ごとに配置されます。固定された既定値はありません。**/filealign** が指定されていない場合は、共通言語ランタイムによりコンパイル時に既定値が選択されます。

セクションのサイズを指定すると、出力ファイルのサイズに影響します。セクションのサイズの変更は、容量の小さなデバイス上で動作するプログラムに対して有効な場合があります。

出力ファイルのセクションに関する情報を参照するには、DUMPBIN を使用します。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [詳細] をクリックします。
4. [ファイル アライメント] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「FileAlignment」を参照してください。

参照
その他の技術情報
C# コンパイラ オプション

/fullpaths (コンパイラ出力の絶対パスの指定) (C# コンパイラ オプション)

/fullpaths オプションは、コンパイラに対し、コンパイルのエラーや警告を表示するときにそのファイルの完全パスを指定するように指示します。

```
/fullpaths
```

解説

既定では、コンパイル時のエラーおよび警告では、エラーが見つかったファイルの名前が示されます。**/fullpaths** オプションは、コンパイラにそのファイルの完全パスを指定するように指示します。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

参照

その他の技術情報

C# コンパイラ オプション

/help、/?(コンパイラのコマンド ラインのヘルプ) (C# コンパイラ オプション)

このオプションは、標準出力にコンパイラ オプションの一覧と各オプションの簡単な説明を出力します。

```
/help
/?
```

解説

コンパイル時にこのオプションを指定すると、出力ファイルは作成されず、コンパイルも実行されません。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

参照

その他の技術情報

C# コンパイラ オプション

/keycontainer (厳密名キー コンテナの指定) (C# コンパイラ オプション)

暗号化キー コンテナの名前を指定します。

```
/keycontainer:string
```

引数

用語	定義
string	厳密名キーのコンテナ名。

解説

/keycontainer オプションが指定された場合、コンパイラは、指定されたコンテナに格納された公開キーをアセンブリ マニフェストに追加し、最終的なアセンブリを秘密キーで署名することによって、共有可能なコンポーネントを生成します。キー ファイルを生成するには、コマンド ラインで「sn -k *file*」と入力します。sn -i は、キー ペアをコンテナに組み込みます。

/target:module を使用してコンパイルすると、キー ファイル名はモジュールに保持され、/addmodule を使用してこのモジュールをコンパイルするときに作成されるアセンブリに組み込まれます。

このオプションは、任意の Microsoft Intermediate Language (MSIL) モジュールのソース コードでカスタム属性 (System.Reflection.AssemblyKeyNameAttribute) として指定することもできます。

/keyfile を使用して、暗号に関する情報をコンパイラに渡すこともできます。公開キーのみアセンブリ マニフェストに追加しておき、アセンブリへの署名については、テストが終わるまで保留にする場合は、/delaysign を使用します。

詳細については、「厳密な名前付きアセンブリ」および「アセンブリへの遅延署名」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

- このコンパイラ オプションは、Visual Studio 開発環境では使用できません。

このコンパイラ オプションには、AssemblyKeyContainerName を使用してプログラムでアクセスできます。

参照

その他の技術情報

C# コンパイラ オプション

/keyfile (厳密名キー ファイルの指定) (C# コンパイラ オプション)

暗号化キーの格納されたファイル名を指定します。

```
/keyfile:file
```

引数

用語	定義
file	厳密名キーが格納されたファイルの名前。

解説

このオプションを使用すると、コンパイラによって、指定したファイルに格納されている公開キーがアセンブリ マニフェストに対して追加され、最終的なアセンブリが秘密キーで署名されます。キー ファイルを生成するには、コマンド ラインで「sn -k *file*」と入力します。

/target:module を使用してコンパイルすると、キー ファイル名はモジュールに保持され、/addmodule でアセンブリをコンパイルするときに作成されるアセンブリに組み込まれます。

/keycontainer を使用して、暗号に関する情報をコンパイラに渡すこともできます。部分署名されたアセンブリを作成する場合は、/delaysign を使用します。

コマンド ライン オプションまたはカスタム属性によって、コンパイル時に /keyfile と /keycontainer の両方が同時に指定されると、コンパイラは先にキー コンテナを処理します。コンテナが検出された場合、アセンブリはキー コンテナの情報で署名されます。キー コンテナが見つからない場合、コンパイラは /keyfile で指定されたファイルを処理します。成功すると、キー ファイル内の情報を使用して、アセンブリが署名されます。キー情報はキー コンテナに組み込まれるため (sn -i と同じ)、次回のコンパイルではキー コンテナが有効になります。

キー ファイルには公開キーだけが含まれる場合があることに注意してください。

詳細については、「厳密な名前付きアセンブリの作成と使用」および「アセンブリへの遅延署名」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [署名] プロパティ ページをクリックします。
3. [厳密な名前のキー ファイルを選択してください] プロパティを変更します。

このコンパイラ オプションには、AssemblyOriginatorKeyFile を使用してプログラムでアクセスできます。

参照

その他の技術情報

C# コンパイラ オプション

/langversion (準拠構文) (C# コンパイラ オプション)

コンパイラが、ISO/IEC 23270:2003 の C# 言語仕様に含まれている構文のみを受け入れるようにします。

```
/langversion:option
```

引数
option

option が **ISO-1** の場合、コンパイラは、ISO/IEC 23270:2003 の C# 言語仕様に含まれていない構文に対してエラーを生成します。*option* が **default** の場合、コンパイラは有効なすべての構文を受け入れます。既定値は **/langversion:default** です。

解説

C# 仕様の Version 1.0 は、**/langversion:ISO-1** で利用可能な機能を表しています。http://msdn.microsoft.com/vcsharp/programming/language/default.aspx には、すべての仕様が Microsoft Word ファイルとして登録されています。

C# アプリケーションによって参照されるメタデータは、**/langversion** コンパイラ オプションの影響を受けません。

C# コンパイラの各バージョンには、言語仕様の拡張機能が含まれているため、**/langversion** は、コンパイラの以前のバージョンと同等の機能を提供しません。

どの **/langversion** 設定を使用する場合でも、.exe や .dll を作成するには、共通言語ランタイムの現在のバージョンを使用します。

ただし、**/langversion:ISO-1** に従って動作するフレンド アセンブリと /moduleassemblyname (モジュールに対するフレンド アセンブリの指定) (C# コンパイラ オプション) は例外です。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [詳細] をクリックします。
4. [言語バージョン] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「LanguageVersion」を参照してください。

参照
その他の技術情報
C# コンパイラ オプション

/lib (アセンブリ参照場所の指定) (C# コンパイラ オプション)

/lib オプションでは、/reference (メタデータのインポート) (C# コンパイラ オプション) オプションによって参照されるアセンブリの場所を指定します。

```
/lib:dir1[,dir2]
```

引数
dir1

参照先アセンブリが現在の作業ディレクトリ (コンパイラを起動したディレクトリ) または共通言語ランタイムのシステム ディレクトリに存在しない場合に、コンパイラが探すディレクトリ。

dir2

アセンブリ参照を検索するための 1 つ以上の別のディレクトリ。ディレクトリ名を複数追加する場合は、空白ではなくコンマで区切って指定します。

解説

コンパイラは、完全には修飾されていないアセンブリ参照を次の順序で検索します。

1. 現在の作業ディレクトリ。これは、コンパイラが起動されるディレクトリです。
2. 共通言語ランタイムのシステム ディレクトリ。
3. **/lib** で指定したディレクトリ。
4. LIB 環境変数で指定したディレクトリ。

アセンブリ参照を指定するには、**/reference** を使用します。

/lib は追加して指定できます。繰り返して指定すると前の値に追加されます。

/lib を使用する代わりに、必須のアセンブリをすべて作業ディレクトリにコピーすることもできます。この場合は、アセンブリ名を **/reference** に渡すだけです。その後、作業ディレクトリからアセンブリを削除できます。依存アセンブリのパスはアセンブリ マニフェストで指定されていないため、アプリケーションをターゲット コンピュータで開始でき、グローバル アセンブリ キャッシュ内のアセンブリを検索および使用できます。

コンパイラがアセンブリを参照できる場合でも、共通言語ランタイムが実行時にアセンブリを検索して読み込むことができるとは限りません。実行時に参照アセンブリがどのように検索されるかについては、「ランタイムがアセンブリを検索する方法」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ ページ] ダイアログ ボックスを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [参照パス] プロパティ ページをクリックします。
3. リスト ボックスの内容を変更します。

このコンパイラ オプションをプログラムで設定する方法については、「ReferencePath」を参照してください。

使用例

t2.cs をコンパイルして .exe ファイルを作成するには、次のコードを使用します。コンパイラは、作業ディレクトリと C ドライブのルート ディレクトリでアセンブリ参照を探します。

```
csc /lib:c:\ /reference:t2.dll t2.cs
```

参照
その他の技術情報
C# コンパイラ オプション

/linkresource (.NET Framework リソースへのリンク) (C# コンパイラ オプション)

出力ファイルに、.NET Framework リソースへのリンクを作成します。リソース ファイルを出力ファイルに埋め込む /resource オプションとは異なり、出力ファイルにリソース ファイルは格納されません。

```
/linkresource:filename[,identifier[,accessibility-modifier]]
```

引数
filename

アセンブリからのリンク先である .NET Framework リソース ファイル。

identifier (省略可能)

リソースの論理名。リソースを読み込むときに使用します。既定値はファイル名です。

accessibility-modifier (省略可能)

リソースのアクセシビリティ (public または private)。既定値は public です。

解説

既定では、C# コンパイラを使用して作成したリンク先のリソースは、アセンブリ内でパブリックになります。リソースを private にする場合は、**private** をアクセシビリティ修飾子として指定します。**public** と **private** 以外の修飾子は使用できません。

/linkresource では、**/target:module** 以外のいずれかの /target オプションが必要です。

たとえば、*filename* が Resgen.exe によって作成された .NET Framework リソース ファイルである場合、または開発環境で作成された .NET Framework リソース ファイルである場合は、System.Resources 名前空間のメンバを使用してアクセスできます。詳細については、「System.Resources.ResourceManager」を参照してください。その他のすべてのリソースの場合は、Assembly クラスの **GetManifestResource*** メソッドを使用して実行時にリソースにアクセスします。

filename に指定するファイルの形式は任意です。たとえば、ネイティブ DLL をアセンブリの一部として含め、そのネイティブ DLL をグローバル アセンブリ キャッシュにインストールして、アセンブリ内のマネージ コードからアクセスできるようにすることもできます。以下の 2 つ目の例では、この方法が示されています。これと同じことは、アセンブリ リンカで行うこともできます。以下の 3 つ目の例では、この方法が示されています。詳細については、「アセンブリ リンカ (Al.exe)」および「アセンブリとグローバル アセンブリ キャッシュの使用」を参照してください。

/linkres は **/linkresource** の省略形です。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

使用例

`in.cs` をコンパイルし、リソース ファイル `rf.resource` にリンクさせる例を次に示します。

```
csc /linkresource:rf.resource in.cs
```

`A.cs` をコンパイルして DLL を作成し、ネイティブの DLL N.dll にリンクして、出力をグローバル アセンブリ キャッシュ (GAC) に格納します。次の例では、A.dll および N.dll の両方が GAC に格納されます。

```
csc /linkresource:N.dll /t:library A.cs
gacutil -i A.dll
```

次の例では、前の例と同じことをアセンブリ リンカのオプションを使って実行します。

```
csc /t:module A.cs
al /out:A.dll A.netmodule /link:N.dll
gacutil -i A.dll
```

参照
関連項目

アセンブリ リンカ (Al.exe)

その他の技術情報

C# コンパイラ オプション

アセンブリとグローバル アセンブリ キャッシュの使用

/main (Main メソッドの場所の指定) (C# コンパイラ オプション)

このオプションは、**Main** メソッドを持つクラスが複数存在する場合に、プログラムのエントリ ポイントとして使用するクラスを指定します。

```
/main:class
```

引数
class

Main メソッドを含む型。

解説

Main メソッドを持つ型を複数コンパイルする場合は、プログラムへのエントリ ポイントとして使用する **Main** メソッドをどの型が含んでいるかを指定できます。

このオプションは、.exe ファイルをコンパイルする場合に使用します。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [アプリケーション] プロパティ ページをクリックします。
3. [スタートアップの設定] プロパティを変更します。

   このコンパイラ オプションをコードから設定するには、「StartupObject」を参照してください。

使用例

`t2.cs` および `t3.cs` をコンパイルし、**Main** メソッドが `Test2` にあることを指定するには、次のコードを使用します。

```
csc t2.cs t3.cs /main:Test2
```

参照
その他の技術情報
C# コンパイラ オプション

/moduleassemblyname (モジュールに対するフレンド アセンブリの指定) (C# コンパイラ オプション)

.netmodule からアクセスできる、パブリック型ではないアセンブリを指定します。

```
/moduleassemblyname:assembly_name
```

引数
assembly_name

.netmodule からアクセスできる、パブリック型ではないアセンブリの名前。

解説

/moduleassemblyname は、.netmodule をビルドするとき、次の条件に当てはまる場合に使用します。

- .netmodule から、既存のアセンブリにあるパブリック以外の型にアクセスする必要がある場合。
- .netmodule をビルドするアセンブリの名前がわかっている場合。
- 既存のアセンブリで、.netmodule をビルドするアセンブリに対して、フレンド アセンブリのアクセス権を認めている場合。

.netmodule のビルドの詳細については、「/target:module (アセンブリに追加するモジュールの作成) (C# コンパイラ オプション)」を参照してください。

フレンド アセンブリの詳細については、「フレンド アセンブリ (C# プログラミング ガイド)」を参照してください。

このオプションは開発環境内では利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

使用例

この例では、プライベート型のアセンブリをビルドし、フレンド アセンブリのアクセス権を csman_an_assembly というアセンブリに許可しています。

```
// moduleassemblyname_1.cs
// compile with: /target:library
using System;
using System.Runtime.CompilerServices;

[assembly:InternalsVisibleTo ("csman_an_assembly")]

class An_Internal_Class
{
    public void Test()
    {
        Console.WriteLine("An_Internal_Class.Test called");
    }
}
```

この例では、アセンブリ moduleassemblyname_1.dll に含まれるパブリック以外の型にアクセスする .netmodule をビルドします。この .netmodule は csman_an_assembly というアセンブリにビルドされることがわかっているため、**/moduleassemblyname** を指定して、.netmodule から、csman_an_assembly に対するフレンド アセンブリのアクセス権を許可したアセンブリに含まれる、パブリック以外の型にアクセスできます。

```
// moduleassemblyname_2.cs
// compile with: /moduleassemblyname:csman_an_assembly /target:module /reference:moduleasse
mblyname_1.dll
class B {
    public void Test() {
        An_Internal_Class x = new An_Internal_Class();
        x.Test();
    }
```

```
}
```

このコード例では、アセンブリ csman_an_assembly をビルドし、以前にビルドしたアセンブリと .netmodule を参照します。

```
// csman_an_assembly.cs
// compile with: /addmodule:moduleassemblyname_2.netmodule /reference:moduleassemblyname_1.
dll
class A {
    public static void Main() {
        B bb = new B();
        bb.Test();
    }
}
```

出力

```
An_Internal_Class.Test called
```

参照

その他の技術情報

C# コンパイラ オプション

/noconfig (csc.rsp の無視) (C# コンパイラ オプション)

/noconfig オプションは、コンパイルに csc.rsp ファイルを使用しないようにコンパイラに指示します。csc.rsp ファイルは、csc.exe ファイルと同じディレクトリにあり、そこから読み込まれます。

```
/noconfig
```

解説

csc.rsp ファイルは、.NET Framework に付属のすべてのアセンブリを参照します。Visual Studio .NET 開発環境に含まれる実際の参照は、プロジェクトの種類に応じて異なります。

csc.rsp ファイルを変更し、csc.exe を使用したコマンド ラインからのコンパイルのたびに含める追加のコンパイラ オプションを指定できます (**/noconfig** オプションを除く)。

コンパイラは、**csc** コマンドに渡されるオプションを最後に処理します。このため、コマンド ラインに指定したオプションは、csc.rsp ファイル内の同じオプションの設定をオーバーライドします。

コンパイラで csc.rsp ファイルの設定を検索および使用しない場合は、**/noconfig** を指定します。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

参照

その他の技術情報

C# コンパイラ オプション

/nologo (著作権情報の非表示) (C# コンパイラ オプション)

/nologo オプションは、コンパイラ起動時の著作権情報や、コンパイル中の情報メッセージを表示しないようにします。

```
/nologo
```

解説

このオプションは開発環境内では利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

参照

その他の技術情報

C# コンパイラ オプション

/nostdlib (標準ライブラリのインポート禁止) (C# コンパイラ オプション)

/nostdlib は、System 名前空間の全体を定義する mscorlib.dll がインポートされないようにします。

```
/nostdlib[<U>+</U> | -]
```

解説

独自の System 名前空間およびオブジェクトを定義または作成する場合は、このオプションを使用します。

/nostdlib を指定しないと、**/nostdlib-** を指定したことと同じで、mscorlib.dll がプログラムにインポートされます。**/nostdlib** の指定は、**/nostdlib+** の指定と同じです。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [詳細] をクリックします。
4. [mscorlib.dll を参照しない] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「NoStdLib」を参照してください。

参照
その他の技術情報
C# コンパイラ オプション

/nowarn (指定した警告の非表示) (C# コンパイラ オプション)

/nowarn オプションを使用すると、コンパイラからの警告が出力されないようにできます。警告番号が複数ある場合は、コンマで区切ります。

```
/nowarn:number1[,number2,...]
```

引数
number1, *number2*

コンパイラで表示しないようにする警告の番号

解説

必要なのは、警告 ID の数値を指定することだけです。たとえば、CS0028 を表示しない場合は、`/nowarn:28` と指定します。

コンパイラは、**/nowarn** に渡された警告番号のうち、以前のリリースで有効であり、今はコンパイラからは削除されている番号は無視します。たとえば、CS0679 は、Visual Studio .NET 2002 のコンパイラでは有効でしたが、それ以降は削除されています。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [警告の表示なし] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「DelaySign」を参照してください。

参照
その他の技術情報
C# コンパイラ オプション

/optimize (最適化の有効化/無効化) (C# コンパイラ オプション)

/optimize オプションは、コンパイラで行われる最適化の有効と無効を切り替えて、出力ファイルのサイズを小さくし、速度と効率を高めます。

```
/optimize[+ | <U>-</U>]
```

解説

また、実行時にコードを最適化するように共通言語ランタイムに指示します。

既定では、最適化は無効です。最適化を有効にするには、**/optimize+** を指定します。

アセンブリで使用するモジュールをビルドする場合は、アセンブリと同じ **/optimize** の設定を使用してください。

/o は **/optimize** の省略形です。

/optimize オプションと /debug オプションを組み合わせることができます。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [コードの最適化] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「Optimize」を参照してください。

使用例

`t2.cs` をコンパイルし、コンパイラの最適化を有効にするには、次のコードを使用します。

```
csc t2.cs /optimize
```

参照

その他の技術情報

C# コンパイラ オプション

/out (出力ファイル名の設定) (C# コンパイラ オプション)

/out オプションは、出力ファイルの名前を指定します。

```
/out:filename
```

引数
filename

コンパイラで作成される出力ファイルの名前。

解説

コマンド ラインでは、コンパイルに対して複数の出力ファイルを指定できます。**/out** オプションの後に、1 つ以上のソース コード ファイルを指定します。**/out** オプションの後に指定されたソース コード ファイルはすべて、その **/out** オプションに指定された出力ファイルにコンパイルされます。

作成するファイルについて、フルネームと拡張子を指定します。

出力ファイルの名前を指定しない場合は、次のように処理されます。

- .exe の名前は、**Main** メソッドを含むソース コード ファイルと同じになります。
- .dll または .netmodule の名前は、最初のソース コード ファイルと同じになります。

ある出力ファイルのコンパイルに使用されるソース コード ファイルは、同じコンパイルで別の出力ファイルにコンパイルできません。

複数の出力ファイルを 1 回のコマンド ライン コンパイルで作成する場合、アセンブリにできるのは、出力ファイルのうちの 1 ファイルだけであり、暗黙にまたは **/out** を使用して明示的に指定した最初の出力ファイルだけがアセンブリになります。

コンパイルで作成されたモジュールはすべて、そのコンパイルで作成されたアセンブリに関連付けられるファイルになります。ildasm.exe を使用してアセンブリ マニフェストを表示し、関連付けられたファイルを確認してください。

target オプションに exe ファイルを指定してフレンド アセンブリを追加するには、/out コンパイラ オプションが必要です。詳細については、「フレンド アセンブリ (C# プログラミング ガイド)」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [アプリケーション] プロパティ ページをクリックします。
3. [アセンブリ名] プロパティを変更します。

   このコンパイラ オプションをプログラムから設定するには: OutputFileName は読み取り専用のプロパティです。プロジェクトの種類 (exe、ライブラリなど) と、アセンブリ名の組み合わせに基づいて決定されます。これらのプロパティの一方または両方を変更するには、出力ファイル名を設定する必要があります。

使用例

`t.cs` をコンパイルして出力ファイル `t.exe` を作成し、`t2.cs` をビルドしてモジュール出力ファイル `mymodule.netmodule` を作成するには、次のコードを使用します。

```
csc t.cs /out:mymodule.netmodule /target:module t2.cs
```

参照
関連項目
フレンド アセンブリ (C# プログラミング ガイド)
その他の技術情報
C# コンパイラ オプション

/pdb (デバッグ シンボル ファイルの指定) (C# コンパイラ オプション)

/pdb コンパイラ オプションでは、デバッグ シンボル ファイルの名前と場所を指定します。

```
/pdb:filename
```

引数
filename

デバッグ シンボル ファイルの名前と場所。

解説

/debug (デバッグ情報の生成) (C# コンパイラ オプション) を指定すると、コンパイラにより、出力ファイル (.exe または .dll) が作成されるのと同じディレクトリに、出力ファイルと同じ名前の .pdb ファイルが作成されます。

/pdb を使用すると、.pdb ファイルに既定以外のファイル名と場所を指定できます。

このコンパイラ オプションは、Visual Studio 開発環境で設定することも、プログラムから変更することもできません。

使用例

`t.cs` をコンパイルし、tt.pdb という名前の .pdb ファイルを作成します。

```
csc /debug /pdb:tt t.cs
```

参照
その他の技術情報
C# コンパイラ オプション

/platform (出力プラットフォームの指定) (C# コンパイラ オプション)

アセンブリをどのバージョンの共通言語ランタイム (CLR: Common Language Runtime) で実行するかを指定します。

```
/platform:string
```

パラメータ
string

x86、Itanium、x64、または anycpu (既定値)。

解説

- x86 を指定すると、32 ビットの x86 互換共通言語ランタイムで実行されるアセンブリがコンパイルされます。
- Itanium を指定すると、Itanium プロセッサ搭載コンピュータ上の 64 ビット共通言語ランタイムで実行されるアセンブリがコンパイルされます。
- x64 を指定すると、AMD64 または EM64T 命令セットをサポートするコンピュータ上の 64 ビット共通言語ランタイムで実行されるアセンブリがコンパイルされます。
- anycpu (既定値) を指定すると、任意のプラットフォームで実行されるアセンブリがコンパイルされます。

64 ビットの Windows オペレーティング システムでは次のようになります。

- **/platform:x86** を指定してコンパイルしたアセンブリは、WOW64 環境の 32 ビット CLR 上で実行されます。
- **/platform:anycpu** を指定してコンパイルした実行可能ファイルは、64 ビット CLR 上で実行されます。
- **/platform:anycpu** を指定してコンパイルした DLL は、読み込み先のプロセスと同じ CLR 上で実行されます。

64 ビットの Windows オペレーティング システム上で実行されるアプリケーションの開発の詳細については、「64 ビット アプリケーション」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [プラットフォーム ターゲット] プロパティを変更します。

メモ **/platform** は、Visual C# Express の開発環境では使用できません。

このコンパイラ オプションをプログラムで設定する方法については、「PlatformTarget」を参照してください。

使用例

次の例は、**/platform** オプションを使用して、Itanium 搭載の 64 ビット Windows オペレーティング システム上の 64 ビット CLR でのみ実行されるアプリケーションをコンパイルする方法を示しています。

```
csc /platform:Itanium myItanium.cs
```

参照
その他の技術情報
C# コンパイラ オプション

/recurse (サブディレクトリでのソース ファイルの検索) (C# コンパイラ オプション)

/recurse オプションを使用すると、指定のディレクトリ (dir) またはプロジェクト ディレクトリのすべての子ディレクトリ内のソース コード ファイルをコンパイルできます。

```
/recurse:[dir\]file
```

引数
dir (省略可能)

検索を開始するディレクトリ。指定しない場合は、プロジェクト ディレクトリから検索されます。

file

検索するファイル。ワイルドカード文字を使用できます。

解説

/recurse オプションを使用すると、指定のディレクトリ (*dir*) またはプロジェクト ディレクトリのすべての子ディレクトリ内のソース コード ファイルをコンパイルできます。

/recurse を使用しなくても、ファイル名にワイルドカードを使用すると、プロジェクト ディレクトリ内で一致するすべてのファイルをコンパイルできます。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

使用例

現在のディレクトリ内のすべての C# ファイルをコンパイルするには、次のコードを使用します。

```
csc *.cs
```

dir1\dir2 ディレクトリおよびそのディレクトリの下の全ディレクトリ内の C# ファイルすべてをコンパイルし、dir2.dll を生成するには、次のコードを使用します。

```
csc /target:library /out:dir2.dll /recurse:dir1\dir2\*.cs
```

参照
その他の技術情報
C# コンパイラ オプション

/reference (メタデータのインポート) (C# コンパイラ オプション)

/reference オプションは、指定したファイル内で public (C# リファレンス) として宣言された型の情報を現在のプロジェクトにインポートするよう、コンパイラに対して指示します。こうすることによって、指定したアセンブリ ファイルのメタデータを参照できます。

```
/reference:[alias=]filename
/reference:filename
```

引数
filename

アセンブリ マニフェストを含むファイルの名前。複数のファイルをインポートするには、ファイルごとに **/reference** オプションを指定します。

alias

(アセンブリ内のすべての名前空間を包含する) ルート名前空間を表す有効な C# 識別子。

解説

複数のファイルをインポートするには、ファイルごとに **/reference** オプションを指定します。

インポートするファイルにはマニフェストが必要です。出力ファイル
は、/target:module (アセンブリに追加するモジュールの作成) (C# コンパイラ オプション) 以外のいずれかの
/target (出力ファイル形式の指定) (C# コンパイラ オプション) オプションを使用してコンパイルしておく必要があります。

/r は **/reference** の省略形です。

アセンブリ マニフェストを含まない出力ファイルからメタデータをインポートするに
は、/addmodule (メタデータのインポート) (C# コンパイラ オプション) を使用します。

別のアセンブリ (アセンブリ B) を参照するアセンブリ (アセンブリ A) を参照するときに、次の状況に該当する場合は、アセンブリ B も参照する必要があります。

- アセンブリ A で使用する型がアセンブリ B の型を継承しているか、アセンブリ B のインターフェイスを実装している場合。
- アセンブリ B の戻り値の型やパラメータの型を持つフィールド、プロパティ、イベント、またはメソッドを呼び出す場合。

/lib (アセンブリ参照場所の指定) (C# コンパイラ オプション) を使用して、1 つ以上のアセンブリ参照があるディレクトリを指定します。**/lib** に関するトピックでも、コンパイラがアセンブリを検索するディレクトリについて説明しています。

モジュールではなくアセンブリ内の型をコンパイラで認識するには、型の解決を強制する必要があります。型の解決を実行するには、たとえば、型のインスタンスを定義します。アセンブリの型名を解決する方法は他にもあります。たとえば、アセンブリの型を継承すると、コンパイラで型名が認識されます。

1 つのアセンブリから、同じコンポーネントの 2 種類のバージョンを参照しなければならない場合もあります。そのためには、ファイルごとに **/reference** スイッチで alias サブオプションを使用し、2 つのファイルを区別します。このエイリアスがコンポーネント名の修飾子として使用され、いずれかのファイルのコンポーネントに解決されます。

既定では、頻繁に使用される .NET Framework アセンブリを参照する csc 応答ファイル (.rsp) が使用されます。コンパイラで csc.rsp を使用しない場合は、/noconfig (csc.rsp の無視) (C# コンパイラ オプション) を使用します。

詳細については、「[参照の追加] ダイアログ ボックス」を参照してください。

例

extern エイリアス (C# リファレンス) 機能を使用する方法の例を次に示します。

このソース ファイルをコンパイルして、あらかじめコンパイルされた `grid.dll` および `grid20.dll` からメタデータをインポートします。この 2 つの DLL には、同じコンポーネントの異なるバージョンがそれぞれ格納されています。エイリアス オプションを付けた 2 つの **/reference** を使用して、ソース ファイルをコンパイルします。オプションの指定例を次に示します。

/reference:GridV1=grid.dll and /reference:GridV2=grid20.dll

これにより、外部のエイリアス "GridV1" および "GridV2" が設定され、プログラム内から extern ステートメントで使用できるようになります。

```
extern GridV1;
extern GridV2;
```

```
// Using statements go here.
```

一度このように設定すれば、コントロール名に GridV1 というプレフィックスを指定することにより、grid.dll から、グリッド コントロールを参照できます。

```
GridV1::Grid
```

また、コントロール名に GridV2 というプレフィックスを指定することにより、grid20.dll から、グリッド コントロールを参照できます。

```
GridV2::Grid
```

参照

その他の技術情報

C# コンパイラ オプション

/resource (出力へのリソース ファイルの埋め込み) (C# コンパイラ オプション)

指定されたリソースを出力ファイルに埋め込みます。

```
/resource:filename[,identifier[,accessibility-modifier]]
```

引数
filename

出力ファイルに埋め込む .NET Framework リソース ファイル。

identifier (省略可能)

リソースの論理名。リソースを読み込むときに使用します。既定値はファイルの名前です。

accessibility-modifier (省略可能)

リソースのアクセシビリティ (public または private)。既定値は public です。

解説

/linkresource を使用すると、リソースがアセンブリにリンクされ、リソース ファイルは出力ファイルに組み込まれません。

既定では、C# コンパイラを使用して作成したリンク先のリソースは、アセンブリ内でパブリックになります。リソースを private にする場合は、**private** をアクセシビリティ修飾子として指定します。**public** と **private** 以外のアクセシビリティは使用できません。

たとえば、*filename* が Resgen.exe によって作成された .NET Framework リソース ファイルである場合、または開発環境で作成された .NET Framework リソース ファイルである場合は、System.Resources 名前空間のメンバを使用してアクセスできます。詳細については、「ResourceManager クラス」を参照してください。その他のすべてのリソースの場合は、Assembly クラスの **GetManifestResource*** メソッドを使用して実行時にリソースにアクセスします。

/res は **/resource** の省略形です。

出力ファイルにおけるリソースの順序は、コマンド ラインでの指定順序に基づいて決定されます。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトにリソース ファイルを追加します。
2. ソリューション エクスプローラで、埋め込むファイルを選択します。
3. 選択したファイルの [プロパティ] ウィンドウで、[ビルド アクション] をクリックします。
4. [ビルド アクション] を [埋め込まれたリソース] に設定します。

このコンパイラ オプションをプログラムで設定する方法については、「BuildAction」を参照してください。

使用例

`in.cs` をコンパイルし、リソース ファイル `rf.resource` をアタッチする例を次に示します。

```
csc /resource:rf.resource in.cs
```

参照
その他の技術情報
C# コンパイラ オプション

/target (出力ファイル形式の指定) (C# コンパイラ オプション)

/target コンパイラ オプションを指定するには、次の 4 つの形式のいずれかを使用します。

/target:exe

.exe ファイルを作成する場合。

/target:library

コード ライブラリを作成する場合。

/target:module

モジュールを作成する場合。

/target:winexe

Windows プログラムを作成する場合。

/target:module を指定しない限り、**/target** を使用すると、.NET Framework のアセンブリ マニフェストが出力ファイルに組み込まれます。詳細については、「共通言語ランタイムのアセンブリ」および「グローバル属性 (C# プログラミング ガイド)」を参照してください。

アセンブリ マニフェストは、コンパイル中の最初の .exe 出力ファイルに組み込まれます。.exe 出力ファイルがない場合には、最初の DLL ファイルに組み込まれます。たとえば、次のコマンド ラインでは、マニフェストは `1.exe` に組み込まれます。

```
csc /out:1.exe t1.cs /out:2.netmodule t2.cs
```

コンパイラの 1 回のコンパイルで作成されるアセンブリ マニフェストは 1 つだけです。コンパイルにおけるすべてのファイルの情報は、アセンブリ マニフェストに含まれます。**/target:module** 以外で作成されたすべての出力ファイルは、アセンブリ マニフェストを含むことができます。コマンド ラインで複数の出力ファイルを生成するときは、アセンブリ マニフェストが 1 つだけ作成されます。このアセンブリ マニフェストは、コマンド ラインで指定した最初の出力ファイルに含める必要があります。最初の出力ファイル (**/target:exe**、**/target:winexe**、**/target:library**、または **/target:module**) にかかわらず、同じコンパイル処理で作成する他の出力ファイルはモジュール (**/target:module**) にする必要があります。

アセンブリを作成する場合は、コードのすべてまたは一部を CLSCompliant 属性に準拠した CLS に指定できます。

```
// target_clscompliant.cs
[assembly:System.CLSCompliant(true)]   // specify assembly compliance

[System.CLSCompliant(false)]   // specify compliance for an element
public class TestClass
{
    public static void Main() {}
}
```

このコンパイラ オプションをプログラムで設定する方法の詳細については、「OutputType」を参照してください。

参照
その他の技術情報
C# コンパイラ オプション

/target:exe (コンソール アプリケーションの作成) (C# コンパイラ オプション)

/target:exe オプションは、実行可能な (EXE) コンソール アプリケーションを作成するようにコンパイラに指示します。

```
/target:exe
```

解説

既定では、**/target:exe** オプションが有効になっています。実行可能ファイルは、.exe という拡張子で作成されます。

Windows プログラムの実行形式を作成するには、/target:winexe を使用します。

/out オプションで特に指定しない限り、出力ファイル名は Main メソッドを含む入力ファイルと同じになります。

コマンド ラインで指定すると、次の **/out** または **/target:module** のオプションまでに指定したすべてのファイルが .exe ファイルを作成するために使用されます。

Main メソッドは、.exe ファイルにコンパイルされるソース コード ファイル内で 1 つだけ必要になります。/main コンパイラ オプションを使用すると、コードの複数のクラスに **Main** メソッドが含まれている場合に、どのクラスの **Main** メソッドを使用するのかを指定できます。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [アプリケーション] プロパティ ページをクリックします。
3. [出力の種類] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「OutputType」を参照してください。

使用例

`in.cs` をコンパイルし、`in.exe` を作成するには、次のいずれかのコマンド ラインを使用します。

```
csc /target:exe in.cs
csc in.cs
```

参照

関連項目

/target (出力ファイル形式の指定) (C# コンパイラ オプション)

その他の技術情報

C# コンパイラ オプション

/target:library (コード ライブラリの作成) (C# コンパイラ オプション)

/target:library オプションは、実行可能ファイル (EXE) ではなくダイナミック リンク ライブラリ (DLL: Dynamic-Link Library) を作成するようにコンパイラに指示します。

```
/target:library
```

解説

作成される DLL の拡張子は .dll です。

/out オプションで指定しない限り、出力ファイル名は最初の入力ファイルと同じになります。

コマンド ラインで指定すると、次の **/out** または **/target:module** のオプションまでに指定したすべてのファイルが .dll ファイルを作成するために使用されます。

.dll ファイルのビルドには、Main メソッドは不要です。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [アプリケーション] プロパティ ページをクリックします。
3. [出力の種類] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「OutputType」を参照してください。

使用例

`in.cs` をコンパイルし、`in.dll` を作成するには、次のコードを使用します。

```
csc /target:library in.cs
```

参照

関連項目

/target (出力ファイル形式の指定) (C# コンパイラ オプション)

その他の技術情報

C# コンパイラ オプション

/target:module (アセンブリに追加するモジュールの作成) (C# コンパイラ オプション)

このオプションは、コンパイルでアセンブリ マニフェストを生成しない場合に使用します。

```
/target:module
```

解説

既定では、このオプションを使ってコンパイルした出力ファイルの拡張子は .netmodule です。

アセンブリ マニフェストを持たないファイルは、.NET Framework 共通言語ランタイムで読み込むことができません。ただし、アセンブリ マニフェストを持たないファイルは、/addmodule を使用してアセンブリのアセンブリ マニフェストに組み込むことができます。

1 回のコンパイルで複数のモジュールが作成される場合、あるモジュールの internal 型をコンパイル中の他のモジュールで利用できます。あるモジュールのコードが別のモジュールの **internal** 型を参照する場合は、**/addmodule** を使用して両方のモジュールを 1 つのアセンブリ マニフェストに組み込む必要があります。

Visual Studio 開発環境では、モジュールの作成はサポートされていません。

このコンパイラ オプションをプログラムで設定する方法については、「OutputType」を参照してください。

使用例

`in.cs` をコンパイルし、`in.netmodule` を作成するには、次のコードを使用します。

```
csc /target:module in.cs
```

参照

関連項目

/target (出力ファイル形式の指定) (C# コンパイラ オプション)

その他の技術情報

C# コンパイラ オプション

/target:winexe (Windows プログラムの作成) (C# コンパイラ オプション)

/target:winexe オプションは、実行可能な (EXE) Windows プログラムを作成するようにコンパイラに指示します。

```
/target:winexe
```

解説

実行可能ファイルは、.exe という拡張子で作成されます。Windows プログラムは、.NET Framework クラス ライブラリまたは Win32 API のユーザー インターフェイスを提供するプログラムです。

コンソール アプリケーションを作成するには、/target:exe を使用します。

/out オプションで特に指定しない限り、出力ファイル名は Main メソッドを含む入力ファイルと同じになります。

コマンド ラインで指定すると、次の **/out** オプションまたは /target オプションまでに指定したすべてのファイルが Windows プログラムの作成に使用されます。

Main メソッドは、.exe ファイルにコンパイルされるソース コード ファイル内で 1 つだけ必要になります。/main オプションを使用すると、コードの複数のクラスに **Main** メソッドが含まれている場合に、どのクラスの **Main** メソッドを使用するのかを指定できます。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [アプリケーション] プロパティ ページをクリックします。
3. [出力の種類] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「OutputType」を参照してください。

使用例

`in.cs` をコンパイルし、Windows プログラムを生成する例を次に示します。

```
csc /target:winexe in.cs
```

参照

関連項目

/target (出力ファイル形式の指定) (C# コンパイラ オプション)

その他の技術情報

C# コンパイラ オプション

/unsafe (unsafe モードの有効化) (C# コンパイラ オプション)

/unsafe コンパイラ オプションは、unsafe キーワードを使用するコードをコンパイルできるようにします。

```
/unsafe
```

解説

アンセーフ コードの詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [アンセーフ コードの許可] チェック ボックスをオンにします。

このコンパイラ オプションをプログラムで設定する方法については、「AllowUnsafeBlocks」を参照してください。

使用例

`in.cs` を unsafe モードでコンパイルするには、次のコードを使用します。

```
csc /unsafe in.cs
```

参照

その他の技術情報

C# コンパイラ オプション

/utf8output (UTF-8 を使用したコンパイラ メッセージの表示) (C# コンパイラ オプション)

/utf8output オプションは、UTF-8 エンコーディングを使用してコンパイラ出力を表示します。

```
/utf8output
```

解説

国際対応の構成によっては、コンパイル出力がコンソールに正しく表示されないことがあります。このような構成では、**/utf8output** を使用してコンパイラ出力をファイルにリダイレクトします。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

参照

その他の技術情報

C# コンパイラ オプション

/warn (警告レベルの指定) (C# コンパイラ オプション)

/warn オプションは、コンパイラが表示する警告レベルを指定します。

```
/warn:option
```

引数
option

コンパイルで表示させる警告のレベルです。値が小さいほど、表示される警告が増え、重大度の高い警告だけが表示されます。また、値が大きいほど、より多くの警告が表示されます。次の 0 ～ 4 の値を指定できます。

警告レベル	説明
0	すべての警告メッセージの出力をオフにします。
1	重大な警告メッセージを表示します。
2	レベル 1 の警告に加えて、より重大度が低いいくつかの警告を表示します。表示される警告には、クラス メンバが非表示になっていることについての警告などがあります。
3	レベル 2 の警告に加えて、それより重大度が低いいくつかの警告を表示します。これには、常に **true** または **false** になる式に関する警告などが含まれます。
4 (既定)	レベル 3 のすべての警告と、情報を提供するだけの警告を表示します。

解説

ヘルプ索引でエラー コードを検索することにより、エラーまたは警告に関する情報を参照できます。エラーまたは警告に関する情報を入手するための他の方法については、「方法 : コンパイラ エラーのヘルプを見つける」を参照してください。

すべての警告をエラーとして扱うには、/warnaserror を使用します。特定の警告を無効にするには、/nowarn を使用します。

/w は **/warn** の省略形です。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [警告レベル] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「WarningLevel」を参照してください。

使用例

`in.cs` をコンパイルし、コンパイラでレベル 1 の警告だけを表示する例を次に示します。

```
csc /warn:1 in.cs
```

参照
その他の技術情報
C# コンパイラ オプション

/warnaserror (警告のエラーとしての取り扱い) (C# コンパイラ オプション)

/warnaserror+ オプションは、すべての警告をエラーとして扱います。

```
/warnaserror[<U>+</U> | -][:warning-list]
```

解説

通常は警告としてレポートされるメッセージがエラーとしてレポートされ、ビルド処理が中断します。出力ファイルはビルドされません。

既定では、警告が出力ファイルの生成を妨げない、**/warnaserror-** が有効です。**/warnaserror+** と同じ機能を持つ **/warnaserror** は、警告をエラーとして扱います。

また、エラーと見なす警告の番号をコンマ区切りで指定することにより、一部の特定の警告だけをエラーとして扱うこともできます。

コンパイラで表示する警告のレベルを指定するには、/warn を使用します。特定の警告を無効にするには、/nowarn を使用します。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [ビルド] プロパティ ページをクリックします。
3. [警告をエラーとして扱う] プロパティを変更します。

   このコンパイラ オプションをコードから設定するには、「TreatWarningsAsErrors」を参照してください。

使用例

`in.cs` をコンパイルし、コンパイラで警告を表示しないようにする例を次に示します。

```
csc /warnaserror in.cs
csc /warnaserror:642,649,652 in.cs
```

参照

その他の技術情報

C# コンパイラ オプション

/win32icon (.ico ファイルのインポート) (C# コンパイラ オプション)

/win32icon オプションは、.ico ファイルを出力ファイルに挿入し、Windows エクスプローラでの出力ファイルの表示形式を指定します。

```
/win32icon:filename
```

引数
filename

出力ファイルに加える .ico ファイル。

解説

.ico ファイルは、リソース コンパイラを使用して作成できます。リソース コンパイラは Visual C++ プログラムをコンパイルするときに呼び出され、.ico ファイルは .rc ファイルから作成されます。

.NET Framework のリソース ファイルの参照については、「/linkresource」、アタッチについては「/resource」を参照してください。.res ファイルのインポートについては、「/win32res (Win32 リソース ファイルのインポート)」を参照してください。

Visual Studio 開発環境でこのコンパイラ オプションを設定するには

1. プロジェクトの [プロパティ] ページを開きます。詳細については、「方法 : プロジェクトのプロパティを設定する (C#、J#)」を参照してください。
2. [アプリケーション] プロパティ ページをクリックします。
3. [アプリケーション アイコン] プロパティを変更します。

このコンパイラ オプションをプログラムで設定する方法については、「ApplicationIcon」を参照してください。

使用例

`in.cs` をコンパイルし、.ico ファイル `rf.ico` をアタッチして `in.exe` を生成するには、次のコードを使用します。

```
csc /win32icon:rf.ico in.cs
```

参照
その他の技術情報
C# コンパイラ オプション

/win32res (Win32 リソース ファイルのインポート) (C# コンパイラ オプション)

/win32res オプションは、Win32 リソースを出力ファイルに挿入します。

```
/win32res:filename
```

引数
filename

出力ファイルに加えるリソース ファイル。

解説

Win32 リソース ファイルは、リソース コンパイラを使用して作成できます。リソース コンパイラは、Visual C++ プログラムをコンパイルするときに呼び出されます。.res ファイルは .rc ファイルから作成されます。

Win32 リソースには、Windows エクスプローラでアプリケーションを識別するときに役立つ、バージョンやビットマップ (アイコン) の情報を含めることができます。**/win32res** を指定しない場合は、アセンブリのバージョンに基づくバージョン情報がコンパイラによって生成されます。

.NET Framework のリソース ファイルの参照については、「/linkresource」、アタッチについては「/resource」を参照してください。

このコンパイラ オプションは、Visual Studio で利用できず、プログラムで変更することもできません。

使用例

`in.cs` をコンパイルし、Win32 リソース ファイル `rf.res` をアタッチして `in.exe` を生成する例を次に示します。

```
csc /win32res:rf.res in.cs
```

参照
その他の技術情報
C# コンパイラ オプション

方法：コンパイラ エラーのヘルプを見つける

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

すべての C# コンパイラ エラーには、エラーが生成された理由について説明するトピックが用意されています。また、エラーの解決方法が解説されている場合もあります。特定のエラー メッセージについて調べるには、次のいずれかの方法を使用します。

手順

エラーに対応するヘルプを見つけるには

- [出力] ウィンドウでエラー番号をクリックし、F1 キーを押します。

  または

  [キーワード] の [検索する文字列] ボックスに、エラー番号を入力します。

  または

  [検索] ページでエラー番号を入力します。

参照

その他の技術情報

C# コンパイラ オプション

C# コンパイラ エラー

ここでは、C# コンパイラ エラーを番号ごとに説明しています。すべての C# コンパイラ エラーには、エラーが生成された理由について説明するトピックが用意されています。また、エラーの解決方法が解説されている場合もあります。特定のエラー メッセージについて調べるには、次のいずれかの方法を使用します。

- [出力] ウィンドウでエラー番号をクリックし、F1 キーを押します。
- [キーワード] の [検索する文字列] ボックスに、エラー番号を入力します。
- [検索] ページでエラー番号を入力します。
- [目次] リストでエラーの位置を特定します。

メモ :

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

参照

その他の技術情報

C# コンパイラ オプション

コンパイラ エラー CS0001

エラー メッセージ
内部コンパイル エラー

コンパイラで予測不可能な構文を解析できないためにエラーが発生したのかどうかを調べてください。このケースが該当しない場合は、「テクニカル サポート オプション (Visual Studio)」を参照してください。

コンパイラ エラー CS0003

エラー メッセージ
メモリが不足しています。

コンパイルの完了に必要な仮想メモリを割り当てることができませんでした。不要なアプリケーションをすべて閉じて、もう一度コンパイルしてください。

ページファイルのサイズを増やしたり、ディスクの空き領域を確認したりすることもできます。

このエラーは、.NET Framework SDK と C# コンパイラのバージョンが一致しない場合や、C# コンパイラをサポートするファイルが 1 つ以上破損している場合にも発生することがあります。このような場合は、Visual Studio を再インストールしてください。

コンパイラ エラー CS0004

警告をエラーとして扱う。

コンパイル時に /warnaserror コンパイラ オプションが指定されたため、警告がエラーとして生成されています。

コンパイラ エラー CS0005

エラー メッセージ
コンパイラ オプション 'compiler_option' の後には引数が必要です。

一部のコンパイラ オプションには引数が必要です。コンパイラ オプションに必要な引数を渡さないと、CS0005 が生成されます。

詳細については、「C# コンパイラ オプション」を参照してください。

コンパイラ エラー CS0006

エラー メッセージ
'dll_name' メタデータが見つかりませんでした。

プログラムがコンパイルされ、メタデータを格納したファイル名が明示的に渡されましたが、.dll が見つかりませんでした。詳細については、「/reference (メタデータのインポート) (C# コンパイラ オプション)」を参照してください。

コンパイラ エラー CS0007

エラー メッセージ
予期しない共通言語ランタイム初期化エラーです — 'description'

このエラーは、ランタイムの読み込みに失敗した場合に発生します。原因としては、コンパイラが読み込もうとしている共通言語ランタイムのバージョンがコンピュータに存在しないか、共通言語ランタイムの環境または構成に不具合が生じていることなどが考えられます。

たとえば、`csc.exe.config` ファイルが変更された場合です。このファイルは、セットアップ時に設定されるため、変更することはできません。`csc.exe.config` ファイルが変更されている可能性がある場合は、ファイル内で指定されたバージョンのランタイムがコンピュータにインストールされているかを確認してください。正しいバージョンがインストールされている場合、ランタイムが破損している可能性があります。共通言語ランタイムを再インストールしてください。

コンパイラ エラー CS0008

エラー メッセージ
ファイル 'file' からメタデータを読み込んでいるときに予期しないエラーが発生しました — 'description'

メタデータの取得に必要な DLL が正常に開かれましたが、破損しているためにデータを読み取ることができませんでした。詳細については、「/reference (メタデータのインポート) (C# コンパイラ オプション)」を参照してください。

コンパイラ エラー CS0009

エラー メッセージ
'ファイル' メタデータ ファイルを開けませんでした - '説明'

/reference コンパイラ オプションで指定したファイルに有効なメタデータが格納されていません。

コンパイラ エラー CS0011

エラー メッセージ
型 '型' によって参照されたアセンブリ 'アセンブリ' の基本クラスまたはインターフェイス 'クラス' を解決できませんでした。

/reference を使用してファイルからインポートされたクラスが、あるクラスの派生クラスであるか、または見つからないインターフェイスを実装しています。このエラーは、必要な .DLL が **/reference** を使用したコンパイル時にインクルードされていない場合に発生することがあります。

詳細については、「[参照の追加] ダイアログ ボックス」および「/reference (メタデータのインポート) (C# コンパイラ オプション)」を参照してください。

使用例

```
// CS0011_1.cs
// compile with: /target:library

public class Outer
{
   public class B { }
}
```

2 つ目のファイルでは、前の例で作成した `B` クラスから派生した `C` クラスを定義することによって DLL を作成します。

```
// CS0011_2.cs
// compile with: /target:library /reference:CS0011_1.dll
// post-build command: del /f CS0011_1.dll
public class C : Outer.B {}
```

3 つ目のファイルでは、最初の手順で作成した DLL を置き換え、内部クラスである `B` の定義を省略しています。

```
// CS0011_3.cs
// compile with: /target:library /out:cs0011_1.dll
public class Outer {}
```

最後に、4 つ目のファイルで、2 番目の例で定義した `C` クラスを参照します。C クラスは `B` クラスから派生していますが、今度は B クラスは省略されています。

次の例では CS0011 エラーが生成されます。

```
// CS0011_4.cs
// compile with: /reference:CS0011_1.dll /reference:CS0011_2.dll
// CS0011 expected

class M
{
   public static void Main()
   {
      C c = new C();
   }
}
```

参照

関連項目

[参照の追加] ダイアログ ボックス
/reference (メタデータのインポート) (C# コンパイラ オプション)

コンパイラ エラー CS0012

エラー メッセージ
型 'type' が参照されていないアセンブリで定義されています。アセンブリ 'assembly' に参照を追加してください。

参照される型の定義が見つかりませんでした。このエラーは、必要な .DLL ファイルがコンパイル時にインクルードされていない場合に発生することがあります。詳細については、「[参照の追加] ダイアログ ボックス」および「/reference (メタデータのインポート) (C# コンパイラ オプション)」を参照してください。

次のコンパイル シーケンスは CS0012 になります。

```
// cs0012a.cs
// compile with: /target:library
public class A {}
```

次のコードも同様の結果になります。

```
// cs0012b.cs
// compile with: /target:library /reference:cs0012a.dll
public class B
{
   public static A f()
   {
      return new A();
   }
}
```

次のコードも同様の結果になります。

```
// cs0012c.cs
// compile with: /reference:cs0012b.dll
class C
{
   public static void Main()
   {
      object o = B.f();   // CS0012
   }
}
```

この CS0012 エラーは、`/reference:b.dll;a.dll` を使用してコンパイルすることで解決できる場合があります。

コンパイラ エラー CS0013

エラー メッセージ
メタデータ をファイル 'ファイル' に書き込み中に予期しないエラーが発生しました - '説明'

.NET Framework 共通言語ランタイムでメタデータの出力に失敗しました。パスが正しく指定されていること、および、ディスクの空き容量が十分に残っていることを確認します。それでも問題が解決されない場合は、Visual Studio と .NET Framework の修復インストールまたは再インストールが必要です。

コンパイラ エラー CS0014

エラー メッセージ
必要なファイル 'file' が見つかりませんでした。

コンパイラで必要なファイルがシステムにありません。パスが正しく指定されていることを確認します。ファイルが Visual Studio のシステム ファイルである場合は、修復インストールを試みるか、一度 をアンインストールし、再インストールする必要があります。

コンパイラ エラー CS0015

エラー メッセージ
型名 'type' が長すぎます。

ユーザー定義型の完全限定名は 2048 文字以下にする必要があります。

次の例では CS0015 エラーが生成されます。

```
// CS0015.cs
namespace
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
{
namespace
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
{
namespace
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
{
public class
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC   // CS0015
{
   public static void Main()
   {
   }
}
}
}
}
```

コンパイラ エラー CS0016

エラー メッセージ
出力ファイル 'file' に書き込めませんでした — 'reason'

コンパイラは出力ファイルに書き込むことができませんでした。ファイルのパスをチェックして、対応するファイルが存在するかどうかを確認してください。その場所に、既にビルドされたファイルが存在する場合は、ファイルが書き込み可能になっていること、および、そのファイルが他のプロセスによってロックされていないことを確認します。たとえば、ビルドする際に、実行可能ファイルが使用中でないことを確認します。

コンパイラ エラー CS0017

エラー メッセージ
プログラム 'output file name' で、複数のエントリ ポイントが定義されています : function

1 つのプログラムで使用できる Main メソッドは 1 つだけです。

このエラーを解決するには、コードに含まれる **Main** メソッドを 1 つ残してそれ以外はすべて削除するか、または使用する **Main** メソッドを /main コンパイラ オプションで指定します。

次の例では CS0017 エラーが生成されます。

```
// CS0017.cs
public class clx
{
   static public void Main()
   {
   }
}

public class cly
{
   public static void Main()   // CS0017, delete one Main or use /main
   {
   }
}
```

コンパイラ エラー CS0019

エラー メッセージ
演算子 'operator' を 'type' と 'type' 型のオペランドに適用することはできません。

二項演算子を使用した演算が、対象外のデータ型で行われています。たとえば、文字列で || 演算子は使用できません。

使用例
この例では、条件ロジックを ConditionalAttribute の外側に指定する必要があります。**ConditionalAttribute** に渡すことができる定義済みシンボルは 1 つだけです。

次の例では CS0019 エラーが生成されます。

```
// CS0019.cs
// compile with: /target:library
using System.Diagnostics;
public class MyClass
{
   [ConditionalAttribute("DEBUG" || "TRACE")]   // CS0019
   public void TestMethod() {}

   // OK
   [ConditionalAttribute("DEBUG")]
   public void TestMethod2() {}
}
```

参照
関連項目
演算子 (C# プログラミング ガイド)

コンパイラ エラー CS0020

エラー メッセージ
定数 0 で除算します。

除算の分母で、(変数ではなく) 0 のリテラル値を使用しています。0 による除算は定義されていないため無効です。

次の例では CS0020 エラーが生成されます。

```
// CS0020.cs
namespace x
{
   public class b
   {
      public static void Main()
      {
         1 / 0;   // CS0020
      }
   }
}
```

参照
関連項目
演算子 (C# プログラミング ガイド)

コンパイラ エラー CS0021

エラー メッセージ
角かっこ [] 付きインデックスを 'type' 型の式に適用することはできません。

インデクサ (C# プログラミング ガイド) をサポートしないデータ型に対し、インデクサによって値にアクセスしようとしました。

C++ アセンブリでインデクサの使用を試みると、CS0021 エラーが発生することがあります。この場合は、既定のインデクサを C# コンパイラが判別できるように、C++ クラスに **DefaultMember** 属性を適用します。次の例では CS0021 エラーが生成されます。

使用例

このエラーを生成するには、次のファイルを (**DefaultMember** 属性をコメント アウトして) コンパイルし、.dll ファイルを作成します。

```
// CPP0021.cpp
// compile with: /clr /LD
using namespace System::Reflection;
// Uncomment the following line to resolve
//[DefaultMember("myItem")]
public ref class MyClassMC
{
        public:
        property int myItem[int]
        {
            int get(int i){  return 5; }
            void set(int i, int value) {}
        }
};
```

次のコードは、この .dll ファイルを呼び出す C# ファイルです。このファイルは、インデクサ経由でクラスへのアクセスを試みますが、既定のインデクサとして宣言されたメンバが存在しないため、エラーが生成されます。

```
// CS0021.cs
// compile with: /reference:CPP0021.dll
public class MyClass
{
    public static void Main()
    {
        MyClassMC myMC = new MyClassMC();
        int j = myMC[1]; // CS0021
    }
}
```

コンパイラ エラー CS0022

エラー メッセージ
角かっこ [] 内のインデックス数が正しくありません。正しい数は 'number' です。

配列アクセス操作で、角かっこ内に誤った次元数が指定されています。詳細については、「配列 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0022 エラーが生成されます。

```
// CS0022.cs
public class MyClass
{
    public static void Main()
    {
        int[] a = new int[10];
        a[0] = 0;     // single-dimension array
        a[0,1] = 9;   // CS0022, the array does not have two dimensions
    }
}
```

コンパイラ エラー CS0023

エラー メッセージ
演算子 'operator' を 'type' 型のオペランドには適用することはできません。

演算子が使用できるようにデザインされていない型の変数に対して演算子の使用を試みました。詳細については、「データ型 (C# プログラミング ガイド)」および「C# の演算子」を参照してください。

次の例では CS0023 エラーが生成されます。

```
// CS0023.cs
namespace x
{
   public class a
   {
      public static void Main()
      {
         string s = "hello";
         s = -s;   // CS0023, minus operator not allowed on strings
      }
   }
}
```

コンパイラ エラー CS0026

エラー メッセージ
キーワード this はスタティック プロパティ、スタティック メソッド、またはスタティック フィールド初期化子で無効です。

this (C# リファレンス) キーワードは、オブジェクト (つまり、型のインスタンス) を参照します。静的メソッドは、そのメソッドが配置されたクラスのインスタンスとは独立しているため、"this" キーワードは無意味であり、使用できません。詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」および「オブジェクト (C# プログラミング ガイド)」を参照してください。

使用例
次の例では、CS0026 エラーが生成されます。

```
// CS0026.cs
public class A
{
   public static int i = 0;

   public static void Main()
   {
// CS0026
      this.i = this.i + 1;
      // Try the following line instead:
      // i = i + 1;
   }
}
```

コンパイラ エラー CS0027

エラー メッセージ
キーワード this は現在のコンテキストでは使用できません。

this (C# リファレンス) キーワードが、プロパティ、メソッド、またはコンストラクタ以外の場所で見つかりました。

このエラーを解決するには、**this** を使わないようにステートメントを変更するか、ステートメントの一部または全体をプロパティ、メソッド、またはコンストラクタ内に移動します。

次の例では、CS0027 エラーが生成されます。

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication3
{
    class MyClass
    {

        int err1 = this.Fun() + 1;  // CS0027

        public int Fun()
        {
            return 10;
        }

        public void Test()
        {
            // valid use of this
            int err = this.Fun() + 1;
            Console.WriteLine(err);
        }

        public static void Main()
        {
            MyClass c = new MyClass();
            c.Test();
        }
    }
}
```

コンパイラ エラー CS0029

エラー メッセージ
型 'type' を型 'type' に変換できません。

コンパイラでは明示的な変換が必要です。たとえば、左辺値と同じ型になるように右辺値をキャストする必要が生じることがあります。または、特定の演算子のオーバーロードをサポートする、変換ルーチンを用意する必要があります。

変換は、ある型の変数を別の型の変数に代入する場合に行われます。異なる型の変数間で代入を行う場合、コンパイラでは、代入演算子の右側にある型を、代入演算子の左側にある型に変換する必要があります。次のようなコードがあるとします。

```
int i = 50;
long lng = 100;
i = lng;
```

`i = lng;` は代入を行いますが、代入演算子の左右の変数のデータ型が一致しません。代入を行う前に、コンパイラは変数 `lng` の型を long から int に暗黙に変換します。この変換を実行するようにコンパイラに明示的に指示するコードがないため、この変換は暗黙に行われます。このコードの問題点は、この変換が縮小変換であると見なされることです。データを損失する可能性があるため、コンパイラでは暗黙の縮小変換を許可していません。

変換元のデータ型よりも、変換先のデータ型の方がメモリの記憶領域の占有率が少ない場合に、縮小変換が行われます。たとえば、long を int に変換することは縮小変換と見なされます。long はメモリの 8 バイトを占有し、int は 4 バイトを占有します。データの損失がどのように発生するかを確認するには、次の例を参照してください。

```
int i = 50;
long lng = 3147483647;
i = lng;
```

変数 `lng` には、大きすぎて変数 `i` に格納できない値が含まれます。この値を int 型に変換した場合は、データの一部が損失し、変換された値は変換前の値と同じにはなりません。

拡大変換は、縮小変換の逆の変換です。拡大変換では、変換元のデータ型よりも変換先のデータ型の方が、メモリの記憶領域の占有率が多くなります。拡大変換の例を次に示します。

```
int i = 50;
long lng = 100;
lng = i;
```

このコード サンプルと最初のコード サンプルとの違いに注意してください。このコード サンプルでは、代入の対象となるように、変数 `lng` が代入演算子の左側にあります。変数の値を代入するためには、コンパイラは int 型の変数 `i` を long. 型に暗黙的に変換する必要があります。この場合は、記憶領域が 4 バイトの型 (int) から、8 バイト (long) の型に変換することになるため、拡大変換になります。データを損失する可能性はないため、暗黙に拡大変換を行うことができます。int に格納できる値は、long にも格納できます。

暗黙の縮小変換は許可されないため、このコードをコンパイルするには、データ型を明示的に変換する必要があります。キャストを使用すると、明示的な変換が行われます。キャストとは、あるデータ型から別のデータ型への変換を記述する C# の用語です。コンパイルするコードを取得するには、次の構文を使用する必要があります。

```
int i = 50;
long lng = 100;
i = (int) lng;   // cast to int
```

このコードの 3 行目は、代入を行う前に long 型の変数 `lng` を int に明示的に変換するようにコンパイラに指示します。縮小変換を使用すると、データを損失する可能性があります。縮小変換を使用する場合は注意が必要です。コードがコンパイルできても、実行時に予期しない結果になる可能性があります。

この説明は、値型だけに対するものです。値型を処理する場合は、変数に格納されているデータを直接処理します。ただし、.NET Framework にも参照型があります。参照型を扱う場合は、実際のデータではなく変数への参照を扱います。参照型の例としては、クラス、インターフェイス、および配列があります。コンパイラで特定の変換が許可されているか、適切な変換演算子が実装されていない限り、ある参照型から別の参照型への変換は、暗黙にも明示的にも行うことはできません。

次の例では CS0029 エラーが生成されます。

```
// CS0029.cs
public class MyInt
{
    private int x = 0;

    // Uncomment this conversion routine to resolve CS0029
    /*
    public static implicit operator int(MyInt i)
    {
       return i.x;
    }
    */


    public static void Main()
   {
      MyInt myInt = new MyInt();
      int i = myInt; // CS0029
   }
}
```

参照

概念

変換演算子 (C# プログラミング ガイド)

コンパイラ エラー CS0030

エラー メッセージ
型 '型' を '型' に変換することはできません。

特定の演算子のオーバーロードをサポートする、変換ルーチンを用意する必要があります。詳細については、「変換演算子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0030 エラーが生成されます。

```
// CS0030.cs
namespace x
{
   public class iii
   {
      /*
      public static implicit operator iii(int aa)
      {
         return null;
      }

      public static implicit operator int(iii aa)
      {
         return 0;
      }
      */

      public static iii operator ++(iii aa)
      {
         return (iii)0;   // CS0030
         // uncomment the conversion routines to resolve CS0030
      }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0031

エラー メッセージ
定数値 'value' を 'type' に変換できません。

値を格納できない型を持つ変数への値の代入を試みました。詳細については、「データ型 (C# プログラミング ガイド)」を参照してください。

次の例では CS0031 エラーが生成されます。

```
// CS0031.cs
namespace x
{
   public class a
   {
      public static void Main()
      {
         byte i = 512;   // CS0031, 512 cannot fit in byte
      }
   }
}
```

コンパイラ エラー CS0034

エラー メッセージ
演算子 'operator' は型 'type1' と 'type2' のオペランドに対してあいまいです。

1 つの演算子が 2 つのオブジェクトで使用されており、複数の変換が見つかりました。変換は一意であることが必要です。したがって、キャストを行うか、または変換のうちの 1 つを明示的にする必要があります。詳細については、「変換演算子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0034 エラーが生成されます。

```
// CS0034.cs
public class A
{
   // allows for the conversion of A object to int
   public static implicit operator int (A s)
   {
      return 0;
   }

   public static implicit operator string (A i)
   {
      return null;
   }
}

public class B
{
   public static implicit operator int (B s)
   // one way to resolve this CS0034 is to make one conversion explicit
   // public static explicit operator int (B s)
   {
      return 0;
   }

   public static implicit operator string (B i)
   {
      return null;
   }

   public static implicit operator B (string i)
   {
      return null;
   }

   public static implicit operator B (int i)
   {
      return null;
   }
}

public class C
{
   public static void Main ()
   {
      A a = new A();
      B b = new B();
      b = b + a;   // CS0034
      // another way to resolve this CS0034 is to make a cast
      // b = b + (int)a;
   }
}
```

コンパイラ エラー CS0035

エラー メッセージ
演算子 'operator' は型 'type' のオペランドに対してあいまいです。

使用できる変換が複数あり、コンパイラは演算子を適用する前に変換を選択できません。詳細については、「Templated User Defined Conversions」および「変換演算子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0035 エラーが生成されます。

```
// CS0035.cs
class MyClass
{
   private int i;

   public MyClass(int i)
   {
      this.i = i;
   }

   public static implicit operator double(MyClass x)
   {
      return (double) x.i;
   }

   public static implicit operator decimal(MyClass x)
   {
      return (decimal) x.i;
   }
}

class MyClass2
{
   static void Main()
   {
      MyClass x = new MyClass(7);
      object o = - x;   // CS0035
      // try a cast:
      // object o = - (double)x;
   }
}
```

コンパイラ エラー CS0036

エラー メッセージ
パラメータに in 属性を指定することはできません。

現在、**In** 属性は out パラメータで使用できません。

次の例では CS0036 エラーが生成されます。

```
// CS0036.cs

using System;
using System.Runtime.InteropServices;

public class MyClass
{
   public static void TestOut([In] out char TestChar)   // CS0036
   // try the following line instead
   // public static void TestOut(out char TestChar)
   {
      TestChar = 'b';
      Console.WriteLine(TestChar);
   }

   public static void Main()
   {
      char i;           //variable to receive the value
      TestOut(out i);   // the arg must be passed as out
      Console.WriteLine(i);
   }
}
```

コンパイラ エラー CS0037

エラー メッセージ
値の型なので、null を 'type' に変換することはできません。

値型には null を代入できません。null を代入できるのは参照型または Null 許容型だけです。構造体は値型です。詳細については、「null 許容型 (C# プログラミング ガイド)」を参照してください。

次の例では CS0037 エラーが生成されます。

```
// CS0037.cs
public struct s
{
}

class a
{
   public static void Main()
   {
      int i = null;   // CS0037
      s ss = null;    // CS0037
   }
}
```

コンパイラ エラー CS0038

エラー メッセージ
入れ子にされた型 'type2' を経由して、外の型 'type1' の static でないメンバにアクセスすることはできません。

クラスに含まれるフィールドが、入れ子になったクラスで自動的に使用可能になることはありません。入れ子になったクラスで使用可能にするには、フィールドが静的であることが必要です。それ以外の場合は、外部クラスのインスタンスを作成する必要があります。詳細については、「入れ子にされた型 (C# プログラミング ガイド)」を参照してください。

次の例では CS0038 エラーが生成されます。

```
// CS0038.cs
class OuterClass
{
   public int count;
   // try the following line instead
   // public static int count;

   class InnerClass
   {
      void func()
      {
         // or, create an instance
         // OuterClass class_inst = new OuterClass();
         // int count2 = class_inst.count;
         int count2 = count;   // CS0038
      }
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0039

エラー メッセージ
ビルトイン変換で、型 'type1' を 'type2' に変換できません。

as (C# リファレンス) 演算子を使った変換は、継承や参照変換、およびボックス化変換にのみ対応しています。詳細については、「変換演算子 (C# プログラミング ガイド)」を参照してください。

使用例

次の例では、CS0039 エラーが生成されます。

```
// CS0039.cs
using System;
class A
{
}
class B: A
{
}
class C: A
{
}
class M
{
    static void Main()
    {
        A a = new C();
        B b = new B();
        C c;

        // This is valid; there is a built-in reference
        // conversion from A to C.
        c = a as C;

        //The following generates CS0039; there is no
        // built-in reference conversion from B to C.
        c = b as C;  // CS0039
    }
}
```

コンパイラ エラー CS0040

エラー メッセージ
予期しないデバッグ情報の初期化エラーが発生しました — 'reason'

このエラーは、/debug コンパイラ オプションの使用時に発生することがあり、コンパイラが .pdb ファイルに書き込むことができなかったことを示します。このエラーを解決するには、Visual Studio を再インストールするか、コンパイラにファイルやフォルダへの書き込みアクセス権があることを確認するか、またはコンパイルで /debug を使用しないようにします。

コンパイラ エラー CS0041

エラー メッセージ
ファイル 'file' にデバッグ情報を書き込んでいるときに予期しないエラーが発生しました — 'reason'

このエラーは、/debug コンパイラ オプションの使用時に発生することがあります。このエラーが発生した場合は、bin ディレクトリの PDB ファイルを削除し、再コンパイルしてみてください。引き続きエラーが発生する場合は、Visual Studio の修復インストールまたは再インストールが必要です。どうしても解決できない場合は、テクニカル サポート オプション (Visual Studio)までお問い合わせください。

コンパイラ エラー CS0042

エラー メッセージ
デバッグ情報ファイル 'file' を作成中に予期しないエラーが発生しました — 'reason'

コンパイラでデバッグ情報を作成できませんでした。エラー状態に関する追加情報が *reason* に含まれています。

コンパイラ エラー CS0043

エラー メッセージ
PDB ファイル 'file' の日付形式が正しくないか最新ではありません。削除してリビルドしてください。

このコンパイルの .pdb ファイルを削除し、再コンパイルしてください。

コンパイラ エラー CS0050

エラー メッセージ
アクセシビリティに一貫性がありません。戻り値の型 '型' のアクセシビリティはメソッド 'メソッド' よりも低く設定されています。

戻り値の型と、メソッドの仮パラメータ リストで参照される各型は、少なくとも、メソッド自体と同程度にアクセスできる必要があります。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

使用例

次の例では、CS0050 エラーが生成されます。`MyClass` に対するアクセシビリティ修飾子が指定されておらず、既定で **private** が適用されるためです。

```
// CS0050.cs
class MyClass //accessibility defaults to private
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public static MyClass MyMethod()   // CS0050
    {
        return new MyClass();
    }

    public static void Main() { }
}
```

コンパイラ エラー CS0051

エラー メッセージ
アクセシビリティに一貫性がありません。パラメータの型 'type' のアクセシビリティはメソッド 'method' よりも低く設定されています。

戻り値の型と、メソッドの仮パラメータ リストで参照される各型は、少なくとも、メソッド自体と同程度にアクセスできる必要があります。メソッド シグネチャで使用されている型が、**public** 修飾子を誤って省略したなどの理由により、プライベートになっていないか確認してください。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0051 エラーが生成されます。

```
// CS0051.cs
public class A
{
    // Try making B public since F is public
    // B is implicitly private here
    class B
    {
    }

    public static void F(B b)  // CS0051
    {
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0052

エラー メッセージ
アクセシビリティに一貫性がありません。フィールドの型 'type' のアクセシビリティはフィールド 'field' よりも低く設定されています。

パブリックな構成要素は、パブリックにアクセス可能なオブジェクトを返す必要があります。したがって、フィールドで参照されている型は、フィールド自体よりも高いアクセシビリティ レベルが割り当てられている必要があります。

使用例
次の例では CS0052 エラーが生成されます。

```
// CS0052.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
   public MyClass mf;   // CS0052
}

public class MyClass3
{
   public static void Main()
   {
   }
}
```

参照
関連項目
C# のキーワード
アクセス修飾子 (C# リファレンス)
アクセシビリティ レベル (C# リファレンス)
修飾子 (C# リファレンス)

コンパイラ エラー CS0053

エラー メッセージ
アクセシビリティに一貫性がありません。プロパティの型 'type のアクセシビリティはプロパティ 'property' よりも低く設定されています。

パブリックな構成要素は、パブリックにアクセス可能なオブジェクトを返す必要があります。詳細については、
「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0053 エラーが生成されます。

```
// CS0053.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
   public MyClass myProperty   // CS0053
   {
      get
      {
         return new MyClass();
      }
      set
      {
      }
   }
}

public class MyClass3
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0054

エラー メッセージ
アクセシビリティに一貫性がありません。インデクサの戻り値の型 'type' のアクセシビリティはインデックス 'indexer' よりも低く設定されています。

パブリックな構成要素は、パブリックにアクセス可能なオブジェクトを返す必要があります。詳細については、
「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0054 エラーが生成されます。

```
// CS0054.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class MyClass3
{
   public MyClass this[int i]   // CS0054
   {
      get
      {
         return new MyClass();
      }
   }
}

public class MyClass2
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0055

エラー メッセージ
アクセシビリティに一貫性がありません。パラメータの型 'type' のアクセシビリティはインデクサ 'indexer' よりも低く設定されています。

パブリックな構成要素は、パブリックにアクセス可能なオブジェクトを返す必要があります。詳細については、
「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0055 エラーが生成されます。

```
// CS0055.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
   public int this[MyClass myClass]   // CS0055
   {
      get
      {
         return 0;
      }
   }
}

public class MyClass3
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0056

エラー メッセージ
アクセシビリティに一貫性がありません。戻り値の型 '型' のアクセシビリティは演算子 '演算子' よりも低く設定されています。

パブリックな構成要素は、パブリックにアクセス可能なオブジェクトを返す必要があります。詳細については、
「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0056 エラーが生成されます。

```
// CS0056.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class A
{
   public static implicit operator MyClass(A a)   // CS0056
   {
      return new MyClass();
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0057

エラー メッセージ
アクセシビリティに一貫性がありません。パラメータの型 'type' のアクセシビリティは演算子 'operator' よりも低く設定されています。

パブリックな構成要素は、パブリックにアクセス可能なオブジェクトを返す必要があります。詳細については、
「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0057 エラーが生成されます。

```
// CS0057.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
   public static implicit operator MyClass2(MyClass iii)   // CS0057
   {
      return new MyClass2();
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0058

エラー メッセージ
アクセシビリティに一貫性がありません。戻り値の型 '型' のアクセシビリティはデリゲート 'デリゲート' よりも低く設定されています。

パブリックな構成要素は、パブリックにアクセス可能なオブジェクトを返す必要があります。詳細については、
「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

次のサンプルでは、MyClass にアクセス修飾子が適用されておらず、既定でプライベートなアクセシビリティが割り当てられるため、CS0058 エラーが生成されます。

```
// CS0058.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public delegate MyClass MyClassDel();   // CS0058

public class A
{
   public static void Main()
   {
   }
}
```

参照

関連項目

private (C# リファレンス)

コンパイラ エラー CS0059

エラー メッセージ
アクセシビリティに一貫性がありません。パラメータの型 'type' のアクセシビリティはデリゲート 'delegate' よりも低く設定されています。

戻り値の型と、メソッドの仮パラメータ リストで参照される各型は、少なくとも、メソッド自体と同程度にアクセスできる必要があります。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0059 エラーが生成されます。

```
// CS0059.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public delegate void MyClassDel( MyClass myClass);   // CS0059

public class Program
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0060

エラー メッセージ
アクセシビリティに一貫性がありません。基本クラス 'class1' のアクセシビリティはクラス 'class2' よりも低く設定されています。

クラスのアクセシビリティは、基本クラスと継承クラスの間で一貫している必要があります。

次の例では CS0060 エラーが生成されます。

```
// CS0060.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class MyClass2 : MyClass   // CS0060
{
   public static void Main()
   {
   }
}
```

参照

関連項目

アクセス修飾子 (C# プログラミング ガイド)

コンパイラ エラー CS0061

エラー メッセージ
アクセシビリティに一貫性がありません。基本インターフェイス 'interface 1' のアクセシビリティはインターフェイス 'interface 2' よりも低く設定されています。

パブリックな構成要素は、パブリックにアクセス可能なオブジェクトを返す必要があります。

インターフェイス アクセシビリティは、派生インターフェイスで制限できません。詳細については、「インターフェイス (C# プログラミング ガイド)」および「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0061 エラーが生成されます。

```
// CS0061.cs
// compile with: /target:library
internal interface A {}
public interface AA : A {}  // CS0061

// OK
public interface B {}
internal interface BB : B {}

internal interface C {}
internal interface CC : C {}
```

コンパイラ エラー CS0065

エラー メッセージ
'event' : すべてのイベント プロパティで add と remove の両方のアクセサが必要です。

フィールドではないイベントには、両方のアクセス メソッドが必要です。

次の例では CS0065 エラーが生成されます。

```
// CS0065.cs
using System;
public delegate void Eventhandler(object sender, int e);
public class MyClass
{
   public event EventHandler Click   // CS0065,
   {
      // to fix, uncomment the add and remove definitions
      /*
      add
      {
         Click += value;
      }
      remove
      {
         Click -= value;
      }
      */
   }

   public static void Main()
   {
   }
}
```

参照

概念

イベント (C# プログラミング ガイド)

コンパイラ エラー CS0066

エラー メッセージ
'event': イベントはデリゲート型でなければなりません。

event キーワードには、delegate 型を指定する必要があります。詳細については、「イベント (C# プログラミング ガイド)」および「デリゲート (C# プログラミング ガイド)」を参照してください。

次の例では CS0066 エラーが生成されます。

```
// CS0066.cs
using System;

public class EventHandler
{
}

// to fix the error, remove the event declaration and the
// EventHandler class declaration, and uncomment the following line
// public delegate void EventHandler();

public class a
{
   public event EventHandler Click;   // CS0066

   private void TestMethod()
   {
      if (Click != null)
         Click();
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0068

エラー メッセージ
'event': インターフェイスのイベントに初期化子を含めることはできません。

インターフェイス内のイベントでは初期化子を使用できません。詳細については、「インターフェイス (C# プログラミング ガイド)」を参照してください。

次の例では CS0068 エラーが生成されます。

```
// CS0068.cs

delegate void MyDelegate();

interface I
{
   event MyDelegate d = new MyDelegate(M.f);   // CS0068
   // try the following line instead
   // event MyDelegate d2;
}

class M
{
   event MyDelegate d = new MyDelegate(M.f);

   public static void f()
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0069

エラー メッセージ
インターフェイスのイベントに、add または remove アクセサを指定することはできません

interface 内ではイベントのアクセサを定義できません。詳細については、「イベント (C# プログラミング ガイド)」および「インターフェイス (C# プログラミング ガイド)」を参照してください。

次の例では CS0069 エラーが生成されます。

```
// CS0069.cs
// compile with: /target:library

public delegate void EventHandler();

public interface a
{
   event EventHandler Click { remove {} }   // CS0069
   event EventHandler Click2;   // OK
}
```

コンパイラ エラー CS0070

エラー メッセージ
イベント 'event' は +=、-= の左辺にのみ使用できます。ただし、'type' 型内から使用されている場合を除きます。

イベントは、定義されているクラスの外側では、参照の加算または減算しかできません。詳細については、「イベント (C# プログラミング ガイド)」を参照してください。

次の例では CS0070 エラーが生成されます。

```
// CS0070.cs
using System;
public delegate void EventHandler();

public class A
{
   public event EventHandler Click;

   public static void OnClick()
   {
      EventHandler eh;
      A a = new A();
      eh = a.Click;
   }

   public static void Main()
   {
   }
}

public class B
{
   public int Foo ()
   {
      EventHandler eh = new EventHandler(A.OnClick);
      A a = new A();
      eh = a.Click;   // CS0070
      // try the following line instead
      // a.Click += eh;
      return 1;
   }
}
```

コンパイラ エラー CS0071

エラー メッセージ
イベントのインターフェイスを明示的に実装するには、プロパティの構文を使用する必要があります。

インターフェイス内で宣言されたイベントを明示的に実装するときは、通常はコンパイラによって提供されるイベント アクセサ (**add** および **remove**) を手動で提供する必要があります。アクセサ コードは、インターフェイス イベントをクラス内の別のイベント (下記参照) または固有のデリゲート型に接続できます。詳細については、「方法 : インターフェイス イベントを実装する (C# プログラミング ガイド)」を参照してください。

使用例

次の例では、CS0071 エラーが生成されます。

```
// CS0071.cs
public delegate void MyEvent(object sender);

interface ITest
{
    event MyEvent Clicked;
}

class Test : Itest
{
    event MyEvent ITest.Clicked;  // CS0071

    // try the following code instead
/*
private MyEvent clicked;

    event MyEvent Itest.Clicked
    {
        add
        {
            clicked += value;
        }
        remove
        {
            clicked -= value;
        }
    }
*/
    public static void Main() { }
}
```

コンパイラ エラー CS0072

エラー メッセージ
'event' : オーバーライドできません。'method' はイベントではありません。

イベントがオーバーライドできるのは別のイベントだけです。詳細については、「イベント (C# プログラミング ガイド)」を参照してください。

次の例では CS0072 エラーが生成されます。

```
// CS0072.cs
delegate void MyDelegate();

class Test1
{
   public virtual event MyDelegate MyEvent;
   public virtual void VMeth()
   {
   }

   public void FireAway()
   {
      if (MyEvent != null)
         MyEvent();
   }
}

class Test2 : Test1
{
   public override event MyDelegate VMeth   // CS0072
   // uncomment the following lines to resolve
   // public override event MyDelegate MyEvent
   {
      add
      {
         VMeth += value;
         // MyEvent += value;
      }
      remove
      {
         VMeth -= value;
         // MyEvent -= value;
      }
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0073

エラー メッセージ
add または remove アクセサには本体が必要です。

イベント定義に含まれる **add** キーワードまたは **remove** キーワードには本体が必要です。詳細については、「イベント (C# プログラミング ガイド)」を参照してください。

次の例では CS0073 エラーが生成されます。

```
// CS0073.cs
delegate void del();

class Test
{
   public event del MyEvent
   {
      add;   // CS0073
      // try the following lines instead
      // add
      // {
      //    MyEvent += value;
      // }
      remove
      {
         MyEvent -= value;
      }

   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0074

エラー メッセージ
'event': 抽象イベントに初期化子を指定することはできません。

abstract としてマークされているイベントは初期化できません。詳細については、「イベント (C# プログラミング ガイド)」を参照してください。

次の例では CS0074 エラーが生成されます。

```
// CS0074.cs
delegate void D();

abstract class Test
{
   public abstract event D e = null;   // CS0074
   // try the following line instead
   // public abstract event D e;

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0075

エラー メッセージ
負の値をキャストするには、値をかっこで囲んでください。

定義済みの型を指定するキーワードを使ってキャストしている場合、かっこは不要です。それ以外の場合は、かっこを指定する必要があります。(x) –y はキャスト式とは見なされません。以降、C# 仕様のセクション 7.6.6 からの抜粋です。

あいまいさを排除するという観点から、x と y が識別子の場合、(x)y、(x)(y)、および (x)(-y) はキャスト式と見なされますが、(x)-y は x で型が指定されていたとしてもキャスト式とは見なされません。ただし、x が定義済みの型 (int など) を示すキーワードである場合は、このようなキーワードがそれ自体で式になることはないため、4 つの形式はいずれもキャスト式と見なされます。

次のコードでは、CS0075 エラーが生成されます。

```
// CS0075
namespace MyNamespace
{
    enum MyEnum { }
    public class MyClass
    {
        public static void Main()
        {
            // To fix the error, place the negative
            // values below in parentheses
            int i = (System.Int32) - 4; //CS0075
            MyEnum e = (MyEnum) - 1;    //CS0075
            System.Console.WriteLine(i); //to avoid warning
            System.Console.WriteLine(e); //to avoid warning
        }
    }
}
```

参照

関連項目

キャスト (C# プログラミング ガイド)

コンパイラ エラー CS0076

エラー メッセージ
列挙子名 'value__' は予約されているため、使用できません。

列挙体に含まれる項目では、**value__** という識別子を使用できません。

コンパイラ エラー CS0077

エラー メッセージ
as オペレータは参照型で使用してください ('type' は値の型です)。

as 演算子に値型が渡されました。**as** は null を返すことができるため、渡すことができるのは参照型だけです。

次の例では CS0077 エラーが生成されます。

```
// CS0077.cs
using System;

class C
{
}

struct S
{
}

class M
{
   public static void Main()
   {
      object o1, o2;
      C c;
      S s;

      o1 = new C();
      o2 = new S();

      s = o2 as S;  // CS0077, S is not a reference type.
      // try the following line instead
      // c = o1 as C;
   }
}
```

コンパイラ エラー CS0079

エラー メッセージ
イベント 'event' は += または -= の左側にのみ表示されます。

イベントが誤って呼び出されました。詳細については、「イベント (C# プログラミング ガイド)」および「デリゲート (C# プログラミング ガイド)」を参照してください。

次の例では CS0079 エラーが生成されます。

```
// CS0079.cs
using System;

public delegate void MyEventHandler();

public class Class1
{
   private MyEventHandler _e;

   public event MyEventHandler Pow
   {
      add
      {
         _e += value;
         Console.WriteLine("in add accessor");
      }
      remove
      {
         _e -= value;
         Console.WriteLine("in remove accessor");
      }
   }

   public void Handler()
   {
   }

   public void Fire()
   {
      if (_e != null)
      {
         Pow();   // CS0079
         // try the following line instead
         // _e();
      }
   }

   public static void Main()
   {
      Class1 p = new Class1();
      p.Pow += new MyEventHandler(p.Handler);
      p._e();
      p.Pow += new MyEventHandler(p.Handler);
      p._e();
      p._e -= new MyEventHandler(p.Handler);
      if (p._e != null)
      {
         p._e();
      }
      p.Pow -= new MyEventHandler(p.Handler);
      if (p._e != null)
      {
         p._e();
      }
   }
}
```

コンパイラ エラー CS0080

エラー メッセージ
制約は非ジェネリック宣言では許可されません。

この構文は、ジェネリック宣言で、型パラメータに制約を適用する場合にのみ使用できます。詳細については、「ジェネリック (C# プログラミング ガイド)」を参照してください。

次の例では、MyClass クラスも Foo メソッドもジェネリックではないため、CS0080 エラーが生成されます。

```
namespace MyNamespace
{
    public class MyClass where MyClass : System.IDisposable // CS0080    //the following li
ne shows an example of correct syntax
    //public class MyClass<T> where T : System.IDisposable
    {
        public void Foo() where Foo : new() // CS0080
        //the following line shows an example of correct syntax
        //public void Foo<U>() where U : struct
        {
        }
    }

    public class Program
    {
        public static void Main()
        {
        }
    }
}
```

コンパイラ エラー CS0081

エラー メッセージ
型パラメータの宣言は型ではなく識別子でなければなりません。

ジェネリック メソッドまたはジェネリック型を宣言する場合は、識別子として型パラメータ ("T"、"inputType" など) を指定します。クライアント コードで型を指定してこのメソッドを呼び出すと、メソッドまたはクラス本体に出現する各識別子が置き換えられます。詳細については、「ジェネリック型の型パラメータ (C# プログラミング ガイド)」を参照してください。

```
// CS0081.cs
class MyClass
{
   public void F<int>() {}   // CS0081
   public void F<T>(T input) {}   // OK

   public static void Main()
   {
      MyClass a = new MyClass();
      a.F<int>(2);
      a.F<double>(.05);
   }
}
```

参照

概念
ジェネリック (C# プログラミング ガイド)

コンパイラ エラー CS0100

エラー メッセージ
パラメータ名 'parameter name' が重複しています。

メソッドの宣言で同じパラメータ名を複数回使用しました。メソッドの宣言では、パラメータ名が一意であることが必要です。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

次の例では CS0100 エラーが生成されます。

```
// CS0100.cs
namespace x
{
   public class a
   {
      public static void f(int i, char i)   // CS0100
      // try the following line instead
      // public static void f(int i, char j)
      {
      }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0101

エラー メッセージ
名前空間 'namespace' に 'type' の定義が既に含まれています。

名前空間の識別子が重複しています。重複している識別子のうちの 1 つの名前を変更するか、または削除してください。詳細については、「名前空間 (C# プログラミング ガイド)」を参照してください。

次の例では CS0101 エラーが生成されます。

```
// CS0101.cs
namespace MyNamespace
{
   public class MyClass
   {
      static public void Main()
      {
      }
   }

   public class MyClass   // CS0101
   {
   }
}
```

コンパイラ エラー CS0102

エラー メッセージ
型 '型名' は '識別子' の定義を既に含んでいます。

1 つのクラス内で、同じ名前の識別子が同じスコープで複数宣言されています。このエラーを解決するには、重複する識別子の名前を変更します。

使用例
次の例では CS0102 エラーが生成されます。

```
// CS0102.cs
// compile with: /target:library
namespace MyApp
{
   public class MyClass
   {
      string s = "Hello";
      string s = "Goodbye";   // CS0102

      public void GetString()
      {
         string s = "Hello again";   // method scope, no error
      }
   }
}
```

コンパイラ エラー CS0103

エラー メッセージ
名前 '識別子' は現在のコンテキスト内に存在しません。

クラス、名前空間、またはスコープに存在しない名前が使用されています。名前が正しく入力されているかどうかを確認してください。また、using ステートメントやアセンブリ参照をチェックして、使おうとしている名前が本当に利用できるかどうかを確認します。陥りがちなミスとして、変数をループや try ブロック内で宣言し、外部のコード ブロックからこの変数にアクセスしていることが考えられます。次にその例を示します。

次の例では CS0103 エラーが生成されます。

```
// CS0103.cs
using System;

class MyClass
{
   public static void Main()
   {
      // MyClass conn = null;
      try
      {
         MyClass conn = new MyClass();   // delete this line
         // and uncomment the following line and the line above the try
         // conn = new MyClass();
      }
      catch(Exception e)
      {
         if (conn != null)   // CS0103
            Console.WriteLine("{0}", e);
      }
   }
}
```

コンパイラ エラー CS0104

エラー メッセージ
'参照' は '識別子' 間の不適切な参照です。

2 つの名前空間に対する using ディレクティブがプログラムに含まれており、コードが両方の名前空間で使用されている名前を参照しています。

次の例では CS0104 エラーが生成されます。

```
// CS0104.cs
using x;
using y;

namespace x
{
   public class Test
   {
   }
}

namespace y
{
   public class Test
   {
   }
}

public class a
{
   public static void Main()
   {
      Test test = new Test();   // CS0104, is Test in x or y namespace?
      // try the following line instead
      // y.Test test = new y.Test();
   }
}
```

コンパイラ エラー CS0106

エラー メッセージ
修飾子 'modifier' はこの項目に対して使用できません。

クラスまたはインターフェイスのメンバが、無効なアクセス修飾子でマークされました。無効な修飾子の例を次に示します。

- static 修飾子と public 修飾子は、インターフェイス メソッドでは使用できません。
- **public** キーワードは、明示的なインターフェイス宣言では使用できません。この場合は、明示的なインターフェイス宣言から **public** キーワードを削除してください。
- 明示的なインターフェイスの実装はオーバーライドできないため、abstract キーワードは明示的なインターフェイス宣言で使用できません。

旧バージョンの Visual Studio では、クラスに対して **static** 修飾子を使用することはできませんでした。**static** クラスは、Microsoft Visual Studio 2005 以降でのみサポートされます。

詳細については、「インターフェイス (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0106 エラーが生成されます。

```
// CS0106.cs
namespace MyNamespace
{
   interface I
   {
      void m();
      static public void f();   // CS0106
   }

   public class MyClass
   {
      public void I.m() {}   // CS0106
      public static void Main() {}
   }
}
```

コンパイラ エラー CS0107

エラー メッセージ
複数の保護修飾子があります。

クラス メンバでは、**internal protected** を除いて、アクセス修飾子 (public、private、protected、または internal) を 1 つしか使用できません。

次の例では CS0107 エラーが生成されます。

```
// CS0107.cs
public class C
{
   private internal void f()   // CS0107, delete private or internal
   {
   }

   public static int Main()
   {
      return 1;
   }
}
```

コンパイラ エラー CS0110

エラー メッセージ
'const declaration' の定数値の評価により、循環定義が発生します。

const 変数 (a) の宣言は、(a) を参照する別の const 変数 (b) を参照できません。

次の例では CS0110 エラーが生成されます。

```
// CS0110.cs
namespace MyNamespace
{
   public class A
   {
      public static void Main()
      {
      }
   }

   public class B : A
   {
      public const int i = c + 1;   // CS0110, c already references i
      public const int c = i + 1;
      // the following line would be OK
      // public const int c = 10;
   }
}
```

参照

関連項目

定数 (C# プログラミング ガイド)

コンパイラ エラー CS0111

エラー メッセージ
型 'クラス' は、'メンバ' と呼ばれるメンバを同じパラメータの型で既に定義しています。

CS0111 エラーは、クラスに、同じ名前とパラメータの型で定義された、2 つのメンバ宣言が存在する場合に発生します。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0111 エラーが生成されます。

```
// CS0111.cs
class A
{
   void Test() { }
   public static void Test(){}   // CS0111

   public static void Main() {}
}
```

コンパイラ エラー CS0112

エラー メッセージ
静的メンバ 'function' を override、virtual、または abstract とすることはできません。

override、**virtual**、**abstract** の各キーワードを使用するメソッドの宣言では、**static** キーワードを使用できません。

詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

次の例では CS0112 エラーが生成されます。

```
// CS0112.cs
namespace MyNamespace
{
   abstract public class MyClass
   {
      public abstract void Foo();
   }
   public class MyClass2 : MyClass
   {
      override public static void Foo()   // CS0112, remove static keyword
      {
      }
      public static int Main()
      {
         return 0;
      }
   }
}
```

コンパイラ エラー CS0113

エラー メッセージ
override 型のメンバ 'function' を、new または virtual にすることはできません。

メソッドの new キーワードと override キーワードによるマーキングは、同時に行うことはできません。

次の例では CS0113 エラーが生成されます。

```
// CS0113.cs
namespace MyNamespace
{
   abstract public class MyClass
   {
      public abstract void Foo();
   }

   public class MyClass2 : MyClass
   {
      override new public void Foo()   // CS0113, remove new keyword
      {
      }

      public static int Main()
      {
         return 0;
      }
   }
}
```

コンパイラ エラー CS0115

エラー メッセージ
'function' : オーバーライドする適切なメソッドが見つかりませんでした。

メソッドがオーバーライドとしてマークされましたが、オーバーライドするメソッドが見つかりませんでした。詳細については、「override (C# リファレンス)」および「Override キーワードと New キーワードを使用する場合について (C# プログラミング ガイド)」を参照してください。

使用例

次の例では、CS0115 エラーが生成されます。CS0115 エラーは、次のいずれかの方法で解決できます。

- `MyClass2` のメソッドから **override** キーワードを削除します。
- `MyClass2` の基本クラスとして `MyClass1` を使用します。

```
// CS0115.cs
namespace MyNamespace
{
    abstract public class MyClass1
    {
        public abstract int f();
    }

    abstract public class MyClass2
    {
        public override int f()   // CS0115
        {
            return 0;
        }

        public static void Main()
        {
        }
    }
}
```

コンパイラ エラー CS0116

エラー メッセージ
名前空間にフィールドやメソッドのようなメンバを直接含めることはできません。

名前空間の内部で使用できるのは、クラス、構造体、共用体、列挙体、インターフェイス、およびデリゲートだけです。C/C++ の経験を持つ開発者が陥りやすいエラーです。C/C++ とは異なり、C# では、メソッドおよび変数は構造体内またはクラス内で宣言および定義する必要があります。詳細については、「C# プログラムの一般構造 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0116 エラーが生成されます。

```
// CS0116.cs
namespace x
{
    using System;

    // method must be in class/struct
    void Method(string str) // CS0116
    {
        Console.WriteLine(str);
    }
}
```

コンパイラ エラー CS0117

'型' に '識別子' の定義がありません。

このエラーは、特定のデータ型に存在しないメンバを参照しようとした場合に発生します。

このエラーが発生する一般的な原因としては、次のようなケースが考えられます。

- 存在しないメソッドを呼び出している。
- インデクサの前に **Item** プロパティが指定されている。
- クラスと、それが属する名前空間とが同じ名前であるときに修飾されたメソッドを呼び出している。
- インターフェイス内で静的メンバを使用できる言語で記述されたインターフェイスを呼び出している。

次の例では CS0117 エラーが生成されます。

```
// CS0117_1.cs
namespace MyNamespace
{
   public class MyClass
   {
      public static void Main()
      {
         int i;
         i = i.get();   // CS0117
      }
   }
}
```

使用例

次の例では、**Item** プロパティがインデクサと同時に使用されています。C# では、プロパティまたはインデクサを使用してメンバにアクセスできますが、両方を使用することはできません。次の例では CS0117 エラーが生成されます。

```
// CS0117_2.cs
using System;
using System.Collections;
class Test
{
   public static void Main()
   {
      ArrayList al = new ArrayList();
      al.Add( new Test() );
      Console.WriteLine("{0}", al.Item[0]);   // CS0117
      Console.WriteLine("{0}", al[0]);   // OK
   }
}
```

CS0017 エラーは、インターフェイス内での静的メンバの使用が認められた言語で記述されたライブラリを使用し、C# から静的メンバにアクセスしようとした場合にも発生します。

```
// CS0117_3.jsl
// compile with: /target:library
public interface IMyJSharpInterface
{
   static int MyStaticMember = 0;
   void NonStaticMember();
}
```

次の例では CS0117 エラーが生成されます。

```
// CS0117_4.cs
// compile with: /reference:CS0117_3.dll
class MyCSharpClass : IMyJSharpInterface
{
   public void NonStaticMember() {}

   public static void Main()
   {
      IMyJSharpInterface myObj = new MyCSharpClass();
      myObj.NonStaticMember();
      int i = myObj.MyStaticMember;   // CS0117
   }
}
```

コンパイラ エラー CS0118

エラー メッセージ
'construct1_name' は 'construct1' ですが、'construct2' のように使用されています。

コンストラクトの使用方法が正しくないか、コンストラクトに対して許可されていない操作を実行しようとしました。一般的な例外は次のとおりです。

- クラスではなく名前空間をインスタンス化しようとしました。
- メソッドではなくフィールドを呼び出そうとしました。
- 型を変数として使おうとしました。
- extern エイリアスを型として使おうとしました。

このエラーを解決するには、実行しようとしている操作が、対象となる型に対して適切であることを確認します。

使用例
次の例では CS0118 エラーが生成されます。

```
// CS0118.cs
// compile with: /target:library
namespace MyNamespace
{
   class MyClass
   {
      // MyNamespace not a class
      MyNamespace ix = new MyNamespace ();   // CS0118
   }
}
```

コンパイラ エラー CS0119

エラー メッセージ
construct1_name' は 'construct2' ですが、指定されたコンテキストでは有効ではありません

予期しない制御構文がコンパイラによって検出されました。

- クラスのコンストラクタは、条件付きステートメントで有効なテスト式ではありません。
- 配列要素を参照するために、インスタンス名ではなくクラス名を使用しました。
- メソッド識別子が構造体またはクラスのように使用されました。

使用例
次の例では CS0119 エラーが生成されます。

```
// CS0119.cs
using System;
public class MyClass
{
   public static void Test() {}

   public static void Main()
   {
      Console.WriteLine(Test.x);   // CS0119
   }
}
```

コンパイラ エラー CS0120

エラー メッセージ
静的でないフィールド、メソッド、またはプロパティ 'member' で、オブジェクト参照が必要です。

静的でないフィールド、メソッド、またはプロパティを使用するには、最初にオブジェクト インスタンスを作成する必要があります。詳細については、「インスタンス コンストラクタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0120 エラーが生成されます。

```
// CS0120_1.cs
public class MyClass
{
   // Non-static field
   public int i;
   // Non-static method
   public void f(){}
   // Non-static property
   int Prop
   {
      get
      {
         return 1;
      }
   }

   public static void Main()
   {
      i = 10;   // CS0120
      f();   // CS0120
      int p = Prop;   // CS0120
      // try the following lines instead
      // MyClass mc = new MyClass();
      // mc.i = 10;
      // mc.f();
      // int p = mc.Prop;
   }
}
```

CS0120 エラーは、次のように、静的メソッドから非静的メソッドの呼び出しがあった場合にも発生します。

```
// CS0120_2.cs
// CS0120 expected
using System;

public class MyClass
{
   public static void Main()
   {
      TestCall();   // CS0120
      // To call a non-static method from Main,
      // first create an instance of the class.
      // Use the following two lines instead:
      // MyClass anInstanceofMyClass = new MyClass();
      // anInstanceofMyClass.TestCall();
   }

   public void TestCall()
   {
   }
}
```

同様に、クラスのインスタンスを明示的に与えない限り、次のように静的メソッドはインスタンス メソッドを呼び出すことができません。

```
// CS0120_3.cs
using System;

public class MyClass
{
   public static void Main()
   {
      do_it("Hello There");   // CS0120
   }

   private void do_it(string sText)
   // You could also add the keyword static to the method definition:
   // private static void do_it(string sText)
   {
      Console.WriteLine(sText);
   }
}
```

コンパイラ エラー CS0121

エラー メッセージ
次のメソッドまたはプロパティ間の呼び出しがあいまいです : 'method1' と 'method2'

暗黙の型変換が行われたため、コンパイラではオーバーロードされたメソッドの一方の形式を呼び出すことができませんでした。このエラーは以下の方法で解決できます。

- 暗黙の変換が行われないような方法でメソッド パラメータを指定します。
- メソッドのオーバーロードをすべて削除します。
- メソッドを呼び出す前に適切な型にキャストします。

次の例では CS0121 エラーが生成されます。

```
// CS0121.cs
public class C
{
   void f(int i, double d)
   {
   }

   void f(double d, int i)
   {
   }

   public static void Main()
   {
      C c = new C();

      c.f(1, 1);   // CS0121
      // try the following line instead
      // c.f(1, 1.0);
      // or
      // c.f(1.0, 1);
      // or
      // c.f(1, (double)1);   // cast and specify which method to call
   }
}
```

コンパイラ エラー CS0122

エラー メッセージ
'member' はアクセスできない保護レベルになっています。

クラス メンバのアクセス修飾子が原因で、そのメンバにアクセスできません。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

この原因の 1 つとして、フレンド アセンブリの出力先に対し、**/out** コンパイラ フラグが省略されていることが考えられます (以下の例には示されていません)。詳細については、「フレンド アセンブリ (C# プログラミング ガイド)」および「/out (出力ファイル名の設定) (C# コンパイラ オプション)」を参照してください。

使用例
次の例では CS0122 エラーが生成されます。

```
// CS0122.cs
public class MyClass
{
    // Make public to resolve CS0122
    void Foo()
    {
    }
}

public class MyClass2
{
    public static int Main()
    {
        MyClass a = new MyClass();
        // Foo is private
        a.Foo();   // CS0122
        return 0;
   }
}
```

コンパイラ エラー CS0123

エラー メッセージ
デリゲート 'デリゲート' に一致する 'メソッド' のオーバーロードはありません。

デリゲートの作成を試みましたが、正しいシグネチャを使用しなかったために失敗しました。デリゲートのインスタンスは、デリゲート宣言の同じシグネチャを使用して宣言する必要があります。

このエラーは、メソッドまたはデリゲートのシグネチャを調整することによって解決できます。詳細については、
「デリゲート (C# プログラミング ガイド)」を参照してください。

次の例では CS0123 エラーが生成されます。

```
// CS0123.cs
delegate void D();
delegate void D2(int i);

public class C
{
   public static void f(int i) {}

   public static void Main()
   {
      D d = new D(f);   // CS0123
      D2 d2 = new D2(f);   // OK
   }
}
```

コンパイラ エラー CS0126

エラー メッセージ
'type' に変換可能な型のオブジェクトが必要です。

return ステートメントがありますが、そのステートメントから予期した型の値が返されません。詳細については、「プロパティ (C# プログラミング ガイド)」を参照してください。

次の例では CS0126 エラーが生成されます。

```
// CS0126.cs
public class a
{
   public int i
   {
      set
      {
      }
      get
      {
         return;   // CS0126, specify a value to return
      }
   }
}
```

コンパイラ エラー CS0127

エラー メッセージ
'function' は void 型を返すため、キーワード return の後にオブジェクト式を指定することはできません。

戻り値の型が void のメソッドから、値を返すことはできません。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

次の例では CS0127 エラーが生成されます。

```
// CS0127.cs
namespace MyNamespace
{
   public class MyClass
   {
      public int hiddenMember2
      {
         get
         {
            return 0;
         }
         set   // CS0127, set has an implicit void return type
         {
            return 0;   // remove return statement to resolve this CS0127
         }
      }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0128

エラー メッセージ
ローカル変数 'variable' はこのスコープで既に定義されています。

同じ名前を持つ 2 つのローカル変数の宣言が検出されました。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

次の例では CS0128 エラーが生成されます。

```
// CS0128.cs
namespace MyNamespace
{
   public class MyClass
   {
      public static void Main()
      {
         char i;
         int i;   // CS0128
      }
   }
}
```

コンパイラ エラー CS0131

エラー メッセージ
代入式の左辺には変数、プロパティ、またはインデクサを指定してください。

代入ステートメントでは、右辺の値が左辺に代入されます。代入式の左辺は変数、プロパティ、インデクサのいずれかであることが必要です。

このエラーを解決するには、すべての演算子が右辺にあり、左辺が変数、プロパティ、インデクサのいずれかであることを確認します。詳細については、「ステートメント、式、および演算子 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では、CS0131 エラーが生成されます。

```
// CS0131.cs
public class MyClass
{
    public int i = 0;
    public void MyMethod()
    {
        i++ = 1;   // CS0131
        // try the following line instead
        // i = 1;
    }
    public static void Main() { }
}
```

このエラーは、代入演算子の左辺で、次のような算術演算を実行しようとした場合にも発生します。

```
// CS0131b.cs
public class C
{
    public static int Main()
    {
        int a = 1, b = 2, c = 3;
        if (a + b = c) // CS0131
        // try this instead
        // if (a + b == c)
            return 0;
        return 1;
    }
}
```

コンパイラ エラー CS0132

エラー メッセージ
'constructor' : スタティックコンストラクタにパラメータを指定することはできません。

パラメータを指定して static コンストラクタを宣言することはできません。詳細については、「コンストラクタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0132 エラーが生成されます。

```
// CS0132.cs
namespace MyNamespace
{
   public class MyClass
   {
      public MyClass(int i)
      {
      }
   }

   public class MyClass2 : MyClass
   {
      static MyClass2(int i)   // CS0132
      {
      }
   }
}
```

コンパイラ エラー CS0133

エラー メッセージ
'variable' に割り当てられた式は定数でなければなりません。

const 変数には、定数以外の式を値として使用できません。詳細については、「定数 (C# プログラミング ガイド)」を参照してください。

次の例では CS0133 エラーが生成されます。

```
// CS0133.cs
public class MyClass
{
   public const int i = c;   // CS0133, c is not constant
   public static int c = i;
   // try the following line instead
   // public const int i = 6;

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0134

エラー メッセージ
'変数' の型は '型' です。文字列以外の参照型の const は null と共にのみ初期化できます。

constant-expression は、コンパイル時に完全に評価できる式です。参照型の非 null 値を作成するには new 演算子を適用するしかなく、new 演算子は constant-expression で許可されていないので、string 以外の "参照型" の定数が取り得る値は null です。

const 文字配列を作成しようとしてこのエラーが発生した場合、配列を readonly にし、コンストラクタ内で初期化します。

使用例
次の例では CS0134 エラーが生成されます。

```
// CS0134.cs
// compile with: /target:library
class MyTest {}

class MyClass
{
   const MyTest test = new MyTest();   // CS0134

   //OK
   const MyTest test2 = null;
   const System.String test3 = "test";
}
```

コンパイラ エラー CS0135

エラー メッセージ
'declaration1' は宣言 'declaration2' と競合しています。

コンパイラでは名前を隠ぺいできません。通常は、名前を隠ぺいするとコード内で論理エラーが発生します。

次の例では CS0135 エラーが生成されます。

```
// CS0135.cs
public class MyClass2
{
   public static int i = 0;

   public static void Main()
   {
      {
         int i = 4;
         i++;
      }
      i = 0;   // CS0135
   }
}
```

コンパイラ エラー CS0136

エラー メッセージ
ローカルの変数 'var' をこのスコープで宣言することはできません。これは、'親または現在の/子' スコープで別の意味を持つ 'var' の意味が変更されるのを避けるためです。

変数宣言が、スコープ内にある別の宣言を隠ぺいしています。CS0136 エラーが発生した行で宣言されている変数の名前を変更してください。

次の例では CS0136 エラーが生成されます。

```
// CS0136.cs
namespace MyNamespace
{
   public class MyClass
   {
      public static void Main()
      {
         int i = 0;
         {
            char i = 'a';   // CS0136, hides int i
         }
         i++;
      }
   }
}
```

コンパイラ エラー CS0138

エラー メッセージ
using namespace ディレクティブは名前空間に対してのみ使用できます。'型' は名前空間ではなく型です。

using ディレクティブがパラメータとして受け取るとができるのは、名前空間の名前だけです。詳細については、「名前空間 (C# プログラミング ガイド)」を参照してください。

次の例では CS0138 エラーが生成されます。

```
// CS0138.cs
using System.Object;   // CS0138
```

コンパイラ エラー CS0139

エラー メッセージ
break または continue を使用できるループがありません。

ループの外側で **break** ステートメントまたは **continue** ステートメントが検出されました。

詳細については、「ジャンプ ステートメント」を参照してください。

次の例では CS0139 エラーが 2 回発生します。

```
// CS0139.cs
namespace x
{
   public class a
   {
      public static void Main()
      {
         continue;  // CS0139
         break;     // CS0139
      }
   }
}
```

コンパイラ エラー CS0140

エラー メッセージ
ラベル 'label' は重複しています。

同じ名前を持つラベルを 2 回使用しています。詳細については、「goto (C# リファレンス)」を参照してください。

次の例では CS0140 エラーが生成されます。

```
// CS0140.cs
namespace MyNamespace
{
   public class MyClass
   {
      public static void Main()
      {
         label1: int i = 0;
         label1: int j = 0;   // CS0140, comment this line to resolve
         goto label1;
      }
   }
}
```

コンパイラ エラー CS0143

エラー メッセージ
型 'class' のコンストラクタが定義されていません。

使用できる適切なコンストラクタがありません。

次の例では CS0143 エラーが生成されます。

```
// CS0143.cs
class MyClass
{
   static public void Main ()
   {
      double d = new double(4.5);   // CS0143
   }
}
```

コンパイラ エラー CS0144

エラー メッセージ
抽象クラスまたはインターフェイス 'interface' のインスタンスを作成できません。

抽象クラスまたはインターフェイスのインスタンスを作成できません。詳細については、「インターフェイス (C# プログラミング ガイド)」を参照してください。

次の例では CS0144 エラーが生成されます。

```
// CS0144.cs
interface MyInterface
{
}
public class MyClass
{
   public static void Main()
   {
      MyInterface myInterface = new MyInterface ();   // CS0144
   }
}
```

コンパイラ エラー CS0145

エラー メッセージ
const フィールドに値を指定する必要があります。

const 変数を初期化する必要があります。詳細については、「定数 (C# プログラミング ガイド)」を参照してください。

次の例では CS0145 エラーが生成されます。

```
// CS0145.cs
class MyClass
{
   const int i;   // CS0145
   // try the following line instead
   // const int i = 0;

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0146

エラー メッセージ
'クラス 1' と 'クラス 2' を含む、循環する基本クラスの依存関係です。

クラスの継承リストに、クラス自身への直接参照または間接参照が含まれています。クラスは自身を継承できません。詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

次の例では CS0146 エラーが生成されます。

```
// CS0146.cs
namespace MyNamespace
{
   public interface InterfaceA
   {
   }

   public class MyClass : InterfaceA, MyClass2
   {
      public void Main()
      {
      }
   }

   public class MyClass2 : MyClass   // CS0146
   {
   }
}
```

コンパイラ エラー CS0148

エラー メッセージ
デリゲート 'delegate' には有効なコンストラクタがありません。

別のコンパイラで作成したマネージ プログラム (.NET Framework 共通言語ランタイムを使用するプログラム) をインポートして使用しました。そのコンパイラでは、不正な形式の delegate コンストラクタが許可されています。詳細については、「デリゲート (C# プログラミング ガイド)」を参照してください。

コンパイラ エラー CS0149

エラー メッセージ
メソッド名が必要です。

delegate を作成するときは、メソッドを指定してください。詳細については、「デリゲート (C# プログラミング ガイド)」を参照してください。

次の例では CS0149 エラーが生成されます。

```
// CS0149.cs
using System;

delegate string MyDelegate(int i);

class MyClass
{
   // class member-field of the declared delegate type
   static MyDelegate dt;

   public static void Main()
   {
      dt = new MyDelegate(17.45);   // CS0149
      // try the following line instead
      // dt = new MyDelegate(Func2);
      F(dt);
   }

   public static string Func2(int j)
   {
      Console.WriteLine(j);
      return j.ToString();
   }

   public static void F(MyDelegate myFunc)
   {
      myFunc(8);
   }
}
```

コンパイラ エラー CS0150

エラー メッセージ
定数値が必要です。

定数を使用する必要がある箇所で変数を使用していました。詳細については、「switch (C# リファレンス)」を参照してください。

次の例では CS0150 エラーが生成されます。

```
// CS0150.cs
namespace MyNamespace
{
   public class MyClass
   {
      public static void Main()
      {
         int i = 0;
         int j = 0;

         switch(i)
         {
            case j:   // CS0150, j is a variable int, not a constant int
            // try the following line instead
            // case 1:
         }
      }
   }
}
```

コンパイラ エラー CS0151

エラー メッセージ
整数型の値が必要です。

整数型が必要な状況で変数を使用しました。詳細については、「データ型 (C# プログラミング ガイド)」を参照してください。

使用例

このエラーが発生するのは、変換が行われない場合、または暗黙の型変換の結果があいまいな場合です。次の例では CS0151 エラーが生成されます。

```
// CS0151.cs
public class MyClass
{
   public static implicit operator int (MyClass aa)
   {
      return 0;
   }

   public static implicit operator long (MyClass aa)
   {
      return 0;
   }

   public static void Main()
   {
      MyClass a = new MyClass();

      // Compiler cannot choose between int and long
      switch (a)   // CS0151
      // try the following line instead
      // switch ((int)a)
      {
         case 1:
            break;
      }
   }
}
```

コンパイラ エラー CS0152

エラー メッセージ
ラベル 'label' は既にこの switch ステートメントで使用されています。

switch ステートメントでラベルが繰り返し使用されました。詳細については、「switch (C# リファレンス)」を参照してください。

次の例では CS0152 エラーが生成されます。

```
// CS0152.cs
namespace MyNamespace
{
   public class MyClass
   {
      public static void Main()
      {
         int i = 0;

         switch (i)
         {
            case 1:
               i++;
               return;

            case 1:   // CS0152, two case 1 statements
               i++;
               return;
         }
      }
   }
}
```

コンパイラ エラー CS0153

エラー メッセージ
goto は switch ステートメント内でのみ有効です。

switch ステートメントの外側で switch 構文の使用を試みました。詳細については、「switch (C# リファレンス)」を参照してください。

次の例では CS0153 エラーが生成されます。

```
// CS0153.cs
public class a
{
   public static void Main()
   {
      goto case 5;   // CS0153
   }
}
```

コンパイラ エラー CS0154

エラー メッセージ
get アクセサがないため、プロパティまたはインデクサ 'property' をこのコンテキストで使用することはできません。

プロパティの使用を試みましたが、get アクセサ メソッドがプロパティで定義されていなかったために失敗しました。詳細については、「フィールド (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0154 エラーが生成されます。

```
// CS0154.cs
public class MyClass2
{
    public int i
    {
        set
        {
        }
        // uncomment the get method to resolve this error
        /*
        get
        {
            return 0;
        }
        */
    }
}

public class MyClass
{
    public static void Main()
    {
        MyClass2 myClass2 = new MyClass2();
        int j = myClass2.i;   // CS0154, no get method
    }
}
```

コンパイラ エラー CS0155

エラー メッセージ
キャッチ、またはスローされた型は System.Exception から派生したものでなければなりません。

System.Exception から派生しないデータ型を catch ブロックに渡そうとしました。**catch** ブロックに渡すことができるのは、**System.Exception** から派生するデータ型だけです。詳細については、「例外処理ステートメント」および「例外と例外処理 (C# プログラミング ガイド)」を参照してください。

次の例では CS0155 エラーが生成されます。

```
// CS0155.cs
using System;

namespace MyNamespace
{
    public class MyClass2
    // try the following line instead
    // public class MyClass2 : Exception
    {
    }
    public class MyClass
    {
        public static void Main()
        {
            try
            {
            }
            catch (MyClass2)   // CS0155, resolves if you derive MyClass2 from Exception
            {
            }
        }
    }
}
```

コンパイラ エラー CS0156

エラー メッセージ
引数なしの throw ステートメントは catch 句以外では使えません。

パラメータがない throw ステートメントを使用できるのは、パラメータを受け取らない **catch** 句だけです。

詳細については、「例外処理ステートメント」および「例外と例外処理 (C# プログラミング ガイド)」を参照してください。

次の例では CS0156 エラーが生成されます。

```
// CS0156.cs
using System;

namespace MyNamespace
{
   public class MyClass2 : Exception
   {
   }

   public class MyClass
   {
      public static void Main()
      {
         try
         {
            throw;   // CS0156
         }

         catch(MyClass2)
         {
            throw;   // this throw is valid
         }
      }
   }
}
```

コンパイラ エラー CS0157

エラー メッセージ
コントロールが finally 句の本体から出られません。

finally 句内のすべてのステートメントを実行する必要があります。詳細については、「例外処理ステートメント」および「例外と例外処理 (C# プログラミング ガイド)」を参照してください。

次の例では CS0157 エラーが生成されます。

```
// CS0157.cs
using System;
namespace MyNamespace
{
   public class MyClass2 : Exception
   {
   }

   public class MyClass
   {
      public static void Main()
      {
         try
         {
         }

         finally
         {
            return;   // CS0157, cannot leave finally clause
         }
      }
   }
}
```

コンパイラ エラー CS0158

エラー メッセージ
スコープ内に、ラベル 'label' と同じ名前のラベルが存在しますが、無視されます。

内部スコープのラベルが、同じ名前を持つ外部スコープのラベルを隠ぺいしています。詳細については、「goto (C# リファレンス)」を参照してください。

次の例では CS0158 エラーが生成されます。

```
// CS0158.cs
namespace MyNamespace
{
   public class MyClass
   {
      public static void Main()
      {
         goto lab1;
         lab1:
         {
            lab1:
            goto lab1;   // CS0158
         }
      }
   }
}
```

コンパイラ エラー CS0159

エラー メッセージ
goto ステートメントのスコープに 'label' というラベルはありません。

goto ステートメントで参照されるラベルが、**goto** ステートメントのスコープ内で見つかりませんでした。

次の例では CS0159 エラーが生成されます。

```
// CS0159.cs
public class Class1
{
   public static void Main()
   {
      int i = 0;

      switch (i)
      {
         case 1:
            goto case 3;   // CS0159, case 3 label does not exist
         case 2:
            break;
      }
      goto NOWHERE;   // CS0159, NOWHERE label does not exist
   }
}
```

コンパイラ エラー CS0160

エラー メッセージ
前の catch 句はこれ、またはスーパー型 ('型') の例外のすべてを既にキャッチしました。

一連の **catch** ステートメントは、派生の降順になっている必要があります。たとえば、最派生オブジェクトを最初に指定する必要があります。

詳細については、「例外処理ステートメント」および「例外と例外処理 (C# プログラミング ガイド)」を参照してください。

次の例では CS0160 エラーが生成されます。

```
// CS0160.cs
public class MyClass2 : System.Exception {}
public class MyClass
{
   public static void Main()
   {
      try {}

      catch(System.Exception) {}   // Second-most derived; should be second catch
      catch(MyClass2) {}   // CS0160  Most derived; should be first catch
   }
}
```

コンパイラ エラー CS0161

エラー メッセージ
'method' : 値を返さないコード パスがあります。

値を返すメソッドでは、すべてのコード パスに **return** ステートメントが必要です。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

次の例では CS0161 エラーが生成されます。

```
// CS0161.cs
public class Test
{
   public static int Main() // CS0161
   {
      int i = 10;
      if (i < 10)
      {
         return i;
      }
      else
      {
         // uncomment the following line to resolve
         // return 1;
      }
   }
}
```

コンパイラ エラー CS0163

エラー メッセージ
コントロールはひとつの case ラベル ('label') から別のラベルへ流れ落ちることはできません。

case ステートメントに 1 つ以上のステートメントが含まれており、その後に別の **case** ステートメントが続く場合は、以下のいずれかのステートメントによってこの **case** ステートメントを明示的に終了する必要があります。

- **return**
- **goto**
- **break**

"移動" 動作を実装する場合は、`goto case #` を使用してください。詳細については、「switch (C# リファレンス)」を参照してください。

次の例では CS0163 エラーが生成されます。

```
// CS0163.cs
public class MyClass
{
   public static void Main()
   {
      int i = 0;

      switch (i)   // CS0163
      {
         case 1:
            i++;
            // uncomment one of the following lines to resolve
            // return;
            // break;
            // goto case 3;

         case 2:
            i++;
            return;

         case 3:
            i = 0;
            return;
      }
   }
}
```

コンパイラ エラー CS0165

エラー メッセージ
未割り当てのローカル変数 'var' が使用されました。

C# コンパイラでは、初期化されていない変数を使用できません。初期化されていない可能性のある変数を検出した場合、コンパイラは CS0165 を生成します。詳細については、「フィールド (C# プログラミング ガイド)」を参照してください。

new を使用してオブジェクトのインスタンスを作成するか、値を代入してください。

次の例では CS0165 エラーが生成されます。

```
// CS0165.cs
using System;

class MyClass
{
   public int i;
}

class MyClass2
{
   public static void Main(string [] args)
   {
      int i, j;
      if (args[0] == "test")
      {
         i = 0;
      }

      /*
      // to resolve, either initialize the variables when declared
      // or provide for logic to initialize them, as follows:
      else
      {
         i = 1;
      }
      */

      j = i;   // CS0165, i might be uninitialized

      MyClass myClass;
      myClass.i = 0;   // CS0165
      // use new as follows
      // MyClass myClass = new MyClass();
      // myClass.i = 0;
   }
}
```

コンパイラ エラー CS0167

エラー メッセージ
デリゲート 'delegate' に Invoke メソッドがありません。

別のコンパイラで作成したマネージ プログラム (.NET Framework 共通言語ランタイムを使用するプログラム) をインポートして使用しました。そのコンパイラでは、不正な形式のデリゲートが許可されています。このため、**Invoke** メソッドを使用できませんでした。詳細については、「デリゲート (C# プログラミング ガイド)」を参照してください。

コンパイラ エラー CS0170

エラー メッセージ
フィールド 'field' は、割り当てられていない可能性があります。

構造体のフィールドが、最初に初期化されずに使用されました。この問題を解決するには、まず初期化されなかったフィールドを特定し、初期化してからアクセスを試みます。構造体の初期化の詳細については、「構造体 (C# プログラミング ガイド)」および「構造体の使用 (C# プログラミング ガイド)」を参照してください。

次の例では CS0170 エラーが生成されます。

```
// CS0170.cs
public struct error
{
   public int i;
}

public class MyClass
{
   public static void Main()
   {
      error e;
      // uncomment the next line to resolve this error
      // e.i = 0;
      System.Console.WriteLine( e.i );   // CS0170 because
                                         //e.i was never assigned
   }
}
```

コンパイラ エラー CS0171

エラー メッセージ
フィールド 'field' は、コントロールがコンストラクタを抜ける前に割り当てられている必要があります。

構造体のコンストラクタでは、構造体内のすべてのフィールドを初期化する必要があります。詳細については、「コンストラクタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0171 エラーが生成されます。

```
// CS0171.cs
struct MyStruct
{
   MyStruct(int initField)   // CS0171
   {
      // i = initField;      // uncomment this line to resolve this error
   }
   public int i;
}

class MyClass
{
   public static void Main()
   {
      MyStruct aStruct = new MyStruct();
   }
}
```

コンパイラ エラー CS0172

エラー メッセージ
'型 1' と '型 2' が暗黙的に変換し合うため、条件式の型がわかりません。

条件付きステートメントでは、: 演算子の両側の型を変換できる必要があります。また、相互変換ルーチンは使用できません。必要な変換は 1 つだけです。詳細については、「変換演算子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0172 エラーが生成されます。

```
// CS0172.cs
public class Square
{
   public class Circle
   {
      public static implicit operator Circle(Square aa)
      {
         return null;
      }

      public static implicit operator Square(Circle aa)
      // using explicit resolves this error
      // public static explicit operator Square(Circle aa)
      {
         return null;
      }
   }

   public static void Main()
   {
      Circle aa = new Circle();
      Square ii = new Square();
      object o = (1 == 1) ? aa : ii;   // CS0172
      // the following cast would resolve this error
      // (1 == 1) ? aa : (Circle)ii;
   }
}
```

コンパイラ エラー CS0173

エラー メッセージ
'型 1' と '型 2' の間に暗黙的な変換がないため、条件式の型がわかりません。

クラスの異なるオブジェクトを 1 つのコードで使用する場合は、クラス間での変換が便利です。ただし、同時に使用する 2 つのクラスで相互変換と冗長変換を行うことはできません。

CS0173 エラーを解決するには、変換の方向や変換が含まれるクラスにかかわらず、*class1* と *class2* の間の暗黙の変換が 1 つだけであるようにします。詳細については、「暗黙的な数値変換の一覧表 (C# リファレンス)」および「変換演算子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0173 エラーが生成されます。

```
// CS0173.cs
public class C {}
public class A {}

public class MyClass
{
   public static void F(bool b)
   {
      A a = new A();
      C c = new C();
      object o = b ? a : c;  // CS0173
   }

   public static void Main()
   {
      F(true);
   }
}
```

コンパイラ エラー CS0174

エラー メッセージ
'base' 参照には基本クラスが必要です。

このエラーが発生するのは、基本クラスを持たない唯一のクラスである **System.Object** クラスのソース コードを .NET Framework 共通言語ランタイムでコンパイルするときだけです。

コンパイラ エラー CS0175

エラー メッセージ
キーワード base はこのコンテキストで無効です。

base (C# リファレンス) キーワードは、基本クラスの特定のメンバを指定するときに使用する必要があります。詳細については、「コンストラクタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0175 エラーが生成されます。

```
// CS0175.cs
using System;
class BaseClass
{
    public int TestInt = 0;
}

class MyClass : BaseClass
{
    public static void Main()
    {
        MyClass aClass = new MyClass();
        aClass.BaseTest();
    }

    public void BaseTest()
    {
        Console.WriteLine(base); // CS0175
        // Try the following line instead:
        // Console.WriteLine(base.TestInt);
        base = 9;   // CS0175

        // Try the following line instead:
        // base.TestInt = 9;
    }
}
```

コンパイラ エラー CS0176

エラー メッセージ
インスタンス参照で静的メンバ 'member' にアクセスできません。typename を代わりに使用してください。

static 変数の修飾に使用できるのはクラス名だけです。インスタンス名は修飾子として使用できません。詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」を参照してください。

次の例では CS0176 エラーが生成されます。

```
// CS0176.cs
public class MyClass2
{
   public static int ii;
}

public class a
{
   public static void Main()
   {
      MyClass2 myClass2 = new MyClass2 ();
      int i = myClass2.ii;   // CS0176
      // try the following line instead
      // int i = MyClass2.ii;
   }
}
```

コンパイラ エラー CS0177

エラー メッセージ
out パラメータ 'parameter' はコントロールが現在のメソッドを抜ける前に割り当てられる必要があります。

out キーワードでマークされたパラメータに、メソッド本体で値が代入されませんでした。詳細については、「パラメータの引き渡し (C# プログラミング ガイド)」を参照してください。

次の例では CS0177 エラーが生成されます。

```
// CS0177.cs
public class MyClass
{
   public static void Foo(out int i)   // CS0177
   {
   // uncomment the following line to resolve this error
   //   i = 0;
   }

   public static void Main()
   {
       int x = -1;
       Foo(out x);
   }
}
```

コンパイラ エラー CS0178

エラー メッセージ
無効な次元指定子です: ',' または ']' を指定してください。

配列の初期化の形式が不正です。たとえば、配列の次元は、以下の方法で指定できます。

- 数値を角かっこ ([]) で囲む。
- 空の角かっこ ([]) を使用する。
- コンマを角かっこ ([]) で囲む。

詳細については、「配列 (C# プログラミング ガイド)」および C# 仕様 (「C# 言語仕様」) の配列初期化子に関するセクションを参照してください。

使用例
次の例では CS0178 エラーが生成されます。

```
// CS0178.cs
class MyClass
{
   public static void Main()
   {
      int a = new int[5][,][][5;   // CS0178
      int[,] b = new int[3,2];   // OK

      int[][] c = new int[10][];
      c[0] = new int[5][5];   // CS0178
      c[0] = new int[2];   // OK
      c[1] = new int[2]{1,2};   // OK
   }
}
```

コンパイラ エラー CS0179

エラー メッセージ
'member' を extern にして、本体を宣言することはできません。

クラス メンバが extern でマークされている場合は、メンバの定義が別のファイルにあることを示します。したがって、**extern** としてマークされたクラス メンバは、クラスで定義できません。**extern** キーワードを削除するか、または定義を削除してください。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

次の例では CS0179 エラーが生成されます。

```
// CS0179.cs
public class MyClass
{
   public extern int ExternMethod(int aa)   // CS0179
   {
      return 0;
   }
   // try the following line instead
   // public extern int ExternMethod(int aa);

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0180

エラー メッセージ
'member' に extern と abstract の両方を指定することはできません。

abstract キーワードと extern キーワードは、同時に指定できません。**extern** キーワードはメンバがファイルの外側で定義されることを示し、**abstract** は実装が派生クラス内で提供されることを示します。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

次の例では CS0180 エラーが生成されます。

```
// CS0180.cs
namespace MyNamespace
{
   public class MyClass
   {
      public extern abstract int Foo(int a);   // CS0180

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0182

エラー メッセージ
属性引数は、定数式、typeof 式、または配列の作成式でなければなりません。

属性の引数が正しく指定されませんでした。詳細については、「グローバル属性 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0182 エラーが生成されます。

```
// CS0182.cs
public class MyClass
{
    static string s = "Test";

    [System.Diagnostics.ConditionalAttribute(s)]   // CS0182
    // try the following line instead
    // [System.Diagnostics.ConditionalAttribute("Test")]
    void NonConstantArgumentToConditional()
    {
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0185

エラー メッセージ
'type' は lock ステートメントによって要求された参照型ではありません。

lock ステートメントが評価できるのは、参照型だけです。詳細については、「スレッドの同期 (C# プログラミング ガイド)」および「参照型 (C# リファレンス)」を参照してください。

使用例
次の例では CS0185 エラーが生成されます。

```
// CS0185.cs
public class MainClass
{
    public static void Main ()
    {
        lock (1)   // CS0185
        // try the following lines instead
        // MainClass x = new MainClass();
        // lock(x)
        {
        }
    }
}
```

コンパイラ エラー CS0186

エラー メッセージ
null はこのコンテキストでは使用できません。

次の例では CS0186 エラーが生成されます。

```
// CS0186.cs
using System;
using System.Collections;

class MyClass
{
   static void Main()
   {
      // Each of the following lines generates CS0186:
      foreach (int i in null) {}   // CS0186
      foreach (int i in (IEnumerable) null) { };   // CS0186
   }
}
```

コンパイラ エラー CS0188

エラー メッセージ
すべてのフィールドが割り当てられるまでは、'this' オブジェクトは使用できません。

struct では、コンストラクタがメソッドを呼び出す前に、コンストラクタによって **struct** 内のすべてのフィールドに値が代入されている必要があります。詳細については、「構造体の使用 (C# プログラミング ガイド)」を参照してください。

次の例では CS0188 エラーが生成されます。

```
// CS0188.cs
// compile with: /t:library
namespace MyNamespace
{
    class MyClass
    {
        struct S
        {
            public int a;

            void Foo()
            {
            }

            S(int i)
            {
                // a = i;
                Foo();  // CS0188
            }
        }
        public static void Main()
        { }

    }
}
```

コンパイラ エラー CS0191

エラー メッセージ
読み取り専用フィールドに割り当てることはできません (コンストラクタ、変数初期化子では可)。

readonly フィールドでは、コンストラクタまたは宣言時の代入だけを行うことができます。詳細については、
「コンストラクタ (C# プログラミング ガイド)」を参照してください。

readonly フィールドが static であり、コンストラクタが **static** としてマークされていない場合にも、CS0191 エラーが生成されます。

使用例
次の例では、CS0191 エラーが生成されます。

```
// CS0191.cs
class MyClass
{
    public readonly int TestInt = 6;  // OK to assign to readonly field in declaration

    MyClass()
    {
        TestInt = 11; // OK to assign to readonly field in constructor
    }

    public void TestReadOnly()
    {
        TestInt = 19;                  // CS0191
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0192

エラー メッセージ
読み取り専用フィールドに ref または out を渡すことはできません (コンストラクタでは可)。

readonly キーワードでマークされたフィールド (変数) は、コンストラクタ内部を除き、ref パラメータや out パラメータに渡すことはできません。詳細については、「フィールド (C# プログラミング ガイド)」を参照してください。

readonly フィールドが static であり、コンストラクタが **static** としてマークされていない場合にも、CS0192 エラーが生成されます。

使用例
次の例では、CS0192 エラーが生成されます。

```
// CS0192.cs
class MyClass
{
    public readonly int TestInt = 6;
    static void TestMethod(ref int testInt)
    {
        testInt = 0;
    }

    MyClass()
    {
        TestMethod(ref TestInt);   // OK
    }

    public void PassReadOnlyRef()
    {
        TestMethod(ref TestInt);   // CS0192
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0193

エラー メッセージ
* または -> 演算子はポインタに対して使用してください。

* 演算子または -> 演算子が、ポインタ以外の型と一緒に使用されました。詳細については、「ポインタ型 (C# プログラミング ガイド)」を参照してください。

次の例では CS0193 エラーが生成されます。

```
// CS0193.cs
using System;

public struct Age
{
   public int AgeYears;
   public int AgeMonths;
   public int AgeDays;
}

public class MyClass
{
   public static void SetAge(ref Age anAge, int years, int months, int days)
   {
      anAge->Months = 3;   // CS0193, anAge is not a pointer
      // try the following line instead
      // anAge.AgeMonths = 3;
   }

   public static void Main()
   {
      Age MyAge = new Age();
      Console.WriteLine(MyAge.AgeMonths);
      SetAge(ref MyAge, 22, 4, 15);
      Console.WriteLine(MyAge.AgeMonths);
   }
}
```

コンパイラ エラー CS0196

エラー メッセージ
ポインタのインデックスを複数指定しないでください。

ポインタは複数のインデックスを持つことはできません。詳細については、「ポインタ型 (C# プログラミング ガイド)」を参照してください。

次の例では CS0196 エラーが生成されます。

```
// CS0196.cs
public class MyClass
{
   public static void Main ()
   {
      int *i = null;
      int j = 0;
      j = i[1,2];   // CS0196
      // try the following line instead
      // j = i[1];
   }
}
```

コンパイラの警告 (レベル 1) CS0197

エラー メッセージ
参照マーシャリング クラスのフィールドであるため、'引数' を ref または out として渡す、またはそのアドレスを取得すると、ランタイム例外が発生する可能性があります。

MarshalByRefObject から直接または間接的に派生するクラスは、参照渡しによるマーシャリング クラスです。このようなクラスは、プロセスとコンピュータの境界にまたがって、参照渡しによるマーシャリングができます。したがって、このクラスのインスタンスは、リモート オブジェクトに対するプロキシ オブジェクトになる場合があります。プロキシ オブジェクトのフィールドを ref または out として渡すことはできません。このため、プロキシ オブジェクトにできない this がインスタンスでない場合は、このようなクラスのフィールドを **ref** または **out** として渡すことはできません。

使用例
次の例では CS0197 エラーが生成されます。

```
// CS0197.cs
// compile with: /W:1
class X : System.MarshalByRefObject
{
   public int i;
}

class M
{
   public int i;
   static void AddSeventeen(ref int i)
   {
      i += 17;
   }

   static void Main()
   {
      X x = new X();
      x.i = 12;
      AddSeventeen(ref x.i);   // CS0197

      // OK
      M m = new M();
      m.i = 12;
      AddSeventeen(ref m.i);
   }
}
```

コンパイラ エラー CS0198

エラー メッセージ
静的読み取り専用フィールドへの割り当てはできません (静的コンストラクタまたは変数初期化子では可)。

readonly 変数では、変数を初期化するコンストラクタと同じ static の使用が必要です。詳細については、「静的コンストラクタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0198 エラーが生成されます。

```
// CS0198.cs
class MyClass
{
   public static readonly int TestInt = 6;

   MyClass()
   {
      TestInt = 11;   // CS0198, constructor is not static and readonly field is
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0199

エラー メッセージ
読み取り専用フィールドに ref または out を渡すことはできません (静的コンストラクタでは可)。

readonly 変数では、変数を ref パラメータまたは out パラメータとして渡すコンストラクタと同じ static の使用が必要です。詳細については、「パラメータの引き渡し (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0199 エラーが生成されます。

```
// CS0199.cs
class MyClass
{
    public static readonly int TestInt = 6;

    static void TestMethod(ref int testInt)
    {
        testInt = 0;
    }

    MyClass()
    {
        TestMethod(ref TestInt);   // CS0199, TestInt is static
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0200

エラー メッセージ
プロパティまたはインデクサ 'property' は読み取り専用なので割り当てすることはできません。

プロパティへの値の代入を試みましたが、そのプロパティに set アクセサがありません。**set** アクセサを追加してエラーを解決してください。詳細については、「方法 : 読み取り/書き込みプロパティを宣言および使用する (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0200 エラーが生成されます。

```
// CS0200.cs
public class MainClass
{
    // private int _mi;
    int I
    {
        get
        {
            return 1;
        }

        // uncomment the set accessor and declaration for _mi
        /*
        set
        {
            _mi = value;
        }
        */
    }

    public static void Main ()
    {
        MainClass II = new MainClass();
        II.I = 9;   // CS0200
    }
}
```

コンパイラ エラー CS0201

エラー メッセージ
割り当て、呼び出し、インクリメント、デクリメント、および新しいオブジェクトの式のみがステートメントとして使用できます。

意味を持たないステートメントが検出されると、コンパイラでこのエラーが生成されます。

使用例
次の例では CS0201 エラーが生成されます。

```
// CS0201.cs
public class MainClass
{
   public static void Main()
   {
      2 * 3;   // CS0201
   }
}
```

次の例では CS0201 エラーが生成されます。

```
// CS0201_b.cs
// compile with: /target:library
public class MyList<T>
{
   public void Add(T x)
   {
      int i = 0;
      if ( (object)x == null)
      {
         checked(i++);   // CS0201

         // OK
         checked {
            i++;
         }
      }
   }
}
```

コンパイラ エラー CS0202

エラー メッセージ
foreach では、戻り値の型 'type.GetEnumerator()' の '型' に適切なパブリック MoveNext メソッドおよびパブリック Current プロパティが含まれている必要があります。

GetEnumerator は、foreach ステートメントの使用を可能にする関数です。ポインタや配列を返すことはできません。この関数が列挙子として機能するためにはクラスのインスタンスを返すことが必要です。厳密には public の Current プロパティと、public の MoveNext メソッドがあって初めて、列挙子としての役割を果たすことができます。

メモ :

C# 2.0 では、Current と MoveNext がコンパイラによって自動的に生成されます。詳細については、「ジェネリック インターフェイス (C# プログラミング ガイド)」のコード例を参照してください。

次の例では CS0202 エラーが生成されます。

```
// CS0202.cs

public class C1
{
   public int Current
   {
      get
      {
         return 0;
      }
   }

   public bool MoveNext ()
   {
      return false;
   }

   public static implicit operator C1 (int c1)
   {
      return 0;
   }
}

public class C2
{
   public int Current
   {
      get
      {
         return 0;
      }
   }

   public bool MoveNext ()
   {
      return false;
   }

   public C1[] GetEnumerator ()
   // try the following line instead
   // public C1 GetEnumerator ()
   {
      return null;
   }
}

public class MainClass
```

```
{
   public static void Main ()
   {
      C2 c2 = new C2();

      foreach (C1 x in c2)   // CS0202
      {
         System.Console.WriteLine(x.Current);
      }
   }
}
```

コンパイラ エラー CS0204

エラー メッセージ
使用できるローカル数は 65535 です。

メソッドで、ローカル変数の制限 65535 を超えました。

コンパイラ エラー CS0205

エラー メッセージ
抽象基本メンバを呼び出すことはできません : 'method'

抽象メソッドは本体を持たないため、呼び出すことはできません。詳細については、
「抽象クラスとシール クラス、およびクラス メンバ (C# プログラミング ガイド)」を参照してください。

次の例では CS0205 エラーが生成されます。

```
// CS0205.cs
abstract public class MyClass
{
   abstract public void mf();
}

public class MyClass2 : MyClass
{
   public override void mf()
   {
      base.mf();   // CS0205, delete this line
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0206

エラー メッセージ
プロパティまたはインデクサを out か ref のパラメータとして渡すことはできません。

プロパティを ref パラメータまたは out パラメータとして渡すことはできません。詳細については、「パラメータの引き渡し (C# プログラミング ガイド)」を参照してください。

使用例

次の例では CS0206 エラーが生成されます。

```
// CS0206.cs
public class MyClass
{
    public static int P
    {
        get
        {
            return 0;
        }
        set
        {
        }
    }

    public static void MyMeth(ref int i)
    // public static void MyMeth(int i)
    {
    }

    public static void Main()
    {
        MyMeth(ref P);   // CS0206
        // try the following line instead
        // MyMeth(P);   // CS0206
    }
}
```

コンパイラ エラー CS0208

エラー メッセージ
マネージ型のアドレスの取得、マネージ型のサイズの取得、またはマネージ型へのポインタの宣言が実行できません ('型')

unsafe キーワードを使用した場合でも、マネージ オブジェクトのアドレスやサイズを取得したり、マネージ型へのポインタを宣言したりすることはできません。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0208 エラーが生成されます。

```
// CS0208.cs
// compile with: /unsafe

class S
{
    public int a = 98;
}

public class MyClass
{
    unsafe public static int Main()
    {
        S s = new S();  // S is managed
        S * s2 = &s;    // CS0208
        return 1;
    }
}
```

コンパイラ エラー CS0209

エラー メッセージ
fixed ステートメントで宣言されたローカルの型は、ポインタ型でなければなりません。

fixed ステートメントで宣言する変数は、ポインタであることが必要です。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0209 エラーが生成されます。

```
// CS0209.cs
// compile with: /unsafe

class Point
{
   public int x, y;
}

public class MyClass
{
   unsafe public static void Main()
   {
      Point pt = new Point();

      fixed (int i)    // CS0209
      {
      }
      // try the following lines instead
      /*
      fixed (int* p = &pt.x)
      {
      }
      fixed (int* q = &pt.y)
      {
      }
      */
   }
}
```

コンパイラ エラー CS0210

エラー メッセージ
fixed または using ステートメントの宣言の中に、初期化子を指定してください。

変数は fixed ステートメントで宣言および初期化する必要があります。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0210 エラーが生成されます。

```
// CS0210a.cs
// compile with: /unsafe

class Point
{
   public int x, y;
}

public class MyClass
{
   unsafe public static void Main()
   {
      Point pt = new Point();

      fixed (int i)    // CS0210
      {
      }
      // try the following lines instead
      /*
      fixed (int* p = &pt.x)
      {
      }
      fixed (int* q = &pt.y)
      {
      }
      */
   }
}
```

次の例では、using ステートメントに初期化子がないため、CS0210 エラーが生成されます。

```
// CS0210b.cs

using System.IO;
class Test
{
   static void Main()
   {
      using (StreamWriter w) // CS0210
      // Try this line instead:
      // using (StreamWriter w = new StreamWriter("TestFile.txt"))
      {
         w.WriteLine("Hello there");
      }
   }
}
```

コンパイラ エラー CS0211

エラー メッセージ
式のアドレスを取得できません。

フィールドのアドレス、ローカル変数、およびポインタの間接は使用できますが、たとえば 2 つのローカル変数の合計のアドレスを使用することはできません。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0211 エラーが生成されます。

```
// CS0211.cs
// compile with: /unsafe

public class MyClass
{
   unsafe public void mf()
   {
      int a = 0, b = 0;
      int *i = &(a + b);   // CS0211, the addition of two local variables
      // try the following line instead
      // int *i = &a;
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0212

エラー メッセージ
fixed ステートメントの初期化子内の fixed でないステートメントのアドレスのみを取得できます。

詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

次の例では、固定されていない式のアドレスの使用方法を示します。次の例では CS0212 エラーが生成されます。

```
// CS0212a.cs
// compile with: /unsafe /target:library

public class A {
   public int iField = 5;

   unsafe public void mf() {
      A a = new A();
      int* ptr = &a.iField;   // CS0212
   }

   // OK
   unsafe public void mf2() {
      A a = new A();
      fixed (int* ptr = &a.iField) {}
   }
}
```

次のサンプルも CS0212 エラーになります。このエラーの解決方法も示しています。

```
// CS0212b.cs
// compile with: /unsafe /target:library
using System;

public class MyClass
{
   unsafe public void mf()
   {
      // Null-terminated ASCII characters in an sbyte array
      sbyte[] sbArr1 = new sbyte[] { 0x41, 0x42, 0x43, 0x00 };
      sbyte* pAsciiUpper = &sbArr1[0];   // CS0212
      // To resolve this error, delete the previous line and
      // uncomment the following code:
      // fixed (sbyte* pAsciiUpper = sbArr1)
      // {
      //    String szAsciiUpper = new String(pAsciiUpper);
      // }
   }
}
```

コンパイラ エラー CS0213

エラー メッセージ
既に fixed が使用されている式のアドレスを取得するために、fixed ステートメントを使用することはできません。

unsafe メソッド内やパラメータ内のローカル変数は既にスタックに固定されているため、fixed 式ではこれら 2 つの変数のどちらのアドレスも使用できません。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0213 エラーが生成されます。

```
// CS0213.cs
// compile with: /unsafe
public class MyClass
{
   unsafe public static void Main()
   {
      int i = 45;
      fixed (int *j = &i) { }  // CS0213
      // try the following line instead
      // int* j = &i;

      int[] a = new int[] {1,2,3};
      fixed (int *b = a)
      {
         fixed (int *c = b) { }  // CS0213
         // try the following line instead
         // int *c = b;
      }
   }
}
```

コンパイラ エラー CS0214

エラー メッセージ
ポインタおよび固定サイズ バッファは、unsafe コンテキストでのみ使用することができます。

ポインタと共に使用できるキーワードは、unsafe キーワードだけです。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0214 エラーが生成されます。

```
// CS0214.cs
// compile with: /target:library /unsafe
public struct S
{
   public int a;
}

public class MyClass
{
   public static void Test()
   {
      S s = new S();
      S * s2 = &s;    // CS0214
      s2->a = 3;      // CS0214
      s.a = 0;
   }

   // OK
   unsafe public static void Test2()
   {
      S s = new S();
      S * s2 = &s;
      s2->a = 3;
      s.a = 0;
   }
}
```

コンパイラ エラー CS0215

エラー メッセージ
演算子 true または false の戻り値はブール型でなければなりません。

ユーザー定義の true 演算子と false 演算子の戻り値の型は、bool 型であることが必要です。詳細については、「オーバーロードされた演算子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0215 エラーが生成されます。

```
// CS0215.cs
class MyClass
{
   public static int operator true (MyClass MyInt)   // CS0215
   // try the following line instead
   // public static bool operator true (MyClass MyInt)
   {
      return true;
   }

   public static int operator false (MyClass MyInt)   // CS0215
   // try the following line instead
   // public static bool operator false (MyClass MyInt)
   {
      return true;
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0216

エラー メッセージ
演算子 'operator' を定義するには、合致する演算子 'missing_operator' が必要です。

ユーザー定義の true 演算子にはユーザー定義の false 演算子が必要であり、その逆もまた同様です。詳細については、「演算子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0216 エラーが生成されます。

```
// CS0216.cs
class MyClass
{
   public static bool operator true (MyClass MyInt)   // CS0216
   {
      return true;
   }

   // to resolve, uncomment the following operator definition
   /*
   public static bool operator false (MyClass MyInt)
   {
      return true;
   }
   */

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0217

エラー メッセージ
short circuit 演算子として適用するためには、ユーザー定義の論理演算子 ('operator') がその 2 つのパラメータと同じ戻り値の型を持つ必要があります。

ユーザー定義型として定義した後にショートサーキット演算子として使用する演算子には、同じ型のパラメータと戻り値が必要です。ショートサーキット演算子の詳細については、「&& 演算子」および「|| 演算子」を参照してください。

次の例では CS0217 エラーが生成されます。

```
// CS0217.cs
using System;

public class MyClass
{
   public static bool operator true (MyClass f)
   {
      return false;
   }

   public static bool operator false (MyClass f)
   {
      return false;
   }

   public static implicit operator int(MyClass x)
   {
      return 0;
   }

   public static int operator & (MyClass f1, MyClass f2)   // CS0217
   // try the following line instead
   // public static MyClass operator & (MyClass f1, MyClass f2)
   {
      return new MyClass();
   }

   public static void Main()
   {
      MyClass f = new MyClass();
      int i = f && f;
   }
}
```

参照

関連項目

オーバーロードされた演算子 (C# プログラミング ガイド)

コンパイラ エラー CS0218

エラー メッセージ
型 ('type') に演算子 true および演算子 false の宣言が含まれている必要があります。

ユーザー定義型として定義した後にショートサーキット演算子として使用する演算子には、定義済みの **true** 演算子と **false** 演算子が必要です。ショートサーキット演算子の詳細については、「&& 演算子」および「|| 演算子」を参照してください。

次の例では CS0218 エラーが生成されます。

```
// CS0218.cs
using System;
public class MyClass
{
   // uncomment these operator declarations to resolve this CS0218
   /*
   public static bool operator true (MyClass f)
   {
      return false;
   }

   public static bool operator false (MyClass f)
   {
      return false;
   }
   */

   public static implicit operator int(MyClass x)
   {
      return 0;
   }

   public static MyClass operator & (MyClass f1, MyClass f2)
   {
      return new MyClass();
   }

   public static void Main()
   {
      MyClass f = new MyClass();
      int i = f && f;   // CS0218, requires operators true and false
   }
}
```

参照

概念

変換演算子 (C# プログラミング ガイド)

コンパイラ エラー CS0220

エラー メッセージ
この操作はチェック モードでコンパイルしたときにオーバーフローします。

既定の checked によって、データの損失となった演算が検出されました。このエラーを解決するには、代入に対する入力を訂正するか、または unchecked を使用します。詳細については、「チェックありとチェックなし (C# リファレンス)」を参照してください。

次の例では CS0220 エラーが生成されます。

```
// CS0220.cs
using System;

class TestClass
{
   const int x = 1000000;
   const int y = 1000000;

   public int MethodCh()
   {
      int z = (x * y);   // CS0220
      return z;
   }

   public int MethodUnCh()
   {
      unchecked
      {
         int z = (x * y);
         return z;
      }
   }

   public static void Main()
   {
      TestClass myObject = new TestClass();
      Console.WriteLine("Checked  : {0}", myObject.MethodCh());
      Console.WriteLine("Unchecked: {0}", myObject.MethodUnCh());
   }
}
```

コンパイラ エラー CS0221

エラー メッセージ
定数値 'value' は 'type' に変換できません。(unchecked 構文を使ってオーバーライドしてください。)

既定でオンになっている checked によって、データの損失となる代入演算が検出されました。このエラーを解決するには、代入を訂正するか、または unchecked を使用します。詳細については、「チェックありとチェックなし (C# リファレンス)」を参照してください。

次の例では CS0221 エラーが生成されます。

```
// CS0221.cs
public class MyClass
{
   public static void Main()
   {
      // unchecked
      // {
         int a = (int)0xFFFFFFFF;   // CS0221
         a++;
      // }
   }
}
```

コンパイラ エラー CS0225

エラー メッセージ
params パラメータは 1 次元配列でなければなりません。

params キーワードを使用するときは、データ型の 1 次元配列を指定する必要があります。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0225 エラーが生成されます。

```
// CS0225.cs
public class MyClass
{
    public static void TestParams(params int a)   // CS0225
    // try the following line instead
    // public static void TestParams(params int[] a)
    {
    }

    public static void Main()
    {
        TestParams(1);
    }
}
```

コンパイラ エラー CS0227

エラー メッセージ
unsafe コードは /unsafe でコンパイルした場合のみ有効です。

ソース コードに unsafe キーワードがある場合は、/unsafe コンパイラ オプションも使用する必要があります。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

次の例では、**/unsafe** を使用せずにコンパイルすると CS0227 エラーが生成されます。

```
// CS0227.cs
public class MyClass
{
   unsafe public static void Main()   // CS0227
   {
   }
}
```

コンパイラ エラー CS0228

エラー メッセージ
'type' に 'member' の定義がないか、アクセスできません。

System.Hashtable、**System.String**、または **System.Array** のシステム バージョンを置き換えた可能性があり、その置き換えには *member* が含まれていません。

それ以外の場合は、Visual Studio を修復または再インストールする必要があります。

コンパイラ エラー CS0229

エラー メッセージ
'member1' と 'member2' があいまいです。

異なるインターフェイスのメンバの名前が同じです。同じ名前を使用する場合は、名前を限定する必要があります。詳細については、「インターフェイス (C# プログラミング ガイド)」を参照してください。

メモ :

このようなあいまいさは、using エイリアスを使用して識別子に明示的なプリフィックスを指定すると、解決できる場合があります。

使用例
次の例では、CS0229 エラーが生成されます。

```
// CS0229.cs

interface IList
{
    int Count
    {
        get;
        set;
    }

    void Counter();
}

interface Icounter
{
    double Count
    {
        get;
        set;
    }
}

interface IListCounter : IList , Icounter {}

class MyClass
{
    void Test(IListCounter x)
    {
        x.Count = 1;  // CS0229
        // Try one of the following lines instead:
        // ((IList)x).Count = 1;
        // or
        // ((Icounter)x).Count = 1;
    }

    public static void Main() {}
}
```

コンパイラ エラー CS0230

エラー メッセージ
foreach ステートメントには、型と識別子の両方が必要です。

foreach ステートメントの形式が不正です。

次の例では CS0230 エラーが生成されます。

```
// CS0230.cs
using System;

class MyClass
{
   public static void Main()
   {
      int[] myarray = new int[3] {1,2,3};

      foreach (int in myarray)   // CS0230
      // try the following line instead
      // foreach (int x in myarray)
      {
         Console.WriteLine(x);
      }
   }
}
```

コンパイラ エラー CS0231

エラー メッセージ
params パラメータは正式なパラメータ リストで最後のパラメータでなければなりません。

params パラメータは、引数の数を可変にすることができ、他のすべてのパラメータの後に指定する必要があります。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

次の例では CS0231 エラーが生成されます。

```
// CS0231.cs
class Test
{
   public void TestMethod(params int[] p, int i) {} // CS0231
   // To resolve the error, use the following line instead:
   // public void TestMethod(int i, params int[] p) {}

   static void Main()
   {
   }
}
```

コンパイラ エラー CS0233

エラー メッセージ
'識別子' には定義済みの型が指定されていないため、sizeof は unsafe コンテキスト でのみ使用できます
(System.Runtime.InteropServices.Marshal.SizeOf の使用をお勧めします)

sizeof 演算子は、コンパイル時の定数である型に対してのみ使用できます。このエラーが発生した場合は、コンパイル時にサイズを取得することが可能な識別子であるかどうかを確認してください。そうでない場合は、**sizeof** ではなく、SizeOf を使用します。

使用例
次の例では、CS0233 エラーが生成されます。

```
// CS0233.cs
using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Sequential)]
public struct S
{
    public int a;
}

public class MyClass
{
    public static void Main()
    {
        S myS = new S();
        Console.WriteLine(sizeof(S));   // CS0233
        // Try the following line instead:
        // Console.WriteLine(Marshal.SizeOf(myS));
   }
}
```

コンパイラ エラー CS0234

エラー メッセージ
型または名前空間名 '名前' は名前空間 '名前空間' に存在しません。アセンブリ参照が不足しています。

型を指定する必要があります。以下の原因が考えられます。

- 型の定義を含むアセンブリがコンパイルで参照されませんでした。/reference (メタデータのインポート) を使用してアセンブリを指定してください。
- typeof 演算子に変数名を渡しました。

開発環境で参照を追加する方法については、「[参照の追加] ダイアログ ボックス」を参照してください。

次の例では CS0234 エラーが生成されます。

```
// CS0234.cs
public class C
{
   public static void Main()
   {
      System.DateTime x = new System.DateTim();   // CS0234
      // try the following line instead
      // System.DateTime x = new System.DateTime();
   }
}
```

コンパイラ エラー CS0236

エラー メッセージ
フィールド初期化子は静的でないフィールド、メソッド、またはプロパティ 'field' を参照できません。

インスタンス フィールドは、メソッドの外側にあるほかのインスタンス フィールドの初期化には使用できません。メソッドの外部で変数を初期化する場合は、クラス コンストラクタの内部で初期化を実行することをお勧めします。詳細については、「メソッド (C# プログラミング ガイド)」を参照してください。

次の例では CS0236 エラーが生成されます。

```
// CS0236.cs
public class MyClass
{
   public int i = 5;
   public int j = i;  // CS0236
   public int k;      // initialize in constructor

   MyClass()
   {
      k = i;
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0238

エラー メッセージ
override ではないため、'member' をシールすることはできません。

override としてマークされていないメンバで sealed が使用されました。詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

次の例では CS0238 エラーが生成されます。

```
// CS0238.cs
abstract class MyClass
{
   public abstract void f();
}

class MyClass2 : MyClass
{
   public static void Main()
   {
   }

   public sealed void f() // CS0238
   // Try the following definition instead:
   // public override sealed void f()
   {
   }
}
```

コンパイラ エラー CS0239

エラー メッセージ
'member' : シールされているため、メンバ 'inherited member' をオーバーライドできません。

継承された sealed メンバをメンバでオーバーライドできません。詳細については、「チェックありとチェックなし (C# リファレンス)」を参照してください。

次の例では CS0239 エラーが生成されます。

```
// CS0239.cs
abstract class MyClass
{
   public abstract void f();
}

class MyClass2 : MyClass
{
   public static void Main()
   {
   }

   public override sealed void f()
   {
   }
}

class MyClass3 : MyClass2
{
   public override void f()   // CS0239
   // Try the following definition instead:
   // public new void f()
   {
   }
}
```

コンパイラ エラー CS0241

エラー メッセージ
既定のパラメータ指定子は使用できません。

メソッドのパラメータには既定値を設定できません。同様の機能が必要な場合は、メソッドのオーバーロードを使用します。詳細については、「パラメータの引き渡し (C# プログラミング ガイド)」を参照してください。

使用例

次の例では CS0241 エラーが生成されます。またこの例では、オーバーロードを使用して、既定の引数でメソッドをシミュレートする方法も示します。

```
// CS0241.cs
public class A
{
   public void Test(int i = 9) {}   // CS0241
}

public class B
{
   public void Test() { Test(9); }
   public void Test(int i)  {}
}

public class C
{
   public static void Main()
   {
      B x = new B();
      x.Test();
   }
}
```

コンパイラ エラー CS0242

エラー メッセージ
問題の操作は void ポインタで定義されていません。

void ポインタはインクリメントできません。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0242 エラーが生成されます。

```
// CS0242.cs
// compile with: /unsafe
class TestClass
{
   public unsafe void Test()
   {
      void * p = null;
      p++;   // CS0242, incrementing a void pointer not allowed
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0243

エラー メッセージ
オーバーライド メソッドであるため、条件付き属性は 'メソッド' では無効です。

override キーワードでマークされたメソッドでは、Conditional 属性を使用できません。詳細については、
「Override キーワードと New キーワードを使用する場合について (C# プログラミング ガイド)」を参照してください。

コンパイラではオーバーライド メソッドにバインドせず、基本メソッドにバインドし、共通言語ランタイムが必要に応じてオーバーライドを呼び出します。

次の例では CS0243 エラーが生成されます。

```
// CS0243.cs
// compile with: /target:library
public class MyClass
{
   public virtual void M() {}
}

public class MyClass2 : MyClass
{
   [System.Diagnostics.ConditionalAttribute("MySymbol")]   // CS0243
   // remove Conditional attribute or remove override keyword
   public override void M() {}
}
```

コンパイラ エラー CS0244

エラー メッセージ
"is" と "as" のどちらもポインタ型では無効です

is キーワードと as キーワードは、ポインタ型に対しては使用できません。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0244 エラーが生成されます。

```
// CS0244.cs
// compile with: /unsafe

class UnsafeTest
{
   unsafe static void SquarePtrParam (int* p)
   {
      bool b = p is object;   // CS0244 p is pointer
   }

   unsafe public static void Main()
   {
      int i = 5;
      SquarePtrParam (&i);
   }
}
```

コンパイラ エラー CS0245

エラー メッセージ
デストラクタと object.Finalize を直接呼び出すことはできません。使用可能であれば IDisposable.Dispose を呼び出してください。

詳細については、「ガベージ コレクションのプログラミング」および「デストラクタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0245 エラーが生成されます。

```
// CS0245.cs
using System;
using System.Collections;

class MyClass // : IDisposable
{
   /*
   public void Dispose()
   {
      // cleanup code goes here
   }
   */

   void m()
   {
      this.Finalize();   // CS0245
      // this.Dispose();
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0246

エラー メッセージ
型または名前空間名 'type/namespace' が見つかりませんでした。ディレクティブを使うかアセンブリ参照を使ってください。

型が見つかりませんでした。型を含むアセンブリを参照 (/reference) するのを忘れているか、または using ディレクティブを使用して正しく限定していない可能性があります。

このエラーには次のようないくつかの原因があります。

1. 使用する型または名前空間のスペルが間違っている可能性があります (大文字と小文字の区別も含む)。名前が正しくないと、コンパイラは、コード内で参照されている型または名前空間の定義を検索できません。C# では大文字と小文字が区別されるので、大文字と小文字が間違っていると、型の参照時に正しいキャストが使用されません。このエラーが最も頻繁に発生します。たとえば、`Dataset ds;` では CS0246 エラーが生成されます。これは、Dataset の s が大文字になっていないためです。
2. 名前空間名にエラーがある場合、その名前空間を含むアセンブリを正しく参照 (/reference) できていない可能性があります。たとえば、コードに `using Accessibility;` という指定があるとします。しかし、プロジェクトが Accessibility.dll アセンブリを参照していない場合、CS0246 エラーが生成されます。開発環境で参照を追加する方法については、「[参照の追加] ダイアログ ボックス」を参照してください。
3. 型名にエラーがある場合は、適切な using ディレクティブを使用していないか、または型名が完全には限定されていない可能性があります。`DataSet ds;` というコード行があるとします。DataSet 型を使用するには、次の 2 つが必要になります。第 1 に、DataSet 型の定義を含むアセンブリへの参照が必要です。第 2 に、DataSet が配置される名前空間に using ディレクティブが必要です。たとえば、DataSet は **System.Data** 名前空間に配置されるため、コードの先頭に `using System.Data;` というステートメントが必要です。

   第 2 のステップは必須ではありません。ただし、このステップを省略した場合は、参照時に DataSet 型を完全限定する必要があります。完全限定とは、コードで参照するたびに名前空間と型を使用することです。このため、第 2 のステップをスキップする場合は、上で示した宣言コードを `System.Data.DataSet ds;` に変更する必要があります。
4. 型そのものに問題がない場合、型を指定しなければならない場所で変数などを使用している可能性があります。たとえば、**is** ステートメントで、実際の型ではなく型オブジェクトを使用した場合、このエラーが発生します。

次の例では CS0246 エラーが生成されます。

```
// CS0246.cs
// using System.Diagnostics;

public class MyClass
{
   [Conditional("A")]   // CS0246, uncomment using directive to resolve
   public void Test()
   {
   }

   public static void Main()
   {
   }
}
```

実際の型を指定しなければならない場所で型オブジェクトが使用されている例を次に示します。前述の 4 つ目のケースに相当します。

```
// CS0246b.cs
using System;

class C
{
    public bool supports(object o, Type t)
    {
         if (o is t)  // CS0246 - t is not a type
         {
            return true;
         }
         return false;
    }
```

```
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0247

エラー メッセージ
stackalloc で負のサイズを使うことはできません。

負の数が stackalloc ステートメントに渡されました。

次の例では CS0247 エラーが生成されます。

```
// CS0247.cs
// compile with: /unsafe
public class MyClass
{
   unsafe public static void Main()
   {
      int *p = stackalloc int [-30];   // CS0247
   }
}
```

コンパイラ エラー CS0248

エラー メッセージ
負のサイズで配列を作成することはできません。

配列のサイズが負の数で指定されました。詳細については、「配列 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0248 エラーが生成されます。

```
// CS0248.cs
class MyClass
{
    public static void Main()
    {
        int[] myArray = new int[-3] {1,2,3};   // CS0248, pass a nonnegative number
    }
}
```

コンパイラ エラー CS0249

エラー メッセージ
object.Finalize をオーバーライドしないでください。代わりにデストラクタを提供してください。

デストラクタの構文を使用して、オブジェクトが破棄されるときに実行する命令を指定してください。

詳細については、「C# および C++ のデストラクタ構文」を参照してください。

次の例では CS0249 エラーが生成されます。

```
// CS0249.cs
class MyClass
{
   protected override void Finalize()   // CS0249
   // try the following line instead
   // ~MyClass()
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0250

エラー メッセージ
基本クラスの Finalize メソッドを直接呼び出さないでください。デストラクタから自動的に呼び出されます。

プログラムでは基本クラスのリソースを強制的にクリーンアップできません。

詳細については、「Finalize メソッドおよびデストラクタ」を参照してください。

次の例では CS0250 エラーが生成されます。

```
// CS0250.cs
class B
{
}

class C : B
{
   ~C()
   {
      base.Finalize();   // CS0250
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0254

エラー メッセージ
固定ステートメントの代入式の右辺はキャスト式ではない可能性があります。

fixed 式の右辺ではキャストを使用できません。詳細については、「unsafe コードとポインタ (C# プログラミング ガイド)」を参照してください。

次の例では CS0254 エラーが生成されます。

```
// CS0254.cs
// compile with: /unsafe
class Point
{
   public uint x, y;
}

class FixedTest
{
   unsafe static void SquarePtrParam (int* p)
   {
      *p *= *p;
   }

   unsafe public static void Main()
   {
      Point pt = new Point();
      pt.x = 5;
      pt.y = 6;

      fixed (int* p = (int*)&pt.x)   // CS0254
      // try the following line instead
      // fixed (uint* p = &pt.x)
      {
         SquarePtrParam ((int*)p);
      }
   }
}
```

コンパイラ エラー CS0255

エラー メッセージ
stackalloc は catch または finally ブロックで使用されない可能性があります。

stackalloc キーワードは、catch ブロックや finally ブロックでは使用できません。詳細については、「例外と例外処理 (C# プログラミング ガイド)」を参照してください。

次の例では CS0255 エラーが生成されます。

```
// CS0255.cs
// compile with: /unsafe
using System;

public class TestTryFinally
{
   public static unsafe void Test()
   {
      int i = 123;
      string s = "Some string";
      object o = s;

      try
      {
         // Conversion is not valid; o contains a string not an int
         i = (int) o;
      }

      finally
      {
         Console.Write("i = {0}", i);
         int* fib = stackalloc int[100];   // CS0255
      }
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0260

エラー メッセージ
部分識別子が型 '型' にありません。この型の別の部分宣言が存在します。

このエラーは、部分型としては宣言されていない、同じ名前の複数のクラス宣言が存在する場合に発生します。クラスを分割して定義する場合、分割定義されたすべてのクラスは、**partial** というキーワードで宣言されている必要があります。このエラーは、新しいクラスを作成したとき、偶然同じ名前の部分クラスが、同一名前空間内で宣言されていた場合にも発生します。

次の例では CS0260 エラーが生成されます。

```
// CS0260.cs
class C  // CS0260
{
}

partial class C
{
}
```

コンパイラ エラー CS0261

エラー メッセージ
'型' の部分宣言は、すべてのクラス、すべての構造体、またはすべてのインターフェイスにする必要があります。

このエラーは、部分型が場所によって異なる型として宣言されている場合に発生します。詳細については、「部分クラス定義 (C# プログラミング ガイド)」を参照してください。

次の例では CS0261 エラーが生成されます。

```
// CS0261.cs
partial class A  // CS0261 – A declared as a class here, but as a struct below
{
}

partial struct A
{
}
```

コンパイラ エラー CS0262

エラー メッセージ
'型' の部分宣言には競合するアクセシビリティ修飾子が含まれています。

このエラーは、部分型に割り当てられた修飾子 (public、private、protected、internal、abstract) が矛盾している場合に発生します。同じ型に対する修飾子は、すべての部分宣言で一貫している必要があります。詳細については、「部分クラス定義 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0262 エラーが生成されます。

```
// CS0262.cs
class A
{
    public partial class C  // CS0262
    {
    }
    private partial class C
    {
    }
}
```

コンパイラ エラー CS0263

エラー メッセージ
'型' の部分宣言では、異なる基本クラスを指定してはいけません。

部分宣言で型を定義するとき、部分宣言では、すべて同じ基本型を指定する必要があります。詳細については、「部分クラス定義 (C# プログラミング ガイド)」を参照してください。

次の例では CS0263 エラーが生成されます。

```
// CS0263.cs
// compile with: /target:library
class B1
{
}

class B2
{
}
partial class C : B1  // CS0263 – is the base class B1 or B2?
{
}

partial class C : B2
{
}
```

コンパイラ エラー CS0264

エラー メッセージ
'型' の部分宣言では、同じ型パラメータ名を同じ順序で指定しなければなりません。

このエラーは、ジェネリック型を部分宣言で定義するとき、型パラメータの名前または指定順序が、すべての部分宣言で一貫していない場合に発生します。このエラーを回避するには、各部分宣言について型パラメータをチェックし、パラメータの名前と指定順序が統一されていることを確認します。詳細については、「部分クラス定義 (C# プログラミング ガイド)」および「ジェネリック型の型パラメータ (C# プログラミング ガイド)」を参照してください。

使用例
次のコードでは CS0264 エラーが生成されます。

```
// CS0264.cs

partial class MyClass<T>  // CS0264
{
}

partial class MyClass <MyType>
{
}
```

コンパイラ エラー CS0265

エラー メッセージ
'型' の部分宣言には、型パラメータ '型パラメータ' に対して矛盾する制約が含まれています。'

このエラーは、ジェネリック クラスを部分クラスとして定義したとき、つまり部分定義が複数の場所で実行され、ジェネリック型に対する制約が異なるか、矛盾している場合に発生します。制約を複数の場所で指定する場合は、すべて統一する必要があります。最も簡単な解決策は、制約を 1 か所で指定し、それ以外の場所では制約を省略することです。詳細については、「部分クラス定義 (C# プログラミング ガイド)」および「型パラメータの制約 (C# プログラミング ガイド)」を参照してください。

次のコードでは、CS0265 エラーが生成されます。

使用例

次のコードでは、部分クラスのすべての定義が 1 つのファイル内に記述されていますが、複数のファイルに分けて記述することもできます。

```
// CS0265.cs
public class GenericsErrors
{
    interface IFace1 { }
    interface IFace2 { }
    partial class PartialBadBounds<T> where T : IFace1 { } // CS0265
    partial class PartialBadBounds<T> where T : IFace2 { }
}
```

コンパイラ エラー CS0266

エラー メッセージ
型 'type1' を 'type2' に暗黙的に変換できません。明示的な変換が存在します。(cast が不足していないかどうかを確認してください)

このエラーは、暗黙的に変換することのできない 2 つの型に対して変換処理を適用しようとした場合に発生します。たとえば、基本型を派生型に代入するとき、明示的なキャストをしていない場合などが該当します。詳細については、「変換演算子 (C# プログラミング ガイド)」を参照してください。

次の例では CS0266 エラーが生成されます。

```
// CS0266.cs
class MyClass
{
    public static void Main()
    {
        object obj = "MyString";
        // Cannot implicitly convert 'object' to 'MyClass'
        MyClass myClass = obj;  // CS0266
        // Try this line instead
        // MyClass c = ( MyClass )obj;
    }
}
```

コンパイラ エラー CS0267

エラー メッセージ
部分識別子は、'class'、'struct' または 'interface' の直前にのみ指定できます。

クラス、構造体、またはインターフェイスの宣言で、**partial** 修飾子が誤った位置に記述されています。このエラーを解決するには、修飾子の順序を正しく指定します。詳細については、「部分クラス定義 (C# プログラミング ガイド)」を参照してください。

次の例では CS0267 エラーが生成されます。

```
// CS0267.cs
public partial class MyClass
{
   public MyClass()
   {
   }
}

partial public class MyClass  // CS0267
// Try this line instead:
// public partial class MyClass
{
   public void Foo()
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0268

エラー メッセージ
インポートされた型 '型' は無効です。この型には循環する基本クラスの依存関係が含まれています。

このエラーは、別の言語からインポートされた型に、基本クラスの循環依存関係がある場合に発生します。このような型を C# プログラムで使用することはできません。このエラーを解決するには、他の言語からインポートしている型を、参照アセンブリまたはモジュールでチェックします。

コンパイラ エラー CS0269

エラー メッセージ
割り当てのない out パラメータ 'パラメータ' の使用です。

Out パラメータの使用前に値が代入されるかどうかをコンパイラが確認できませんでした。代入の時点で値が定義されない可能性があります。パラメータの値にアクセスする前に、**out** パラメータが確実に初期化されるようにしてください。パラメータで渡された変数の値を使用する場合は、代わりに **ref** パラメータを使用します。詳細については、「パラメータの引き渡し (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0269 エラーが生成されます。

```
// CS0269.cs
class C
{
    public static void F(out int i)
    // Try this instead:
    // public static void F(ref int i)
    {
        int k = i;  // CS0269
        i = 1;
    }

    public static void Main()
    {
        int myInt;
        F(out myInt);
    }
}
```

このエラーは、変数が try ブロックで初期化されているために、正常に実行できるかどうかをコンパイラが確認できなかった場合にも発生します。

```
// CS0269b.cs
class C
{
    public static void F(out int i)
    {
        try
        {
            // Assignment occurs, but compiler can't verify it
            i = 1;
        }
        catch
        {
        }

        int k = i;  // CS0269
        i = 1;
    }

    public static void Main()
    {
        int myInt;
        F(out myInt);
    }
}
```

コンパイラ エラー CS0270

エラー メッセージ
配列のサイズは変数宣言の中で指定できません ('new' を使用して初期化してください)

このエラーは、配列宣言の一部としてサイズが指定されている場合に発生します。このエラーを解決するには、new 演算子式を使用します。

次の例では、CS0270 エラーが生成されます。

```
// CS0270.cs
// compile with: /t:module

public class Test
{
   int[10] a;   // CS0270
   // To resolve, use the following line instaead:
   // int[] a = new int[10];
}
```

コンパイラ エラー CS0271

エラー メッセージ
get アクセサにアクセスできないため、プロパティまたはインデクサ 'プロパティ/インデクサ' はこのコンテキストでは使用できません。

このエラーは、アクセスできない **get** アクセサにアクセスしようとした場合に発生します。このエラーを解決するには、アクセサのアクセシビリティ レベルを上げるか、別の場所から呼び出すようにします。詳細については、「アクセサのアクセシビリティ」および「プロパティ (C# プログラミング ガイド)」を参照してください。

次の例では、CS0271 エラーが生成されます。

```
// CS0271.cs
public class MyClass
{
   public int Property
   {
      private get { return 0; }
      set { }
   }

   public int Property2
   {
      get { return 0; }
      set { }
   }
}

public class Test
{
   public static void Main(string[] args)
   {
      MyClass c = new MyClass();
      int a = c.Property;   // CS0271
      int b = c.Property2;   // OK
   }
}
```

コンパイラ エラー CS0272

エラー メッセージ
set アクセサにアクセスできないため、プロパティまたはインデクサ 'プロパティ/インデクサ' はこのコンテキストでは使用できません。

このエラーは、プログラム コードが **set** アクセサにアクセスできない場合に発生します。このエラーを解決するには、アクセサのアクセシビリティ レベルを上げるか、別の場所から呼び出すようにします。詳細については、「非対称アクセサのアクセシビリティ (C# プログラミング ガイド)」を参照してください。

使用例
次の例では、CS0272 エラーが生成されます。

```
// CS0272.cs
public class MyClass
{
    public int Property
    {
        get { return 0; }
        private set { }
    }
}

public class Test
{
    static void Main()
    {
        MyClass c = new MyClass();
        c.Property = 10;      // CS0272
        // To resolve, remove the previous line
        // or use an appropriate modifier on the set accessor.
    }
}
```

コンパイラ エラー CS0273

エラー メッセージ
'property_accessor' アクセサのアクセシビリティ修飾子には、プロパティまたはインデクサの 'property' よりも強いアクセス制限が設定されている必要があります。

set/get アクセサのアクセシビリティ修飾子には、プロパティまたはインデクサの 'property/indexer' よりも強いアクセス制限が設定されている必要があります。

このエラーは、プロパティまたはインデクサに対し、そのいずれかのアクセサよりも制限の厳しいアクセス修飾子が指定されている場合に発生します。このエラーを解決するには、プロパティまたは set アクセサに適切なアクセス修飾子を使用します。詳細については、「アクセサのアクセシビリティ」を参照してください。

使用例
この例には、内部 set メソッドを持つ内部プロパティが含まれています。次の例では CS0273 エラーが生成されます。

```
// CS0273.cs
// compile with: /target:library
public class MyClass
{
   internal int Property
   {
      get { return 0; }
      internal set {}   // CS0273
      // try the following line instead
      // private set {}
   }
}
```

コンパイラ エラー CS0274

エラー メッセージ
アクセシビリティ修飾子は、プロパティまたはインデクサ 'プロパティ/インデクサ' の両方のアクセサに指定できません。

このエラーは、プロパティまたはインデクサを宣言するときに、両方のアクセサにアクセス修飾子を指定した場合に発生します。このエラーを解決するには、2 つのアクセサのうち、一方にのみアクセス修飾子を付けます。詳細については、「アクセサのアクセシビリティ」を参照してください。

次の例では、CS0274 エラーが生成されます。

```
// CS0274.cs
public class MyClass
{
    public int Property   // CS0274
    {
        public get { return 0; }
        protected set { }
    }
}
```

コンパイラ エラー CS0275

エラー メッセージ
'アクセサ': アクセシビリティ修飾子をインターフェイスのアクセサで使用することはできません。

このエラーは、プロパティのアクセサまたはインターフェイスのインデクサでアクセス修飾子を使用した場合に発生します。このエラーを解決するには、アクセス修飾子を削除します。

使用例
次の例では、CS0275 エラーが生成されます。

```
// CS0275.cs
public interface MyInterface
{
    int Property
    {
        get;
        internal set;   // CS0275
    }
}
```

コンパイラ エラー CS0276

エラー メッセージ
'プロパティ/インデクサ' : アクセサのアクセシビリティ修飾子は、プロパティまたはインデクサが get アクセサおよび set アクセサの両方を含む場合にのみ、使用されます。

このエラーは、プロパティまたはインデクサを 1 つのアクセサだけで宣言し、そのアクセサに対してアクセス修飾子を使用した場合に発生します。このエラーを解決するには、アクセス修飾子を削除するか、別のアクセサを追加します。

使用例
次の例では、CS0276 エラーが生成されます。

```
// CS0276.cs
public class MyClass
{
    public int Property
    {
        protected set { }   // CS0276
    }
    public int Property2
    {
        internal get { }   // CS0276
    }
}
```

コンパイラ エラー CS0277

エラー メッセージ
'クラス' はインターフェイス メンバ 'アクセサ' を実装しません。'クラス アクセサ' は、パブリックではありません。

このエラーは、インターフェイスのプロパティを実装するとき、クラスに実装されたプロパティのアクセサがパブリックで宣言されていない場合に発生します。インターフェイス メンバを実装するメソッドには、パブリックなアクセシビリティが必要です。このエラーを解決するには、プロパティのアクセサに指定されているアクセス修飾子を削除します。

使用例
次の例では、CS0277 エラーが生成されます。

```
// CS0277.cs
public interface MyInterface
{
    int Property
    {
        get;
        set;
    }
}

public class MyClass : MyInterface   // CS0277
{
    public int Property
    {
        get { return 0; }
        // Try this instead:
        //set { }
        protected set { }
    }
}
```

コンパイラ エラー CS0281

エラー メッセージ
フレンド アクセスは 'AssemblyName1' に許可されましたが、出力アセンブリは 'AssemblyName2' という名前です。'AssemblyName1' への参照を追加するか、または出力アセンブリ名が一致するように変更してください。

フレンド アクセスは、異なるアセンブリ間でパブリック以外の型へのアクセスを可能にする、共通言語ランタイム (CLR) の新しい機能です。このエラーは、フレンド アクセスを許可する側のアセンブリで、許可を受ける側のアセンブリ名が間違って指定されている場合に発生します。詳細については、「フレンド アセンブリ (C# プログラミング ガイド)」を参照してください。

使用例
次のコード サンプル シーケンスでは CS0281 エラーが生成されます。

厳密な名前付きアセンブリの作成に使用するファイルは、次のように生成されます。

- sn -d CS0281.snk
- sn -k CS0281.snk
- sn -i CS0281.snk CS0281.snk
- sn -pc CS0281.snk key.publickey
- sn -tp key.publickey

```
// CS0281.cs
// compile with: /target:library /keyfile:CS0281.snk
public class A {}
```

```
// CS0281_b.cs
// compile with: /target:library /keyfile:CS0281.snk /reference:CS0281.dll
[assembly:System.Runtime.CompilerServices.InternalsVisibleTo("CS0281 , PublicKey=0024000004
800000940000000602000000240000525341310004000001000100 4b2d4d56af7c50be2fcbbf97cb880b9e73ad8
4467a587191fef63aadc118a96cecf9d508cd679c907b6e20f71684300bdc2c0a851019af0c96b29bf8f1339753
276041aefd67db46139e6348b3a12f29537b4dc6c2c19829df2c9ed6803f3c63c3b84cfa2728849386aea575c54
3a5f70fa85793d2946f15f7fe1ccb0c5e8fe0")]
class B : A {}
```

次の例では CS0281 エラーが生成されます。

この例では、最初の例の出力ファイルと同じ名前の出力ファイルが作成されることに注意してください。解決するには、コンポーネントのアセンブリ属性を変更したり、クラス C を追加したりしないでください。

```
// CS0281_c.cs
// compile with: /target:library /out:CS0281.dll /keyfile:CS0281.snk /reference:CS0281_b.dl
l
// CS0281 expected
[assembly:System.Reflection.AssemblyVersion("3")]
[assembly:System.Reflection.AssemblyCulture("en-us")]
class C : B {}
public class A {}
```

コンパイラ エラー CS0283

エラー メッセージ
型 '型' を const 宣言することはできません。

定数の宣言で指定する型は、**byte**、**char**、**short**、**int**、**long**、**float**、**double**、**decimal**、**bool**、**string**、列挙型、または null 値を割り当てられた参照型のいずれかである必要があります。定数式は、結果となる値が対象の型、または対象の型に暗黙に変換できる型である必要があります。

使用例
次の例では CS0283 エラーが生成されます。

```
// CS0283.cs
struct MyTest
{
}
class MyClass
{
    // To resolve the error but retain the "const-ness",
    // change const to readonly.
    const MyTest test = new MyTest();   // CS0283

    public static int Main() {
        return 1;
    }
}
```

コンパイラ エラー CS0304

エラー メッセージ
変数型 '型' のインスタンスは、new() 制約を含まないため、作成できません

このエラーは、new を使用して型変数のインスタンスを作成するとき、型変数に new() 制約が指定されていない場合に発生します。既定のコンストラクタの存在が new() 制約によって保証されない限り、不明な型のコンストラクタを、new を使用して直接呼び出すことはできません。new 制約を使用できない場合は、typeof 式を使って必要なコンストラクタにアクセスすることを検討してください。

次の例では CS0304 エラーが生成されます。

```
// CS0304.cs
// compile with: /target:library
class C<T>
{
   T t = new T();   // CS0304
}
```

クラスのメソッド内で、次のように new ステートメントを使うことはできません。

```
// CS0304_2.cs
// compile with: /target:library
class C<T>
{
   public void f()
   {
      T t = new T();  // CS0304
   }
}
```

コンパイラ エラー CS0305

エラー メッセージ
ジェネリック型 'ジェネリック型' を使用するには、'数' 型引数が必要です。

このエラーは、必要な数の型引数が見つからない場合に発生します。CS0305 エラーを解決するには、必要な数の型引数を使用します。

使用例
次の例では CS0305 エラーが生成されます。

```
// CS0305.cs
public class MyList<T> {}
public class MyClass<T> {}

class MyClass
{
   public static void Main()
   {
      MyList<MyClass, MyClass> list1 = new MyList<MyClass>();   // CS0305
      MyList<MyClass> list2 = new MyList<MyClass>();   // OK
   }
}
```

コンパイラ エラー CS0306

エラー メッセージ
型 '型' は 型引数として使用できません

型パラメータとして使用されている型が無効です。原因として、ポインタ型などの使用が考えられます。

次の例では、CS0306 エラーが生成されます。

```
// CS0306.cs
class C<T>
{
}

class M
{
    // CS0306 – int* not allowed as a type parameter
     C<int*> f;
}
```

コンパイラ エラー CS0307

エラー メッセージ
'制約' '識別子' は型引数と一緒には使用できません

型およびメソッド以外に対して、型引数が指定されています。汎用引数を受け取ることができるのは、型またはメソッドだけです。山かっこ付きで指定した型引数を削除してください。ジェネリックを使用する場合は、該当する構文をジェネリック型またはジェネリック メソッドとして宣言してください。

次の例では CS0307 エラーが生成されます。

```
// CS0307.cs
class C
{
   public int P { get { return 1; } }
   public static void Main()
   {
      C c = new C();
      int p = c.P<int>();  // CS0307 – C.P is a property
      // Try this instead
      // int p = c.P;
   }
}
```

コンパイラ エラー CS0308

エラー メッセージ
非ジェネリック型または非ジェネリック メソッド '識別子' は型引数と一緒には使用できません。

ジェネリックではないメソッドまたは型で、型引数が使用されています。このエラーを回避するには、そのメソッドまたは型から、山かっこおよび型引数を削除するか、ジェネリック メソッドまたはジェネリック型として宣言し直します。

次の例では、CS0308 エラーが生成されます。

```
// CS0308a.cs
class MyClass
{
   public void F() {}
   public static void Main()
   {
      F<int>();  // CS0308 – F is not generic.
      // Try this instead:
      // F();
   }
}
```

次の例でも、CS0308 エラーが生成されます。このエラーを解決するには、"using System.Collections.Generic" ディレクティブを使用します。

```
// CS0308b.cs
// compile with: /t:library
using System.Collections;
// To resolve, uncomment the following line:
// using System.Collections.Generic;
public class MyStack<T>
{
    // Store the elements of the stack:
    private T[] items = new T[100];
    private int stack_counter = 0;

    // Define the iterator block:
    public IEnumerator<T> GetEnumerator()   // CS0308
    {
        for (int i = stack_counter - 1 ; i >= 0; i--)
        yield return items[i];
    }
}
```

コンパイラ エラー CS0309

エラー メッセージ
型 '型名' は、ジェネリック型またはメソッド 'ジェネリック' 内でパラメータ 'パラメータ' として使用するために、'制約型' に変換可能でなければなりません。

ジェネリック クラスまたはジェネリック メソッドを使用する場合は、where キーワードを使って指定されたジェネリック型制約を遵守する必要があります。制約に違反すると、このエラーが発生します。このエラーを解決するには、ジェネリック クラスまたはジェネリック メソッドに別の型を渡すか、制約を変更する必要があります。

次の例では、C<T> に対する制約で T が I を実装していることを条件としているにもかかわらず、B に I が実装されていないため、CS0309 エラーが生成されます。

```
// CS0309.cs
using System;

interface I
{
}

class C<T> where T : I
{
}

class B
{
}

class CMain
{
    public static void Main()
    {
        Console.WriteLine(new C<B>()); // CS0309
    }
}
```

コンパイラ エラー CS0310

エラー メッセージ
型 '型名' には、ジェネリック型またはメソッド 'ジェネリック' 内でパラメータ 'パラメータ' として使用するために、パブリックのパラメータを持たないコンストラクタを指定しなければなりません。

ジェネリック型またはジェネリック メソッドの where 句に new 制約が定義されているため、その型引数には public として宣言された、パラメータなしのコンストラクタを使用する必要があります。このエラーを回避するには、型に適切なコンストラクタを使用するか、ジェネリック型またはジェネリック メソッドの制約句を変更します。

使用例
次の例では CS0310 エラーが生成されます。

```
// CS0310.cs
using System;

class G<T> where T : new()
{
    T t;

    public G()
    {
        t = new T();
        Console.WriteLine(t);
    }
}

class B
{
    private B() { }
    // Try this instead:
    // public B() { }
}

class CMain
{
    public static void Main()
    {
        G<B> g = new G<B>();   // CS0310
        Console.WriteLine(g.ToString());
    }
}
```

コンパイラ エラー CS0400

エラー メッセージ
型または名前空間名 '識別子' はグローバル名前空間に見つかりませんでした。アセンブリ参照が不足しています。

グローバル スコープ演算子 (**::**) で定義された識別子が、グローバル名前空間に見つかりません。その識別子を格納するアセンブリ参照を指定していないか、そのグローバル名前空間以外のクラスまたは名前空間で宣言された識別子を指定した可能性があります。このエラーは、グローバルなスコープを持つ識別子が宣言されていなかったり、スペルに誤りがある場合にも生成されます。

このエラーを回避するには、識別子の宣言を探して正しいスペルを確認し、その宣言が別のアセンブリに存在する場合は、アセンブリ参照が適切であることを確認します。識別子が他の型または名前空間で宣言されていた場合は、:: の後に完全修飾名を指定してください。次の例では CS0400 エラーが生成されます。

```
// CS0400.cs
class C
{
    public static void Main()
    {
        // CS0400 - D could not be found
        // in the global namespace.
        global::D d = new global::D();
   }
}
```

コンパイラ エラー CS0401

エラー メッセージ
new() 制約は最後に指定する制約でなければなりません

複数の制約を使用する場合、new() 制約は他の制約の最後に指定する必要があります。

使用例
次の例では CS0401 エラーが生成されます。

```
// CS0401.cs
// compile with: /target:library
using System;
 class C<T> where T : new(), IDisposable {}  // CS0401

class D<T> where T : IDisposable
{
   static void F<U>() where U : new(), IDisposable{}   // CS0401
}
```

コンパイラ エラー CS0403

エラー メッセージ
値の型である可能性があるため、Null を型パラメータ '型パラメータ' に変換できません。代わりに、既定値 ('T') を使用してください。

未知の型に対して、null を代入することはできません。未知の型が null を代入できない値型に解決されることも考えられるためです。ジェネリック クラスで、値型が渡されることを想定していない場合は、クラス型制約を使用してください。組み込み型などの値型を受け取ることのできるジェネリック クラスについては、例に示すように、null の代入部分を `default(T)` という式で置き換えることができます。

使用例
次の例では CS0403 エラーが生成されます。

```
// CS0403.cs
// compile with: /target:library
class C<T>
{
   public void f()
   {
      T t = null;  // CS0403
      T t2 = default(T);   // OK
    }
}

class D<T> where T : class
{
   public void f()
   {
      T t = null;  // OK
    }
}
```

コンパイラ エラー CS0404

エラー メッセージ
'<' は無効です : 属性はジェネリックにすることができません。

属性でジェネリック型のパラメータを使用することはできません。型パラメータおよび山かっこを削除してください。

次の例では CS0404 エラーが生成されます。

```
// CS0404.cs
[MyAttrib<int>]  // CS0404
class C
{
   public static void Main()
   {

   }
}
```

コンパイラ エラー CS0405

エラー メッセージ
型パラメータ '型パラメータ' の重複する制約 '制約' です。

ジェネリック宣言に指定されている 2 つの制約が重複しています。このエラーを解決するには、重複している制約を削除します。

次の例では CS0405 エラーが生成されます。

```
// CS0405.cs
interface I
{
}

class C<T> where T : I, I  // CS0405.cs
{
}
```

コンパイラ エラー CS0406

エラー メッセージ
クラス型制約 '制約' は、他の制約の前に指定されなければなりません。

ジェネリック型またはジェネリック メソッドの型制約としてクラスが指定されている場合、クラスを制約の最初に指定する必要があります。このエラーを回避するには、クラスの型制約を、制約の先頭に移動します。

使用例
次の例では CS0406 エラーが生成されます。

```
// CS0406.cs
// compile with: /target:library
interface I {}
class C {}
class D<T> where T : I, C {}   // CS0406
class D2<T> where T : C, I {}   // OK
```

コンパイラ エラー CS0407

エラー メッセージ
'戻り値型メソッド' には、不適切な戻り値の型が指定されています。

メソッドとデリゲート型との間に互換性がありません。引数の型は一致していますが、デリゲートの戻り値の型が正しくありません。このエラーを回避するには、別のメソッドを使用するか、メソッドまたはデリゲートの戻り値の型を変更します。

使用例
次の例では CS0407 エラーが生成されます。

```
// CS0407.cs
public delegate int MyDelegate();

class C
{
    MyDelegate d;

    public C()
    {
        d = new MyDelegate(F);  // OK: F returns int
        d = new MyDelegate(G);  // CS0407 – G doesn't return int
    }

    public int F()
    {
        return 1;
    }

    public void G()
    {
    }

    public static void Main()
    {
        C c1 = new C();
    }
}
```

コンパイラ エラー CS0409

エラー メッセージ
制約句が、型パラメータ '型パラメータ' に既に指定されています。型パラメータの制約のすべてが、単一の WHERE 句で指定されなければなりません。

1 つの型パラメータに対して制約句 (where 句など) が重複しています。不要な where 句を削除するか、1 つの句の中で型パラメータが重複しないように where 句を修正してください。

```
// CS0409.cs
interface I
{
}

class C<T1, T2> where T1 : I where T1 : I  // CS0409 – T1 used twice
{
}
```

コンパイラ エラー CS0410

エラー メッセージ
適切なパラメータおよび戻り値の型が指定された 'メソッド' のオーバーロードはありません。

このエラーは、パラメータの型が異なる関数でデリゲートをインスタンス化しようとした場合に発生します。デリゲートのパラメータには、そのデリゲートに割り当てる関数と同じ型が使用されている必要があります。

使用例
次の例では、CS0410 エラーが生成されます。

```
// CS0410.cs
// compile with: /langversion:ISO-1

class Test
{
    delegate void D(double d );
    static void F(int i) { }

    static void Main()
    {
        D d = new D(F);  // CS0410
    }
}
```

コンパイラ エラー CS0411

エラー メッセージ
メソッド 'メソッド' に対する型引数を使い方から推論することはできません。型引数を明示的に指定してください。

このエラーは、型引数を明示的に指定せずにジェネリック メソッドを呼び出したため、選択すべき型引数をコンパイラが判断できなかった場合に発生します。このエラーを回避するには、型引数を山かっこで囲んで明示的に指定します。

使用例
次の例では CS0411 エラーが生成されます。

```
// CS0411.cs
class C
{
    void G<T>()
    {
    }

    public static void Main()
    {
        G();  // CS0411
        // Try this instead:
        // G<int>();
    }
}
```

次のように、型情報を持たない `null` のパラメータを使用した場合も同じエラーが生成されます。

```
// CS0411b.cs
class C
{
    public void F<T>(T t) where T : C
    {
    }

    public static void Main()
    {
        C c = new C();
        c.F(null);  // CS0411
    }
}
```

また、次のように、複数のパラメータのいずれかに対する変換に原因がある場合もあります。

```
// CS0411c.cs

class C
{
    void F<T>(T t1, T t2)
    {
    }

    public static void Main()
    {
        C c = new C();
        c.F(1, 2L);  // CS0411 -- is T int or long?
    }
}
```

コンパイラ エラー CS0412

エラー メッセージ
'ジェネリック': パラメータまたはローカル変数に、メソッド型パラメータと同じ名前を指定することはできません。

メソッドのローカル変数 (またはメソッドのいずれかのパラメータ) と、ジェネリック メソッドの型パラメータとで、名前の競合が生じています。このエラーを回避するには、競合しているパラメータまたはローカル変数の名前を変更します。

使用例
次の例では CS0412 エラーが生成されます。

```
// CS0412.cs
using System;

class C
{
    // Parameter name is the same as method type parameter name
    public void G<T>(int T)  // CS0412
    {
    }
    public void F<T>()
    {
        // Method local variable name is the same as method type
        // parameter name
        double T = 0.0;  // CS0412
        Console.WriteLine(T);
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0413

エラー メッセージ
型パラメータ '型パラメータ' にはクラス型制約、または 'class' 制約が含まれないため、'as' 演算子で使用できません。

このエラーは、ジェネリック型に as 演算子が使用されているとき、その型制約にクラスが指定されていない場合に発生します。**as** 演算子は参照型以外では使用できません。そのため、値型が渡されないように型パラメータで制約を加える必要があります。このエラーを回避するには、クラス型制約または参照型制約を使用します。

これは、**as** 演算子が、値型では使用できない **null** を返す場合があるためです。また、型パラメータは、クラス型制約または参照型制約が指定されていない場合、常に値型として処理される必要があります。

使用例
次の例では CS0413 エラーが生成されます。

```
// CS0413.cs
// compile with: /target:library
class A {}
class B : A {}

class CMain
{
   A a = null;
   public void G<T>()
   {
      a = new A();
      System.Console.WriteLine (a as T);  // CS0413
   }

   // OK
   public void H<T>() where T : A
   {
      a = new A();
      System.Console.WriteLine (a as T);
   }
}
```

コンパイラ エラー CS0415

エラー メッセージ
'インデクサ名' 属性は、明示的なインターフェイス メンバ宣言ではないインデクサ上でのみ有効です。

このエラーは、インターフェイスの明示的な実装メンバであるインデクサに対して、IndexerName 属性を使用した場合に発生します。このエラーを回避するには、インデクサの宣言から不要なインターフェイス名を削除します。詳細については、「IndexerNameAttribute クラス」を参照してください。

次の例では CS0415 エラーが生成されます。

```
// CS0415.cs
using System;
using System.Runtime.CompilerServices;

public interface IA
{
    int this[int index]
    {
        get;
        set;
    }
}

public class A : IA
{
    [IndexerName("Item")]  // CS0415
    int IA.this[int index]
    // Try this line instead:
    // public int this[int index]
    {
        get { return 0; }
        set { }
    }

    static void Main()
    {
    }
}
```

コンパイラ エラー CS0416

エラー メッセージ
'型パラメータ': 属性引数は型パラメータを使用することはできません。

型パラメータが属性の引数として使用されていますが、このような使い方はできません。ジェネリック型以外の型を使用してください。

次の例では CS0416 エラーが生成されます。

```
// CS0416.cs
public class MyAttribute : System.Attribute
{
   public MyAttribute(System.Type t)
   {
   }
}

class G<T>
{

   [MyAttribute(typeof(G<T>))]  // CS0416
   public void F()
   {
   }

}
```

コンパイラ エラー CS0417

エラー メッセージ
'識別子': 変数型のインターフェイスを作成するときに、引数を指定できません。

このエラーは、型パラメータに対する new 演算子の呼び出しで、引数を指定した場合に発生します。型パラメータに new 演算子を使用して呼び出すことができるコンストラクタは、引数を受け取らないコンストラクタだけです。他のコンストラクタを呼び出す場合は、クラス型制約またはインターフェイス制約を使用してください。

使用例
次の例では、CS0417 エラーが生成されます。

```
// CS0417
class C<T> where T : new()
{
    T type = new T(1);   // CS0417
}
```

コンパイラ エラー CS0418

エラー メッセージ
'クラス名': 抽象クラスを static または sealed に指定することはできません。

抽象クラスは、継承することによって初めてオブジェクトを作成できるため、抽象クラスをシールすることは無意味です。また、同様の理由により、抽象クラスを static として宣言することも意味をなしません。オブジェクト階層上、抽象クラスは、基本クラスとして使用することが前提となっています。

使用例
次の例では CS0418 エラーが生成されます。

```
// CS0418.cs
public abstract sealed class C  // CS0418
{
}

sealed static class S  // CS0418
{
}

public class MyClass
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0423

エラー メッセージ
'クラス' は ComImport 属性を含むため、'メソッド' は extern または abstract にしなければなりません。

ComImport 属性は、そのクラスの実装が COM モジュールからインポートされることを意味します。他のメソッドは定義できません。

次の例では CS0423 エラーが生成されます。

```
// CS0423.cs

using System.Runtime.InteropServices;

[
  ComImport,
  Guid("7ab770c7-0e23-4d7a-8aa2-19bfad479829")
]
class ImageProperties
{
  public static void Main()  // CS0423
  {
    ImageProperties i = new ImageProperties();
  }
}
```

コンパイラ エラー CS0424

エラー メッセージ
'クラス': ComImport 属性を含むクラスは、基本クラスを指定できません。

ComImportAttribute 属性は、そのクラスの実装が COM モジュールからインポートされることを意味します。COM モジュールで定義された実装に、基本クラスから継承される他のメソッドまたはフィールドを追加することはできません。

次の例では CS0424 エラーが生成されます。

```
// CS0424.cs
// compile with: /target:library
using System.Runtime.InteropServices;
public class A {}

[ ComImport, Guid("7ab770c7-0e23-4d7a-8aa2-19bfad479829") ]
class B : A {}   // CS0424 error
```

コンパイラ エラー CS0425

エラー メッセージ
メソッド 'メソッド' の型パラメータ '型パラメータ' に対する制約は、インターフェイス メソッド 'メソッド' の型パラメータ '型パラメータ' に対する制約と一致しなければなりません。明示的なインターフェイスの実装を使用することをお勧めします。

このエラーは、仮想ジェネリック メソッドが派生クラスでオーバーライドされているとき、メソッドに対する制約が派生クラスと基本クラスとで一致しない場合に発生します。このエラーを回避するには、両方の宣言で `where` 句を統一するか、インターフェイスを明示的に実装します。

使用例
次の例では、CS0425 エラーが生成されます。

```
// CS0425.cs

class C1
{
}

class C2
{
}

interface IBase
{
    void F<ItemType>(ItemType item) where ItemType : C1;
}

class Derived : IBase
{
    public void F<ItemType>(ItemType item) where ItemType : C2  // CS0425
    {
    }
}

class CMain
{
    public static void Main()
    {
    }
}
```

制約句の意味が一致していればよく、制約名がまったく同じである必要はありません。たとえば、次のような使い方は問題ありません。

```
// CS0425b.cs

interface J<Z>
{
}

interface I<S>
{
    void F<T>(S s, T t) where T: J<S>, J<int>;
}

class C : I<int>
{
    public void F<X>(int s, X x) where X : J<int>
    {
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0426

エラー メッセージ
型名 '識別子' は型 '型' に存在しません。

指定された型には、該当する型名が存在しません。指定した名前のスペルが正しいこと、および、参照先の型に、必要なメンバが存在することを確認してください。

次の例では CS0426 エラーが生成されます。

```
// CS0426.cs

class C
{
}

class D
{
   public static void Main()
   {
      C.A a;  // CS0426
   }
}
```

コンパイラ エラー CS0428

エラー メッセージ
メソッド グループ '識別子' を非デリゲート型 '型' に変換することはできません。このメソッドを呼び出すことはできません。

このエラーは、メソッド グループをデリゲート以外の型に変換しようとしたか、かっこを付けずにメソッドを呼び出そうとした場合に発生します。

使用例
次の例では CS0428 エラーが生成されます。

```
// CS0428.cs

delegate object Del1();
delegate int Del2();

public class C
{
    public static C Method() { return null; }
    public int Foo() { return 1; }

    public static void Main()
    {
        C c = Method; // CS0428, C is not a delegate type.
        int i = (new C()).Foo; // CS0428, int is not a delegate type.

        Del1 d1 = Method; // OK, assign to the delegate type.
        Del2 d2 = (new C()).Foo; // OK, assign to the delegate type.
        // or you might mean to invoke method
        // C c = Method();
        // int i = (new C()).Foo();
    }
}
```

コンパイラ エラー CS0430

エラー メッセージ
extern エイリアス 'エイリアス' は、/reference オプションで指定されませんでした

このエラーは、extern エイリアスが存在するにもかかわらず、/reference コンパイラ オプションでエイリアスが指定されなかった場合に発生します。CS0430 エラーを解決するには、**/reference** を指定してコンパイルします。

使用例

```
// CS0430_a.cs
// compile with: /target:library
public class MyClass {}
```

/reference:MyType=cs0430_a.dll を指定してコンパイルし、上のサンプルで作成された DLL を参照すると、このエラーは解決されます。次の例では CS0430 エラーが生成されます。

```
// CS0430_b.cs
extern alias MyType;   // CS0430
public class Test
{
   public static void Main() {}
}
```

コンパイラ エラー CS0431

エラー メッセージ
'::' を含むエイリアス '識別子' は型を参照するため、使用できません。'.' を使用してください。

型を参照するエイリアスに "::" を使用しています。このエラーを解決するには、"." 演算子を使用します。

次の例では CS0431 エラーが生成されます。

```
// CS0431.cs
using A = Outer;

public class Outer
{
   public class Inner
   {
      public static void Meth() {}
   }
}

public class MyClass
{
   public static void Main()
   {
      A::Inner.Meth();   // CS0431
      A.Inner.Meth();   // OK
   }
}
```

コンパイラ エラー CS0432

エラー メッセージ
エイリアス '識別子' が見つかりません。

このエラーは、エイリアスではない識別子の右側に "::" を使用した場合に発生します。このエラーを解決するには、代わりに "." を使用します。

次の例では、CS0432 エラーが生成されます。

```
// CS0432.cs
namespace A {
    public class B {
        public static void Meth() { }
    }
}

public class Test
{
    public static void Main()
    {
        A::B.Meth();   // CS0432
       // To resolve, use the following line instead:
       // A.B.Meth();
    }
}
```

コンパイラ エラー CS0433

エラー メッセージ
型 'TypeName1' は 'TypeName2' および 'TypeName3' の両方に存在します。

アプリケーション内で参照されている 2 つのアセンブリに同じ名前空間と型が含まれているため、あいまいさが発生しています。

このエラーを解決するには、/reference (メタデータのインポート) (C# コンパイラ オプション) コンパイラ オプションのエイリアス機能を使用するか、アセンブリの 1 つを参照しないようにします。

使用例
次のコードは、1 つ目のあいまいな型で DLL を作成します。

```
// CS0433_1.cs
// compile with: /target:library
namespace TypeBindConflicts
{
   public class AggPubImpAggPubImp {}
}
```

次のコードは、2 つ目のあいまいな型で DLL を作成します。

```
// CS0433_2.cs
// compile with: /target:library
namespace TypeBindConflicts
{
   public class AggPubImpAggPubImp {}
}
```

次の例では CS0433 エラーが生成されます。

```
// CS0433_3.cs
// compile with: /reference:cs0433_1.dll /reference:cs0433_2.dll
using TypeBindConflicts;
public class Test
{
   public static void Main()
   {
      AggPubImpAggPubImp n6 = new AggPubImpAggPubImp();   // CS0433
   }
}
```

次の例では、**/reference** コンパイラ オプションのエイリアス機能を使用して、この CS0433 エラーを解決する方法を示します。

```
// CS0433_4.cs
// compile with: /reference:cs0433_1.dll /reference:TypeBindConflicts=cs0433_2.dll
using TypeBindConflicts;
public class Test
{
   public static void Main()
   {
      AggPubImpAggPubImp n6 = new AggPubImpAggPubImp();
   }
}
```

コンパイラ エラー CS0434

エラー メッセージ
'NamespaceName2' にある名前空間 'NamespaceName1' が、'NamespaceName3' にある型 'TypeName1' と競合しています

このエラーは、インポートされた型とインポートされた名前空間の完全修飾名が同じである場合に発生します。このような名前が参照された場合、コンパイラは両者を区別できなくなります。

次のコードでは、CS0434 エラーが生成されます。

使用例
次のコードでは、同じ完全修飾名を持つ型の 1 つ目を作成します。

```
// CS0434_1.cs
// compile with: /t:library
namespace TypeBindConflicts
{
    namespace NsImpAggPubImp
    {
        public class X { }
    }
}
```

次のコードでは、同じ完全修飾名を持つ型の 2 つ目を作成します。

```
// CS0434_2.cs
// compile with: /t:library
namespace TypeBindConflicts {
    // Conflicts with another import (import2.cs).
    public class NsImpAggPubImp { }
    // Try this instead:
    // public class UniqueClassName { }
}
```

次のコードでは、同じ完全修飾名で型を参照しています。

```
// CS0434.cs
// compile with: /r:cs0434_1.dll /r:cs0434_2.dll
using TypeBindConflicts;
public class Test
{
    public TypeBindConflicts.NsImpAggPubImp.X n2 = null; // CS0434
}
```

コンパイラ エラー CS0438

エラー メッセージ
'module_1' の型 '型' は、'module_2' の名前空間 '名前空間' と競合します。

このエラーは、ソース ファイルに、他のソース ファイルの名前空間と競合する型が存在する場合に発生します。通常、競合する名前の型または名前空間を持つモジュールを追加したことが原因として考えられます。このエラーを解決するには、競合を引き起こしている型または名前空間の名前を変更します。

次の例では、CS0438 エラーが生成されます。

最初に、次のファイルをコンパイルします。

```
// CS0438_1.cs
// compile with: /target:module
public class Util
{
   public class A { }
}
```

続けて、次のファイルをコンパイルします。

```
// CS0438_2.cs
// compile with: /target:module
namespace Util
{
   public class A { }
}
```

最後に、次のファイルをコンパイルします。

```
// CS0438_3.cs
// compile with: /addmodule:CS0438_1.netmodule /addmodule:CS0438_2.netmodule
using System;
public class Test
{
   public static void Main() {
      Console.WriteLine(typeof(Util.A));   // CS0438
   }
}
```

コンパイラ エラー CS0439

エラー メッセージ
extern エイリアス宣言は、他のすべての名前空間要素の前に指定しなければなりません。

このエラーは、**extern** 宣言が、**using** など、同じ名前空間の他の宣言の後ろに指定されている場合に発生します。**extern** 宣言は、他のどの名前空間よりも前に指定されている必要があります。

使用例
次の例では、CS0439 エラーが生成されます。

```
// CS0439.cs
using System;

extern alias MyType;   // CS0439
// To resolve the error, make the extern alias the first line in the file.

public class Test
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0441

エラー メッセージ
'クラス': クラスに static と sealed の両方を指定することはできません。

このエラーは、クラスが static と sealed の両方で宣言されている場合に発生します。静的クラスはもともとシールされ、静的なメンバ以外持てないため、sealed 修飾子は不要です。このエラーを解決するには、いずれかの修飾子を削除します。

次の例では、CS0441 エラーが生成されます。

```
// CS0441.cs
sealed static class MyClass { }   // CS0441
```

コンパイラ エラー CS0442

エラー メッセージ
'プロパティ' : 抽象プロパティにプライベート アクセサは指定できません。

このエラーは、"private" アクセス修飾子を使用して抽象アクセサを修飾した場合に発生します。このエラーを解決するには、アクセス修飾子を変更するか、プロパティを abstract 以外で宣言します。

使用例
次の例では CS0442 エラーが生成されます。

```
// CS0442.cs
public abstract class MyClass
{
    public abstract int AbstractProperty
    {
        get;
        private set;   // CS0442
        // Try this instead:
        // set;
    }
}
```

コンパイラ エラー CS0443

エラー メッセージ
構文エラーです。値が必要です。

このエラーは、インデックス値を指定せずに配列を参照した場合に発生します。

使用例
次のコードでは、CS0443 エラーが生成されます。

```
// CS0443.cs
using System;
class MyClass
{
    public static void Main()
    {
        int[,] x = new int[1,5];
        if (x[] == 5) {} // CS0443
        // if (x[0, 0] == 5) {}
    }
}
```

コンパイラ エラー CS0445

エラー メッセージ
アンボックス変換の結果を変更できません。

ボックス化解除変換の結果はテンポラリ変数に格納されます。テンポラリ変数に対して行った変更は一時的にしか維持されないため、コンパイラでは、このような変数に変更を加えることを許可していません。このエラーを修正するには、中間式を値型に格納します。値を代入する場合は、この値型に対して行うようにしてください。

次のコードでは、CS0455 エラーが生成されます。

```
// CS0445.CS
public struct Point
{
    public int x;
    public static void SetX(object obj, int x)
    {
        ((Point)obj).x = x; // CS0445
    }
}
class UnboxingTest{public static void Main(){}}
```

コンパイラ エラー CS0446

エラー メッセージ
Foreach は、'メソッドまたはデリゲート' 上で使用できません。'メソッドまたはデリゲート' を呼び出すことはできません。

このエラーは、かっこを付けずにメソッドを指定した場合、または、通常はコレクション クラスを記述する **foreach** ステートメントの中で、匿名メソッドをかっこなしで指定した場合に発生します。あまり一般的ではありませんが、コレクション クラスを返すメソッドの場合は、このような場所でもメソッド呼び出しを記述できます。

使用例
次のコードでは、CS0446 エラーが生成されます。

```
// CS0446.cs
using System;
class Tester
{
    static void Main()
    {
        int[] intArray = new int[5];
        foreach (int i in M) { } // CS0446
    }
    static void M() { }
}
```

コンパイラ エラー CS0447

エラー メッセージ
型引数には属性を使用できません。型パラメータでのみ使用できます

このエラーは、呼び出しステートメントに出現する型引数に属性を適用した場合に発生します。次のように、クラスまたはメソッドの宣言ステートメントで、型パラメータに属性を適用することはできます。

```
class C<[some attribute] T> {…}
```

このエラーは次のようなコード行で発生します。ここでは、`C` クラスには、先行するコード行で `MyStaticMethod` という静的メソッドが定義されているものと仮定します。

```
C<[some attribute] T>.MyStaticMethod();
```

使用例
次のコードでは、CS0447 エラーが生成されます。

```
// CS0447.cs
using System;
namespace Test41
{
    public interface I<A>
    {
        void Meth<B>();
    }
    public class B : I<int>
    {
        void I<[Test] int>.Meth<X>() { }  // CS0447
    }
}
```

コンパイラ エラー CS0448

エラー メッセージ
++ または -- 演算子の戻り値の型は、それを含む型であるか、またはそれを含む型から派生しなければなりません

++ 演算子または **--** 演算子をオーバーライドする場合、戻り値の型は、その演算子が属する型と同じ型か、そこから派生した型にする必要があります。

使用例

次の例では CS0448 エラーが生成されます。

```
// CS0448.cs
class C5
{
   public static int operator ++(C5 c) { return null; }   // CS0448
   public static C5 operator --(C5 c) { return null; }   // OK
   public static void Main() {}
}
```

次の例では CS0448 エラーが生成されます。

```
// CS0448_b.cs
public struct S
{
   public static S? operator ++(S s) { return new S(); }   // CS0448
   public static S? operator --(S s) { return new S(); }   // CS0448
}

public struct T
{
// OK
   public static T operator --(T t) { return new T(); }
   public static T operator ++(T t) { return new T(); }

   public static T? operator --(T? t) { return new T(); }
   public static T? operator ++(T? t) { return new T(); }

   public static void Main() {}
}
```

コンパイラ エラー CS0449

エラー メッセージ
'class' または 'struct' 制約は、他の制約の前に指定されなければなりません。

ジェネリック型またはジェネリック メソッドの型パラメータに対する制約は、特定の順序で指定されている必要があります。まず、**class** または **struct** を指定します (存在する場合)。続けて、インターフェイスの制約を指定し、最後にコンストラクタの制約を指定します。このエラーは、**class** または **struct** の制約が最初に指定されなかった場合に発生します。このエラーを解決するには、制約句の指定順序を変更します。

使用例
次の例では CS0449 エラーが生成されます。

```
// CS0449.cs
// compile with: /target:library
interface I {}
public class C4
{
   public void F1<T>() where T : class, struct, I {}   // CS0449
   public void F2<T>() where T : I, struct {}   // CS0449
   public void F3<T>() where T : I, class {}   // CS0449

   // OK
   public void F4<T>() where T : class {}
   public void F5<T>() where T : struct {}
   public void F6<T>() where T : I {}
}
```

コンパイラ エラー CS0450

エラー メッセージ
'型パラメータ名': 制約クラスと 'クラス' または '構造体' 制約の両方を指定することはできません

struct 型とクラス型を同時に指定することはできません。そのため、型パラメータに struct 型制約が指定されているときに、同時に特定のクラス型制約を指定すると、論理的な矛盾が生じます。型パラメータに特定のクラス型制約が指定されている場合は、当然、そのクラス型制約が使用されるため、重ねてクラス型制約を指定することはできません。

使用例

```
// CS0450.cs
// compile with: /t:library
public class GenericsErrors
{
    public class B { }
    public class G3<T> where T : struct, B { } // CS0450
// To resolve, use the following line instead:
// public class G3<T> where T : B { }
}
```

コンパイラ エラー CS0451

エラー メッセージ
'new()' 制約は 'struct' 制約と一緒には使用できません。

ジェネリック型に対して制約を指定するとき、**new()** 制約は、クラス型制約、インターフェイス型制約、参照型制約、および型パラメータ制約と組み合わせてのみ使用できます。値型の制約と組み合わせて使用することはできません。

使用例
次の例では CS0451 エラーが生成されます。

```
// CS0451.cs
using System;
public class C4
{
   public void F4<T>() where T : struct, new() {}   // CS0451
}

// OK
public class C5
{
   public void F5<T>() where T : struct {}
}

public class C6
{
   public void F6<T>() where T : new() {}
}
```

コンパイラ エラー CS0452

エラー メッセージ
型 '型名' は、ジェネリック型のパラメータ 'パラメータ名'、またはメソッド 'ジェネリック 識別子' として使用するために、参照型でなければなりません

このエラーは、**struct** や **int** などの値型を、参照型制約を持つジェネリック型またはジェネリック メソッドのパラメータとして渡した場合に発生します。

使用例
次のコードでは、CS0452 エラーが生成されます。

```
// CS0452.cs
using System;
public class BaseClass<S> where S : class { }
public class Derived1 : BaseClass<int> { } // CS0452
public class Derived2<S> : BaseClass<S> where S : struct { } // CS0452
```

コンパイラ エラー CS0453

エラー メッセージ
型 '型名' は、ジェネリック型のパラメータ 'パラメータ名'、またはメソッド '汎用識別子' として使用するために、Null 非許容値型でなければなりません

このエラーは、**value** 制約が指定されたジェネリック型またはジェネリック メソッドをインスタンス化するときに、値型以外の引数を使用した場合に発生します。また、null 許容値型の引数を使用した場合にも発生します。次の例で、最後の 2 行のコードを参照してください。

使用例
このエラーが発生するコード例を次に示します。

```
// CS0453.cs
using System;
public class HV<S> where S : struct { }
public class H1 : HV<string> { }                    // CS0453
public class H2 : HV<H1> { }                        // CS0453
public class H3<S> : HV<S> where S : class { }      // CS0453
public class H4 : HV<int?> { }                      // CS0453
public class H5 : HV<Nullable<Nullable<int>>> { }  // CS0453
```

コンパイラ エラー CS0454

エラー メッセージ
型パラメータ 1' と '型パラメータ 2' を含む、循環制約の依存関係です。

このエラーは、ある型パラメータが別のパラメータを参照し、さらに、そのパラメータが最初のパラメータを参照している場合に発生します。このエラーを解決するには、いずれかの型制約を削除して、循環する依存関係を解消します。2 つの型制約の間の循環依存関係は、間接的なものである可能性もあります。

使用例

次のコードでは、CS0454 エラーが生成されます。

```
// CS0554
using System;
public class GenericsErrors
{
    public class G4<T> where T : T { } // CS0454
}
```

2 つの型制約間の循環する依存関係の例を次に示します。

```
public class Gen<T,U> where T : U where U : T  // CS0454
{
}
```

コンパイラ エラー CS0455

エラー メッセージ
型パラメータ '型パラメータ名' は、競合する制約 '制約名 1' および '制約名 2' を継承します

一般に、このエラーが発生する状況としては、型パラメータが関連性のない 2 つのクラスから派生するような制約を設定したか、型パラメータがクラス型 (参照型) と **struct** 型 (値型) から派生するような制約を設定したことが考えられます。このエラーを解決するには、継承の階層構造から競合する要素を取り除きます。

使用例
次のコードでは、CS0455 エラーが生成されます。

```
// CS0455.cs
using System;

public class GenericsErrors {
    public class B { }
    public class B2 { }
    public class G6<T> where T : B { public class N<U> where U : B2, T { } } // CS0455
}
```

コンパイラ エラー CS0456

エラー メッセージ
型パラメータ '型パラメータ名 1' は '制約' 制約を含むので、型パラメータ '型パラメータ 2' の制約として '型パラメータ 1' を使用することはできません。

値型の制約は、別の型パラメータで制約として使用できないように、暗黙的にシールされます。値型はオーバーライドできないためです。このエラーを解決するには、値型の制約を 1 つ目の型パラメータを使って間接的に指定するのではなく、2 つ目の型パラメータに直接記述します。

使用例
次の例では CS0456 エラーが生成されます。

```
// CS0456.cs
// compile with: /target:library
public class GenericsErrors
{
   public class G5<T> where T : struct
   {
      public class N<U> where U : T {}   // CS0456
      public class N2<U> where U : struct {}   // OK
   }
}
```

コンパイラ エラー CS0457

エラー メッセージ
'型名 1' から '型名 2' へ変換するときの、あいまいなユーザー定義の変換 '変換メソッド名 1' および '変換メソッド名 2' です

適用可能な変換メソッドが 2 つ存在し、どちらを使えばよいかコンパイラが判断できません。

このエラーが発生する状況としては、次のようなケースが挙げられます。

- クラス A をクラス B に変換したいとします。ここで、両者の間に関連性はありません。
- A は A0 から派生しており、A0 から B への変換を行うメソッドが存在します。
- B は B1 というサブクラスを持ち、A から B1 への変換を行うメソッドが存在します。

コンパイラは、2 つの変換メソッドに対して同等のウェイトを置きます。なぜなら、A0 から B の変換を選択した場合は変換先の型が最適であり、A から B1 の変換を選択した場合は変換元の型が最適であると見なすためです。どちらを使用するべきかコンパイラが判断できず、このエラーが生成されます。このエラーを解決するには、A を B に変換する明示的なメソッドを新たに記述します。

A を B に変換する 2 つのメソッドが存在する場合にもこのエラーが発生します。これを解決するには、使用する変換を、明示的なキャストによって指定する必要があります。

使用例
次の例では CS0457 エラーが生成されます。

```
// CS0457.cs
using System;
public class A { }

public class G0 {  }
public class G1<R> : G0 {  }

public class H0 {
   public static implicit operator G0(H0 h) {
      return new G0();
   }
}
public class H1<R> : H0 {
   public static implicit operator G1<R>(H1<R> h) {
      return new G1<R>();
   }
}

public class Test
{
   public static void F0(G0 g) {  }
   public static void Main()
   {
      H1<A> h1a = new H1<A>();
      F0(h1a);   // CS0457
   }
}
```

コンパイラ エラー CS0459

エラー メッセージ
読み取り専用のローカル変数のアドレスを取得することはできません。

C# 言語では、**foreach**、**using**、および **fixed** の 3 とおりの方法で、読み取り専用のローカル変数を生成できます。これらの方法で生成された読み取り専用のローカル変数は、値を書き込んだり、アドレスを取得したりすることはできません。このエラーは、読み取り専用のローカル変数のアドレスが取得されようとしていることを、コンパイラが認識した場合に生成されます。

使用例

次のコードでは、**foreach** ループおよび **fixed** ステートメント ブロックで読み取り専用のローカル変数のアドレスを取得しようとすると、CS0459 エラーが生成されます。

```
// CS0459.cs
// compile with: /unsafe

class A
{
    public unsafe void M1()
    {
        int[] ints = new int[] { 1, 2, 3 };
        foreach (int i in ints)
        {
            int *j = &i;  // CS0459
        }

        fixed (int *i = &_i)
        {
            int **j = &i;  // CS0459
        }
    }

    private int _i = 0;
}
```

コンパイラ エラー CS0460

エラー メッセージ
オーバーライドおよび明示的なインターフェイスの実装メソッドの制約は、基本メソッドから継承されるので、直接指定できません

派生クラス内のジェネリック メソッドで、基本クラスのメソッドをオーバーライドするとき、オーバーライドしたメソッドに対して制約を指定することはできません。派生クラス内のオーバーライド メソッドは、その制約を基本クラスのメソッドから継承します。

使用例

次の例では CS0460 エラーが生成されます。

```
// CS0460.cs
// compile with: /target:library
class BaseClass
{
   BaseClass() { }
}

interface I
{
   void F1<T>() where T : BaseClass;
   void F2<T>() where T : struct;
   void F3<T>() where T : BaseClass;
}

class ExpImpl : I
{
   void I.F1<T>() where T : BaseClass {}   // CS0460
   void I.F2<T>() where T : class {}  // CS0460
}
```

コンパイラ エラー CS0462

エラー メッセージ
継承されたメンバ 'member1' および 'member2' には型 'type' に同じシグネチャがあるので、オーバーライドできません

このエラーは、ジェネリックが導入されたことに伴い発生します。通常、1 つのクラス内で、同じシグネチャを持つ 2 種類のメソッドを定義することはできません。しかし、ジェネリックを使用すると、特定の型でインスタンス化されたときに別のメソッドのように振る舞う、ジェネリック メソッドを指定できます。

使用例

`C<int>` をインスタンス化すると、`F` メソッドの 2 つのバージョンが同じシグネチャで作成されます。そのため、このメソッドをオーバーライドする `D` クラスでは、どちらのメソッドを優先すべきかを判断できません。

次の例では CS0462 エラーが生成されます。

```
// CS0462.cs
// compile with: /target:library
class C<T>
{
   public virtual void F(T t) {}
   public virtual void F(int t) {}
}

class D : C<int>
{
   public override void F(int t) {}   // CS0462
}
```

コンパイラ エラー CS0463

エラー メッセージ
10 進数の定数式の評価に失敗し、次のエラーが発生しました: 'エラー'

このエラーは、コンパイル時に decimal 型の定数式がオーバーフローした場合に発生します。

オーバーフローのエラーは実行時に発生するのが一般的です。この場合、コンパイラが結果を評価してオーバーフローの発生を認識できるような方法で定数式を定義しています。

使用例
次の例では、CS0463 エラーが生成されます。

```
// CS0463.cs
using System;
class MyClass
{
    public static void Main()
    {
        const decimal myDec = 7900000000000000000000000000.0m + 7900000000000000000000000000
000.0m; // CS0463
        Console.WriteLine(myDec.ToString());
    }
}
```

コンパイラ エラー CS0466

エラー メッセージ
'method2' に指定されていないため、'method1' に params パラメータを指定しないでください

params パラメータは、実装インターフェイスで使用していない場合、クラス メンバで使用できません。

使用例
次の例では CS0466 エラーが生成されます。

```
// CS0466.cs
interface I
{
   void F1(params int[] a);
   void F2(int[] a);
}

class C : I
{
   void I.F1(params int[] a) {}
   void I.F2(params int[] a) {}   // CS0466
   void I.F2(int[] a) {}   // OK

   public static void Main()
   {
      I i = (I) new C();

      i.F1(new int[] {1, 2} );
      i.F2(new int[] {1, 2} );
   }
}
```

コンパイラ エラー CS0468

エラー メッセージ
型 'type1' と型 'type2' があいまいです。

このエラーは、コンパイルされているアセンブリの型に、同じ完全修飾名を持つものが 2 つあるときに生成されます。どちらも追加されたモジュールに含まれる場合、または一方はモジュールに含まれ、もう一方はソースに含まれる場合に、このエラーが発生します。

コンパイラ エラー CS0470

エラー メッセージ
メソッド 'メソッド' は、インターフェイス アクセサ 'アクセサ' を型 'type' に対して実装できません。明示的なインターフェイスの実装を使用してください。

このエラーは、アクセサがインターフェイスを実装しようとしているときに生成されます。明示的なインターフェイスの実装を使用する必要があります。

使用例
次の例では CS0470 エラーが生成されます。

```
// CS0470.cs
// compile with: /target:library

interface I
{
   int P { get; }
}

class MyClass : I
{
   public int get_P() { return 0; }   // CS0470
   public int P2 { get { return 0;} }   // OK
}
```

コンパイラ エラー CS0471

エラー メッセージ
変数 '変数' はジェネリック メソッドではありません。式リストの場合は、< 式をかっこで囲んでください。

このエラーは、かっこのない式リストがコードに含まれるときに生成されます。

使用例
次の例では CS0471 エラーが生成されます。

```
// CS0471.cs
// compile with: /t:library
class Test
{
    public void F(bool x, bool y) {}
    public void F1()
    {
        int a = 1, b = 2, c = 3;
        F(a<b, c>(3));    // CS0471
        // To resolve, try the following instead:
        // F((a<b), c>(3));
    }
}
```

コンパイラ エラー CS0500

エラー メッセージ
'class member' は abstract に指定されているため本体を宣言できません。

抽象メソッドには実装を含めることができません。

次の例では CS0500 エラーが生成されます。

```
// CS0500.cs
namespace x
{
   abstract public class clx
   {
      abstract public void f(){}   // CS0500
      // try the following line instead
      // abstract public void f();
   }

   public class cly
   {
      public static int Main()
      {
         return 0;
      }
   }
}
```

コンパイラ エラー CS0501

エラー メッセージ
'member function' は abstract または extern に指定されていないため、本体を宣言する必要があります。

非抽象メソッドには実装が必要です。

次の例では CS0501 エラーが生成されます。

```
// CS0501.cs
// compile with: /target:library
public class clx
{
   public void f();   // CS0501 declared not defined
   public void g() {}   // OK
}
```

コンパイラ エラー CS0502

エラー メッセージ
'メンバ' に abstract と sealed の両方を指定することはできません。

クラスを abstract と sealed の両方にすることはできません。

次の例では CS0502 エラーが生成されます。

```
// CS0502.cs
public class B
{
   abstract public void F();
}

public class C : B
{
   abstract sealed override public void F()   // CS0502
   {
   }
}

public class CMain
{
    public static void Main()
    { }
}
```

コンパイラ エラー CS0503

エラー メッセージ
抽象メソッド 'method' を virtual に指定することはできません。

abstract は **virtual** を意味するため、メンバ メソッドを abstract と virtual の両方としてマークしても冗長になります。

次の例では CS0503 エラーが生成されます。

```
// CS0503.cs
namespace x
{
   abstract public class clx
   {
      abstract virtual public void f();   // CS0503
   }
}
```

コンパイラ エラー CS0504

エラー メッセージ
定数 'variable' を static に指定することはできません。

変数が const の場合は、static でもあります。**const** であり **static** である変数が必要な場合は、単にその変数を **const** として宣言してください。**static** 変数だけが必要な場合は、単に **static** としてマークしてください。

次の例では CS0504 エラーが生成されます。

```
// CS0504.cs
namespace x
{
   abstract public class clx
   {
      static const int i = 0;   // CS0504, cannot be both static and const
      abstract public void f();
   }
}
```

コンパイラ エラー CS0505

エラー メッセージ
'メンバー1' : 'メンバー2' は関数ではないためオーバーライドできません

クラス宣言で、基本クラスの非メソッドのオーバーライドを試みました。オーバーライドではメンバの型が一致する必要があります。基本クラスのメソッドと同じ名前のメソッドが必要な場合は、基本クラスのメソッド宣言で (override ではなく) new を使用してください。

次の例では CS0505 エラーが生成されます。

```
// CS0505.cs
// compile with: /target:library
public class clx
{
   public int i;
}

public class cly : clx
{
   public override int i() { return 0; }   // CS0505
}
```

コンパイラ エラー CS0506

エラー メッセージ
'function1' : 継承メンバ 'function2' は、virtual、abstract、または override に指定されていないので、オーバーライドできません。

virtual、**abstract**、または **override** として明示的にマークされていないメソッドがオーバーライドされました。

次の例では CS0506 エラーが生成されます。

```
// CS0506.cs
namespace MyNameSpace
{
   abstract public class ClassX
   {
      public int i = 0;

      public int f()
      {
         return 0;
      }
      // Try the following definition for f() instead:
      // abstract public int f();
   }

   public class ClassY : ClassX
   {
      public override int f()   // CS0506
      {
         return 0;
      }

      public static int Main()
      {
         return 0;
      }
   }
}
```

コンパイラ エラー CS0507

エラー メッセージ
'function1' : 'access' 継承メンバ 'function2' をオーバーライドするときに、アクセス修飾子を変更できません。

メソッドのオーバーライドでアクセス指定の変更を試みました。

使用例
次の例では CS0507 エラーが生成されます。

```
// CS0507.cs
abstract public class clx
{
   virtual protected void f() {}
}

public class cly : clx
{
   public override void f() {}   // CS0507
   public static void Main() {}
}
```

CS0507 エラーは、参照されるメタデータで定義済みの **protected internal** とマークされているメソッドをクラスがオーバーライドしようとした場合にも発生することがあります。この場合、オーバーライドするメソッドは **protected** とマークされている必要があります。

```
// CS0507_b.cs
// compile with: /target:library
abstract public class clx
{
   virtual protected internal void f() {}
}
```

次の例では CS0507 エラーが生成されます。

```
// CS0507_c.cs
// compile with: /reference:cs0507_b.dll
public class cly : clx
{
   protected internal override void f() {}   // CS0507
   // try the following line instead
   // protected override void f() {}   // OK

   public static void Main() {}
}
```

コンパイラ エラー CS0508

エラー メッセージ
'Type 1': オーバーライドされたメンバ 'メンバ名' に対応するために戻り値の型は 'Type 2' でなければなりません

メソッドのオーバーライドで戻り値の型の変更を試みました。このエラーを解決するには、戻り値の型の宣言を両方のメソッドで一致させます。

使用例
次の例では CS0508 エラーが生成されます。

```
// CS0508.cs
// compile with: /target:library
abstract public class Clx
{
   public int i = 0;
   // Return type is int.
   abstract public int F();
}

public class Cly : Clx
{
   public override double F()
   {
      return 0.0;   // CS0508
   }
}
```

コンパイラ エラー CS0509

エラー メッセージ
'class1': シール型 'class2' から派生することはできません。

sealed クラスは base クラスとしては機能できません。既定では、構造体はシールされています。

次の例では CS0509 エラーが生成されます。

```
// CS0509.cs
// compile with: /target:library
sealed public class clx {}
public class cly : clx {}   // CS0509
```

コンパイラ エラー CS0513

エラー メッセージ
'function' は抽象ですが、非抽象クラスの 'class' に含まれています。

メソッドを非抽象クラスの abstract メンバにすることはできません。

次の例では CS0513 エラーが生成されます。

```
// CS0513.cs
namespace x
{
   public class clx
   {
      abstract public void f();   // CS0513, abstract member of nonabstract class
   }
}
```

コンパイラ エラー CS0514

エラー メッセージ
'constructor' : 静的コンストラクタに、明示的な this または基本コンストラクタの呼び出しを含めることはできません。

静的コンストラクタはクラスのインスタンスが作成される前に自動的に呼び出されるため、静的コンストラクタで **this** を呼び出すことはできません。また、静的コンストラクタは継承することも、直接呼び出すこともできません。

詳細については、「this (C# リファレンス)」および「base (C# リファレンス)」を参照してください。

使用例
次の例では、CS0514 エラーが生成されます。

```
// CS0514.cs
class A
{
    static A() : base(0) // CS0514
    {
    }

    public A(object o)
    {
    }
}

class B
{
    static B() : this(null) // CS0514
    {
    }

    public B(object o)
    {
    }
}
```

コンパイラ エラー CS0515

エラー メッセージ
'function' : アクセス修飾子を静的コンストラクタで使用できません。

静的コンストラクタではアクセス修飾子を使用できません。

使用例
次の例では CS0515 エラーが生成されます。

```
// CS0515.cs
public class Clx
{
    public static void Main()
    {
    }
}

public class Clz
{
    public static Clz()   // CS0515, remove public keyword
    {
    }
}
```

コンパイラ エラー CS0516

エラー メッセージ
コンストラクタ 'constructor' はそれ自身を呼び出すことはできません。

プログラムでコンストラクタを再帰的に呼び出すことはできません。

次の例では CS0516 エラーが生成されます。

```
// CS0516.cs
namespace x
{
   public class clx
   {
      public clx() : this()   // CS0516, delete "this()"
      {
      }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0517

エラー メッセージ
'class' に基本クラスがないため、基本コンストラクタを呼び出せません。

CS0517 エラーが発生するのは、基本クラスを持たない唯一のクラスであるオブジェクト クラスのソース コードを .NET Framework 共通言語ランタイムでコンパイルするときだけです。

コンパイラ エラー CS0518

エラー メッセージ
定義済みの型 '型' は定義、またはインポートされていません

この問題の主な原因は、System 名前空間全体を定義する mscorlib.dll を、プロジェクトがインポートしていないことです。次のような原因が考えられます。

- コマンド ライン コンパイラの /nostdlib オプションが指定された。/nostdlib オプションは、mscorlib.dll のインポートを防ぎます。このオプションは、ユーザー固有の System 名前空間を定義または作成する場合に使用します。
- 不正な mscorlib.dll が参照された。
- Visual Studio .NET または .NET Framework 共通言語ランタイムのインストールが破損している。
- 以前のインストールに、最新のインストールと互換性のないコンポーネントが残っている。

この問題を解決するには、以下のいずれかの操作を実行します。

- コマンド ライン コンパイラから /nostdlib オプションを指定しない。
- プロジェクトが正しい mscorlib.dll を参照していることを確認する。
- .NET Framework 共通言語ランタイムを再インストールする (上の解決策で問題が解決しない場合)。

コンパイラ エラー CS0520

エラー メッセージ
定義済みの型 '名前' が不適切に宣言されています。

コンパイラで必要なファイルが見つかりません。.NET Framework 共通言語ランタイムを再インストールしてください。

コンパイラ エラー CS0522

エラー メッセージ
'constructor' : 構造体は基本クラスのコンストラクタを呼び出すことができません。

構造体から基本クラスのコンストラクタを呼び出すことはできません。基本クラスのコンストラクタの呼び出しを削除してください。

次の例では CS0522 エラーが生成されます。

```
// CS0522.cs
public class clx
{
   public clx(int i)
   {
   }

   public static void Main()
   {
   }
}

public struct cly
{
   public cly(int i):base(0)   // CS0522
   // try the following line instead
   // public cly(int i)
   {
   }
}
```

コンパイラ エラー CS0523

エラー メッセージ
型 'struct1' の構造体メンバ 'struct2 field' により、構造体レイアウトで循環参照が発生します

2 つの構造体の定義に再帰的な参照が含まれます。それぞれが他方の定義で自身を定義しないように、構造体の定義を変更してください。この制限は、値型である構造体にのみ適用されます。再帰的な参照を使用する場合は、型をクラスとして宣言してください。

次の例では CS0523 エラーが生成されます。

```
// CS0523.cs
// compile with: /target:library
struct RecursiveLayoutStruct1
{
   public RecursiveLayoutStruct2 field;
}

struct RecursiveLayoutStruct2
{
   public RecursiveLayoutStruct1 field;   // CS0523
}
```

コンパイラ エラー CS0524

エラー メッセージ
'type' : インターフェイスで型を宣言することはできません。

インターフェイスでユーザー定義型を宣言することはできません。インターフェイスには、メソッドおよびプロパティのみ含めることができます。

使用例
次の例では CS0524 エラーが生成されます。

```
// CS0524.cs
public interface Clx
{
    public class Cly   // CS0524, delete user-defined type
    {
    }
}
```

コンパイラ エラー CS0525

エラー メッセージ
インターフェイスにフィールドを含めることはできません。

interface にはメソッドとプロパティを含めることはできますが、フィールドを含めることはできません。

次の例では CS0525 エラーが生成されます。

```
// CS0525.cs
namespace x
{
   public interface clx
   {
      public int i;   // CS0525
   }
}
```

コンパイラ エラー CS0526

エラー メッセージ
インターフェイスにコンストラクタを含めることはできません。

コンストラクタは、interface に対して定義できません。クラスと同じ名前で戻り値の型がないメソッドは、コンストラクタと見なされます。

次の例では CS0526 エラーが生成されます。

```
// CS0526.cs
namespace x
{
   public interface clx
   {
      public clx()   // CS0526
      {
      }
   }

   public class cly
   {
      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0527

エラー メッセージ
'type' : インターフェイス リストの型がインターフェイスではありません。

構造体や interface は別のインターフェイスを継承できますが、ほかのどの型も継承できません。

次の例では CS0527 エラーが生成されます。

```
// CS0527.cs
// compile with: /target:library
public struct clx : int {}   // CS0527 int not an interface
```

コンパイラ エラー CS0528

エラー メッセージ
'interface' は既にインターフェイス リストに存在します。

インターフェイス継承リストに重複があります。継承リストで 1 つの interface を指定できるのは 1 回だけです。

次の例では CS0528 エラーが生成されます。

```
// CS0528.cs
namespace x
{
   public interface a
   {
   }

   public class b : a, a   // CS0528
   {
      public void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0529

エラー メッセージ
継承インターフェイス 'interface1' により、'interface2' のインターフェイス階層内で循環参照が発生します。

interface の継承リストに、インターフェイス自身への直接参照または間接参照が含まれています。インターフェイスは自身を継承できません。

次の例では CS0529 エラーが生成されます。

```
// CS0529.cs
namespace x
{
   public interface a
   {
   }

   public interface b : a, c
   {
   }

   public interface c : b   // CS0529, b inherits from c
   {
   }
}
```

コンパイラ エラー CS0531

エラー メッセージ
'member' : インターフェイス メンバを定義することはできません。

interface で宣言されるメソッドは、インターフェイス自身ではなく、インターフェイスを継承するクラスに実装する必要があります。

次の例では CS0531 エラーが生成されます。

```
// CS0531.cs
namespace x
{
   public interface clx
   {
      int xclx()   // CS0531, cannot define xclx
      // Try the following declaration instead:
      // int xclx();
      {
         return 0;
      }
   }

   public class cly
   {
      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0533

エラー メッセージ
'derived-class member' は継承抽象メンバ 'base-class member' を隠します。

基本クラスのメソッドが隠ぺいされています。宣言の構文が正しいかどうかを確認してください。

詳細については、「base」を参照してください。

次の例では CS0533 エラーが生成されます。

```
// CS0533.cs
namespace x
{
   abstract public class a
   {
      abstract public void f();
   }

   abstract public class b : a
   {
      new abstract public void f();   // CS0533
      // try the following lines instead
      // override public void f()
      // {
      // }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0534

エラー メッセージ
'function1' は継承抽象メンバ 'function2' を実装しません。

抽象クラスでない限り、クラスでは、基本クラスのすべての抽象メンバを実装する必要があります。

次の例では CS0534 エラーが生成されます。

```
// CS0534.cs
namespace x
{
   abstract public class clx
   {
      public abstract void f();
   }

   public class cly : clx   // CS0534, no override for clx::f
   {
      // uncomment the following sample override to resolve CS0534
      // override public void f()
      // {
      // }

      public static int Main()
      {
         return 0;
      }
   }
}
```

コンパイラ エラー CS0535

エラー メッセージ
'class' はインターフェイス メンバ 'member' を実装しません。

クラスが interface から派生していますが、そのクラスではインターフェイスのメンバが実装されませんでした。クラスでは、派生元のインターフェイスのすべてのメンバを実装するか、**abstract** として宣言されている必要があります。

使用例

次の例では CS0535 エラーが生成されます。

```
// CS0535.cs
public interface A
{
   void F();
}

public class B : A {}   // CS0535 A::F is not implemented

// OK
public class C : A {
   public void F() {}
   public static void Main() {}
}
```

次の例では CS0535 エラーが生成されます。

```
// CS0535_b.cs
using System;
class C : IDisposable {}   // CS0535

// OK
class D : IDisposable {
   void IDisposable.Dispose() {}
   public void Dispose() {}

   static void Main() {
      using (D d = new D()) {}
   }
}
```

コンパイラ エラー CS0536

エラー メッセージ
'class' はインターフェイス メンバ 'interface member' を実装しません。'class member' が public ではなく、static になっているか、または戻り値の型が正しくありません。

' *class* はインターフェイス メンバ ' *member1* ' を実装しません。' *member2* ' が public ではなく、static になっているか、または戻り値の型が正しくありません。

interface メンバの実装が検出されませんでした。インターフェイス メンバをほとんど実装する宣言が存在する可能性があります。インターフェイス メンバの宣言について、以下の構文エラーがないかを確認してください。

- **public** キーワードが省略されている。
- 戻り値の型が一致しない。
- **static** キーワードがある。

次の例では CS0536 エラーが生成されます。

```
// CS0536.cs
public interface a
{
   void f();
}

public class b : a
{
   public static int f()   // CS0536
   // try the following line instead
   // public void f()
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0537

エラー メッセージ
クラス System.Object は基本クラスを含んだり、インターフェイスを実装したりできません。

CS0537 エラーは、System クラス ライブラリをビルドし直したときに、Object が別のクラスから派生している場合に発生します。このエラーが発生するケースはきわめてまれです。万一このエラーが発生した場合は、**Object** をクラスやインターフェイスから派生させることは避けてください。System.Object は、.NET Framework クラス階層構造のルートであるため、他のクラスから継承することはできません。

コンパイラ エラー CS0538

エラー メッセージ
明示的インターフェイス宣言の中の 'name' はインターフェイスではありません。

interface の明示的な宣言を試みましたが、インターフェイスが指定されませんでした。

次の例では CS0538 エラーが生成されます。

```
// CS0538.cs
interface MyIFace
{
   void F();
}

public class MyClass
{
   public void G()
   {
   }
}

class C: MyIFace
{
   void MyIFace.F()
   {
   }

   void MyClass.G()   // CS0538, MyClass not an interface
   {
   }
}
```

コンパイラ エラー CS0539

エラー メッセージ
明示的インターフェイス宣言の中の 'member' はインターフェイスのメンバではありません。

存在しない interface メンバの明示的な宣言を試みました。宣言を削除するか、または有効なインターフェイス メンバを参照するように宣言を変更する必要があります。

次の例では CS0539 エラーが生成されます。

```
// CS0539.cs
namespace x
{
   interface I
   {
      void m();
   }

   public class clx : I
   {
      void I.x()   // CS0539
      {
      }

      public static int Main()
      {
         return 0;
      }
   }
}
```

コンパイラ エラー CS0540

エラー メッセージ
'インターフェイス メンバ': 含む型は、インターフェイス 'インターフェイス' を実装しません

interface から派生しないクラスに対して、インターフェイス メンバの実装を試みました。インターフェイス メンバの実装を削除するか、またはクラスの基本クラスのリストにインターフェイスを追加する必要があります。

使用例

次の例では CS0540 エラーが生成されます。

```
// CS0540.cs
interface I
{
   void m();
}

public class Clx
{
   void I.m() {}   // CS0540
}

// OK
public class Cly : I
{
   void I.m() {}
   public static void Main() {}
}
```

次の例では CS0540 エラーが生成されます。

```
// CS0540_b.cs
using System;
class C {
   void IDisposable.Dispose() {}   // CS0540
}

class D : IDisposable {
   void IDisposable.Dispose() {}
   public void Dispose() {}

   static void Main() {
      using (D d = new D()) {}
   }
}
```

コンパイラ エラー CS0541

エラー メッセージ
'declaration' : 明示的インターフェイスはクラス、または構造体の中でのみ宣言できます。

明示的な interface 宣言が、クラスまたは構造体の外側で見つかりました。

次の例では CS0541 エラーが生成されます。

```
// CS0541.cs
namespace x
{
   interface IFace
   {
      void F();
   }

   interface IFace2 : IFace
   {
      void IFace.F();   // CS0541
   }
}
```

コンパイラ エラー CS0542

エラー メッセージ
'user-defined type' : メンバ名をそれを囲む型の名前と同じにすることはできません。

1 つの名前が同じ構成要素で複数回使用されました。このエラーは、コンストラクタに間違って戻り値の型を挿入することにより発生する場合があります。

次の例では CS0542 エラーが生成されます。

```
// CS0542.cs
class F
{
   // Remove void from F() to resolve the problem.
   void F()   // CS0542, same name as the class
   {
   }
}

class MyClass
{
   public static void Main()
   {
   }
}
```

'Item' という名前のクラスに、`this` として宣言されたインデクサが存在する場合、このエラーが発生します。出力されたコードでは、インデクサに対し、既定で 'Item' という名前が割り当てられるため、クラス名との競合が生じます。

```
// CS0542b.cs
class Item
{
   public int this[int i]  // CS0542
   {
      get
      {
         return 0;
      }
   }
}

class CMain
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0543

エラー メッセージ
'enumeration' : 列挙子の値が大きすぎます。

列挙体で要素に代入された値は、データ型の範囲外です。

次の例では CS0543 エラーが生成されます。

```
// CS0543.cs
namespace x
{
   enum I : byte
   {a = 255, b, c}   // CS0543
   public class clx
   {
      public static int Main()
      {
         return 0;
      }
   }
}
```

コンパイラ エラー CS0544

エラー メッセージ
プロパティ オーバーライド': '非プロパティ' はプロパティではないため、オーバーライドできません

非プロパティのデータ型をプロパティとしてオーバーライドしようとしましたが、この処理は許可されていません。

次の例では CS0544 エラーが生成されます。

```
// CS0544.cs
// compile with: /target:library
public class a
{
   public int i;
}

public class b : a
{
   public override int i {   // CS0544
   // try the following line instead
   // public new int i {
      get
      {
         return 0;
      }
   }
}
```

コンパイラ エラー CS0545

エラー メッセージ
'function' : 'property' に、オーバーライド可能な get アクセサがないため、オーバーライドできません。

オーバーライドする定義が基本クラスにないときに、プロパティのアクセサに対するオーバーライドの定義を試みました。このエラーは以下の方法で解決できます。

- 基本クラスで **set** アクセサを追加する。
- 派生クラスから **set** アクセサを削除する。
- 派生クラスのプロパティで new キーワードを追加することにより、基本クラスのプロパティを隠ぺいする。
- 基本クラスのプロパティを virtual にする。

詳細については、「プロパティの使用 (C# プログラミング ガイド)」および「プロパティの使用 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0545 エラーが生成されます。

```
// CS0545.cs
// compile with: /target:library
// CS0545
public class a
{
   public virtual int i
   {
      set {}

      // Uncomment the following line to resolve.
      // get { return 0; }
   }
}

public class b : a
{
   public override int i
   {
      get { return 0; }
      set {}   // OK
   }
}
```

コンパイラ エラー CS0546

エラー メッセージ
'accessor' : 'property' に、オーバーライド可能な set アクセサがないため、オーバーライドできません。

アクセサをオーバーライドできないため、プロパティのアクセサ メソッドのオーバーライドに失敗しました。このエラーは以下の方法で解決できます。

- 基本クラスで **set** アクセサを追加する。
- 派生クラスから **set** アクセサを削除する。
- 派生クラスのプロパティで new キーワードを追加することにより、基本クラスのプロパティを隠ぺいする。
- 基本クラスのプロパティを virtual にする。

詳細については、「プロパティの宣言」および「プロパティの使用 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0546 エラーが生成されます。

```
// CS0546.cs
// compile with: /target:library
public class a
{
   public virtual int i
   {
      get
      {
         return 0;
      }
   }

   public virtual int i2
   {
      get
      {
         return 0;
      }

      set {}
   }
}

public class b : a
{
   public override int i
   {
      set {}   // CS0546 error no set
   }

   public override int i2
   {
      set {}   // OK
   }
}
```

コンパイラ エラー CS0547

エラー メッセージ
'property' : プロパティまたはインデクサに void 型を指定できません。

void は、プロパティの戻り値としては無効です。

詳細については、「プロパティ」を参照してください。

次の例では CS0547 エラーが生成されます。

```
// CS0547.cs
public class a
{
   public void i   // CS0547
   // Try the following declaration instead:
   // public int i
   {
      get
      {
         return 0;
      }
   }
}

public class b : a
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0548

エラー メッセージ
'property' : プロパティまたはインデクサは最低 1 つのアクセサを必要とします。

プロパティには少なくとも 1 つのアクセサ (**get** または **set**) メソッドが必要です。

詳細については、「プロパティの宣言」および「プロパティの使用 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0548 エラーが生成されます。

```
// CS0548.cs
// compile with: /target:library
public class b
{
   public int MyProp {}   // CS0548

   public int MyProp2   // OK
   {
      get
      {
         return 0;
      }
      set {}
   }
}
```

コンパイラ エラー CS0549

エラー メッセージ
'function' はシール クラス 'class' の新しい仮想メンバです。

シール クラスは、基本クラスとして使用できません。したがって、シール クラスに仮想メソッドを使用しても意味がありません。

次の例では CS0549 エラーが生成されます。

```
// CS0549.cs
// compile with: /target:library
sealed public class MyClass
{
   virtual public void TestMethod() {}   // CS0549
   public void TestMethod2() {}   // OK
}
```

コンパイラ エラー CS0550

エラー メッセージ
'accessor' はインターフェイス メンバ 'property' にないアクセサを追加します。

派生クラスのプロパティの実装に、基本インターフェイスで指定されていないアクセサが含まれています。

詳細については、「プロパティの使用 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0550 エラーが生成されます。

```
// CS0550.cs
namespace x
{
   interface ii
   {
      int i
      {
         get;
         // add the following accessor to resolve this CS0550
         // set;
      }
   }

   public class a : ii
   {
      int ii.i
      {
         get
         {
            return 0;
         }
         set {}   // CS0550  no set in interface
      }

      public static void Main() {}
   }
}
```

コンパイラ エラー CS0551

エラー メッセージ
明示的なインターフェイスの実装 'implementation' にアクセサ 'accessor' がありません。

インターフェイスのプロパティを明示的に実装するクラスは、インターフェイスが定義するすべてのアクセサを実装する必要があります。

詳細については、「プロパティの宣言」および「プロパティの使用 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0551 エラーが生成されます。

```
// CS0551.cs
// compile with: /target:library
interface ii
{
   int i
   {
      get;
      set;
   }
}

public class a : ii
{
   int ii.i { set {} }   // CS0551

   // OK
   int ii.i
   {
      set {}
      get { return 0; }
   }
}
```

コンパイラ エラー CS0552

エラー メッセージ
'conversion routine' : インターフェイスへ/からのユーザー定義の変換です。

インターフェイスとの間でユーザー定義の変換ができません。変換ルーチンが必要な場合は、インターフェイスをクラスにしてこのエラーを解決するか、またはインターフェイスからクラスを派生してください。

次の例では CS0552 エラーが生成されます。

```
// CS0552.cs
public interface ii
{
}

public class a
{
   // delete the routine to resolve CS0552
   public static implicit operator ii(a aa) // CS0552
   {
      return new ii();
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0553

エラー メッセージ
'conversion routine' : 基本クラスへ/からのユーザー定義の変換です。

基本クラスの値へのユーザー定義の変換はできません。この演算子は不要です。

次の例では CS0553 エラーが生成されます。

```
// CS0553.cs
namespace x
{
   public class ii
   {
   }

   public class a : ii
   {
      // delete the conversion routine to resolve CS0553
      public static implicit operator ii(a aa) // CS0553
      {
         return new ii();
      }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0554

エラー メッセージ
'conversion routine' : 派生クラスへ/からのユーザー定義の変換です。

派生クラスの値へのユーザー定義の変換はできません。この演算子は不要です。

ユーザー定義の変換の詳細については、『C# 言語の仕様』の第 6 章を参照してください。

次の例では CS0554 エラーが生成されます。

```
// CS0554.cs
namespace x
{
   public class ii
   {
      // delete the conversion routine to resolve CS0554
      public static implicit operator ii(a aa) // CS0554
      {
         return new ii();
      }
   }

   public class a : ii
   {
      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0555

エラー メッセージ
ユーザー定義の演算子は、それを囲む型のオブジェクトの取得、およびそれを囲む型のオブジェクトへの変換を行えません。

外側のクラスの値へのユーザー定義の変換はできません。この演算子は不要です。

次の例では CS0555 エラーが生成されます。

```
// CS0555.cs
public class MyClass
{
   // delete the following operator to resolve this CS0555
   public static implicit operator MyClass(MyClass aa)   // CS0555
   {
      return new MyClass();
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0556

エラー メッセージ
ユーザー定義の変換は、それを囲む型に/から変換しなければなりません。

ユーザー定義の変換ルーチンは、そのルーチンを含むクラスとの間で変換を実行する必要があります。

次の例では CS0556 エラーが生成されます。

```
// CS0556.cs
namespace x
{
   public class ii
   {
      public class iii
      {
         public static implicit operator int(byte aa)   // CS0556
         // try the following line instead
         // public static implicit operator int(iii aa)
         {
            return 0;
         }
      }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0557

エラー メッセージ
型 'クラス' で重複するユーザー定義の変換です。

クラスでは変換ルーチンを重複して使用できません。

次の例では、CS0557 エラーが生成されます。

```
// CS0557.cs
namespace x
{
    public class ii
    {
        public class iii
        {
        public static implicit operator int(iii aa)
        {
            return 0;
        }

    // CS0557, delete duplicate
        public static explicit operator int(iii aa)
        {
            return 0;
        }
        }

        public static void Main()
        {
        }
    }
}
```

コンパイラ エラー CS0558

エラー メッセージ
ユーザー定義の演算子 'operator' は static および public として宣言されなければなりません。

ユーザー定義の演算子では、**static** と **public** の両方のアクセス修飾子を指定する必要があります。

次の例では CS0558 エラーが生成されます。

```
// CS0558.cs
namespace x
{
   public class ii
   {
      public class iii
      {
         static implicit operator int(iii aa)   // CS0558, add public
         {
            return 0;
         }
      }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0559

エラー メッセージ
++ か -- 演算子の戻り値の型は、それを含む型でなければなりません。

演算子のオーバーロードのためのメソッド宣言は、一定のガイドラインに沿っている必要があります。++ 演算子および -- 演算子の場合、パラメータの型が、オーバーロードされる側とオーバーロードする側とで一致していることが必要です。

使用例

次の例では CS0559 エラーが生成されます。

```
// CS0559.cs
// compile with: /target:library
public class iii
{
   public static implicit operator int(iii x)
   {
      return 0;
   }

   public static implicit operator iii(int x)
   {
      return null;
   }

   public static int operator ++(int aa)   // CS0559
   // try the following line instead
   // public static iii operator ++(iii aa)
   {
      return (iii)0;
   }
}
```

次の例では CS0559 エラーが生成されます。

```
// CS0559_b.cs
// compile with: /target:library
public struct S
{
   public static S operator ++(S? s) { return new S(); }   // CS0559
   public static S operator --(S? s) { return new S(); }   // CS0559
}

public struct T
{
// OK
   public static T operator --(T t) { return new T(); }
   public static T operator ++(T t) { return new T(); }

   public static T? operator --(T? t) { return new T(); }
   public static T? operator ++(T? t) { return new T(); }
}
```

コンパイラ エラー CS0562

エラー メッセージ
単項演算子のパラメータは、それを含む型でなければなりません。

演算子のオーバーロードのためのメソッド宣言は、一定のガイドラインに沿っている必要があります。詳細については、「オーバーロード可能な演算子」および「演算子のオーバーロードのサンプル」を参照してください。

使用例
次の例では CS0562 エラーが生成されます。

```
// CS0562.cs
public class iii
{
    public static implicit operator int(iii x)
    {
        return 0;
    }

    public static implicit operator iii(int x)
    {
        return null;
    }

    public static iii operator +(int aa)   // CS0562
    // try the following line instead
    // public static iii operator +(iii aa)
    {
        return (iii)0;
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0563

エラー メッセージ
バイナリ演算子のパラメータの 1 つはそれを含む型でなければなりません。

演算子のオーバーロードのためのメソッド宣言は、一定のガイドラインに沿っている必要があります。詳細については、「オーバーロード可能な演算子」および「演算子のオーバーロードのサンプル」を参照してください。

使用例
次の例では CS0563 エラーが生成されます。

```
// CS0563.cs
public class iii
{
    public static implicit operator int(iii x)
    {
        return 0;
    }
    public static implicit operator iii(int x)
    {
        return null;
    }
    public static int operator +(int aa, int bb)   // CS0563
    // Use the following line instead:
    // public static int operator +(int aa, iii bb)
    {
        return 0;
    }
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0564

エラー メッセージ
オーバーロードされた shift 演算子では、最初のオペランドの型はそれを含む型で、2 番目のオペランドの型は int でなければなりません。

間違って入力したオペランドによりシフト演算子 (<< または >>) をオーバーロードしようとしました。最初のオペランドは型であることが必要で、2 番目のオペランドは **int** 型であることが必要です。

次の例では CS0564 エラーが生成されます。

```
// CS0564.cs
using System;
class C
{
   public static int operator << (C c1, C c2) // CS0564
// To correct, change second operand to int, like so:
// public static int operator << (C c1, int c2)
   {
      return 0;
   }
   static void Main()
   {
   }
}
```

コンパイラ エラー CS0567

エラー メッセージ
インターフェイスに演算子を含めることはできません。

演算子は、interface の定義では使用できません。

次の例では CS0567 エラーが生成されます。

```
// CS0567.cs
interface IA
{
   int operator +(int aa, int bb);   // CS0567
}

class Sample
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0568

エラー メッセージ
構造体に明示的なパラメータのないコンストラクタを含めることはできません。

各構造体には、オブジェクトをゼロに初期化する既定のコンストラクタが既にあります。したがって、構造体に対して作成するコンストラクタでは 1 つ以上のパラメータを受け取る必要があります。

次の例では CS0568 エラーが生成されます。

```
// CS0568.cs
public struct ClassY
{
   public int field1;
   public ClassY(){}   // CS0568, cannot have no param constructor
   // Try following instead:
   // public ClassY(int i)
   // {
   //    field1 = i;
   // }
}

public class ClassX
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0569

エラー メッセージ
'method2' : 'method1' はこの言語でサポートされていないため、オーバーライドできません。

このエラーが発生するのは、別の言語で記述された基本クラスから派生する場合と、オーバーライドするメソッドをコンパイラが認識しない場合です。

コンパイラ エラー CS0570

エラー メッセージ
'クラス' はこの言語によってサポートされていません。

このエラーが発生するのは、別のコンパイラによって生成され、インポートされたメタデータを使用するときです。コンパイラで処理できないクラス メンバの使用を試みました。

使用例

次の C++ プログラムでは RequiredAttributeAttribute 属性を使用しますが、この属性は他の言語では利用できません。

```
// CPP0570.cpp
// compile with: /clr /LD

using namespace System;
using namespace System::Runtime::CompilerServices;

namespace CS0570_Server {
   [RequiredAttributeAttribute(Int32::typeid)]
   public ref struct Scenario1 {
      int intVar;
   };

   public ref struct CS0570Class {
      Scenario1 ^ sc1_field;

      property virtual Scenario1 ^ sc1_prop {
         Scenario1 ^ get() { return sc1_field; }
      }

      Scenario1 ^ sc1_method() { return sc1_field; }
   };
};
```

次の例では CS0570 エラーが生成されます。

```
// CS0570.cs
// compile with: /reference:CPP0570.dll
using System;
using CS0570_Server;

public class C {
   public static int Main() {
      CS0570Class r = new CS0570Class();
      r.sc1_field = null;   // CS0570
      object o = r.sc1_prop;   // CS0570
      r.sc1_method();   // CS0570
   }
}
```

コンパイラ エラー CS0571

エラー メッセージ
'function' : 演算子またはアクセサを明示的に呼び出すことはできません。

一部の演算子に内部名があります。たとえば、**op_Increment** は ++ 演算子の内部名です。このようなメソッド名を使用したり、明示的に呼び出したりしないでください。

次の例では CS0571 エラーが生成されます。

```
// CS0571.cs
public class MyClass
{
   public static MyClass operator ++ (MyClass c)
   {
      return null;
   }

   public static int prop
   {
      get
      {
         return 1;
      }
      set
      {
      }
   }

   public static void Main()
   {
      op_Increment(null);   // CS0571
      // use the increment operator as follows
      // MyClass x = new MyClass();
      // x++;

      set_prop(1);      // CS0571
      // try the following line instead
      // prop = 1;
   }
}
```

コンパイラ エラー CS0572

エラー メッセージ
'type' : 式から型を参照することはできません。'path_to_type' を使用してください。

識別子を通じてクラスのメンバにアクセスしようとしましたが、この場合は許可されていません。

次の例では CS0572 エラーが生成されます。

```
// CS0572.cs
using System;
class C
{
   public class Inner
   {
      public static int v = 9;
   }
}

class D : C
{
   public static void Main()
   {
      C cValue = new C();
      Console.WriteLine(cValue.Inner.v);   // CS0572
      // try the following line instead
      // Console.WriteLine(C.Inner.v);
   }
}
```

コンパイラ エラー CS0573

エラー メッセージ
'field declaration' : 構造体にインスタンス フィールド初期化子を指定することはできません。

構造体のインスタンス フィールドは初期化できません。値型のフィールドは既定値に初期化され、参照型のフィールドは **null** に初期化されます。

使用例
次の例では CS0573 エラーが生成されます。

```
// CS0573.cs
namespace x
{
    public class clx
    {
        public static void Main()
        {
        }
    }

    public struct cly
    {
        clx a = new clx();   // CS0573
        // clx a;            // OK
        int i = 7;           // CS0573
        // int i;            // OK
    }
}
```

コンパイラ エラー CS0574

エラー メッセージ
デストラクタの名前をクラスの名前と同じにしてください。

デストラクタの名前は、ティルダ (~) に続くクラス名であることが必要です。

次の例では CS0574 エラーが生成されます。

```
// CS0574.cs
namespace x
{
   public class iii
   {
      ~iiii()   // CS0574
      {
      }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0575

エラー メッセージ
クラスのみがデストラクタを含むことができます。

構造体にデストラクタを含めることはできません。

次の例では CS0575 エラーが生成されます。

```
// CS0575.cs
namespace x
{
   public struct iii
   {
      ~iii()   // CS0575
      {
      }

      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0576

エラー メッセージ
名前空間 'namespace' は、エイリアス 'identifier' と競合する定義を含んでいます

同じ名前空間を 2 回使用しようとしました。

使用例
次の例では CS0576 エラーが生成されます。

```
// CS0576.cs
using SysIO = System.IO;
public class SysIO
{
   public void MyMethod() {}
}

public class Test
{
   public static void Main()
   {
      SysIO.Stream s;   // CS0576
   }
}
```

コンパイラ エラー CS0577

エラー メッセージ
条件付き属性は、コンストラクタ、デストラクタ、演算子または明示的インターフェイスの実装であるため、'関数' では無効です

Conditional は、指定されたメソッドには適用できません。

たとえば、明示的インターフェイス定義には使用できない属性があります。次の例では CS0577 エラーが生成されます。

```
// CS0577.cs
// compile with: /target:library
interface I
{
   void m();
}

public class MyClass : I
{
   [System.Diagnostics.Conditional("a")]   // CS0577
   void I.m() {}
}
```

コンパイラ エラー CS0578

エラー メッセージ
戻り値の型が void でないため、条件付き属性は '関数' では無効です。

ConditionalAttribute は、戻り値の型が **void** 以外のメソッドには適用できません。これは、メソッドの他の戻り値の型は、プログラムの別の部分で必要になる可能性があるためです。

使用例

次の例では、CS0578 エラーが生成されます。このエラーを解決するには、**ConditionalAttribute** を削除するか、またはメソッドの戻り値を **void** に変更する必要があります。

```
// CS0578.cs
// compile with: /target:library
public class MyClass
{
   [System.Diagnostics.ConditionalAttribute("a")]   // CS0578
   public int TestMethod()
   {
      return 0;
   }
}
```

コンパイラ エラー CS0579

エラー メッセージ
'attribute' の属性が重複しています。

属性が AttributeUsage で **AllowMultiple=true** を指定しない場合は、同じ属性を複数回指定できません。

使用例
次のコードは CS0579 を生成します。

```
// CS0579.cs
using System;
public class MyAttribute : Attribute
{
}

[AttributeUsage(AttributeTargets.All,AllowMultiple=true)]
public class MyAttribute2 : Attribute
{
}

public class z
{
    [MyAttribute, MyAttribute]     // CS0579
    public void zz()
    {
    }

    [MyAttribute2, MyAttribute2]   // OK
    public void zzz()
    {
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0582

エラー メッセージ
インターフェイス メンバに対して、条件式は使用できません。

ConditionalAttribute は、インターフェイス メンバでは無効です。

次の例では CS0582 エラーが生成されます。

```
// CS0582.cs
// compile with: /target:library
using System.Diagnostics;
interface MyIFace
{
   [ConditionalAttribute("DEBUG")]   // CS0582
   void zz();
}
```

コンパイラ エラー CS0583

エラー メッセージ
内部コンパイル エラー ('アドレス'): 原因として '原因' が考えられます。

オプションを '/bugreport' にして障害報告ファイルを作成し、そのレポートをサポート担当者に送信してください。

/bugreport コンパイラ オプションを使用して問題を確認してください。

コンパイラ エラー CS0584

エラー メッセージ
内部コンパイル エラー : ステージ 'stage' シンボル 'symbol'

予測不可能な構文を解析できないためにコンパイラでエラーが発生したのかどうかを調べてください。このケースが該当しない場合は、「テクニカル サポート オプション (Visual Studio)」を参照してください。

コンパイラ エラー CS0585

エラー メッセージ
内部コンパイル エラー: ステージ 'ステージ'

予測不可能な構文を解析できないためにコンパイラでエラーが発生したのかどうかを調べてください。このケースが該当しない場合は、「テクニカル サポート オプション (Visual Studio)」を参照してください。

コンパイラ エラー CS0586

エラー メッセージ
内部コンパイル エラー: ステージ 'ステージ'

予測不可能な構文を解析できないためにコンパイラでエラーが発生したのかどうかを調べてください。このケースが該当しない場合は、「テクニカル サポート オプション (Visual Studio)」を参照してください。

コンパイラ エラー CS0587

エラー メッセージ
内部コンパイル エラー: ステージ 'ステージ'

予測不可能な構文を解析できないためにコンパイラでエラーが発生したのかどうかを調べてください。このケースが該当しない場合は、「テクニカル サポート オプション (Visual Studio)」を参照してください。

コンパイラ エラー CS0588

エラー メッセージ
内部コンパイラ エラー : ステージ 'LEX'

予測不可能な構文を解析できないためにコンパイラでエラーが発生したのかどうかを調べてください。このケースが該当しない場合は、「テクニカル サポート オプション (Visual Studio)」を参照してください。

コンパイラ エラー CS0589

エラー メッセージ
内部コンパイラ エラー : ステージ 'PARSE'

予測不可能な構文を解析できないためにコンパイラでエラーが発生したのかどうかを調べてください。このケースが該当しない場合は、「テクニカル サポート オプション (Visual Studio)」を参照してください。

コンパイラ エラー CS0590

エラー メッセージ
ユーザー定義の演算子は void を返すことはできません。

ユーザー定義の演算子を使用する目的は、オブジェクトを返すことです。

次の例では CS0590 エラーが生成されます。

```
// CS0590.cs
namespace x
{
   public class a
   {
      public static void operator+(a A1, a A2)   // CS0590
      {
      }

      // try the following user-defined operator
      /*
      public static a operator+(a A1, a A2)
      {
         return A2;
      }
      */

      public static int Main()
      {
         return 1;
      }
   }
}
```

コンパイラ エラー CS0591

エラー メッセージ
'attribute' 属性の引数の値が無効です。

属性に、無効な引数または同時に指定できない 2 つの引数が渡されました。

使用例
次の例では CS0591 エラーが生成されます。

```
// CS0591.cs
using System;

[AttributeUsage(0)]   // CS0591
class I: Attribute
{
}

public class a
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0592

エラー メッセージ
属性 'attribute' は、この種の宣言では無効です。'type' の宣言でのみ有効です。

属性が、意図していない宣言に適用されました。

使用例
次の例では CS0592 エラーが生成されます。

```
// CS0592.cs
using System;

[AttributeUsage(AttributeTargets.Interface)]
public class MyAttribute : Attribute
{
}

[MyAttribute]
public class A  // CS0592, MyAttribute is not valid for a class
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0594

エラー メッセージ
浮動小数点定数が型 'type' の範囲外にあります。

浮動小数点変数に代入された値が大きすぎます。データ型で使用できる値の範囲については、「整数型の一覧表」を参照してください。

次の例では CS0594 エラーが生成されます。

```
// CS0594.cs
namespace MyNamespace
{
   public class MyClass
   {
      public static void Main()
      {
         float f = 6.77777777777E400;   // CS0594, value too large
      }
   }
}
```

コンパイラ エラー CS0596

エラー メッセージ
Guid 属性は Comimport 属性を使って指定する必要があります。

ComImport 属性を使用するときは **Guid** 属性が必要です。

次の例では CS0596 エラーが生成されます。

```
// CS0596.cs
using System.Runtime.InteropServices;

namespace x
{
   [ComImport]   // CS0596
   // try the following line to resolve this CS0596
   // [ComImport, Guid("00000000-0000-0000-0000-000000000001")]
   public class a
   {
   }

   public class b
   {
      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS0599

エラー メッセージ
名前付き属性引数 'argument' の値が無効です。

属性に渡された引数が無効です。

コンパイラ エラー CS0601

エラー メッセージ
static または extern に指定されているメソッドでは、DllImport 属性を指定する必要があります。

正しいアクセス キーワードを持たないメソッドで **DllImport** 属性を使用しました。

次の例では CS0601 エラーが生成されます。

```
// CS0601.cs
using System.Runtime.InteropServices;
using System.Text;

public class C
{
   [DllImport("KERNEL32.DLL")]
   extern int GetCurDirectory(int bufSize, StringBuilder buf);   // CS0601
   // Try the following line instead:
   // static extern int GetCurDirectory(int bufSize, StringBuilder buf);
}

public class MainClass
{
   public static void Main ()
   {
   }
}
```

コンパイラ エラー CS0609

エラー メッセージ
override として指定されたインデクサに IndexerName 属性を設定することはできません。

名前属性 (**IndexerNameAttribute**) は、オーバーライドであるインデックス付きプロパティには適用できません。詳細については、「インデクサ」を参照してください。

次の例では CS0609 エラーが生成されます。

```
// CS0609.cs
using System;
using System.Runtime.CompilerServices;

public class idx
{
   public virtual int this[int iPropIndex]
   {
      get
      {
         return 0;
      }
      set
      {
      }
   }
}

public class MonthDays : idx
{
   [IndexerName("MonthInfoIndexer")]   // CS0609, delete to resolve this CS0609
   public override int this[int iPropIndex]
   {
      get
      {
         return 0;
      }
      set
      {
      }
   }
}

public class test
{
   public static void Main( string[] args )
   {
   }
}
```

コンパイラ エラー CS0610

エラー メッセージ
フィールドまたはプロパティを型 'type' にすることはできません。

フィールドまたはプロパティとして使用できない型があります。たとえば、**System.ArgIterator** や **System.TypedReference** などです。

次の例では、**System.TypedReference** をフィールドとして使用した結果として CS0610 エラーが生成されます。

```
// CS0610.cs
public class MainClass
{
   System.TypedReference i;   // CS0610
   public static void Main ()
   {
   }

   public static void Test(System.TypedReference i)   // OK
   {
   }
}
```

コンパイラ エラー CS0611

エラー メッセージ
配列の要素を型 'type' にすることはできません。

配列の型として使用できない型があります。たとえば、**System.TypedReference** や **System.ArgIterator** などです。

次の例では、**System.TypedReference** を配列要素として使用した結果として CS0611 エラーが生成されます。

```
// CS0611.cs
public class a
{
   public static void Main()
   {
      System.TypedReference[] ao = new System.TypedReference [10];   // CS0611
      // try the following line instead
      // int[] ao = new int[10];
   }
}
```

コンパイラ エラー CS0616

エラー メッセージ
クラス' は属性クラスではありません。

属性ブロックで属性なしクラスの使用を試みました。すべての属性の型は、System.Attribute から継承する必要があります。

使用例
次の例では CS0616 エラーが生成されます。

```
// CS0616.cs
// compile with: /target:library
[CMyClass(i = 5)]   // CS0616
public class CMyClass {}
```

次のサンプルは属性の定義方法を示しています。

```
// CreateAttrib.cs
// compile with: /target:library
using System;

[AttributeUsage(AttributeTargets.Class|AttributeTargets.Interface)]
public class MyAttr : Attribute
{
   public int Name = 0;
   public int Count = 0;

   public MyAttr (int iCount, int sName)
   {
      Count = iCount;
      Name = sName;
   }
}

[MyAttr(5, 50)]
class Class1 {}

[MyAttr(6, 60)]
interface Interface1 {}
```

コンパイラ エラー CS0617

エラー メッセージ
'参照' は有効な名前付き属性引数ではありません。名前付き属性引数は、readonly、static または const ではないフィールドでなければなりません。また、public であり static ではない読み書き可能プロパティでなければなりません。

属性クラスの private メンバへのアクセスを試みました。

使用例
次の例では CS0617 エラーが生成されます。

```
// CS0617.cs
using System;

[AttributeUsage(AttributeTargets.Struct |
                AttributeTargets.Class |
                AttributeTargets.Interface)]
public class MyClass : Attribute
{
   public int Name;

   public MyClass (int sName)
   {
      Name = sName;
      Bad = -1;
      Bad2 = -1;
   }

   public readonly int Bad;
   public int Bad2;
}

[MyClass(5, Bad=0)] class Class1 {}   // CS0617
[MyClass(5, Bad2=0)] class Class2 {}
```

コンパイラ エラー CS0619

エラー メッセージ
'member' は古い形式です : 'text'

クラス メンバが Obsolete 属性でマークされました。これにより、クラス メンバが参照されるとエラーが発行されます。

次の例では CS0619 エラーが生成されます。

```
// CS0619.cs
using System;

public class C
{
   [Obsolete("Use newMethod instead", true)]   // generates an error on use
   public static void m()
   {
   }

   // this is the method you should be using
   public static void newMethod()
   {
   }
}

class MyClass
{
   public static void Main()
   {
      C.m();   // CS0619
   }
}
```

コンパイラ エラー CS0620

エラー メッセージ
インデクサに void 型を指定できません。

インデクサの戻り値の型を **void** にすることはできません。インデクサは値を返す必要があります。

次の例では CS0620 エラーが生成されます。

```
// CS0620.cs
class MyClass
{
   public static void Main()
   {
      MyClass test = new MyClass();
      System.Console.WriteLine(test[2]);
   }

   void this [int intI]   // CS0620, return type cannot be void
   {
      get
      {
         // will need to return some value
      }
   }
}
```

コンパイラ エラー CS0621

エラー メッセージ
'member' : virtual または abstract メンバを private に指定することはできません。

プライベートな **virtual** またはプライベートな **abstract** メンバは使用できません。

次の例では CS0621 エラーが生成されます。

```
// CS0621.cs
abstract class MyClass
{
   private virtual void DoNothing1()   // CS0621
   {
   }

   private abstract void DoNothing2();   // CS0621

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0622

エラー メッセージ
配列型を割り当てるには配列初期化子式だけを使用してください。新しい式を使用してください。

配列の初期化に適した構文が、非配列の宣言で使用されました。

使用例
次の例では CS0622 エラーが生成されます。

```
// CS0622.cs
using System;

public class Test
{
    public static void Main ()
    {
        Test t = { new Test() };   // CS0622
        // Try the following instead:
        // Test[] t = { new Test() };
    }
}
```

コンパイラ エラー CS0623

エラー メッセージ
配列

配列が正しく初期化されませんでした。

コンパイラ エラー CS0625

エラー メッセージ
'field' : StructLayout(LayoutKind.Explicit) と指定されたインスタンス フィールドには FieldOffset 属性が必要です。

構造体が明示的な **StructLayout** 属性でマークされた場合、その構造体のすべてのフィールドには FieldOffset 属性が必要です。詳細については、「StructLayoutAttribute クラス」を参照してください。

次の例では CS0625 エラーが生成されます。

```
// CS0625.cs
// compile with: /target:library
using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Explicit)]
struct A
{
   public int i;   // CS0625 not static; an instance field
}

// OK
[StructLayout(LayoutKind.Explicit)]
struct B
{
   [FieldOffset(5)]
   public int i;
}
```

コンパイラ エラー CS0629

エラー メッセージ
条件付きメンバ 'member' は、インターフェイス メンバ '基本クラス メンバ' を型 '型名' で実装できません。

Conditional 属性は、インターフェイスの実装では使用できません。

次の例では CS0629 エラーが生成されます。

```
// CS0629.cs
interface MyInterface
{
   void MyMethod();
}

public class MyClass : MyInterface
{
   [System.Diagnostics.Conditional("debug")]
   public void MyMethod()    // CS0629, remove the Conditional attribute
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0631

エラー メッセージ
ref および out はこのコンテキストでは有効ではありません。

インデクサ プロパティの宣言では、ref パラメータまたは out パラメータを使用できません。

使用例
次の例では CS0631 エラーが生成されます。

```
// CS0631.cs
public class MyClass
{
    public int this[ref int i]   // CS0631
    // try the following line instead
    // public int this[int i]
    {
        get
        {
            return 0;
        }
    }
}

public class MyClass2
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0633

エラー メッセージ
'属性' 属性に対する引数は、有効な識別子でなければなりません。

ConditionalAttribute 属性または IndexerNameAttribute 属性に渡す引数は、有効な識別子であることが必要です。つまり、"+" などの文字を識別子に使用することはできません。

使用例

ConditionalAttribute で CS0633 エラーが生成される例を次に示します。次の例では CS0633 エラーが生成されます。

```
// CS0633a.cs
#define DEBUG
using System.Diagnostics;
public class Test
{
   [Conditional("DEB+UG")]   // CS0633
   // try the following line instead
   // [Conditional("DEBUG")]
   public static void Main() { }
}
```

IndexerNameAttribute で CS0633 エラーが生成される例を次に示します。

```
// CS0633b.cs
// compile with: /target:module
#define DEBUG
using System.Runtime.CompilerServices;
public class Test
{
   [IndexerName("Invalid+Identifier")]   // CS0633
   // try the following line instead
   // [IndexerName("DEBUG")]
   public int this[int i]
   {
      get { return i; }
   }
}
```

コンパイラ エラー CS0634

エラー メッセージ
'arg' : System.Interop.UnmanagedType.CustomMarshaller 型のマーシャルに対してのみ引数が有効です。

Marshal 属性に渡された引数の 1 つは、マーシャリング形式が **System.Interop.UnmanagedType.CustomMarshaller** のときだけ使用できます。

コンパイラ エラー CS0635

エラー メッセージ
'attribute' : System.Interop.UnmanagedType.CustomMarshaller は引数 ComType と Marshal を必要とします。

引数 **ComType** と引数 **Marshal** は、マーシャリング形式が **System.Interop.UnmanagedType.CustomMarshaller** のときに指定する必要があります。

コンパイラ エラー CS0636

エラー メッセージ
FieldOffset 属性は StructLayout(LayoutKind.Explicit) に指定された型のメンバでのみ使用できます。

FieldOffset 属性でマークされたメンバが含まれている構造体の宣言では、**StructLayout(LayoutKind.Explicit)** 属性を使用する必要があります。詳細については、「FieldOffsetAttribute クラス」を参照してください。

次の例では CS0636 エラーが生成されます。

```
// CS0636.cs
using System;
using System.Runtime.InteropServices;

// To resolve the error, uncomment the following line:
// [StructLayout(LayoutKind.Explicit)]
struct Worksheet
{
   [FieldOffset(4)]public int i;   // CS0636
}

public class MainClass
{
   public static void Main ()
   {
   }
}
```

コンパイラ エラー CS0637

エラー メッセージ
FieldOffset 属性は static または const フィールドで使用できません。

FieldOffset 属性は、static または const でマークされたフィールドでは使用できません。

次の例では CS0637 エラーが生成されます。

```
// CS0637.cs
using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Explicit)]
public class MainClass
{
   [FieldOffset(3)]   // CS0637
   public static int i;
   public static void Main ()
   {
   }
}
```

コンパイラ エラー CS0641

エラー メッセージ
'attribute' : System.Attribute から派生したクラスの属性のみが有効です。

System.Attribute の派生クラスだけで使用できる属性が使用されました。

次の例では CS0641 エラーが生成されます。

```
// CS0641.cs
using System;

[AttributeUsage(AttributeTargets.All)]
public class NonAttrClass   // CS0641
// try the following line instead
// public class NonAttrClass : Attribute
{
}

class MyClass
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0643

エラー メッセージ
'arg' 属性引数の名前が重複しています。

ユーザー定義の属性のパラメータ *arg* が 2 回指定されました。詳細については、「属性 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0643 エラーが生成されます。

```
// CS0643.cs
using System;
using System.Runtime.InteropServices;

[AttributeUsage(AttributeTargets.Class)]
public class MyAttribute : Attribute
{
    public MyAttribute()
    {
    }

    public int x;
}

[MyAttribute(x = 5, x = 6)]   // CS0643, error setting x twice
// try the following line instead
// [MyAttribute(x = 5)]
class MyClass
{
}

public class MainClass
{
    public static void Main ()
    {
    }
}
```

コンパイラ エラー CS0644

エラー メッセージ
'class1' は特殊クラス 'class2' からクラスを派生できません。

次の基本クラスからクラスを明示的に継承することはできません。

- **System.Enum**
- **System.ValueType**
- **System.Delegate**
- **System.Array**

これらのクラスは、コンパイラで暗黙の基本クラスとして使用されます。たとえば、**System.ValueType** は構造体の暗黙の基本クラスです。

次の例では CS0644 エラーが生成されます。

```
// CS0644.cs
class MyClass : System.ValueType   // CS0644
{
}
```

コンパイラ エラー CS0645

エラー メッセージ
識別子が長すぎます。

クラス名またはその他の識別子は、512 文字以下にする必要があります。

コンパイラ エラー CS0646

エラー メッセージ
インデクサを含む型に対して DefaultMember 属性を指定できません。

クラスまたはその他の型で **System.Reflection.DefaultMemberAttribute** を指定する場合は、インデクサを含めることはできません。詳細については、「プロパティ」を参照してください。

次の例では CS0646 エラーが生成されます。

```
// CS0646.cs
// compile with: /target:library
[System.Reflection.DefaultMemberAttribute("x")]   // CS0646
class MyClass
{
   public int this[int index]   // an indexer
   {
      get
      {
         return 0;
      }
   }

   public int x = 0;
}

// OK
[System.Reflection.DefaultMemberAttribute("x")]
class MyClass2
{
   public int prop
   {
      get
      {
         return 0;
      }
   }

   public int x = 0;
}

class MyClass3
{
   public int this[int index]   // an indexer
   {
      get
      {
         return 0;
      }
   }

   public int x = 0;
}
```

コンパイラ エラー CS0647

エラー メッセージ
'attribute' 属性を作成時にエラーが発生しました --'reason'

次の例では CS0647 エラーが生成されます。

```
// CS0647.cs
using System.Runtime.InteropServices;

[Guid("z")]   // CS0647, incorrect uuid format.
// try the following line instead
// [Guid("00000000-0000-0000-0000-000000000001")]
public class MyClass
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0648

エラー メッセージ
'type' はこの言語によってサポートされていない型です。

別の言語 (C++ など) から生成されたメタデータに、マネージ型としてマークされていない型が含まれていました。この型に関する情報がメタデータに含まれていますが、この型は C# で記述されたプログラムでは使用できません。

コンパイラ エラー CS0650

エラー メッセージ
構文エラーです。不適切な配列の宣言子です。マネージ配列を宣言するには、次元指定子を変数の識別子の前に指定します。固定サイズ バッファ フィールドを宣言するには、フィールド型の前に fixed キーワードを使用します。

配列が正しく宣言されませんでした。固定サイズ バッファの構文は、配列の構文とは異なる点に注意してください。

使用例
次の例では CS0650 エラーが生成されます。

```
// CS0650.cs
public class MyClass
{
   public static void Main()
   {
      int myarray[2];   // CS0650

      // OK
      int[] myarray2 = new int[2] {1,2};
      myarray2[0] = 0;
    }
}
```

コンパイラ エラー CS0653

エラー メッセージ
抽象であるため属性クラス 'class' を適用でません。

abstract カスタム属性クラスは、属性として使用できません。

次の例では CS0653 エラーが生成されます。

```
// CS0653.cs
using System;

public abstract class MyAttribute : Attribute
{
}

public class My2Attribute : MyAttribute
{
}

[My]   // CS0653
// try the following line instead
// [My2]
class MyClass
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0655

エラー メッセージ
'parameter' は有効な属性パラメータ型ではないため、有効な名前付き属性引数ではありません。

属性の有効なパラメータ型については、「属性 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0655 エラーが生成されます。

```
// CS0655.cs
using System;

class MyAttribute : Attribute
{
    // decimal is not valid attribute parameter type
    public decimal d = 0;
    public int e = 0;
}

[My(d = 0)]   // CS0655
// Try the following line instead:
// [My(e = 0)]
class C
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0656

エラー メッセージ
コンパイラが必要とするメンバ 'object.member' がありません。

次のいずれかの問題があります。

- インストールされている共通言語ランタイムが破損している。
- 共通言語ランタイムにも存在する型を定義するアセンブリへの参照があるが、C# コンパイラが予期する方法でアセンブリの型が定義されていない。

参照をチェックして、正しいバージョンの共通言語ランタイムを使用するように設定してください。

コンパイラの警告 (レベル 1) CS0657

エラー メッセージ
'attribute modifier' はこの宣言の有効な有効な属性ではありません。宣言の有効な属性の場所は 'location' です。

無効な場所で属性の修飾子が見つかりました。詳細については、「属性の対象」を参照してください。

次の例では CS0657 エラーが生成されます。

```
// CS0657.cs
// compile with: /target:library
public class TestAttribute : System.Attribute {}
[return: Test]   // CS0657 return not valid on a class
class Class1 {}
```

コンパイラ エラー CS0662

エラー メッセージ
'method' は ref パラメータの Out 属性だけを指定することはできません。In と Out 属性の両方を使用するか、どちらも使用しないでください。

インターフェイス メソッドに、Out 属性だけで ref を使用するパラメータがあります。**Out** 属性を使用する **ref** パラメータでは、In 属性も使用する必要があります。

次の例では CS0662 エラーが生成されます。

```
// CS0662.cs
using System.Runtime.InteropServices;

interface I
{
   void method([Out] ref int i);   // CS0662
   // try one of the following lines instead
   // void method(ref int i);
   // void method([Out, In]ref int i);
}

class test
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0663

エラー メッセージ
ref と out のみが異なるオーバーロードされたメソッドを指定することはできません。

パラメータでの ref と out の使い方だけが異なるメソッドは許可されていません。

次の例では CS0663 エラーが生成されます。

```
// CS0663.cs
class TestClass
{
   public static void Main()
   {
   }

   public void Test(ref int i)
   {
   }

   public void Test(out int i)   // CS0663
   {
   }
}
```

コンパイラ エラー CS0664

エラー メッセージ
型 double のリテラルを暗黙的に型 'type' に変換することはできません。'suffix' サフィックスを使用して、この型のリテラルを作成してください。

代入が完了しませんでした。サフィックスを使用して命令を修正してください。各型のドキュメントは、その型に対応するサフィックスを特定します。

次の例では CS0664 エラーが生成されます。

```
// CS0664.cs
class M
{
   static void Main()
   {
      decimal m = 1.0;   // CS0664
      // try the following line instead
      // decimal m = 1.0M;
      System.Console.WriteLine(m);
   }
}
```

コンパイラ エラー CS0666

エラー メッセージ
'member' : 新しい protected メンバが構造体で宣言されています。

構造体を **abstract** にすることはできません。また、構造体は常に暗黙の **sealed** と見なされます。詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS0666 エラーが生成されます。

```
// CS0666.cs
class M
{
    static void Main()
    {
    }
}

struct S
{
    protected int x;   // CS0666
}
```

コンパイラ エラー CS0667

エラー メッセージ
機能 '無効な機能' は使用できません。'有効な機能' を使用してください。

使用しようとしている機能は、現在推奨されていません。有効な機能を使用するようにコードを変更してください。

コンパイラ エラー CS0668

エラー メッセージ
2 つのインデクサの名前が違います。1 つの型の中のそれぞれのインデクサの IndexerName 属性は、同じでなければなりません。

IndexerName 属性に渡される値は、型のすべてのインデクサに対して同じであることが必要です。**IndexerName** 属性の詳細については、「IndexerNameAttribute クラス」を参照してください。

次の例では CS0668 エラーが生成されます。

```
// CS0668.cs
using System;
using System.Runtime.CompilerServices;

class IndexerClass
{
   [IndexerName("IName1")]
   public int this [int index]   // indexer declaration
   {
      get
      {
         return index;
      }
      set
      {
      }
   }

   [IndexerName("IName2")]
   public int this [string s]    // CS0668, change IName2 to IName1
   {
      get
      {
         return int.Parse(s);
      }
      set
      {
      }
   }

   void Main()
   {
   }
}
```

コンパイラ エラー CS0669

エラー メッセージ
ComImport 属性を持つクラスはユーザー定義のコンストラクタを持てません。

ComImport クラスのコンストラクタは、共通言語ランタイムの COM 相互運用層に用意されています。そのため、COM オブジェクトは、実行時にマネージ オブジェクトとして使用できます。

次の例では CS0669 エラーが生成されます。

```
// CS0669.cs
using System.Runtime.InteropServices;
[ComImport, Guid("00000000-0000-0000-0000-000000000001")]
class TestClass
{
   TestClass()   // CS0669, delete constructor to resolve
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0670

エラー メッセージ
フィールドは void 型を持てません。

フィールドが void 型になるように宣言されました。

次の例では CS0670 エラーが生成されます。

```
// CS0670.cs
class C
{
   void f;   // CS0670
   // try the following line instead
   // public int f;

   public static void Main()
   {
      C myc = new C();
      myc.f = 0;
   }
}
```

コンパイラ エラー CS0673

エラー メッセージ
System.Void は C# から使用できません。void 型オブジェクトを取得するには typeof(void) を使用してください。

System.Void は C# では使用できません。

次の例では CS0673 エラーが生成されます。

```
// CS0673.cs
class MyClass
{
   public static void Main()
   {
      System.Type t = typeof(System.Void);   // CS0673
      // try the following line instead
      // System.Type t = typeof(void);
   }
}
```

コンパイラ エラー CS0674

エラー メッセージ
'System.ParamArrayAttribute' を使用しないでください。代わりに 'params' キーワードを使用してください。

C# コンパイラでは、System.ParamArrayAttribute を使用できません。代わりに、params を使用してください。

次の例では CS0674 エラーが生成されます。

```
// CS0674.cs
using System;
public class MyClass
{

   public static void UseParams([ParamArray] int[] list)   // CS0674
   // try the following line instead
   // public static void UseParams(params int[] list)
   {
      for ( int i = 0 ; i < list.Length ; i++ )
         Console.WriteLine(list[i]);
      Console.WriteLine();
   }

   public static void Main()
   {
      UseParams(1, 2, 3);
   }
}
```

コンパイラ エラー CS0677

エラー メッセージ
'variable' : volatile フィールドは 'type' 型にはなれません。

volatile キーワードを使用して宣言したフィールドの型は、次のいずれかであることが必要です。

- 参照型
- ポインタ型 (**unsafe** コンテキスト内)
- **sbyte**、**byte**、**short**、**ushort**、**int**、**uint**、**char**、**float**、**bool** の各型
- 上のいずれかの型に基づく列挙型

次の例では CS0677 エラーが生成されます。

```
// CS0677.cs
class TestClass
{
   private volatile long i;   // CS0677

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0678

エラー メッセージ
'variable': フィールドは同時に volatile と readonly にはなれません。

volatile キーワードと readonly キーワードの使用は、相互に排他的です。

次の例では CS0678 エラーが生成されます。

```
// CS0678.cs
using System;

class TestClass
{
   private readonly volatile int i;   // CS0678
   // try the following line instead
   // private volatile int i;

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0681

エラー メッセージ
修飾子 '抽象' はフィールドで有効ではありません。プロパティを使用してください

フィールドを abstract として宣言することはできません。ただし、抽象プロパティを使用してフィールドにアクセスすることはできます。

使用例
次の例では CS0681 エラーが生成されます。

```
// CS0681.cs
// compile with: /target:library
abstract class C
{
    abstract int num;  // CS0681
}
```

代わりに、次のコードを試してください。

```
// CS0681b.cs
// compile with: /target:library
abstract class C
{
    public abstract int num
    {
       get;
       set;
    }
}
```

コンパイラ エラー CS0682

エラー メッセージ

'type2' **はこの言語でサポートされていないため、**'type1' **で実装できません。**

このエラーは、実装しようとしたインターフェイスがコンパイラのサポートしていない言語で記述されている場合に発生します。

コンパイラ エラー CS0683

エラー メッセージ
'explicitmethod' 明示的なメソッドの実装で、アクセサである 'method' を実装することはできません。

次の例では CS0683 エラーが生成されます。

```
// CS0683.cs
interface IExample
{
   int Test { get; }
}

class CExample : IExample
{
   int IExample.get_Test() { return 0; } // CS0683
   int IExample.Test { get{ return 0; } } // correct
}
```

コンパイラ エラー CS0685

エラー メッセージ
条件付きメンバ 'メンバ' には out パラメータを指定できません。

メソッドに対して ConditionalAttribute 属性を使用する場合、そのメソッドに out パラメータが存在しないことが必要です。これは、コンパイラがメソッド呼び出しの部分を出力しなかった場合、out パラメータに使用される変数の値が未定義になるためです。このエラーを回避するには、条件付きメソッドの宣言から out パラメータを削除するか、Conditional 属性を使わないようにします。

使用例
次の例では CS0685 エラーが生成されます。

```
// CS0685.cs
using System.Diagnostics;

class C
{
    [Conditional("DEBUG")]
    void trace(out int i)  // CS0685
    {
        i = 1;
    }
}
```

コンパイラ エラー CS0686

エラー メッセージ
アクセサ 'accessor' は、インターフェイス メンバ 'member' を型 'type' に対して実装できません。明示的なインターフェイスの実装を使用してください。

推奨事項 : このエラーは、実装しているインターフェイスに、プロパティまたはイベントに対して自動的に生成されるメソッドと同じ名前のメソッドが存在する場合に発生します。プロパティの get/set メソッドは get_<プロパティ> および set_<プロパティ> として生成されます。また、イベントの場合は、add/remove メソッドが add_<イベント> および remove_<イベント> として生成されます。インターフェイスに、これらと同じ名前のメソッドが存在すると、名前の競合が生じます。このエラーを回避するには、明示的なインターフェイスの実装を使用してメソッドを実装します。そのためには、関数を次のように指定します。

```
      Interface.get_property() { /* */ }
Interface.set_property() { /* */ }
```

使用例

次の例では CS0686 エラーが生成されます。

```
// CS0686.cs
interface I
{
    int get_P();
}

class C : I
{
    public int P
    {
        get { return 1; }  // CS0686
    }
}
// But the following is valid:
class D : I
{
    int I.get_P() { return 1; }
    public static void Main() {}
}
```

このエラーは、イベントの宣言で発生する場合もあります。イベントの制御構文では、`add_event` および `remove_event` というメソッドが自動的に生成されるため、インターフェイス内の同じ名前のメソッドと競合します。

```
// CS0686b.cs
using System;

interface I
{
    void add_OnMyEvent(EventHandler e);
}

class C : I
{
    public event EventHandler OnMyEvent
    {
        add { }  // CS0686
        remove { }
    }
}

// Correct (using explicit interface implementation):
class D : I
{
```

```
    void I.add_OnMyEvent(EventHandler e) {}
    public static void Main() {}
}
```

コンパイラ エラー CS0687

エラー メッセージ
名前空間エイリアス修飾子 '::' は、常に型または名前空間を解決するので、ここでは無効です。代わりに、'.' を使用してください。

このエラーは、なんらかのコーディング ミスが、予期しない場所に存在する型であるとして、パーサーによって解釈された場合に発生します。型または名前空間の名前は、メンバ アクセス演算子 (.) を使ったメンバ アクセス式でのみ使用できます。グローバル スコープ演算子 (::) が、別のコンテキストで使用されている場合などに、このエラーが発生します。

使用例
次の例では CS0687 エラーが生成されます。

```
// CS0687.cs

using M = Test;
using System;

public class Test
{
    public static int x = 77;

    public static void Main()
    {
        Console.WriteLine(M::x); // CS0687
        // To resolve use the following line instead:
        // Console.WriteLine(M.x);
    }
}
```

コンパイラ エラー CS0689

エラー メッセージ
'識別子' は型パラメータであるため、派生させることはできません。

ジェネリック クラスの基本クラスやインターフェイスを型パラメータで指定することはできません。特定のクラスまたはインターフェイス (または特定のジェネリック クラス) から派生させるか、未知の型をメンバとして追加してください。

次の例では CS0689 エラーが生成されます。

```
// CS0689.cs
class A<T> : T   // CS0689
{
}
```

コンパイラ エラー CS0690

エラー メッセージ
入力ファイル 'ファイル' は無効なメタデータを含んでいます。

メタデータ ファイルを開くことはできますが、一部のローカリゼーションの問題のためにメタデータが破損しました。このエラーは、メタデータ ファイルを開くことができる点以外は CS0009 エラーと同じです。

PE (Process Executable) は実行可能ファイルです。

コンパイラ エラー CS0692

エラー メッセージ
重複する型パラメータ '識別子' です。'

型パラメータ リスト内で、同じ名前を重複して使用することはできません。重複する型パラメータを削除するか、名前を変更してください。

使用例
次の例では CS0692 エラーが生成されます。

```
// CS0692.cs
// compile with: /target:library
class C <T, A, T>   // CS0692
{
}

class D <T, T>   // CS0692
{
}
```

コンパイラ エラー CS0694

エラー メッセージ
型パラメータ '識別子' には含んでいる型またはメソッドと同じ名前が付いています。

型パラメータに対して別の名前を使用する必要があります。型パラメータ名を、その型パラメータを含む型やメソッドと同じ名前にすることはできません。

使用例
次の例では CS0694 エラーが生成されます。

```
// CS0694.cs
// compile with: /target:library
class C<C> {}   // CS0694
```

このエラーは、上記のようなジェネリック クラスが関係する例の他に、メソッドで発生する場合もあります。

```
// CS0694_2.cs
// compile with: /target:library
class A
{
   public void F<F>(F arg);   // CS0694
}
```

コンパイラ エラー CS0695

エラー メッセージ
型パラメータの代用に対して統合している可能性があるため、'ジェネリック型' は 'ジェネリック インターフェイス' と 'ジェネリック インターフェイス' の両方を実装することはできません。

このエラーは、1 つのジェネリック インターフェイスが 2 種類以上にパラメータ化されて、ジェネリック クラスに実装されている場合に発生します。仮に型パラメータの置換がなされた場合、その 2 つのインターフェイスが同じものになってしまいます。このエラーを回避するには、一方のインターフェイスだけを実装するか、競合が生じないように型パラメータを変更します。

次の例では CS0695 エラーが生成されます。

```
// CS0695.cs
// compile with: /target:library

interface I<T>
{
}

class G<T1, T2> : I<T1>, I<T2>  // CS0695
{
}
```

コンパイラ エラー CS0698

エラー メッセージ
ジェネリック型は属性クラスであるため、'クラス' から派生できません

属性クラスから派生するすべてのクラスは、属性であることが必要です。属性をジェネリック型にすることはできません。

次の例では CS0698 エラーが生成されます。

```
// CS0698.cs
class C<T> : System.Attribute  // CS0698
{
}
```

コンパイラ エラー CS0699

エラー メッセージ
'ジェネリック' は、型パラメータ '識別子' を定義していません。

ジェネリックの定義部分で、型パラメータで宣言されていないパラメータが使用されています。このエラーは、型パラメータに使用された名前に矛盾が見られる場合に発生します。

次の例では CS0699 エラーが生成されます。

```
// CS0699.cs
class C<T> where U : I   // CS0699 – U is not a valid type parameter
{
}
```

コンパイラ エラー CS0701

エラー メッセージ
'識別子' は有効な制約ではありません。制約として使用された型はインターフェイス、非シール クラス、または型パラメータでなければなりません。

このエラーは、シールされた型を制約として使用した場合に発生します。このエラーを解決するには、シールされていない型以外は、制約に使わないようにします。

使用例
次の例では CS0701 エラーが生成されます。

```
// CS0701.cs
// compile with: /target:library
class C<T> where T : System.String {}   // CS0701
class D<T> where T : System.Attribute {}   // OK
```

コンパイラ エラー CS0702

エラー メッセージ
制約は特殊クラス '識別子' にはなれません。

`System.Array` 型、 `System.Delegate` 型、 `System.Enum` 型、 `System.ValueType` 型を制約として使用することはできません。

使用例
次の例では CS0702 エラーが生成されます。

```
// CS0702.cs
class C<T> where T : System.Array  // CS0702
{
}
```

コンパイラ エラー CS0703

エラー メッセージ
アクセシビリティに一貫性がありません。制約の型 '識別子' のアクセシビリティはフィールド '識別子' よりも低く設定されています

ジェネリック クラスそのもののアクセシビリティよりも制限の強いジェネリック パラメータ制約は使用できません。次の例では、ジェネリック クラス C<T> は public として宣言されているにもかかわらず、T に対する制約で強制的に internal なインターフェイスを実装しようとしています。仮にこれが許可された場合、internal なアクセシビリティを持つクライアント以外、そのクラスのパラメータを作成できないことになります。つまり、実質的に、そのクラスを使用できるのは、internal なアクセシビリティを持つクライアントに限定されてしまいます。

このエラーを回避するには、ジェネリック クラスのアクセス レベルを、制約に指定されたクラスまたはインターフェイスよりも狭い範囲に限定する必要があります。

次の例では CS0703 エラーが生成されます。

```
// CS0703.cs
internal interface I {}
public class C<T> where T : I  // CS0703 – I is internal; C<T> is public
{
}
```

コンパイラ エラー CS0704

エラー メッセージ
型パラメータであるため、'型' でメンバの照合を行えません

型パラメータを介して内部の型を指定することはできません。目的の型を明示的に指定してください。

使用例
次の例では CS0704 エラーが生成されます。

```
// CS0704.cs
class B
{
    public class I { }
}

class C<T> where T : B
{
    T.I f;  // CS0704 – member lookup on type parameter
    // Try this instead:
    // B.I f;
}

class CMain
{
    public static void Main() {}
}
```

コンパイラ エラー CS0706

エラー メッセージ
無効な制約型です。制約として使用された型はインターフェイス、非シール クラス、または型パラメータでなければなりません。

このエラーは、制約句で無効な構文が使用された場合に発生します。このエラーを回避するには、エラーの発生した構文を、インターフェイスまたはシールされていないクラスで置き換えます。

使用例
次の例では CS0706 エラーが生成されます。

```
// CS0706.cs
// compile with: /target:library
class A {}
class C<T> where T : int[] {}  // CS0706
class D<T> where T : A {}  // OK
```

コンパイラ エラー CS0708

エラー メッセージ
'フィールド': 静的クラスでインスタンス メンバを宣言することはできません。

このエラーは static として宣言したクラスで、static 以外のメンバを宣言した場合に発生します。静的クラスのインスタンスを作成することはできないため、インスタンス変数は意味を持ちません。**static** キーワードは、静的クラスのすべてのメンバに適用する必要があります。

次の例では CS0708 エラーが生成されます。

```
// CS0708.cs
// compile with: /target:library
public static class C
{
   int i;  // CS0708
   static int j;  // OK
}
```

コンパイラ エラー CS0709

エラー メッセージ
'派生クラス':静的クラス '基本クラス' から派生することはできません。

静的クラスをインスタンス化したり、派生させたりすることはできません。つまり、静的クラスは他のクラスの基本クラスになることはできません。

使用例
次の例では CS0709 エラーが生成されます。

```
// CS0709.cs
// compile with: /target:library
public static class Base {}
public class Derived : Base {}   // CS0709
```

コンパイラ エラー CS0710

エラー メッセージ
静的クラスにはコンストラクタを指定できません。

静的クラスはインスタンス化できないため、コンストラクタは不要です。このエラーを回避するには、静的クラスからコンストラクタを削除します。どうしてもインスタンスを構築する必要がある場合は、そのクラスを非静的クラスに変更してください。

次の例では CS0710 エラーが生成されます。

```
// CS0710.cs
public static class C
{
   public C()  // CS0710
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0711

エラー メッセージ
静的クラスにデストラクタを含めることはできません。

静的クラスはインスタンス化できないため、コンストラクタやデストラクタは不要です。このエラーを回避するには、静的クラスからデストラクタを削除します。どうしてもインスタンスを構築または破棄する必要がある場合は、そのクラスを非静的クラスに変更してください。

次の例では CS0711 エラーが生成されます。

```
// CS0711.cs
public static class C
{
   ~C()  // CS0711
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0712

エラー メッセージ
静的クラス 'static class' のインスタンスを作成することはできません。

静的クラスのインスタンスを作成することはできません。静的クラスは、静的なフィールドとメソッドを持ち、インスタンス化できないようになっています。

使用例
次の例では CS0712 エラーが生成されます。

```
// CS0712.cs
public static class SC
{
}

public class CMain
{
    public static void Main()
    {
        SC sc = new SC();  // CS0712
    }
}
```

コンパイラ エラー CS0713

エラー メッセージ
静的クラス 'static type' は型 'type' から派生できません。静的クラスはオブジェクトから派生しなければなりません。

この操作が仮に許可されていると、静的クラスが基本クラスのメソッドおよび非静的なメンバを継承できることになり、その点で静的とは言えなくなってしまいます。この操作が許可されないのは、このような理由によるものです。

次の例では CS0713 エラーが生成されます。

```
// CS0713.cs
public class Base
{
}

public static class Derived : Base  // CS0713
{
}

public class CMain
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0714

エラー メッセージ
'静的な型': 静的クラスで、インターフェイスを実装することはできません。

インターフェイスは、オブジェクトに対して非静的なメソッドを定義するため、静的なクラスがインターフェイスを実装することはできません。このエラーを解決するには、静的なクラスではインターフェイスを実装しないようにします。

使用例
次の例では CS0714 エラーが生成されます。

```
// CS0714.cs
interface I
{
}

public static class C : I  // CS0714
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0715

エラー メッセージ
'静的クラス': 静的クラスに、ユーザー定義された演算子を含むことはできません。

ユーザー定義の演算子は、クラスのインスタンスに対して作用します。静的クラスはインスタンス化できないため、演算子に必要なインスタンスを作成できません。したがって、ユーザー定義の演算子を静的クラスに対して使用することはできません。

次の例では CS0715 エラーが生成されます。

```
// CS0715.cs
public static class C
{
   public static C operator+(C c)  // CS0715
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0716

エラー メッセージ
スタティック型 '型' に変換することはできません。

このエラーは、静的な型に変換するキャストを使用した場合に発生します。オブジェクトを静的な型のインスタンスにすることはできないため、静的な型へのキャストは一切認められません。

使用例
次の例では CS0716 エラーが生成されます。

```
// CS0716.cs

public static class SC
{
    static void F() { }
}

public class Test
{
    public static void Main()
    {
        object o = new object();
        System.Console.WriteLine((SC)o);  // CS0716
    }
}
```

コンパイラ エラー CS0717

エラー メッセージ
'静的クラス': 静的クラスは、制約として使用することはできません。

静的クラスは静的なメンバしか持たず、インスタンス メンバは存在しないため、拡張することはできません。拡張できないという点で、型パラメータや制約として使用することは無意味です。静的クラスから特化したクラスとして型が存在することはないためです。

使用例
次の例では CS0717 エラーが生成されます。

```
// CS0717.cs

public static class SC
{
    public static void F()
    {
    }
}

public class G<T> where T : SC  // CS0717
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS0718

エラー メッセージ
'型': スタティック型を汎用引数として使用することはできません。

静的な型は、インスタンス化できないため、汎用引数として使用することはできません。このエラーを解決するには、汎用引数から静的な型を削除します。

使用例
次の例では CS0718 エラーが生成されます。

```
// CS0718.cs
public static class SC
{
    public static void F()
    {
    }
}

public class G<T>
{
}

public class CMain
{
    public static void Main()
    {
        G<SC> gsc = new G<SC>();  // CS0718
    }
}
```

コンパイラ エラー CS0719

エラー メッセージ
'型': 配列要素をスタティック型にすることはできません。

配列に静的な型を割り当てることは意味をなしません。配列の要素はインスタンスであり、静的な型のインスタンスを作成することはできません。

次の例では CS0719 エラーが生成されます。

```
// CS0719.cs
public static class SC
{
   public static void F()
   {
   }
}


public class CMain
{
   public static void Main()
   {
      SC[] sca = new SC[10];  // CS0719
   }
}
```

コンパイラ エラー CS0720

エラー メッセージ
'静的クラス': 静的クラスでインデクサを宣言することはできません。

インデクサはインスタンス以外では使用できず、また、静的な型はインスタンスを作成できないため、静的クラスでインデクサを使用することは意味を持ちません。

使用例
次の例では CS0720 エラーが生成されます。

```
// CS0720.cs

public static class Test
{
    public int this[int index]  // CS0720
    {
        get { return 1; }
        set {}
    }

    static void Main() {}
}
```

コンパイラ エラー CS0721

エラー メッセージ
'型': スタティック型をパラメータとして使用することはできません。

パラメータに静的な型を使用しても意味がありません。静的な型はインスタンスを作成することが不可能であるため、パラメータとして渡すことはできません。

次の例では CS0721 エラーが生成されます。

```
// CS0721.cs
public static class SC
{
}


public class CMain
{
   public void F(SC sc)  // CS0721
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0722

エラー メッセージ
'型': スタティック型を戻り値の型として使用することはできません。

静的な型はインスタンスを作成できないため、戻り値に静的な型を使用することは意味がありません。

次の例では CS0722 エラーが生成されます。

```
// CS0722.cs
public static class SC
{
}


public class CMain
{
   public SC F()  // CS0722
   {
      return null;
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS0723

エラー メッセージ
スタティック型 '型' の変数を宣言することはできません。

静的な型のインスタンスは作成できません。

次の例では CS0723 エラーが生成されます。

```
// CS0723.cs
public static class SC
{
}


public class CMain
{
   public static void Main()
   {
      SC sc = null;  // CS0723
   }
}
```

コンパイラ エラー CS0724

エラー メッセージ
引数のない throw ステートメントは、最も内側の catch 句の中に入れ子にされた finally 句の中で使用することはできません。

次の例では、**finally** 句ブロックに **throw** ステートメントがあるため、CS0724 エラーが生成されます。

使用例
次の例では CS0724 エラーが生成されます。

```
// CS0724.cs
using System;

class X
{
    static void Test()
    {
        try
        {
            throw new Exception();
        }
        catch
        {
            try
            {
            }
            finally
            {
                throw; // CS0724
            }
        }
    }

    static void Main()
    {
    }
}
```

コンパイラ エラー CS0726

エラー メッセージ
'形式指定子' は有効な形式指定子ではありません。

このエラーは、デバッガで発生します。デバッガ ウィンドウでは、変数名を入力するときに、変数名に続けて、コンマおよび書式指定子を入力できます。たとえば、「`myInt, h`」や「`myString,nq`」のように入力できます。このエラーは、コンパイラがC# の書式指定子を認識できなかった場合に発生します。

コンパイラ エラー CS0727

エラー メッセージ
無効な形式指定子です。

このエラーは、デバッガで発生します。デバッガ ウィンドウでは、変数名を入力するときに、変数名に続けて、コンマおよび書式指定子を入力できます。たとえば、「myInt, h」や「myString,nq」のように入力できます。このエラーは、入力された内容をコンパイラが完全に解析できなかった場合に発生します。このエラーを解決するには、変数名と、(オプションで) コンマおよび有効な書式指定子を入力し直します。

コンパイラ エラー CS0729

エラー メッセージ
型 '型' はこのアセンブリ内で定義されていますが、これには型フォワーダが指定されています。

同じアセンブリで定義されている型には型フォワーダを使用できません。

使用例
次の例では CS0729 エラーが生成されます。

```
// CS0729.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:TypeForwardedTo(typeof(TestClass))]   // CS0729
class TestClass {}
```

コンパイラ エラー CS0730

エラー メッセージ
型 '型' は、'型' の入れ子にされた型なので、転送できません。

このエラーは、入れ子になったクラスを転送しようとしたときに生成されます。

使用例
次の例では CS0730 エラーが生成されます。この例は 2 つのソース ファイルで構成されます。まず、ライブラリ ファイル `CS0730a.cs` をコンパイルし、そのライブラリ ファイルを参照するファイル `CS0730.cs` をコンパイルします。

```
// CS0730a.cs
// compile with: /t:library
public class Outer
{
   public class Nested {}
}
```

```
// CS0730.cs
// compile with: /t:library /r:CS0730a.dll
using System.Runtime.CompilerServices;

[assembly:TypeForwardedToAttribute(typeof(Outer.Nested))]   // CS0730

[assembly:TypeForwardedToAttribute(typeof(Outer))]   // OK
```

コンパイラ エラー CS0731

エラー メッセージ
アセンブリ 'アセンブリ' にある '型' の型フォワーダで循環が発生します

このエラーは、不適切な形式のメタデータをインポートした場合にのみ発生します。C# ソースのみの場合には発生しません。

使用例
次の例では CS0731 エラーが生成されます。この例は、次の 3 つのファイルで構成されます。

1. Circular.IL

2. Circular2.IL

3. CS0731.cs

まず .IL ファイルをライブラリとしてコンパイルしてから、2 つのファイルを参照する .cs コードをコンパイルします。

```
// Circular.il
// compile with: /DLL /out=Circular.dll
.assembly extern circular2
{
  .ver 0:0:0:0
}
.assembly extern circular3
{
  .ver 0:0:0:0
}
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )   // .z\V.4..
  .ver 2:0:0:0
}
.assembly Circular
{
  .custom instance void [mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAtt
ribute::.ctor(int32) = ( 01 00 08 00 00 00 00 00 )
  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.class extern forwarder Circular.Referenced.TypeForwarder
{
  .assembly extern circular2
}
.module Circular.dll
// MVID: {880C2329-C915-42A0-83E9-9D10C3E6DBD0}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003       // WINDOWS_CUI
.corflags 0x00000001    //  ILONLY
// Image base: 0x04E40000
// ======== CLASS MEMBERS DECLARATION =========
.class public abstract auto ansi sealed beforefieldinit User
       extends [mscorlib]System.Object
{
  .method public hidebysig static class [circular2]Circular.Referenced.TypeForwarder
          F() cil managed
  {
    .maxstack  1
    newobj     instance void [circular2]Circular.Referenced.TypeForwarder::.ctor()
    ret
  }
}
```

```
// Circular2.il
// compile with: /DLL /out=Circular2.dll
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )   // .z\V.4..
  .ver 2:0:0:0
}
.assembly extern Circular
{
  .ver 0:0:0:0
}
.assembly circular2
{
  .custom instance void [mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAtt
ribute::.ctor(int32) = ( 01 00 08 00 00 00 00 00 )
  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.class extern forwarder Circular.Referenced.TypeForwarder
{
  .assembly extern Circular
}
.module circular2.dll
// MVID: {8B3BE5C8-DBE1-49C4-BC72-DF35F0387C21}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003       //   WINDOWS_CUI
.corflags 0x00000001    //  ILONLY
// Image base: 0x04E40000
```

```
// CS0731.cs
// compile with: /reference:circular.dll /reference:circular2.dll
// CS0731 expected
class A {
   public static void Main() {
      User.F();
   }
}
```

コンパイラ エラー CS0733

エラー メッセージ
ジェネリック型 'GenericType<>' を転送できません。

使用例
次の例では CS0733 エラーが生成されます。最初のファイルをライブラリとしてコンパイルし、2 つ目のファイルをコンパイルするときに参照します。

```
// CS0733a.cs
// compile with: /target:library
public class GenericType<T>
{
}
```

```
// CS0733.cs
// compile with: /target:library /r:CS0733a.dll
[assembly: System.Runtime.CompilerServices.TypeForwardedTo(typeof(GenericType<int>))]   //
CS0733
```

コンパイラ エラー CS0734

エラー メッセージ
/moduleassemblyname オプションは 'module' のターゲット型をビルドするときのみ指定できます。

コンパイラ オプション **/moduleassemblyname** は .netmodule をビルドするときにのみ使用します。詳細については、「/moduleassemblyname (モジュールに対するフレンド アセンブリの指定) (C# コンパイラ オプション)」を参照してください。

.netmodule のビルドの詳細については、「/target:module (アセンブリに追加するモジュールの作成) (C# コンパイラ オプション)」を参照してください。

使用例
次の例では CS0734 エラーが生成されます。解決するには、コンパイル オプションに **/target:module** を追加します。

```
// CS0734.cs
// compile with: /moduleassemblyname:A
// CS0734 expected
public class Test {}
```

コンパイラ エラー CS1001

エラー メッセージ
識別子が必要です。

識別子が指定されていません。たとえば、列挙型を宣言するときはメンバを指定する必要があります。

次の例では CS1001 エラーが生成されます。

```
// CS1001.cs
public class clx
{
   enum splitch : int
   {
      'a', 'b' // CS1001, 'a' is not a valid int identifier
   };

   public static void Main()
   {
   }
}
```

パラメータ名は、インターフェイス定義など、コンパイラによって使用されない場合でも指定する必要があります。これらのパラメータは、そのインターフェイスを利用するプログラマが、パラメータの意味を理解するうえで必要になります。

```
// CS1001-2.cs
// compile with: /target:library
interface IMyTest
{
   void TestFunc1(int, int);  // CS1001
}

class CMyTest : IMyTest
{
   void IMyTest.TestFunc1(int a, int b)
   {
   }
}
```

コンパイラ エラー CS1002

エラー メッセージ
; が必要です。

セミコロン (;) が付けられていません。

次の例では CS1002 エラーが生成されます。

```
// CS1002.cs
namespace x
{
   abstract public class clx
   {
      int i   // CS1002, missing semicolon

      public static int Main()
      {
         return 0;
      }
   }
}
```

コンパイラ エラー CS1003

エラー メッセージ
構文エラーです。'char' が必要です。

コンパイラは、いくつかのエラー条件に対してこのエラーを生成します。コードを確認して構文エラーを見つけてください。

次の例では CS1003 エラーが生成されます。

```
// CS1003.cs
public class b
{
   public static void Main()
   {
      int[] a;
      a();   // CS1003
   }
}
```

コンパイラ エラー CS1004

エラー メッセージ
'modifier' 修飾子が重複しています。

アクセス修飾子などの修飾子の重複が検出されました。

次の例では CS1004 エラーが生成されます。

```
// CS1004.cs
public class clx
{
   public public static void Main()   // CS1004, two public keywords
   {
   }
}
```

コンパイラ エラー CS1007

エラー メッセージ
プロパティ アクセサは既に定義されています。

プロパティを宣言するときは、そのアクセサ メソッドも宣言する必要があります。ただし、プロパティには複数の **get** アクセサ メソッドまたは **set** アクセサ メソッドを使用できません。

使用例
次の例では CS1007 エラーが生成されます。

```
// CS1007.cs
public class clx
{
    public int MyProperty
    {
        get
        {
            return 0;
        }
        get   // CS1007, this is the second get method
        {
            return 0;
        }
    }

    public static void Main() {}
}
```

コンパイラ エラー CS1008

エラー メッセージ
byte、sbyte、short、ushort、int、uint、long または ulong のいずれかの型を使用してください。

列挙型などの特定のデータ型は、指定された型のデータを保持するようにしか宣言できません。

次の例では CS1008 エラーが生成されます。

```
// CS1008.cs
abstract public class clx
{
   enum splitch : char   // CS1008, char not valid type for enums
   {
      x, y, z
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1009

エラー メッセージ
認識できないエスケープ シーケンスです。

文字列で、円記号 (\) の後ろに予測不可能な文字がありました。コンパイラは、有効なエスケープ文字を予期しています。詳細については、「文字のエスケープ」を参照してください。

次の例では CS1009 エラーが生成されます。

```
// CS1009-a.cs
class MyClass
{
   static void Main()
   {
      string a = "\m";   // CS1009
      // try the following line instead
      // string a = "\t";
   }
}
```

このエラーに共通するのは、たとえば、次のように円記号 (\) をファイル名に使用していることです。

```
string filename = "c:\myFolder\myFile.txt";
```

このエラーを解決するには、次のコード例のように "\\" を使用するか、@ と二重引用符で囲まれたリテラル文字列を使用してください。

```
// CS1009-b.cs
class MyClass
{
   static void Main()
   {
      string filename = "c:\myFolder\myFile.txt";   // CS1009
      // try the one of the following lines instead
      // string filename = "c:\\myFolder\\myFile.txt";
      // string filename = @"c:\myFolder\myFile.txt";
   }
}
```

コンパイラ エラー CS1010

エラー メッセージ
定数が 2 行目に続いています。

文字列が正しく区切られていません。

次の例では CS1010 エラーが生成されます。

```
// CS1010.cs
class Sample
{
   static void Main()
   {
      string a = "Hello World;   // CS1010, add end quote
   }
}
```

コンパイラ エラー CS1011

エラー メッセージ
空の文字リテラルです。

char が宣言され、null に初期化されました。**char** の初期化では、文字を指定する必要があります。

次の例では CS1011 エラーが生成されます。

```
// CS1011.cs
class Sample
{
   public char CharField = '';   // CS1011
}
```

コンパイラ エラー CS1012

エラー メッセージ
文字リテラルに文字が多すぎます。

複数の文字を使用して char 定数の初期化を試みました。

CS1012 エラーは、データ バインディングの実行時にも発生する場合があります。たとえば、次のコード行はエラーになります。

```
<%# DataBinder.Eval(Container.DataItem, 'doctitle') %>
```

代わりに、次のコード行を試してください。

```
<%# DataBinder.Eval(Container.DataItem, "doctitle") %>
```

次の例では CS1012 エラーが生成されます。

```
// CS1012.cs
class Sample
{
   static void Main()
   {
      char a = 'xx';   // CS1012
      char a2 = 'x';   // OK
      System.Console.WriteLine(a2);
   }
}
```

コンパイラ エラー CS1013

エラー メッセージ
無効な数字です。

不正な形式の数値が検出されました。整数型の詳細については、「整数型の一覧表 (C# リファレンス)」を参照してください。

使用例
次の例では CS1013 エラーが生成されます。

```
// CS1013.cs
class Sample
{
    static void Main()
    {
        int i = 0x;   // CS1013
    }
}
```

コンパイラ エラー CS1014

エラー メッセージ
get または set アクセサが必要です。

プロパティの宣言でメソッドの宣言が見つかりました。プロパティで宣言できるのは **get** メソッドおよび **set** メソッドのみです。

プロパティの詳細については、「プロパティの使用 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS1014 エラーが生成されます。

```
// CS1014.cs
// compile with: /target:library
class Sample
{
   public int TestProperty
   {
      get
      {
         return 0;
      }
      int z;   // CS1014  not get or set
   }
}
```

コンパイラ エラー CS1015

エラー メッセージ
オブジェクト、文字列またはクラスが必要です。

定義済みのデータ型を catch ブロックに渡そうとしました。**catch** ブロックに渡すことができるのは、System.Exception から派生するデータ型だけです。例外の詳細については、「例外処理ステートメント (C# リファレンス)」を参照してください。

使用例

次の例では CS1015 エラーが生成されます。

```
// CS1015.cs
class Sample
{
    static void Main()
    {
        try
        {
        }
        catch(int)   // CS1015, int is not derived from System.Exception
        {
        }
    }
}
```

コンパイラ エラー CS1016

エラー メッセージ
名前付き属性引数が必要です。

名前のない属性引数は、名前付き引数の前に使用する必要があります。

使用例
次の例では CS1016 エラーが生成されます。

```
// CS1016.cs
using System;

[AttributeUsage(AttributeTargets.Class)]
public class HelpAttribute : Attribute
{
    public HelpAttribute(string url)   // url is a positional parameter
    {
        m_url = url;
    }

    public string Topic = null;      // Topic is a named parameter
    private string m_url = null;
}

[HelpAttribute(Topic="Samples", "http://intranet/inhouse")]   // CS1016
// try the following line instead
//[HelpAttribute("http://intranet/inhouse", Topic="Samples")]
public class MainClass
{
    public static void Main ()
    {
    }
}
```

コンパイラ エラー CS1017

エラー メッセージ
Try ステートメントに既に空の catch ブロックがあります。

パラメータを一切受け取らない **catch** ブロックは、一連の **catch** ブロックの最後に配置する必要があります。例外の詳細については、「例外処理ステートメント (C# リファレンス)」を参照してください。

使用例
次の例では CS1017 エラーが生成されます。

```
// CS1017.cs
using System;

namespace x
{
    public class b : Exception
    {
    }

    public class a
    {
        public static void Main()
        {
            try
            {
            }

            catch   // CS1017, must be last catch
            {
            }

            catch(b)
            {
                throw;
            }
        }
    }
}
```

コンパイラ エラー CS1018

エラー メッセージ
キーワード 'this' または 'base' が必要です。

不完全なコンストラクタ宣言が検出されました。

使用例
次の例では CS1018 エラーが生成されます。このサンプルでは、いくつかの解決方法も示されています。

```
// CS1018.cs
public class C
{
}

public class a : C
{
    public a(int i)
    {
    }

    public a () :   // CS1018
    // possible resolutions:
    // public a () resolves by removing the colon
    // public a () : base() calls C's default constructor
    // public a () : this(1) calls the assignment constructor of class a
    {
    }

    public static int Main()
    {
        return 1;
    }
}
```

コンパイラ エラー CS1019

エラー メッセージ
オーバーロード可能な単項演算子が必要です。

別のクラスの値を返す単項演算子があります。この変換を行うには、**implicit** または **explicit** のキャストが必要です。

次の例では CS1019 エラーが生成されます。

```
// CS1019.cs
public class ii
{
   int i
   {
      get
      {
         return 0;
      }
   }
}

public class a
{
   public static a operator ii(a aa)   // CS1019
   // try the following line instead
   //public static a operator ++(a aa)
   {
      return new a();
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1020

エラー メッセージ
オーバーロード可能な 2 項演算子が必要です。

演算子のオーバーロードの定義を試みましたが、演算子が 2 つのパラメータを受け取る 2 項演算子ではありませんでした。

次の例では CS1020 エラーが生成されます。

```
// CS1020.cs
public class iii
{
   public static int operator ++(iii aa, int bb)   // CS1020, change ++ to +
   {
      return 0;
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1021

エラー メッセージ
整数定数が大きすぎます。

符号なし整数の最大値よりも大きい値が、整数型に代入されました。

次の例では CS1021 エラーが生成されます。

```
// CS1021.cs
enum F : int
{
   a=(int)259000000000000000000   // CS1021
}

public class clx
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1022

エラー メッセージ
型、名前空間の定義、またはファイルの終わりが必要です。

ソース コード ファイルに、対応する中かっこ ({}) のセットがありません。

次の例では CS1022 エラーが生成されます。

```
// CS1022.cs
namespace x
{
}
}   // CS1022
```

コンパイラ エラー CS1023

エラー メッセージ
埋め込みステートメントを宣言やラベル付きのステートメントにすることはできません。

if ステートメントに続くステートメントなどの埋め込みステートメントには、宣言もラベル付きステートメントも組み込むことができません。

次の例では CS1023 エラーが 2 回生成されます。

```
// CS1023.cs
public class a
{
   public static void Main()
   {
      if (1)
         int i;      // CS1023, declaration is not valid here

      if (1)
         xx : i++;   // CS1023, labeled statement is not valid here
   }
}
```

コンパイラ エラー CS1024

エラー メッセージ
プリプロセッサ ディレクティブが必要です。

行の先頭がシャープ記号 (#) ですが、後続の文字列が有効なプリプロセッサ ディレクティブではありませんでした。

次の例では CS1024 エラーが生成されます。

```
// CS1024.cs
#import System   // CS1024
```

コンパイラ エラー CS1025

エラー メッセージ
単一行コメントか行の終わりが必要です。

プリプロセッサ ディレクティブがある行では、複数行のコメントを使用できません。

次の例では CS1025 エラーが生成されます。

```
#if true /* hello
*/   // CS1025
#endif   // this is a good comment
```

次のように、無効なプリプロセッサ ディレクティブを使用した場合にも、CS1025 エラーが発生することがあります。

```
// CS1025.cs
#define a

class Sample
{
   static void Main()
   {
      #if a 1   // CS1025, invalid syntax
         System.Console.WriteLine("Hello, World!");
      #endif
   }
}
```

コンパイラ エラー CS1026

エラー メッセージ
) が必要です。

不完全なステートメントが見つかりました。

このエラーに共通するのは、ASP.NET ページのインライン式の中に、式ではなくステートメントを使用していることです。たとえば、次の式は間違っています。

```
<%=new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days;%>
```

次の式が正しい式です。

```
<%=new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days %>
```

この式は、次のように解釈されます。

```
<% Response.Write(new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days);
%>
```

次の例では、CS1026 エラーが生成されます。

```
// CS1026.cs
#if (a == b   // CS1026, add closing )
#endif

class x
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1027

エラー メッセージ
#endif ディレクティブが必要です。

指定された **#if** ディレクティブに対応する **#endif** プリプロセッサ ディレクティブが見つかりませんでした。または、見つかった **#endregion** ディレクティブに対応する **#region** ディレクティブが、**#if** ブロック内に見つからなかった可能性があります。

次の例では CS1027 エラーが生成されます。

```
// CS1027.cs
#if true   // CS1027, uncomment next line to resolve
// #endif

namespace x
{
   public class clx
   {
      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS1028

エラー メッセージ
不適切なプリプロセッサ ディレクティブです。

予期しないプリプロセッサ ディレクティブが見つかりました。

たとえば、**#endif** が見つかりましたが、その前に **#if** がありません。

次の例では CS1028 エラーが生成されます。

```
// CS1028.cs
#endif   // CS1028, no matching #if
namespace x
{
   public class clx
   {
      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS1029

エラー メッセージ
#error: 'text'

#error ディレクティブで定義されたエラーのテキストを表示します。

次の例では、ユーザー定義のエラーの作成方法を示しています。

```
// CS1029.cs
class Sample
{
   static void Main()
   {
      #error Let's give an error here   // CS1029
   }
}
```

コンパイラ エラー CS1031

エラー メッセージ
型が必要です。

型パラメータが必要です。

使用例
次の例では CS1031 エラーが生成されます。

```
// CS1031.cs
namespace x
{
    public class ii
    {
    }

    public class a
    {
        public static operator +(a aa)   // CS1031
        // try the following line instead
        // public static ii operator +(a aa)
        {
            return new ii();
        }

        public static void Main()
        {
            e = new base;   // CS1031, not a type
            e = new this;   // CS1031, not a type
            e = new ();     // CS1031, not a type
        }
    }
}
```

コンパイラ エラー CS1032

エラー メッセージ
ファイルの最初のトークンの後でプリプロセッサのシンボルの定義または定義の解除を行えませんでした。

プログラムの先頭にプリプロセッサ ディレクティブの **#define** および **#undef** を付けてから、名前空間の宣言などのキーワードを使用する必要があります。

次の例では CS1032 エラーが生成されます。

```
// CS1032.cs
namespace x
{
   public class clx
   {
      #define a   // CS1032, put before namespace
      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS1033

エラー メッセージ
コンパイラの制限を超えています : ファイルは 'number' 行を超えることはできません。

ソース コード ファイルの行数が、コンパイラで処理できる行数の最大値を超えています。このエラーを解決するには、元のファイルから 2 つ以上のソース コード ファイルを作成します。最大行数は、268,435,454 行です。/debug オプションを指定した場合、行数が 16,707,566 を超えると、デバッグ情報が破損します。

コンパイラ エラー CS1034

エラー メッセージ
コンパイラの制限を超えています : 行の文字数は 'number' までです。

1 行に使用できる最大文字数は 16,777,214 文字です。

コンパイラ エラー CS1035

エラー メッセージ
ファイルの終わりが見つかりました。'*/' が必要です。

開始コメントの区切り符合と終了の区切り符号が対応していません。

次の例では CS1035 エラーが生成されます。

```
// CS1035.cs
public class a
{
   public static void Main()
   {
   }
}
/*   // CS1035, needs closing comment
```

コンパイラ エラー CS1036

エラー メッセージ
(または . が必要です。

/doc コメントの XML の形式が不正です。

コンパイラ エラー CS1037

エラー メッセージ
オーバーロード可能な演算子が必要です。

/doc でコメントを指定するときに、無効なリンクが検出されました。

コンパイラ エラー CS1038

エラー メッセージ
#endregion ディレクティブが必要です。

#region ディレクティブに対応する #endregion ディレクティブがありませんでした。

次の例では CS1038 エラーが生成されます。

```
// CS1038.cs
#region testing

public class clx
{
   public static void Main()
   {
   }
}
// CS1038
// uncomment the next line to resolve
// #endregion
```

コンパイラ エラー CS1039

エラー メッセージ
未終了の文字列です

不正な形式のリテラル文字列が検出されました。

使用例
次の例では、CS1039 エラーが生成されます。このエラーを解決するには、終了の引用符を追加します。

```
// CS1039.cs
public class MyClass
{
    public static void Main()
    {
        string b = @"hello, world;   // CS1039
    }
}
```

コンパイラ エラー CS1040

エラー メッセージ
プリプロセッサ ディレクティブは行でスペース以外の最初の文字でなければなりません。

行で見つかったプリプロセッサ ディレクティブが、その行の最初のトークンではありませんでした。ディレクティブは、指定した行の最初のトークンであることが必要です。

次の例では CS1040 エラーが生成されます。

```
// CS1040.cs
/* Define a symbol, X */ #define X   // CS1040

// try the following two lines instead
// /* Define a symbol, X */
// #define X

public class MyClass
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1041

エラー メッセージ
識別子が必要です。キー ワードは 'keyword' です。

識別子を置く必要がある場所に C# 言語の予約語が見つかりました。キーワードをユーザー指定の識別子で置き換えてください。

使用例
次の例では CS1041 エラーが生成されます。

```
// CS1041a.cs
class MyClass
{
    public void f(int long)   // CS1041
    // Try the following instead:
    // public void f(int i)
    {
    }

    public static void Main()
    {
    }
}
```

予約語のセットが異なる別のプログラミング言語からインポートする場合は、次のサンプルで示すように、@ プリフィックスを使用して、予約済みの識別子を変更できます。

@ プリフィックスを持つ識別子は、逐語的識別子と呼ばれます。

```
// CS1041b.cs
class MyClass
{
    public void f(int long)   // CS1041
    // Try the following instead:
    // public void f(int @long)
    {
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS1043

エラー メッセージ
{ または ; が必要です。

プロパティのアクセサが正しく宣言されませんでした。詳細については、「プロパティの使用 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS1043 エラーが生成されます。

```
// CS1043.cs
// compile with: /target:library
public class MyClass
{
   public int DoSomething
   {
      get return 1;   // CS1043
      set {}
   }

   // OK
   public int DoSomething2
   {
      get { return 1;}
   }
}
```

コンパイラ エラー CS1044

エラー メッセージ
for、using、fixed または declaration ステートメントに 1 つ以上の型を使用することはできません。

無効なステートメントが見つかりました。

次の例では CS1044 エラーが生成されます。

```
// CS1044.cs
using System;

public class MyClass : IDisposable
{
   public void Dispose()
   {
      Console.WriteLine("Res1.Dispose()");
   }

   public static void Main()
   {
      using (MyClass mc1 = new MyClass(),
             MyClass mc2 = new MyClass())   // CS1044, remove an instantiation
      {
      }
   }
}
```

コンパイラ エラー CS1055

エラー メッセージ
add または remove アクセサが必要です。

イベントがフィールドとして宣言されない場合は、**add** アクセサと **remove** アクセサの両方を定義する必要があります。

次の例では CS1055 エラーが生成されます。

```
// CS1055.cs
delegate void del();
class Test
{
   public event del MyEvent
   {
      int i;   // CS1055
      // uncomment accessors and delete previous line to resolve
      // add
      // {
      //    MyEvent += value;
      // }
      // remove
      // {
      //    MyEvent -= value;
      // }
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1056

エラー メッセージ
予期しない文字 '文字' です。

予期しない文字が見つかったため、C# コンパイラが現在処理中のトークンを識別できません。たとえば、識別子の処理中にユーロ文字が見つかった場合にこの問題が発生します。ユーロ文字は文字列の中でのみ有効であり、文字列処理以外では、コンパイラが識別子を正しく認識できません。

コンパイラ エラー CS1501

エラー メッセージ
引数を 'number' 個指定できる、メソッド 'method' のオーバーロードはありません。

クラスのメソッドの呼び出しを試みましたが、必要な数の引数を受け取るメソッドの形式を指定していません。

参照先アセンブリのクラスでメソッドを呼び出した場合に、そのメソッドのパラメータに既定値があると、CS1501 エラーが発生することがあります。既定値を持つパラメータを受け取るメソッドは、C# では作成されませんが、ランタイムを対象とする別の言語では作成される場合があります。参照先アセンブリのメソッドのパラメータに既定値がある場合でも、そのメソッドを呼び出してすべてのパラメータを明示的に渡す必要があります。

デリゲートにメモリを割り当てるときに CS1501 エラーが発生することがあります。以下の 2 つ目の例のように、デリゲートが表すメソッドの名前も指定する必要があります。

基本クラスに既定のコンストラクタが存在せず、代わりに、引数を少なくとも 1 つ受け取る、既定以外のコンストラクタが存在するとき、その基本クラスの派生クラスをインスタンス化しようとした場合にも、CS1501 エラーが発生します。このエラーを解決するには、基本クラスに既定のコンストラクタを追加するか、base (C# リファレンス) キーワードを使用して、基本クラスの適切なコンストラクタを呼び出します (以下の 3 つ目の例を参照)。詳細については、「コンストラクタの使用 (C# プログラミング ガイド)」を参照してください。

使用例

次の例では、CS1501 エラーが生成されます。

```
// CS1501.cs
public class a
{
    // declare the following constructor to resolve this CS1501
    /*
    public a(int i)
    {
    }
    */

    public static void Main()
    {
        a aa = new a(2);   // CS1501
    }
}
```

次の例では、CS1501 エラーが生成されます。

```
// CS1501a.cs
using System;

delegate string func(int i);   // declare the delegate

class a
{
    static func dt;   // class member-field of the declared delegate type

    public static void Main()
    {
        dt = new func();   // CS1501
        // try the following line instead
        // dt = new func(z);   // this works
        x(dt);
    }

    public static string z(int j)
    {
        Console.WriteLine(j);
        return j.ToString();
    }

    public static void x(func hello)
```

```
    {
        hello(8);
    }
}
```

次の例では、CS1501 エラーが生成されます。

```
// CS1501b.cs
class Base
{
    public Base(string s)
    {
    }
}

class Derived : Base
{ // CS1501
    public Derived(string s)
    {
    }

    // Try this instead:
    // public Derived(string s) : base(s)
    // {
    // }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS1502

エラー メッセージ
'declaration' に最も適しているオーバーロード メソッドには無効な引数がいくつか含まれています。

このエラーは、メソッドに渡された引数の型が、そのメソッドのパラメータ型と一致しない場合に発生します。呼び出されたメソッドがオーバーロードされている場合は、オーバーロードされたバージョンのどのシグネチャも、渡された引数の型と一致しません。

この問題を解決するには、以下のいずれかの操作を実行します。

- 渡される引数の型を再度チェックします。渡される引数の型が、呼び出されるメソッドの引数と一致していることを確認してください。
- 必要に応じて、System.Convert クラスを使用し、不一致のパラメータを変換します。
- 必要に応じて、メソッドが使用する型と一致するように不一致パラメータをキャストします。
- 必要に応じて、送信されるパラメータ型と一致するように、メソッドの別のオーバーロードされたバージョンを定義します。

次の例では CS1502 エラーが生成されます。

```
// CS1502.cs
namespace x
{
   public class a
   {
      public a(char i)
      // try the following constructor instead
      // public a(int i)
      {
      }

      public static void Main()
      {
         a aa = new a(2222);   // CS1502
      }
   }
}
```

コンパイラ エラー CS1503

エラー メッセージ
引数 'arg': '型 1' から '型 2' に変換できません。

メソッドの 1 つの引数の型が、クラスをインスタンス化したときに渡された型と一致しません。

このエラーを解決する方法については、「コンパイラ エラー CS1502」を参照してください。

コンパイラ エラー CS1504

エラー メッセージ
ソース ファイル 'file' を開くことができませんでした ('reason')。

コンパイラでソース ファイルのオープンまたは読み込みができませんでした。ファイルが別のアプリケーションによってロックされているか、オペレーティング システムに別の問題がある可能性があります。メッセージには、コンパイラがファイルを開いたり読み込んだりできなかった、オペレーティング システム側の原因が記載されています。

コンパイラ エラー CS1507

エラー メッセージ
モジュールをビルド中にリソース ファイル 'ファイル' にリンクできません。

/target:module と同じコンパイルで /linkresource が使用されましたが、この処理は許可されていません。たとえば、次のオプションでは CS1507 エラーが生成されます。

```
csc /linkresource:rf.resource /target:module in.cs
```

ただし、リソースの埋め込み (/resource) は許可されます。

コンパイラ エラー CS1508

エラー メッセージ
リソース識別子 'identifier' はアセンブリで既に使用されています。

コンパイルで、同じ識別子 (***identifier***) が複数の /resource コンパイラ オプションまたは複数の /linkresource コンパイラ オプションに渡されました。

たとえば、次のオプションでは CS1508 エラーが生成されます。

```
/resource:anyfile.bmp,DuplicatIdent /linkresource:a.bmp,DuplicatIdent
```

コンパイラ エラー CS1509

エラー メッセージ
参照したファイル 'file' はアセンブリではありません。/addmodule オプションを使用してください。

/target:module (アセンブリ マニフェストを持たない) を使用したコンパイルで生成された出力ファイル (出力ファイル 1) が、/reference に対して指定されました。したがって、現在のプログラムのアセンブリには、アセンブリが追加されるのではなく、出力ファイル 1 に含まれるメタデータ情報が追加されます。

コンパイラ エラー CS1510

エラー メッセージ
ref または out 引数は、割り当て可能な変数でなければなりません。

メソッドの呼び出しで ref パラメータとして渡すことができるのは変数だけです。**ref** の値の指定はポインタを渡すことに相当します。

使用例
次の例では CS1510 エラーが生成されます。

```
// CS1510.cs
public class C
{
   public static int j = 0;

   public static void mf(ref int j)
   {
      j++;
   }

   public static void Main ()
   {
      mf (ref 2);   // CS1510, can't pass a number as a ref parameter
      // try the following to resolve the error
      // mf (ref j);
   }
}
```

コンパイラ エラー CS1511

エラー メッセージ
キーワード 'base' は静的メソッドでは使用できません。

base キーワードが static なメソッドで使用されています。**base** キーワードで基本クラスを呼び出すことができるのは、インスタンスのコンストラクタ内、インスタンスのメソッド内、またはインスタンスのアクセサ内だけです。

使用例

次の例では CS1511 エラーが生成されます。

```
// CS1511.cs
// compile with: /target:library
public class A
{
   public int j = 0;
}

class C : A
{
   public void Method()
   {
      base.j = 3;   // base allowed here
   }

   public static int StaticMethod()
   {
      base.j = 3;   // CS1511
      return 1;
   }
}
```

コンパイラ エラー CS1512

エラー メッセージ
キーワード 'base' は現在のコンテキストでは使用できません。

base キーワードが、メソッド、プロパティ、またはコンストラクタの外側で使用されました。

次の例では、CS1512 エラーが生成されます。

```
// CS1512.cs
using System;

class Base {}

class CMyClass : Base
{
    private String xx = base.ToString();   // CS1512
    // Try putting this initialization in the constructor instead:
    // public CMyClass()
    // {
    //    xx = base.ToString();
    // }

    public static void Main()
    {
        CMyClass z = new CMyClass();
    }
}
```

コンパイラ エラー CS1513

エラー メッセージ
} が必要です。

右中かっこ (}) が必要ですが、見つかりませんでした。

次の例では CS1513 エラーが生成されます。

```
// CS1513
namespace y   // CS1513, no close curly brace
{
   class x
   {
      public static void Main()
      {
      }
   }
```

コンパイラ エラー CS1514

エラー メッセージ
{ が必要です。

左中かっこ ({) が必要ですが、見つかりませんでした。

次の例では CS1514 エラーが生成されます。

```
// CS1514.cs
namespace x
// CS1514, no open curly brace
```

コンパイラ エラー CS1515

エラー メッセージ
in が必要です。

foreach、in ステートメントに、"in" 部分がありません。

使用例
次の例では CS1515 エラーが生成されます。

```
using System;

class Driver
{
   static void Main()
   {
      int[] arr = new int[] {1, 2, 3};

      // try the following line instead
      // foreach (int x in arr)
      foreach (int x arr)      // CS1515, "in" is missing
      {
         Console.WriteLine(x);
      }
   }
}
```

コンパイラ エラー CS1517

エラー メッセージ
無効なプリプロセッサの式です。

無効なプリプロセッサ式が検出されました。

詳細については、「C# プリプロセッサ ディレクティブ」を参照してください。

有効なプリプロセッサ式と無効なプリプロセッサ式の例を次に示します。

```
// CS1517.cs
#if symbol      // OK
#endif
#if !symbol     // OK
#endif
#if (symbol)    // OK
#endif
#if true        // OK
#endif
#if false       // OK
#endif
#if 1           // CS1517
#endif
#if ~symbol     // CS1517
#endif
#if *           // CS1517
#endif

class x
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1518

エラー メッセージ
クラス、デリゲート、列挙型、インターフェイス または構造体が必要です。

名前空間でサポートされない宣言が見つかりました。名前空間の内部で使用できるのは、クラス、構造体、列挙体、インターフェイス、名前空間、およびデリゲートだけです。

使用例
次の例では CS1518 エラーが生成されます。

```
// CS1518.cs
namespace x
{
   sealed class c1 {};      // OK
   namespace f2 {};         // OK
   sealed f3 {};            // CS1518
}
```

コンパイラ エラー CS1519

エラー メッセージ
無効なトークン 'token' がクラス、構造体またはインターフェイスのメンバ宣言で使用されています。

型より前に、無効な修飾子を含むクラス、構造体、またはインターフェイス メンバ宣言があると、このエラーが生成されます。このエラーを解決するには、無効な修飾子を削除します。

次の例では CS1519 エラーが生成されます。

```
// CS1519.cs
public class IMyInterface
{
   checked void f4();   // CS1519
   // Remove "checked" from the line above.
   lock void f5();      // CS1519
   // Remove "lock" from the line above
   namespace;           // CS1519
   // The line above should be removed entirely.
   int i;               // OK
}
```

コンパイラ エラー CS1520

エラー メッセージ
クラス、構造体またはインターフェイスのメソッドには戻り値の型が必要です。

クラス、構造体、またはインターフェイスで宣言されるメソッドには、明示的な戻り値の型が必要です。

次の例では CS1520 エラーが生成されます。

```
// CS1520a.cs
public class x
{
   f7()   // CS1520, needs return type
   // try the following definition
   // void f7()
   {
   }

   public static void Main()
   {
   }
}
```

また、このエラーは、次のサンプルに示すように、コンストラクタの名前の大文字と小文字がクラスまたは構造体宣言の名前と異なる場合に発生することがあります。

```
// CS1520b.cs
public class Class1
{
   public class1()   // CS1520, incorrect case
   {
   }
   static void Main()
   {
   }
}
```

コンパイラ エラー CS1521

エラー メッセージ
無効なベース型です。

base クラスが不正な形式で指定されています。

次の例では CS1521 エラーが生成されます。

```
// CS1521.cs
class CMyClass
{
   public static void Main()
   {
   }
}

class CMyClass1 : CMyClass     // OK
{
}

class CMyClass2 : CMyClass[]   // CS1521
{
}

class CMyClass3 : CMyClass*    // CS1521
{
}
```

コンパイラ エラー CS1524

エラー メッセージ
catch または finally が必要です。

try ブロックの後には、**catch** ブロックまたは **finally** ブロックのみ記述できます。

例外の詳細については、「例外処理ステートメント (C# リファレンス)」を参照してください。

使用例
次の例では CS1524 エラーが生成されます。

```
// CS1524.cs
class x
{
    public static void Main()
    {
        try
        {
            // Code here
        }
        catch
        {
        }
        try
        {
            // Code here
        }
        finally
        {
        }
        try
        {
            // Code here
        }
    }     // CS1524, missing catch or finally
}
```

コンパイラ エラー CS1525

エラー メッセージ
'character' は無効です。

式の中で無効な文字が検出されました。

次の例では CS1525 エラーが生成されます。

```
// CS1525.cs
class x
{
   public static void Main()
   {
      int i = 0;
      i = i +   // OK - identifier
      'c' +     // OK - character
      (5) +     // OK - parenthesis
      [ +       // CS1525, operator not a valid expression element
      throw +   // CS1525, keyword not allowed in expression
      void;     // CS1525, void not allowed in expression
   }
}
```

次の例で示すように、空のラベルが原因で CS1525 エラーが発生することもあります。

```
// CS1525b.cs
using System;
public class MyClass
{
   public static void Main()
   {
      goto FoundIt;
      FoundIt:      // CS1525
      // Uncomment the following line to resolve:
      // System.Console.Write("Hello");
   }
}
```

コンパイラ エラー CS1526

エラー メッセージ
新しい式は型の後に丸かっこ ()、または、角かっこ [] を必要とします。

オブジェクトにメモリを動的に割り当てるときに使用する new 演算子が、正しく指定されていません。

使用例

new を使用して配列とオブジェクトに領域を割り当てる方法を次に示します。

```
// CS1526.cs
public class y
{
   public static int i = 0;
   public int myi = 0;
}

public class z
{
   public static void Main()
   {
      y py = new y;   // CS1526
      y[] aoys = new y[10];   // Array of Ys

      for (int i = 0; i < aoys.Length; i++)
         aoys[i] = new y();   // an object of type y
   }
}
```

コンパイラ エラー CS1527

エラー メッセージ
名前空間の要素は明示的に private、protected、または protected internal に宣言することはできません。

名前空間の型宣言には、public アクセスまたは internal アクセスがあります。アクセシビリティが指定されていない場合は、**internal** が既定値です。

次の例では CS1527 エラーが生成されます。

```
// CS1527.cs
namespace Sample
{
   private class C1 {};             // CS1527
   protected class C2 {};           // CS1527
   protected internal class C3 {};  // CS1527
}
```

コンパイラ エラー CS1528

エラー メッセージ
; または = を指定してください (宣言の中にコンストラクタ引数は指定できません)。

クラスに対する参照が、クラスに対するオブジェクトが生成されているときのように行われました。たとえば、変数をコンストラクタに渡そうとしました。new 演算子を使用して、クラスのオブジェクトを作成してください。

次の例では CS1528 エラーが生成されます。

```
// CS1528.cs
using System;

public class B
{
   public B(int i)
   {
      _i = i;
   }

   public void PrintB()
   {
      Console.WriteLine(_i);
   }

   private int _i;
}

public class mine
{
   public static void Main()
   {
      B b(3);   // CS1528, reference is not an object
      // try one of the following
      // B b;
      // or
      // B bb = new B(3);
      // bb.PrintB();
   }
}
```

コンパイラ エラー CS1529

エラー メッセージ
using 句は、extern エイリアス宣言以外の、すべての名前空間要素の前に使用しなければなりません

using 句は、名前空間で最初に指定する必要があります。

使用例
次の例では CS1529 エラーが生成されます。

```
// CS1529.cs
namespace X
{
    namespace Subspace
    {
        using Microsoft;

        class SomeClass
        {
        };

        using Microsoft;      // CS1529, place before class definition
    }

    using System.Reflection;  // CS1529, place before namespace 'Subspace'
}

using System;                 // CS1529, place at the beginning of the file
```

コンパイラ エラー CS1530

エラー メッセージ
キーワード new は名前空間の要素に対して使用できません。

名前空間内の構成要素では、new キーワードを指定する必要はありません。

次の例では CS1530 エラーが生成されます。

```
// CS1530.cs
namespace a
{
   new class i   // CS1530
   {
   }

   // try the following instead
   class ii
   {
      public static void Main()
      {
      }
   }
}
```

コンパイラ エラー CS1534

エラー メッセージ
オーバーロードされた 2 項演算子 '演算子' に指定できるパラメータ数は 2 です。

二項のオーバーロード可能な演算子の定義では、2 つのパラメータを受け取る必要があります。

次の例では CS1534 エラーが生成されます。

```
// CS1534.cs
class MyClass
{
   public static MyClass operator - (MyClass MC1, MyClass MC2, MyClass MC3)   // CS1534
   // try the following line instead
   // public static MyClass operator - (MyClass MC1, MyClass MC2)
   {
      return new MyClass();
   }

   public static int Main()
   {
      return 1;
   }
}
```

コンパイラ エラー CS1535

エラー メッセージ
オーバーロードされた単項演算子 'operator' に指定できるパラメータ数は 1 です。

単項のオーバーロード可能な演算子の定義では、1 つのパラメータを受け取る必要があります。

使用例
次の例では CS1535 エラーが生成されます。

```
// CS1535.cs
class MyClass
{
    // uncomment the method parameter to resolve CS1535
    public static MyClass operator ++ (/*MyClass MC1*/)   // CS1535
    {
        return new MyClass();
    }

    public static int Main()
    {
        return 1;
    }
}
```

コンパイラ エラー CS1536

エラー メッセージ
void は無効なパラメータ型です。

void ポインタ以外の **void** パラメータを指定することは無効であり、また必要でもありません。

次の例では CS1536 エラーが生成されます。

```
// CS1536.cs
class a
{
   public static int x( void )   // CS1536
   // try the following line instead
   // public static int x()
   {
      return 0;
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1537

エラー メッセージ
using エイリアス 'alias' は以前にこの名前空間で使用されています。

名前空間のエイリアスとして、1 つのシンボルを 2 回定義しています。シンボルを定義できるのは 1 回だけです。

次の例では CS1537 エラーが生成されます。

```
// CS1537.cs
namespace x
{
   using System;
   using Object = System.Object;
   using Object = System.Object;   // CS1537, delete this line to resolve
   using System = System;
}
```

コンパイラ エラー CS1540

エラー メッセージ
'type1' 型の修飾子を通してプロテクト メンバ 'member' にアクセスすることはできません。修飾子は 'type2' 型、またはそれから派生したものでなければなりません。

派生クラスはその基本クラスのプロテクト メンバにアクセスできますが、基本クラスのインスタンスを通じてアクセスすることはできません。

次の例では CS1540 エラーが生成されます。

```
// CS1540.cs
public class Base
{
   protected void func()
   {
   }
}

public class Derived : Base
{
   public static void test(Base anotherInstance)
   // the method declaration could be changed as follows
   // public static void test(Derived anotherInstance)
   {
      anotherInstance.func();   // CS1540
   }
}

public class Tester : Derived
{
   public static void Main()
   {
      Base pBase = new Base();
      // the allocation could be changed as follows
      // Derived pBase = new Derived();
      test(pBase);
   }
}
```

コンパイラ エラー CS1541

エラー メッセージ
無効な参照オプション: 'シンボル' - ディレクトリを参照できません

特定のファイルではなくディレクトリの指定が検出されました。たとえば、/reference コンパイラ オプションを使用する場合は、ファイルを指定する必要があります。ディレクトリは指定できません。

たとえば、コンパイラ オプションに「`/reference:c:\`」のように指定した場合、CS1541 エラーが生成されます。

コンパイラ エラー CS1542

エラー メッセージ
'dll' は既にアセンブリなのでこのアセンブリに加えることはできません。代わりに /R オプションを使ってください。

/addmodule コンパイラ オプションを使用して参照されたファイルは、/target:module で作成されていません。/reference を使用してこのコンパイルのファイルを参照してください。

コンパイラ エラー CS1545

エラー メッセージ
プロパティ、インデクサまたはイベント 'プロパティ' は、この言語でサポートされていません。アクセサ メソッドの 'set アクセサ' または 'get アクセサ' を直接呼び出してください。

既定のインデクサを持つオブジェクトを処理して、インデックス付きの構文の使用を試みました。このエラーを解決するには、プロパティのアクセサ メソッド (**get** または **set**) を呼び出します。

使用例

```
// CPP1545.cpp
// compile with: /clr /LD
// a Visual C++ program
using namespace System;
public ref struct Employee {
   Employee( String^ s, int d ) {}

   property String^ name {
      String^ get() {
         return nullptr;
      }
   }
};

public ref struct Manager {
   property Employee^ Report [String^] {
      Employee^ get(String^ s) {
         return nullptr;
      }

      void set(String^ s, Employee^ e) {}
   }
};
```

次の例では CS1545 エラーが生成されます。

```
// CS1545.cs
// compile with: /r:CPP1545.dll

class x {
   public static void Main() {
      Manager Ed = new Manager();
      Employee Bob = new Employee("Bob Smith", 12);
      Ed.Report( Bob.name ) = Bob;   // CS1545
      Ed.set_Report( Bob.name, Bob);   // OK
   }
}
```

コンパイラ エラー CS1546

エラー メッセージ
プロパティ、インデクサまたはイベント 'property' はこの言語でサポートされていません。アクセサ メソッドの 'accessor' を直接呼び出してください。

既定のインデックス付きプロパティを持つオブジェクトを処理して、インデックス付きの構文の使用を試みました。このエラーを解決するには、プロパティのアクセサ メソッドを呼び出します。インデクサおよびプロパティの詳細については、「インデクサ (C# プログラミング ガイド)」および「プロパティの定義と使用」を参照してください。

次の例では CS1546 エラーが生成されます。

使用例
このコードは、.dll にコンパイルされる .cpp ファイルと、その .dll を使用する .cs ファイルから構成されます。次のコードは、.cs ファイルのコードからアクセスされるプロパティが定義された .dll ファイル用のコードです。

```
// CPP1546.cpp
// compile with: /clr /LD
using namespace System;
public ref class MCPP
{
public:
    property int Prop [int,int]
    {
        int get( int i, int b )
        {
            return i;
        }
    }
};
```

次に C# ファイルを示します。

```
// CS1546.cs
// compile with: /r:CPP1546.dll
using System;
public class Test
{
    public static void Main()
    {
        int i = 0;
        MCPP mcpp = new MCPP();
        i = mcpp.Prop(1,1); // CS1546
        // Try the following line instead:
        // i = mcpp.get_Prop(1,1);
    }
}
```

コンパイラ エラー CS1547

エラー メッセージ
キーワード void はこのコンテキストで使用できません。

void キーワードの使用が無効です。

次の例では CS1547 エラーが生成されます。

```
// CS1547.cs
public class MyClass
{
   void BadMethod()
   {
      void i;   // CS1547, cannot have variables of type void
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1548

エラー メッセージ
アセンブリ 'assembly' を署名しているときに暗号に失敗しました -- 'reason'

アセンブリの署名に失敗すると、CS1548 エラーが発生します。これは、通常、無効なキー ファイル名、無効なキー ファイル パス、または破損したキー ファイルなどに起因します。

アセンブリに完全署名するには、公開キーと秘密キーに関する情報を含む有効なキー ファイルを提供する必要があります。アセンブリに遅延署名するには、[遅延署名のみ] チェック ボックスをオンにし、公開キーの情報を含む有効なキー ファイルを提供する必要があります。アセンブリに遅延署名する場合、秘密キーは必要ありません。

詳細については、「方法 : アセンブリに署名する (Visual Studio)」、「/keyfile (厳密名キー ファイルの指定) (C# コンパイラ オプション)」、および「/delaysign (アセンブリの遅延署名) (C# コンパイラ オプション)」を参照してください。

アセンブリの作成時に、C# コンパイラは al.exe というユーティリティを呼び出します。アセンブリの作成時にエラーが発生すると、al.exe によってエラーの原因が報告されます。その場合は、「Al.exe ツールのエラーと警告」を参照し、そのトピックで、コンパイラが 'reason' で報告したテキストを検索してください。

参照
処理手順
方法 : アセンブリに署名する (Visual Studio)

コンパイラ エラー CS1549

エラー メッセージ
適切な暗号化サービスが見つかりません。

RSA 互換の暗号サービス プロバイダがインストールされていません。

コンパイラ エラー CS1551

エラー メッセージ
インデクサには最低パラメータが 1 つ必要です。

引数を受け取らないインデクサが宣言されました。

次の例では CS1551 エラーが生成されます。

```
// CS1551.cs
public class MyClass
{
   int intI;

   int this[]   // CS1551
   // try the following line instead
   // int this[int i]
   {
      get
      {
         return intI;
      }
      set
      {
         intI = value;
      }
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1552

エラー メッセージ
配列型の指定子の角かっこ、[]、は、パラメータ名の前に使用してください。

配列の型指定子が、配列宣言の変数名の後にあります。

使用例
次の例では CS1552 エラーが生成されます。

```
// CS1552.cs
public class C
{
    public static void Main(string args[])   // CS1552
    // try the following line instead
    // public static void Main(string [] args)
    {
    }
}
```

コンパイラ エラー CS1553

エラー メッセージ
不適切な宣言です。代わりに 'modifier 演算子 <dest-type> (...' を使用してください。

演算子の戻り値の型はパラメータ リストの直前に置く必要があり、*modifier* は **implicit** または **explicit** です。

次の例では CS1553 エラーが生成されます。

```
// CS1553.cs
class MyClass
{
   public static int implicit operator (MyClass f)   // CS1553
   // try the following line instead
   // public static implicit operator int (MyClass f)
   {
      return 6;
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1554

エラー メッセージ
不適切な宣言です。代わりに '<type> 演算子 op (...' を使用してください。

ユーザー定義の演算子の戻り値の型は、キーワード operator の前に置く必要があります。

次の例では CS1554 エラーが生成されます。

```
// CS1554.cs
class MyClass
{
   public static operator ++ MyClass (MyClass f)    // CS1554
   // try the following line instead
   // public static MyClass operator ++ (MyClass f)
   {
      return new MyClass ();
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1555

エラー メッセージ
Main メソッドに指定された 'クラス' が見つかりませんでした。

クラスが /main コンパイラ オプションに対して指定されましたが、ソース コードでそのクラス名が見つかりませんでした。

コンパイラ エラー CS1556

エラー メッセージ
Main メソッドに指定された 'construct' は有効なクラスか構造体でなければなりません。

/main コンパイラ オプションに、クラス名ではない識別子が渡されました。

コンパイラ エラー CS1557

エラー メッセージ
'クラス' は別の出力ファイルに含まれているため、Main メソッドに対して使うことはできません。

/main コンパイラ オプションが、複数出力ファイルのコンパイルに含まれる出力ファイルのうちの 1 つに対して指定されました。しかし、/main を指定したコンパイルのソース コードでクラスが見つかりませんでした。クラスは、コンパイルに含まれるほかの出力ファイルのソース コード ファイルで見つかりました。

コンパイラ エラー CS1558

エラー メッセージ
'クラス' は適切なスタティック Main メソッドを含んでいません。

/main コンパイラ オプションで、**Main** メソッドを検索するクラスを指定しました。しかし、Main メソッドが正しく定義されていません。

次の例は、戻り値の型が無効であるため、CS1558 エラーが生成されます。

```
// CS1558.cs
// compile with: /main:MyNamespace.MyClass

namespace MyNamespace
{
   public class MyClass
   {
      public static float Main()
      {
         return 0.0; // CS1558 because the return type is a float.
      }
   }
}
```

コンパイラ エラー CS1559

エラー メッセージ
'object'がインポートされており、プログラム 'program' のエントリ ポイントとして使用できません。

/main コンパイラ オプションに対して無効なクラスを指定しました。このクラスは Main メソッドの場所として使用できません。

コンパイラ エラー CS1560

エラー メッセージ
プリプロセッサ ディレクティブに対して無効なファイル名が指定されました。ファイル名は長すぎるか、または有効なファイル名ではありません。

#line (C# リファレンス) で指定したファイル名が _MAX_PATH (256 文字) を超えたか、または **#line** が見つかった行の文字が 2000 文字を超えました。

使用例
次の例では CS1560 エラーが生成されます。

```
// cs1560.cs
using System;
class MyClass
{
   public static void Main()
   {
      Console.WriteLine("Normal line #1.");
      #line 21 "MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile12345
67890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile
1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890.
txt"   // CS1560
    }
}
```

コンパイラ エラー CS1561

エラー メッセージ
出力ファイル名が長すぎるか、または無効です。

出力ファイル名に 256 文字を超える文字数は使用できません。また、ユーロ記号、疑問符、円記号などの無効な文字を使用することもできません。

コンパイラ エラー CS1562

エラー メッセージ
ソースのない出力には、/out オプションを指定しなければなりません。

コンパイルでは出力ファイルを作成できますが、出力ファイルの名前を示す入力としてのソース コード ファイルがありませんでした。たとえば、メタデータ専用ファイルやリソース専用ファイルをコンパイルしようとすると、このエラーが生成されます。

/out コンパイラ オプションを使用して、出力ファイルの名前を指定してください。

コンパイラ エラー CS1563

エラー メッセージ
出力 'output file' にソース ファイルがありません。

/out コンパイラ オプションが指定されましたが、ソース コード ファイルがありません。**/out** を使用しないか、または **/out** の後にソース コード ファイルを指定する必要があります。

コンパイラ エラー CS1565

エラー メッセージ
競合するオプションが指定されました : Win32 リソース ファイル、Win32 アイコン

同じコンパイルで /win32res コンパイラ オプションと /win32ico コンパイラ オプションの両方を指定することはできません。

コンパイラ エラー CS1566

エラー メッセージ
リソース ファイル 'ファイル' を読み込み中にエラーが発生しました - '理由'

/resource コンパイラ オプションに渡されるファイル名で問題が発生しました。

コンパイラ エラー CS1567

エラー メッセージ
Win32 リソースを生成中にエラーが発生しました : 'file'

コンパイルで /win32icon コンパイラ オプションを使用した場合や /win32res コンパイラ オプションを使用しなかった場合は、リソース情報を含むファイルが生成されますが、ディスク容量が不足しているか、またはその他のエラーが原因でファイルを作成できませんでした。

ファイル生成の問題を解決できない場合は、リソース情報を含むファイルを生成しない /win32res コンパイラ オプションを使用できます。

コンパイラ エラー CS1569

エラー メッセージ
XML ドキュメント ファイル 'ファイル名' を生成中にエラーが発生しました ('理由')。

ファイルに XML ドキュメントを書き込もうとしているときにエラーが発生しました。エラーの理由とファイル名については、エラー メッセージを参照してください。理由としては、" ネットワーク ドライブが見つかりません" や "アクセスが拒否されました" などがあります。また、この理由から、エラーを修正するために必要な措置がわかる場合もあります。たとえば、"アクセスが拒否されました" という種類のエラーの場合、そのファイルに対する書き込み権限を持っているかどうかを確認する必要があります。

使用例

```
// CS1569.cs
// compile with: /doc:CS1569.xml
// post-build command: attrib +r CS1569.xml
class Test
{
   /// <summary>Test.</summary>
   public static void Main() {}
}
```

上の例では、.xml ファイルが生成され、読み取り専用に設定されました。次の例では、同じファイルへの書き込みを試みます。次の例では CS1569 エラーが生成されます。

```
// CS1569_a.cs
// compile with: /doc:CS1569.xml
// CS1569 expected
class Test
{
   /// <summary>Test.</summary>
   public static void Main() {}
}
```

コンパイラ エラー CS1575

エラー メッセージ
stackalloc の式は型の後に角かっこ [] が必要です。

stackalloc を使用して要求する割り当てのサイズは、角かっこで囲む必要があります。

次の例では CS1575 エラーが生成されます。

```
// CS1575.cs
// compile with: /unsafe
public class MyClass
{
   unsafe public static void Main()
   {
      int *p = stackalloc int (30);   // CS1575
      // try the following line instead
      // int *p = stackalloc int [30];
   }
}
```

コンパイラ エラー CS1576

エラー メッセージ
#line ディレクティブの行数が指定されていないか、無効です。

#line ディレクティブに渡される値でエラーが検出されました。

次の例では CS1576 エラーが生成されます。

```
// CS1576.cs
public class MyClass
{
   static void Main()
   {
      #line "abc.sc"         // CS1576
      // try the following line instead
      //#line  101 "abc.sc"
      intt i;  // error will be reported on line 101
   }
}
```

コンパイラ エラー CS1577

エラー メッセージ
アセンブリの生成に失敗しました

コンパイルのアセンブリ生成部分が失敗しました。詳細については、alink ユーティリティ (AI.exe ツールのエラーと警告) のエラーの説明を参照してください。

コンパイラ エラー CS1578

エラー メッセージ
ファイル名、単一行コメント、または行の終わりが必要です。

#line ディレクティブの後は、二重引用符で囲まれたファイル名か、単一行コメントだけが指定できます。

次の例では CS1578 エラーが生成されます。

```
// CS1578.cs
class MyClass
{
   static void Main()
   {
      #line 101 abc.cs   // CS1578
      // try the following line instead
      //#line 101 "abc.cs"
      intt i;          // error will be reported on line 101
   }
}
```

コンパイラ エラー CS1579

エラー メッセージ
foreach ステートメントは、'type2' が '識別子' のパブリック定義を含んでいないため、型 'type1' の変数に対して使用できません。

コレクションを foreach ステートメントで反復処理するには、そのコレクションが次の要件を満たしていることが必要です。

- インターフェイス、クラス、構造体のいずれかであること。
- 型を返すパブリックな GetEnumerator メソッドが存在すること。
- 戻り値の型に Current という名前のパブリック プロパティと、MoveNext という名前のパブリック メソッドが存在すること。
- 詳細については、「方法 : foreach を使用してコレクション クラスにアクセスする (C# プログラミング ガイド)」を参照してください。

使用例
この例では、**public GetEnumerator** メソッドが `MyCollection` にないため、foreach はコレクションを反復処理できません。

次の例では CS1579 エラーが生成されます。

```
// CS1579.cs
using System;
public class MyCollection
{
   int[] items;
   public MyCollection()
   {
      items = new int[5] {12, 44, 33, 2, 50};
   }

   // Delete the following line to resolve.
   MyEnumerator GetEnumerator()

   // Uncomment the following line to resolve:
   // public MyEnumerator GetEnumerator()
   {
      return new MyEnumerator(this);
   }

   // Declare the enumerator class:
   public class MyEnumerator
   {
      int nIndex;
      MyCollection collection;
      public MyEnumerator(MyCollection coll)
      {
         collection = coll;
         nIndex = -1;
      }

      public bool MoveNext()
      {
         nIndex++;
         return(nIndex < collection.items.GetLength(0));
      }

      public int Current
      {
         get
         {
            return(collection.items[nIndex]);
         }
      }
   }

   public static void Main()
```

```
    {
        MyCollection col = new MyCollection();
        Console.WriteLine("Values in the collection are:");
        foreach (int i in col)   // CS1579
        {
            Console.WriteLine(i);
        }
    }
}
```

コンパイラ エラー CS1583

エラー メッセージ
'file' は有効な Win32 のリソース ファイルではありません。

有効なリソース ファイルではないファイルが、/win32res コンパイラ オプションに渡されました。

コンパイラ エラー CS1585

エラー メッセージ
メンバ型と名前の前にメンバ修飾子 'keyword' が必要です。

メソッドのシグネチャのアクセス指定子を置く位置が間違っています。

次の例では CS1585 エラーが生成されます。

```
// CS1585.cs
public class Class1
{
   public void static Main(string[] args)   // CS1585
   // try the following line instead
   // public static void Main(string[] args)
   {
   }
}
```

コンパイラ エラー CS1586

エラー メッセージ
配列を作成するには、配列のサイズまたは配列の初期化子を指定する必要があります。

配列が正しく宣言されませんでした。

次の例では CS1586 エラーが生成されます。

```
// CS1586.cs
using System;
class MyClass
{
   public static void Main()
   {
      int[] a = new int[];   // CS1586
      // try the following line instead
      int[] b = new int[5];
   }
}
```

コンパイラ エラー CS1588

エラー メッセージ
共通言語ランタイム ディレクトリが決定できません -- 'reason'

.NET Framework 共通言語ランタイムがインストールされているディレクトリを取得できません。このエラーは、共通言語ランタイムが正しくインストールされていないことを示します。

コンパイラ エラー CS1593

エラー メッセージ
デリゲート 'del' に 'number' 個の引数を指定することはできません。

デリゲート呼び出しに渡された引数の数が、デリゲート宣言のパラメータの数と一致しません。

次の例では CS1593 エラーが生成されます。

```
// CS1593.cs
using System;
delegate string func(int i);   // declare delegate

class a
{
   public static void Main()
   {
      func dt = new func(z);
      x(dt);
   }

   public static string z(int j)
   {
      Console.WriteLine(j);
      return j.ToString();
   }

   public static void x(func hello)
   {
      hello(8, 9);   // CS1593
      // try the following line instead
      // hello(8);
   }
}
```

コンパイラ エラー CS1594

エラー メッセージ
デリゲート 'delegate' に無効な引数があります。

デリゲート呼び出しに渡された引数の型が、デリゲート宣言のパラメータの型と一致しません。

次の例では CS1594 エラーが生成されます。

```
// CS1594.cs
using System;
delegate string func(int i);   // declare delegate

class a
{
   public static void Main()
   {
      func dt = new func(z);
      x(dt);
   }

   public static string z(int j)
   {
      Console.WriteLine(j);
      return j.ToString();
   }

   public static void x(func hello)
   {
      hello("8");   // CS1594
      // try the following line instead
      // hello(8);
   }
}
```

コンパイラ エラー CS1597

エラー メッセージ
メソッドまたはアクセサ ブロックの後のセミコロンの使用が正しくありません。

メソッドやアクセサ ブロックの終端には、セミコロン (;) が不要か、または使用できません。

次の例では CS1597 エラーが生成されます。

```
// CS1597.cs
class TestClass
{
   public static void Main()
   {
   };   // CS1597, remove semicolon
}
```

コンパイラ エラー CS1599

エラー メッセージ
メソッドまたはデリゲートは型 'type' を返すことはできません。

基本クラス ライブラリ内の一部の型 (たとえば、TypedReference と ArgIterator) は、戻り値の型として使用できません。

次の例では CS1599 エラーが生成されます。

```
// CS1599.cs
using System;

class MyClass
{
   public static void Main()
   {
   }

   public TypedReference Test1()   // CS1599
   {
      return null;
   }

   public ArgIterator Test2()   // CS1599
   {
      return null;
   }
}
```

コンパイラ エラー CS1600

エラー メッセージ
ユーザーによりコンパイルがキャンセルされました。

Visual Studio IDE の使用中に、C# コンパイラによるコンパイルがキャンセルされました。

コンパイラ エラー CS1601

エラー メッセージ
メソッドまたはデリゲート パラメータを型 'type' にすることはできません。

基本クラス ライブラリ内の一部の型 (たとえば、TypedReference と ArgIterator) は、ref キーワードまたは out キーワードのパラメータ型として使用できません。

次の例では CS1601 エラーが生成されます。

```
// CS1601.cs
using System;

class MyClass
{
   public void Test1 (ref TypedReference t)   // CS1601
   {
   }

   public void Test2 (out ArgIterator t)   // CS1601
   {
   }
}
```

コンパイラ エラー CS1604

エラー メッセージ
読み取り専用なので 'variable' に割り当てできません。

読み取り専用の変数に対して代入が行われました。このエラーを回避するには、このコンテキストでは、この変数に対して値を代入するなどの変更を加えないようにします。

コンパイラ エラー CS1605

エラー メッセージ
読み取り専用なので 'var' は ref または out 引数として渡せません。

ref パラメータまたは out パラメータとして渡される変数は、呼び出されたメソッドで変更されると見なされます。したがって、読み取り専用パラメータを **ref** または **out** として渡すことはできません。

コンパイラ エラー CS1606

エラー メッセージ
アセンブリ署名ができませんでした。出力は署名されていない可能性があります -- reason

アセンブリが生成されましたが、コンパイラがそのアセンブリの署名を終了しようとしたときにエラーが発生しました。

コンパイラ エラー CS1608

エラー メッセージ
Required 属性は C# 型で許可されていません

C# コードで定義された型では、RequiredAttributeAttribute を使用できません。

コンパイラ エラー CS1609

エラー メッセージ
修飾子をイベント アクセサ宣言に付属させることはできません。

修飾子を使用できるのはイベントの宣言だけです。イベントのアクセサの宣言では使用できません詳細については、
「プロパティの使用 (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS1609 エラーが生成されます。

```
// CS1609.cs
// compile with: /target:library
delegate int Del();
class A
{
   public event Del MyEvent
   {
      private add {}   // CS1609
      // try the following line instead
      // add {}
      remove {}
   }
}
```

コンパイラ エラー CS1611

エラー メッセージ
params パラメータは ref または out として宣言できません。

ref キーワードまたは out キーワードは、params キーワードと同時に使用できません。

次の例では CS1611 エラーが生成されます。

```
// CS1611.cs
public class MyClass
{
   public static void Test(params ref int[] a)   // CS1611, remove ref
   {
   }

   public static void Main()
   {
      Test(1);
   }
}
```

コンパイラ エラー CS1612

エラー メッセージ
変数ではないため、'expression' の戻り値を変更できません。

中間式の結果である値型の変更を試みました。この値は永続化されないため、変更されません。

このエラーを解決するには、式の結果を中間値に格納するか、または中間式の参照型を使用します。

使用例
次のコードでは、CS1612 エラーが生成されます。

```
// CS1612.cs
public struct MyStruct
{
    public int Width;
}

public class ListView
{
    public MyStruct Size
    {
        get { return new MyStruct(); }
    }
}

public class MyClass
{
    public MyClass()
    {
        ListView lvi;
        lvi = new ListView();
        lvi.Size.Width = 33;  // CS1612
        // Use this instead:
        // MyStruct temp = lvi.Size;
        // temp.Width = 33;
    }

    public static void Main() {}
}
```

コンパイラ エラー CS1613

エラー メッセージ
インターフェイス 'インターフェイス' のマネージ コクラス ラッパー クラス 'クラス' が見つかりません。アセンブリ 参照が存在することを確認してください。

インターフェイスで COM オブジェクトをインスタンス化しようとしました。インターフェイスには **ComImport** 属性と **CoClass** 属性がありますが、コンパイラで **CoClass** 属性に与えられた型が見つかりません。

このエラーを解決するには、次のいずれかの処理を行ってください。

- コクラスを持つアセンブリへの参照を追加する。ほとんどの場合、インターフェイスとコクラスは同じアセンブリ内にあります。詳細については、「/reference (メタデータのインポート)」または「[参照の追加] ダイアログ ボックス」を参照してください。
- インターフェイスの **CoClass** 属性を修正する。

次の例では、**CoClassAttribute** の適切な使用法を示します。

```
// CS1613.cs
using System;
using System.Runtime.InteropServices;

[Guid("1FFD7840-E82D-4268-875C-80A160C23296")]
[ComImport()]
[CoClass(typeof(A))]
public interface IA{}
public class A : IA {}

public class AA
{
   public static void Main()
   {
      IA i;
      i = new IA(); // This is equivalent to new A().
                    // because of the CoClass attribute on IA
   }
}
```

コンパイラ エラー CS1614

エラー メッセージ
'属性' が不適切です。'@attribute' または 'attributeAttribute' のどちらか一方を使用してください。

コンパイラで、あいまいな属性の指定が検出されました。

便宜上、C# コンパイラでは、単なる `[Example]` として **ExampleAttribute** を指定することを許可しています。ただし、**Example** という名前の属性クラスが **ExampleAttribute** と共に存在する場合は、`[Example]` が **Example** 属性と **ExampleAttribute** 属性のどちらを表しているかをコンパイラが判別できないため、あいまいさが発生します。明確にするために、**Example** 属性には `[@Example]` を使用し、**ExampleAttribute** には `[ExampleAttribute]` を使用してください。

次の例では CS1614 エラーが生成されます。

```
// CS1614.cs
using System;

// Both of the following classes are valid attributes with valid
// names (MySpecial and MySpecialAttribute). However, because the lookup
// rules for attributes involves auto-appending the 'Attribute' suffix
// to the identifier, these two attributes become ambiguous; that is,
// if you specify MySpecial, the compiler can't tell if you want
// MySpecial or MySpecialAttribute.

public class MySpecial : Attribute {
   public MySpecial() {}
}

public class MySpecialAttribute : Attribute {
   public MySpecialAttribute() {}
}

class MakeAWarning {
   [MySpecial()] // CS1614
                 // Ambiguous: MySpecial or MySpecialAttribute?
   public static void Main() {
   }

   [@MySpecial()] // This isn't ambiguous, it binds to the first attribute above.
   public static void NoWarning() {
   }

   [MySpecialAttribute()] // This isn't ambiguous, it binds to the second attribute above.
   public static void NoWarning2() {
   }

   [@MySpecialAttribute()] // This is also legal.
   public static void NoWarning3() {
   }
}
```

コンパイラ エラー CS1615

エラー メッセージ
引数 '数' を 'キーワード' キーワードと共に渡すことはできません。

引数に **ref** パラメータも **out** パラメータも受け取らない関数で、**ref** または **out** のいずれかのキーワードが使用されました。このエラーを解決するには、不適切なキーワードを削除し、関数の宣言と一致する適切なキーワードを使用します。

次の例では CS1615 エラーが生成されます。

```
// CS1615.cs
class C
{
   public void f(int i) {}
   public static void Main()
   {
      int i = 1;
      f(ref i);  // CS1615
   }
}
```

コンパイラ エラー CS1617

エラー メッセージ
/langversion に対する無効なオプション 'オプション' です。ISO-1 または Default でなければなりません。

このエラーは、コマンド ライン スイッチまたはプロジェクト設定で、有効な言語オプションを指定せずに /langversion を使用した場合に発生します。このエラーを解決するには、コマンド ライン構文またはプロジェクト設定を確認し、正しいオプションに変更します。

たとえば、`csc /langversion:ISO` でコンパイルすると、CS1617 エラーが生成されます。

コンパイラ エラー CS1618

エラー メッセージ
'method'' は条件付き属性なので、この属性でデリゲートを作成できません。

メソッドが一部のビルドに存在しない可能性があるため、条件付きメソッドではデリゲートを作成できません。

次の例では CS1618 エラーが生成されます。

```
// CS1618.cs
using System;
using System.Diagnostics;

delegate void del();

class MakeAnError {
   public static void Main() {
      del d = new del(ConditionalMethod);   // CS1618
      // Invalid because on builds where DEBUG is not set,
      // there will be no "ConditionalMethod".
   }
   // To fix the error, remove the next line:
   [Conditional("DEBUG")]
   public static void ConditionalMethod()
   {
      Console.WriteLine("Do something only in debug");
   }
}
```

コンパイラ エラー CS1619

エラー メッセージ
一時ファイル 'ファイル名' -- 理由を作成できません。

コンパイラは、特定の理由 (ディスクの空き領域不足) で一時ファイルを作成できませんでした。

コンパイラ エラー CS1620

エラー メッセージ
引数 '数' は 'キーワード' キーワードと共に渡されなければなりません。

このエラーは、ref パラメータや out パラメータを受け取る関数に対して引数を渡すとき、呼び出しの時点で **ref** キーワードまたは **out** キーワードを指定しなかったか、誤ったキーワードを指定した場合に発生します。エラー メッセージには、正しいキーワードと、エラーの原因となった引数が示されます。

次の例では CS1620 エラーが生成されます。

```
// CS1620.cs
class C
{
    void f(ref int i) {}
    public static void Main()
    {
        int x = 1;
        f(out x);  // CS1620 – f takes a ref parameter, not an out parameter
        // Try this line instead:
        // f(ref x);
    }
}
```

コンパイラ エラー CS1621

エラー メッセージ
yield ステートメントは、匿名メソッド ブロックの内部では使用できません

yield ステートメントを反復子の匿名メソッド ブロックに記述することはできません。

使用例
次の例では CS1621 エラーが生成されます。

```
// CS1621.cs

using System.Collections;

delegate object MyDelegate();

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        MyDelegate d = delegate
        {
            yield return this; // CS1621
            return this;
        };
        d();
        // Try this instead:
        // MyDelegate d = delegate { return this; };
        // yield return d();
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS1622

エラー メッセージ
反復子から値を返すことができません。yield return ステートメントを使用して値を返すか、yield break ステートメントを使用して反復子を終了してください。

反復子は、return ステートメントではなく yield ステートメントで値を返す特殊な関数です。詳細については、反復子のトピックを参照してください。

次の例では CS1622 エラーが生成されます。

```
// CS1622.cs
// compile with: /target:library
using System.Collections;

class C : IEnumerable
{
   public IEnumerator GetEnumerator()
   {
      return (IEnumerator) this;  // CS1622
      yield return this;   // OK
   }
}
```

コンパイラ エラー CS1623

エラー メッセージ
反復子には ref または out パラメータを指定できません。

このエラーは、反復子のメソッドに **ref** パラメータまたは **out** パラメータが定義されている場合に発生します。このエラーを回避するには、メソッド シグネチャから **ref** キーワードまたは **out** キーワードを削除します。

使用例

次の例では CS1623 エラーが生成されます。

```
// CS1623.cs
using System.Collections;

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        yield return 0;
    }

    // To resolve the error, remove ref
    public IEnumerator GetEnumerator(ref int i)  // CS1623
    {
        yield return i;
    }

    // To resolve the error, remove out
    public IEnumerator GetEnumerator(out float f)  // CS1623
    {
        f = 0.0F;
        yield return f;
    }
}
```

コンパイラ エラー CS1624

エラー メッセージ
'型' は反復子インターフェイス型ではないため、'アクセサ' の本体は反復子ブロックにできません

このエラーは、反復子のアクセサが使用されているとき、戻り値の型が反復子インターフェイス (IEnumerable、IEnumerable、IEnumerator、IEnumerator) のいずれの型にも該当しない場合に発生します。このエラーを回避するには、戻り値の型として、反復子インターフェイスのいずれかの型を使用します。

使用例
次の例では CS1624 エラーが生成されます。

```
// CS1624.cs
using System;
using System.Collections;

class C
{
    public int Iterator
    // Try this instead:
    // public IEnumerable Iterator
    {
        get  // CS1624
        {
            yield return 1;
        }
    }
}
```

コンパイラ エラー CS1625

エラー メッセージ
finally 句の本体で生成することはできません。

finally 句の本体に yield ステートメントを記述することはできません。このエラーを回避するには、yield ステートメントを finally 句の外に記述します。

次の例では CS1625 エラーが生成されます。

```
// CS1625.cs
using System.Collections;

class C : IEnumerable
{
   public IEnumerator GetEnumerator()
   {
      try
      {
      }
      finally
      {
        yield return this;  // CS1625
      }
   }
}

public class CMain
{
   public static void Main() { }
}
```

コンパイラ エラー CS1626

エラー メッセージ
catch 句を含む try ブロックの本体で値を生成することはできません。

try ブロックに対応する catch 句が存在する場合、その try ブロックで yield ステートメントを使用することはできません。このエラーを回避するには、yield ステートメントを catch 句の外に記述します。

次の例では CS1626 エラーが生成されます。

```
// CS1626.cs
using System.Collections;

class C : IEnumerable
{
   public IEnumerator GetEnumerator()
   {
      try
      {
         yield return this;  // CS1626
      }
      catch
      {

      }
   }
}

public class CMain
{
   public static void Main() { }
}
```

コンパイラ エラー CS1627

エラー メッセージ
yield の戻り値の後に式が必要です。

このエラーは、**yield** ステートメントから、式が欠落している場合に発生します。このエラーを回避するには、ステートメントに適切な式を記述します。

次の例では CS1627 エラーが生成されます。

```
// CS1627.cs
using System.Collections;

class C : IEnumerable
{
   public IEnumerator GetEnumerator()
   {
      yield return;   // CS1627
      // To resolve, add the following line:
      // yield return 0;
   }
}

public class CMain
{
   public static void Main() { }
}
```

コンパイラ エラー CS1628

エラー メッセージ
ref または out パラメータ 'パラメータ' は、匿名メソッド ブロックの内部では使用できません

このエラーは、ref パラメータまたは out パラメータを匿名メソッド ブロックで使用した場合に発生します。このエラーを回避するには、ローカル変数を使用するか、匿名メソッドを使わない方法を検討します。

次の例では CS1628 エラーが生成されます。

```
// CS1628.cs

delegate int MyDelegate();

class C
{
  public static void F(ref int i)
  {
      MyDelegate d = delegate { return i; };  // CS1628
      // Try this instead:
      // int tmp = i;
      // MyDelegate d = delegate { return tmp; };
  }

  public static void Main()
  {

  }
}
```

コンパイラ エラー CS1629

エラー メッセージ
アンセーフ コードは反復子には記述できません。

C# の言語仕様により、反復子にアンセーフ コードを使用することはできません。

次の例では CS1629 エラーが生成されます。

```
// CS1629.cs
// compile with: /unsafe
using System.Collections.Generic;
class C
{
   IEnumerator<int> IteratorMeth() {
      int i;
      unsafe  // CS1629
      {
         int *p = &i;
         yield return *p;
      }
   }
}
```

コンパイラ エラー CS1630

エラー メッセージ
/errorreport に対する無効なオプション 'オプション' です。prompt、send、queue、または none のいずれかでなければなりません。

コマンド ライン オプション /errorreport に続けて、**prompt**、**send**、**queue**、または **none** を付け、内部コンパイラ エラーが発生したときの処理を指定する必要があります。

コンパイラ エラー CS1631

エラー メッセージ
catch 句の本体で値を生成することはできません。

catch 句の本体に yield ステートメントを記述することはできません。このエラーを回避するには、yield ステートメントを catch 句の本体の外に記述します。

次の例では CS1631 エラーが生成されます。

```
// CS1631.cs
using System;
using System.Collections;

public class C : IEnumerable
{
   public IEnumerator GetEnumerator()
   {
      try
      {
      }
      catch(Exception e)
      {
        yield return this;  // CS1631
      }
   }

   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1632

エラー メッセージ
コントロールを匿名メソッドの本体外に出すことはできません。

このエラーは、ジャンプ ステートメント (**break**、**goto**、**continue** など) で、プログラムの制御を匿名メソッド ブロックの外に移動しようとした場合に発生します。匿名メソッド ブロックは関数本体であり、return ステートメントで終了するか、匿名メソッド ブロックの終わりに達して初めて処理を抜けることができます。

次の例では CS1632 エラーが生成されます。

```
// CS1632.cs
// compile with: /target:library
delegate void MyDelegate();
class MyClass
{
   public void Test()
   {
      for (int i = 0 ; i < 5 ; i++)
      {
         MyDelegate d = delegate {
            break;   // CS1632
          };
      }
   }
}
```

コンパイラ エラー CS1637

エラー メッセージ
反復子には unsafe パラメータまたは yield 型を指定できません。

反復子の引数リストおよび yield ステートメントの型をチェックし、アンセーフな型が使用されていないことを確認してください。

使用例
次の例では CS1637 エラーが生成されます。

```
// CS1637.cs
// compile with: /unsafe
using System.Collections;

public unsafe class C
{
    public IEnumerator Iterator1(int* p)  // CS1637
    {
        yield return null;
    }
}
```

コンパイラ エラー CS1638

エラー メッセージ
'識別子' は予約された識別子で、ISO 言語バージョン モードが使用されたときに使用することはできません。

このエラーは、**/langversion** コンパイラ スイッチで ISO 言語互換を選択するとき、識別子に 2 つのアンダースコアを含めた場合に発生します。このエラーを回避するには、2 つのアンダースコアの付いた識別子を削除するか、ISO-1 言語バージョン オプションを使用しないようにします。

使用例
次の例では CS1638 エラーが生成されます。

```
// CS1638.cs
// compile with: /langversion:ISO-1
class bad__identifer // CS1638 (double underscores are not ISO compliant)
{
}

// Try this instead:
//class GoodIdentifier
//{
//}

class CMain
{
    public static void Main() { }
}
```

参照
関連項目
/langversion (準拠構文) (C# コンパイラ オプション)

コンパイラ エラー CS1639

エラー メッセージ
インターフェイス 'インターフェイス' のマネージ コクラス ラッパー クラス 'シグネチャ' は、有効なクラス名 シグネチャではありません

このエラーは、インターフェイスのメタデータが無効である場合に発生します。このエラーは、インターフェイスが C# またはサポートされている .NET 言語を使って生成されている限り発生しません。インターフェイスの販売元に問い合わせるか、インターフェイス アセンブリが正しく生成されているかを確認してください。

コンパイラ エラー CS1640

エラー メッセージ
'インターフェイス' の複数のインスタンスを実装するため、foreach ステートメントは、型 '型' の変数では操作できません。特定のインターフェイスのインスタンス化にキャストしてください。

型が IEnumerator<T> の複数のインスタンスから継承されています。**foreach** の列挙子として使用できる一意の型が存在しません。IEnumerator<T> の型を指定するか、他のループ制御構文を使用してください。

使用例
次の例では CS1640 エラーが生成されます。

```
// CS1640.cs

using System;
using System.Collections;
using System.Collections.Generic;

public class C : IEnumerable, IEnumerable<int>, IEnumerable<string>
{
    IEnumerator<int> IEnumerable<int>.GetEnumerator()
    {
        yield break;
    }

    IEnumerator<string> IEnumerable<string>.GetEnumerator()
    {
        yield break;
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return (IEnumerator)((IEnumerable<string>)this).GetEnumerator();
    }
}

public class Test
{
    public static int Main()
    {
        foreach (int i in new C()){}    // CS1640

        // Try specifing the type of IEnumerable<T>
        // foreach (int i in (IEnumerable<int>)new C()){}
        return 1;
    }
}
```

コンパイラ エラー CS1641

エラー メッセージ
固定サイズ バッファ フィールドには、フィールド名の後に配列サイズの指定子が必要です。

通常の配列とは異なり、固定サイズのバッファは、宣言の時点で一定のサイズが指定されている必要があります。このエラーを解決するには、正の整数リテラル (一定の値を持つ正の整数) を追加し、識別子の後に角かっこを付けます。

次の例では CS1641 エラーが生成されます。

```
// CS1641.cs
// compile with: /unsafe /target:library
unsafe struct S {
   fixed int [] a;  // CS1641

   // OK
   fixed int b [10];
   const int c = 10;
   fixed int d [c];
}
```

コンパイラ エラー CS1642

エラー メッセージ
固定サイズ バッファ フィールドは、構造体のメンバにしかなれません。

このエラーは、**struct** ではなく、**class** 内で、固定サイズ バッファのフィールドを使用した場合に発生します。このエラーを解決するには、**class** を **struct** に変更するか、フィールドを通常の配列として宣言します。

使用例
次の例では CS1642 エラーが生成されます。

```
// CS1642.cs
// compile with: /unsafe /target:library
unsafe class C
{
   fixed int a[10];   // CS1642
}

unsafe struct D
{
    fixed int a[10];
}

unsafe class E
{
   public int[] a = null;
}
```

コンパイラ エラー CS1643

エラー メッセージ
型 '型' の匿名メソッドです: 値を返さないコントロール パスがあります。

このエラーは、デリゲートの本体に return ステートメントが存在しない、または存在したとしても、その return ステートメントまで到達可能なことをコンパイラが確認できない場合に発生します。次の例では、匿名メソッドのブロックから常に値が戻るか検査するために分岐条件の結果を予測する、ということをコンパイラは行いません。

使用例
次の例では CS1643 エラーが生成されます。

```
// CS1643.cs
delegate int MyDelegate();

class C
{
    static void Main()
    {
        MyDelegate d = delegate
        {                 // CS1643
            int i = 0;
            if (i == 0)
                return 1;
        };
    }
}
```

コンパイラ エラー CS1644

エラー メッセージ
標準 ISO C# 言語仕様の一部ではないため、機能 '機能' を使用することはできません。

このエラーは、/langversion オプションに ISO-1 を指定してコンパイルしたとき、対象となるコードで、ISO 1.0 規格に準拠していない機能が使われている場合に発生します。このエラーを解決するには、ISO-1 互換オプションを指定する場合に、C# コンパイラの新しい機能は一切使わないようにする必要があります。C# コンパイラの新機能については、「C# 2.0 言語およびコンパイラの新機能」を参照してください。

次の例では CS1644 エラーが生成されます。

```
// CS1644.cs
// compile with: /langversion:ISO-1 /target:library
class C<T> {}   // CS1644
```

コンパイラ エラー CS1646

エラー メッセージ
verbatim 識別子の後にはキーワード、識別子、または文字列が必要です: @

リテラル文字列で、verbatim 識別子 '@' の使い方を確認してください。verbatim 識別子は、文字列、キーワード、または識別子の前にのみ使用できます。このエラーを解決するには、不適切な場所に指定されている @ 記号を削除するか、@ 記号に続けて目的の文字列、キーワード、または識別子を追加してください。

次の例では CS1646 エラーが生成されます。

```
// CS1646
class C
{
   int i = @5;  // CS1646
   // Try this line instead:
   // int i = 5;
}
```

コンパイラ エラー CS1647

エラー メッセージ
'コード' の付近でコンパイルするには、式が長すぎるか、または複雑すぎます。

コンパイラがコードを処理しているときに、スタック オーバーフローが発生しました。このエラーを解決するには、コードを単純化します。コードに問題がない場合は、製品サポートまでお問い合わせください。

コンパイラ エラー CS1648

エラー メッセージ
読み取り専用フィールド '識別子' のメンバは変更できません (コンストラクタまたは変数初期化子では可)。

このエラーは、変更することのできない、読み取り専用フィールドのメンバを変更しようとした場合に発生します。このエラーを解決するには、読み取り専用フィールドへの代入をコンストラクタまたは変数初期化子に限定するか、フィールドの宣言から readonly キーワードを削除します。

次の例では CS1648 エラーが生成されます。

```
// CS1648.cs
public struct Inner
  {
    public int i;
  }

class Outer
{
  public readonly Inner inner = new Inner();
}

class D
{
   static void Main()
   {
      Outer outer = new Outer();
      outer.inner.i = 1;  // CS1648
   }
}
```

コンパイラ エラー CS1649

エラー メッセージ
読み取り専用フィールド '識別子' のメンバに ref または out を渡すことはできません (静的コンストラクタでは可)

このエラーは、関数の **ref** 引数または **out** 引数として、**readonly** フィールドのメンバ変数を渡した場合に発生します。**ref** パラメータおよび **out** パラメータは関数によって変更される可能性があるため、このような操作は許可されません。このエラーを解決するには、該当するフィールドから **readonly** キーワードを削除するか、**readonly** フィールドのメンバを関数に渡さないようにする必要があります。たとえば、次の例に示すように、変更可能なテンポラリ変数を作成し、その変数を **ref** 引数として渡すようにします。

使用例
次の例では CS1649 エラーが生成されます。

```
// CS1649.cs
public struct Inner
    {
        public int i;
    }

class Outer
{
    public readonly Inner inner = new Inner();
}

class D
{
    static void f(ref int iref)
    {
    }

    static void Main()
    {
        Outer outer = new Outer();
        f(ref outer.inner.i);  // CS1649
        // Try this code instead:
        // int tmp = outer.inner.i;
        // f(ref tmp);
    }
}
```

コンパイラ エラー CS1650

エラー メッセージ
静的読み取り専用フィールド '識別子' のフィールドへの割り当てはできません (静的コンストラクタまたは変数初期化子では可)

このエラーは、変更することのできない、読み取り専用の静的フィールドのメンバを変更しようとした場合に発生します。このエラーを解決するには、読み取り専用フィールドへの代入をコンストラクタまたは変数初期化子に限定するか、フィールドの宣言から **readonly** キーワードを削除します。

```
// CS1650.cs
public struct Inner
{
    public int i;
}

class Outer
{
    public static readonly Inner inner = new Inner();
}

class D
{
    static void Main()
    {
        Outer.inner.i = 1;  // CS1650
    }
}
```

コンパイラ エラー CS1651

エラー メッセージ
スタティック読み取り専用フィールド '識別子' には、スタティック コンストラクタ内を除き、ref または out を渡すことはできません。

このエラーは、ref 引数として、静的な読み取り専用フィールドのメンバ変数を関数に渡した場合に発生します。ref パラメータは関数によって変更される可能性があるため、このような操作は許可されません。このエラーを解決するには、該当するフィールドから **readonly** キーワードを削除するか、読み取り専用フィールドのメンバを関数に渡さないようにする必要があります。たとえば、次の例に示すように、変更可能なテンポラリ変数を作成し、その変数を ref 引数として渡すようにします。

次の例では CS1651 エラーが生成されます。

```
// CS1651.cs
public struct Inner
  {
    public int i;
  }

class Outer
{
  public static readonly Inner inner = new Inner();
}

class D
{
   static void f(ref int iref)
   {
   }

   static void Main()
   {
      f(ref Outer.inner.i);  // CS1651
      // Try this instead:
      // int tmp = Outer.inner.i;
      // f(ref tmp);
   }
}
```

コンパイラ エラー CS1652

エラー メッセージ
'識別子' は読み取り専用であるため、このメンバを変更できません。

このエラーは、コンテキストのために読み取り専用になっている変数のメンバに変更を加えようとした場合に発生します。

コンパイラ エラー CS1653

エラー メッセージ
読み取り専用であるため、'識別子' のフィールドを ref または out 引数として渡せません

このエラーは、変数がコンテキスト上、読み取り専用であるときに、この変数のメンバを ref または out パラメータとして渡そうとしたときに発生します。

コンパイラ エラー CS1654

エラー メッセージ
'変数' は '読み取り専用の変数型' であるため、このメンバを変更できません。

このエラーは、特別な構造のために読み取り専用になっている変数のメンバに変更を加えようとした場合に発生します。

使用例
次の例では CS1654 エラーが生成されます。

```
// CS1654.cs
using System;
using System.Collections;

public struct Test : IEnumerable
{
    private int index;
    public int Index
    {
        get { return index; }
        set { index = value; }
    }

    public IEnumerator GetEnumerator()
    {
        for(int i = 0; i< 10; i++)
            yield return this;
        yield break;
    }
}

public class Repro
{
    static int Main()
    {
        int i = 0;
        Test t = new Test();
        foreach (Test tt in t)
        {
            tt.Index = i++;    // CS1654
        }
        return 1;
    }
}
```

コンパイラ エラー CS1655

エラー メッセージ
'読み取り専用の変数型'であるため、'変数' のフィールドを ref または out 引数として渡せません

このエラーは、foreach、using、fixed のいずれかの変数のメンバを ref 引数または out 引数として関数に渡そうとした場合に発生します。これらのコンテキストでは、変数は読み取り専用と見なされるため、このような記述は認められません。

次の例では CS1655 エラーが生成されます。

```
// CS1655.cs
struct S
{
   public int i;
}

class CMain
{
  static void f(ref int iref)
  {
  }

  public static void Main()
  {
     S[] sa = new S[10];
     foreach(S s in sa)
     {
        CMain.f(ref s.i);  // CS1655
     }
  }
}
```

コンパイラ エラー CS1656

エラー メッセージ
'読み取り専用の変数型' なので '変数' に割り当てできません。

このエラーは、変数に対する値の代入が、読み取り専用のコンテキストで行われた場合に発生します。読み取り専用になるコンテキストとしては、foreach の反復子、using の変数、fixed の変数などがあります。このエラーを解決するには、**using** ブロック、**foreach** ステートメント、**fixed** ステートメントの変数に値が代入されている箇所を修正します。

使用例
次の例では CS1656 エラーが生成されます。

```
// CS1656.cs
// compile with: /unsafe
using System;

class C : IDisposable
{
    public void Dispose() { }
}

class CMain
{
    unsafe public static void Main()
    {
        using (C c = new C())
        {
            c = new C(); // CS1656
        }

        foreach (object o in new string[] { "1", "2" })
        {
            o = "10"; // CS1656
        }

        int[] ary = new int[] { 1, 2, 3, 4 };
        fixed (int* p = ary)
        {
            p = null; // CS1656
        }
    }
}
```

コンパイラ エラー CS1657

エラー メッセージ
読み取り専用なので '変数' は ref または out 引数として渡せません。

このエラーは、変数が読み取り専用になるコンテキストで、その変数を ref 引数またはout 引数として渡した場合に発生します。読み取り専用になるコンテキストとしては、foreach の反復子、using の変数、**fixed** の変数などがあります。このエラーを解決するには、**using** ブロック、**foreach** ステートメント、および **fixed** ステートメントでは、これらの変数 (つまり、**foreach**、**using**、**fixed** で使用される変数) を **ref** パラメータや **out** パラメータとして受け取るような関数を呼び出さないようにします。

使用例
次の例では CS1657 エラーが生成されます。

```
// CS1657.cs
using System;
class C : IDisposable
{
    public int i;
    public void Dispose() {}
}

class CMain
{
    static void f(ref C c)
    {
    }
    static void Main()
    {
        using (C c = new C())
        {
            f(ref c);  // CS1657
        }
    }
}
```

次のコードには、**fixed** ステートメントにおける同様の問題が示されています。

```
// CS1657b.cs
// compile with: /unsafe
unsafe class C
{
    static void F(ref int* p)
    {
    }

    static void Main()
    {
        int[] a = new int[5];
        fixed(int* p = a) F(ref p); // CS1657
    }
}
```

コンパイラ エラー CS1660

エラー メッセージ
デリゲート型ではないため、匿名メソッド ブロックを型 '型' に変換できません

このエラーは、匿名メソッド ブロックをデリゲート以外の型に代入または変換しようとした場合に発生します。

次の例では CS1660 エラーが生成されます。

```
// CS1660.cs
delegate int MyDelegate();
class C {
   static void Main()
   {
     int i = delegate { return 1; };  // CS1660
     // Try this instead:
     // MyDelegate myDelegate = delegate { return 1; };
     // int i = myDelegate();
   }
}
```

コンパイラ エラー CS1661

エラー メッセージ
指定されたブロックのパラメータ型がデリゲート パラメータ型と一致しないため、匿名メソッド ブロックをデリゲート型 'デリゲート型' に変換することはできません。

このエラーは、匿名メソッドの定義で、匿名メソッドとデリゲートのパラメータの型が一致しない場合に発生します。パラメータの数、パラメータの型、ref パラメータや out パラメータの有無、これらがすべて一致していることを確認します。

次の例では CS1661 エラーが生成されます。

```
// CS1661.cs

delegate void MyDelegate(int i);

class C
{
    public static void Main()
    {
        MyDelegate d = delegate(string s) { };  // CS1661
    }
}
```

コンパイラ エラー CS1662

エラー メッセージ
デリゲート戻り値の型に暗黙的に変換できない戻り値の型がブロック内にあるため、匿名メソッド ブロックをデリゲート型 'デリゲート型' に変換することはできません。

このエラーは、匿名メソッド ブロックの return ステートメントに、デリゲートの戻り値の型として暗黙的に変換することのできない型が存在する場合に発生します。

次の例では CS1662 エラーが生成されます。

```
// CS1662.cs

delegate int MyDelegate(int i);

class C
{

  public static void Main()
  {
     MyDelegate d = delegate(int i) { return 1.0; };  // CS1662
     // Try this instead:
     // MyDelegate d = dekegate(int i) { return (int)1.0; };
  }
}
```

コンパイラ エラー CS1663

エラー メッセージ
固定サイズ バッファの型は次のうちの 1 つでなければなりません: bool、byte、short、int、long、char、sbyte、ushort、uint、ulong、float または double

固定サイズのバッファに、エラー メッセージで示されている以外の型を使用することはできません。このエラーを回避するには、他の型を使用するか、固定サイズの配列を使わないようにします。

使用例
次の例では CS1663 エラーが生成されます。

```
// CS1663.cs
// compile with: /unsafe /target:library

unsafe struct C
{
   fixed string ab[10];   // CS1663
}
```

コンパイラ エラー CS1664

エラー メッセージ
'長さ' の長さで型 '型' の固定サイズ バッファは大きすぎます

(長さに要素サイズを乗じて計算される) 固定サイズ バッファの最大サイズは、2^31 (= 268,435,455) です。

コンパイラ エラー CS1665

エラー メッセージ
固定サイズ バッファには、0 よりも大きい値を指定しなければなりません。

このエラーは、固定サイズのバッファが、ゼロまたは負のサイズで宣言された場合に発生します。固定サイズのバッファは、長さを正の整数にする必要があります。

使用例
次の例では CS1665 エラーが生成されます。

```
// CS1665.cs
// compile with: /unsafe /target:library
struct S
{
   public unsafe fixed int A[0];   // CS1665
}
```

コンパイラ エラー CS1666

エラー メッセージ
fixed でない式に含まれる固定サイズ バッファは使用できません。fixed ステートメントを使用してください

このエラーは、アドレスが固定指定されていないクラスを含む式で、固定サイズのバッファを使用した場合に発生します。アドレスが固定指定されていないクラスは、メモリへのアクセスを最適化するために、ランタイムによって移動されることがあります。ここで、固定サイズのバッファを使用するとエラーになる場合があります。このエラーを回避するには、該当するステートメントに **fixed** キーワードを使用します。

使用例
次の例では CS1666 エラーが生成されます。

```
// CS1666.cs
// compile with: /unsafe /target:library
unsafe struct S
{
   public fixed int buffer[1];
}

unsafe class Test
{
   S field = new S();

   private bool example1()
   {
      return (field.buffer[0] == 0);   // CS1666 error
   }

   private bool example2()
   {
      // OK
      fixed (S* p = &field)
      {
         return (p->buffer[0] == 0);
      }
   }

   private bool example3()
   {
      S local = new S();
      return (local.buffer[0] == 0);
   }
}
```

コンパイラ エラー CS1667

エラー メッセージ
属性 '属性' は、プロパティまたはアクセサで有効ではありません。'宣言型' 宣言でのみ有効です。

このエラーは、属性をプロパティまたはイベントそのものに対して使用すべき場合に、プロパティまたはイベントのアクセサに対して属性を使用した場合に発生します。このエラーは、CLSCompliantAttribute、ConditionalAttribute、ObsoleteAttribute の各属性で発生します。

使用例
次の例では CS1667 エラーが生成されます。

```
// CS1667.cs
using System;

public class C
{
    private int i;

    //Try this instead:
    //[Obsolete]
    public int ObsoleteProperty
    {
        [Obsolete]  // CS1667
        get { return i; }
        set { i = value; }
    }

    public static void Main()
    {
    }
}
```

コンパイラ エラー CS1670

エラー メッセージ
params はこのコンテキストでは有効ではありません。

C# のいくつかの機能は、可変個引数リストと互換性がありません。次のような **params** キーワードの使い方はできません。

- 匿名メソッドのパラメータ リスト
- オーバーロードされた演算子

使用例
次の例では CS1670 エラーが生成されます。

```
// CS1670.cs
public class C
{
    public bool operator +(params int[] paramsList)  // CS1670
    {
        return false;
    }

    static void Main()
    {
    }
}
```

コンパイラ エラー CS1671

エラー メッセージ
名前空間の宣言に、修飾子または属性を指定することはできません。

修飾子を名前空間に適用することは無意味であるため、使用できません。

次の例では CS1671 エラーが生成されます。

```
// CS1671.cs
public namespace NS // CS1671
{

}
```

コンパイラ エラー CS1672

エラー メッセージ
/platform に対する無効なオプション 'オプション' です。anycpu、x86、Itanium または x64 でなければなりません。

このオプションでは、プロセッサの種類を、メッセージに示された、いずれかの形式で指定する必要があります。

コンパイラ エラー CS1673

エラー メッセージ
構造体内の匿名メソッドは、'this' のインスタンス メンバにアクセスできません。'this' を匿名メソッド外のローカル変数にコピーして、そのローカルを使用してください。

次の例では CS1673 エラーが生成されます。

```
// CS1673.cs
delegate int MyDelegate();

public struct S
{
   int member;

   public int F(int i)
   {
       member = i;
       // Try assigning to a local variable
       // S s = this;
       MyDelegate d = delegate()
       {
          i = this.member;  // CS1673
          // And use the local variable instead of "this"
          // i =  s.member;
          return i;

       };
       return d();
   }
}

class CMain
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1674

エラー メッセージ
'T': using ステートメントで使用される型は、暗黙的に 'System.IDisposable' への変換が可能でなければなりません。

using ステートメントは、**using** ブロックの最後に、オブジェクトを確実に破棄することを目的としています。そのため、このステートメントでは、破棄可能な型以外は使用できません。たとえば、値型は破棄できません。また、型パラメータも、解決後の型をクラスに限定しない限り、破棄可能な型とは見なされません。

使用例
次の例では CS1674 エラーが生成されます。

```
// CS1674.cs
class C
{
   public static void Main()
   {
      int a = 0;
      a++;

      using (a) {}   // CS1674
   }
}
```

次の例では CS1674 エラーが生成されます。

```
// CS1674_b.cs
using System;
class C {
   public void Test() {
      using (C c = new C()) {}   // CS1674
   }
}

// OK
class D : IDisposable {
   void IDisposable.Dispose() {}
   public void Dispose() {}

   public static void Main() {
      using (D d = new D()) {}
   }
}
```

次の例は、未知の型パラメータを破棄可能として扱う場合にクラス型制約が必要になるケースを示しています。次の例では CS1674 エラーが生成されます。

```
// CS1674_c.cs
// compile with: /target:library
using System;
public class C<T>
// Add a class type constraint that specifies a disposable class.
// Uncomment the following line to resolve.
// public class C<T> where T : IDisposable
{
   public void F(T t)
   {
      using (t) {}   // CS1674
   }
}
```

コンパイラ エラー CS1675

エラー メッセージ
Enums に型パラメータを指定することはできません。

このエラーを解決するには、**enum** 宣言から型パラメータを削除します。

使用例
次の例では CS1675 エラーが生成されます。

```
// CS1675.cs
enum E<T>  // CS1675
{
}

class CMain
{
    public static void Main()
    {
    }
}
```

コンパイラ エラー CS1676

エラー メッセージ
パラメータ '番号' は 'キーワード' キーワードで宣言されなければなりません。

このエラーは、匿名メソッドのパラメータの型修飾子が、キャスト先のデリゲートの宣言と異なる場合に発生します。

次の例では CS1676 エラーが生成されます。

```
// CS1676.cs
delegate void E(ref int i);
class Errors
{
   static void Main()
   {
      E e = delegate(out int i) { };   // CS1676
      // To resolve, use the following line instead:
      // E e = delegate(ref int i) { };
   }
}
```

コンパイラ エラー CS1677

エラー メッセージ
パラメータ '番号' を 'キーワード' キーワードで宣言してはなりません。

このエラーは、匿名メソッドのパラメータの型修飾子が、メソッドのキャスト先であるデリゲートの宣言と異なる場合に発生します。

使用例
次の例では CS1677 エラーが生成されます。

```
// CS1677.cs
delegate void D(int i);
class Errors
{
    static void Main()
    {
        D d = delegate(out int i) { };   // CS1677
        // To resolve, use the following line instead:
        // D d = delegate(int i) { };

        D d = delegate(ref int j){}; // CS1677
        // To resolve, use the following line instead:
        // D d = delegate(int j){};
    }
}
```

コンパイラ エラー CS1678

エラー メッセージ
パラメータ '番号' は、型 'type1' として宣言されていますが、'type2' でなければなりません。

このエラーは、匿名メソッドのパラメータの型が、キャスト先のデリゲートの宣言と異なる場合に発生します。

次の例では CS1678 エラーが生成されます。

```
// CS1678
delegate void D(int i);
class Errors
{
   static void Main()
   {
      D d = delegate(string s) { };   // CS1678
      // To resolve, use the following line instead:
      // D d = delegate(int s) { };
   }
}
```

コンパイラ エラー CS1679

エラー メッセージ
'/reference' の無効な extern エイリアスです。'indentifier' は識別子ではありません。

/reference オプションで、外部アセンブリ エイリアスの機能を使用した場合、**/reference:** の後ろに指定するテキストと、"=" の前に指定するテキストは、C# の言語仕様に準拠した有効な識別子またはキーワードであることが必要です。

このエラーを解決するには、"=" の前のテキストを C# の有効な識別子またはキーワードに変更します。

使用例
次の例では CS1679 エラーが生成されます。

```
// CS1679.cs
// compile with: /reference:123$BadIdentifier%=System.dll
class TestClass {
    static void Main()
    {
    }
}
```

コンパイラ エラー CS1680

エラー メッセージ
無効な参照エイリアス オプションです: 'エイリアス=' -- ファイル名が指定されていません。

このエラーは、**/reference** コンパイラ オプションの `alias` 機能を使用するとき、有効なファイル名が指定されなかった場合に発生します。

次の例では CS1680 エラーが生成されます。

```
// CS1680.cs
// compile with: /reference:alias=
// CS1680 expected
// To resolve, specify the name of a file with an assembly manifest
class MyClass {}
```

コンパイラ エラー CS1681

エラー メッセージ
グローバルの extern エイリアスは再定義できません。

グローバル エイリアスは、エイリアスを使用しないすべての参照をインクルードするよう既に定義されているため、再定義することはできません。

使用例
次の例では CS1681 エラーが生成されます。

```
// CS1681.cs
// compile with: /reference:global=System.dll
// CS1681 expected

// try this instead: /reference:System.dll
class A
{
   static void Main() {}
}
```

コンパイラ エラー CS1686

エラー メッセージ
ローカル '変数' またはそのメンバがアドレスを渡すことができず、匿名メソッド ブロック内で使用することができません。

このエラーは、変数のアドレスを取得し、そのアドレスを匿名メソッド内で使用した場合に発生します。

使用例
次の例では CS1686 エラーが生成されます。

```
// CS1686.cs
// compile with: /unsafe /target:library
class MyClass
{
   public unsafe delegate int * MyDelegate();

   public unsafe int * Test()
   {
      int j = 0;
      MyDelegate d = delegate { return &j; };   // CS1686
      return &j;   // OK
   }
}
```

コンパイラ エラー CS1688

エラー メッセージ
デリゲート型 'デリゲート' には 1 つ以上の out パラメータが含まれているため、パラメータ リストを含まない匿名メソッド ブロックをこのデリゲート型に変換することはできません。

匿名メソッド ブロックでは、多くの場合、パラメータを省略できます。このエラーは、デリゲートが *out* パラメータを受け取るにもかかわらず、匿名メソッド ブロックでパラメータ リストが省略されている場合に発生します。コンパイラでは、*out* パラメータの存在を無視することはできないため、このような状況はエラーと見なされます。

使用例
次のコードでは、CS1688 エラーが生成されます。

```
// CS1688.cs
using System;
delegate void OutParam(out int i);
class ErrorCS1676
{
    static void Main()
    {
        OutParam o;
        o = delegate  // CS1688
        // Try this instead:
        // o = delegate(out int i)
        {
            Console.WriteLine("");
        };
    }
}
```

コンパイラ エラー CS1689

エラー メッセージ
属性 '属性名' は、メソッド クラスまたは属性クラスでのみ有効です

このエラーは、**ConditionalAttribute** 属性を使用した場合にのみ発生します。メッセージが示しているように、この属性は、メソッドまたは属性クラスに対してのみ有効です。たとえば、この属性をクラスに対して適用しようとすると、このエラーが発生します。

使用例
次の例では CS1689 エラーが生成されます。

```
// CS1689.cs
// compile with: /target:library
[System.Diagnostics.Conditional("A")]   // CS1689
class MyClass {}
```

コンパイラの警告 (レベル 1) CS1690

エラー メッセージ
参照マーシャリング クラスのフィールドであるため、'メンバ' のメンバにアクセスすると、ランタイム例外が発生する可能性があります

この警告は、MarshalByRefObject から派生するクラスの値型のメンバに対して、メソッド、プロパティ、またはインデクサを呼び出そうとした場合に発生します。警告を解決するには、メンバをローカル変数にコピーし、この変数でメソッドを呼び出します。

次の例では CS1690 エラーが生成されます。

```
// CS1690.cs
using System;

class WarningCS1690: MarshalByRefObject
{
   int i = 5;

   public static void Main()
   {
     WarningCS1690 e = new WarningCS1690();
     e.i.ToString();   // CS1690

     // OK
     int i = e.i;
     i.ToString();
     e.i = i;
   }
}
```

コンパイラ エラー CS1703

エラー メッセージ
同じ ID "アセンブリ名" のアセンブリが既にインポートされています。重複している参照の一方を削除してください。

同じパスおよびファイル名を持つ参照はコンパイラによって削除されますが、同じファイルが別々の場所に存在していたり、バージョン番号を変更するのを忘れている可能性があります。このエラーは、同じアセンブリ ID を持つ 2 つの参照が存在するため、コンパイラがメタデータ内で両者を区別できないことを示しています。重複する参照の一方を削除するか、アセンブリのバージョン番号を繰り上げるなどして、参照が重複しないようにする必要があります。

次のコードでは、CS1703 エラーが生成されます。

使用例
次のコードは、アセンブリ A を .\bin1 ディレクトリに作成します。

このサンプルを CS1703a1.cs という名前のファイルで保存し、`/t:library /out:.\bin1\cs1703.dll /keyfile:key.snk` のようにフラグを指定してコンパイルします。

```
using System;
public class A { }
```

次のコードは、アセンブリ A のコピーを .\bin2 ディレクトリに作成します。

このサンプルを CS1703a2.cs という名前のファイルで保存し、`/t:library /out:.\bin2\cs1703.dll /keyfile:key.snk` のようにフラグを指定してコンパイルします。

```
using System;
public class A { }
```

次のコードでは、前述の 2 つのモジュールのアセンブリ A を参照します。

このサンプルを CS1703ref.cs という名前のファイルで保存し、`/t:library /r:A2=.\bin2\cs1703.dll /r:A1=.\bin1\cs1703.dll` のようにフラグを指定してコンパイルします。

```
extern alias A1;
extern alias A2;
```

コンパイラ エラー CS1704

エラー メッセージ
同じ簡易名 'アセンブリ名' でアセンブリが既にインポートされています。参照の 1 つを削除するか、サイド バイ サイドを有効にするために署名してください。

このエラーは、同じアセンブリ ID を持つ 2 つの参照が存在することを示しています。問題となっているアセンブリは厳密な名前を持たず、署名もされていないため、コンパイラがメタデータ内で両者を区別できません。その結果、アセンブリ名のプロパティであるバージョンとカルチャが実行時に無視されてしまいます。このエラーを解決するには、重複する参照を削除する、いずれかの参照の名前を変更する、または、厳密な名前を付ける必要があります。

使用例

次の例では、アセンブリが作成されてルート ディレクトリに保存されます。

```
// CS1704_a.cs
// compile with: /target:library /out:c:\\cs1704.dll
public class A {}
```

次の例では、上の例と同じ名前のアセンブリが作成されますが、別の場所に保存されます。

```
// CS1704_b.cs
// compile with: /target:library /out:cs1704.dll
public class A {}
```

次の例では、両方のアセンブリの参照を試みます。この例では CS1704 エラーが生成されます。

```
// CS1704_c.cs
// compile with: /target:library /r:A2=cs1704.dll /r:A1=c:\\cs1704.dll
// CS1704 expected
extern alias A1;
extern alias A2;
```

コンパイラ エラー CS1705

エラー メッセージ
アセンブリ 'AssemblyName1' は、参照されているアセンブリ 'AssemblyName2' よりも新しいバージョンを含む 'TypeName' を使用します

参照アセンブリのバージョン番号より新しいバージョン番号の型を参照しています。

たとえば、A および B という 2 つのアセンブリがあるとします。A は、バージョン 2.0 でアセンブリ B に追加された `myClass` クラスを参照します。ところが、アセンブリ B への参照ではバージョン 1.0 が指定されています。コンパイラには、参照のバインディングに関して統一規則があり、バージョン 2.0 への参照を、バージョン 1.0 で解決することはできません。

使用例
このサンプルは、次の 4 つのコード モジュールで構成されています。

- バージョン属性以外はまったく同じ 2 つの DLL
- これらを参照する DLL
- クライアント

等価な DLL の 1 つ目を次に示します。

```
// CS1705_a.cs
// compile with: /target:library /out:c:\\cs1705.dll /keyfile:mykey.snk
[assembly:System.Reflection.AssemblyVersion("1.0")]
public class A
{
   public void M1() {}
   public class N1 {}
   public void M2() {}
   public class N2 {}
}

public class C1 {}
public class C2 {}
```

次に、AssemblyVersionAttribute 属性でバージョン 2.0 として指定されたアセンブリを示します。

```
// CS1705_b.cs
// compile with: /target:library /out:cs1705.dll /keyfile:mykey.snk
using System.Reflection;
[assembly:AssemblyVersion("2.0")]
public class A
{
   public void M2() {}
   public class N2 {}
   public void M1() {}
   public class N1 {}
}

public class C2 {}
public class C1 {}
```

このサンプルを CS1705ref.cs という名前のファイルで保存し、`/t:library /r:A2=.\bin2\CS1705a.dll /r:A1=.\bin1\CS1705a.dll` のようにフラグを指定してコンパイルします。

```
// CS1705_c.cs
// compile with: /target:library /r:A2=c:\\CS1705.dll /r:A1=CS1705.dll
extern alias A1;
extern alias A2;
using a1 = A1::A;
using a2 = A2::A;
using n1 = A1::A.N1;
```

```
using n2 = A2::A.N2;
public class Ref
{
   public static a1 A1() { return new a1(); }
   public static a2 A2() { return new a2(); }
   public static A1::C1 M1() { return new A1::C1(); }
   public static A2::C2 M2() { return new A2::C2(); }
   public static n1 N1() { return new a1.N1(); }
   public static n2 N2() { return new a2.N2(); }
}
```

次の例では、バージョン 1.0 の CS1705.dll アセンブリを参照しています。ところが、`Ref.A2().M2()` ステートメントで、CS1705_c.dll のクラスの A2 メソッドが参照されています。このメソッドは A2::A にエイリアスされた a2 を返しますが、A2 は **extern** ステートメントによってバージョン 2.0 を参照しているため、バージョンの不一致が発生します。

次の例では CS1705 エラーが生成されます。

```
// CS1705_d.cs
// compile with: /reference:c:\\CS1705.dll /reference:CS1705_c.dll
// CS1705 expected
class Tester
{
   static void Main()
   {
      Ref.A1().M1();
      Ref.A2().M2();
   }
}
```

コンパイラ エラー CS1706

エラー メッセージ
式に匿名メソッドを含めることはできません。

式の内部に匿名メソッドを挿入することはできません。

このエラーを解決するには

- 式の中では、通常の **delegate** を使用してください。

使用例
次のコードは CS1706 を生成します。

```
// CS1706.cs
using System;

delegate void MyDelegate();
class MyAttribute : Attribute
{
    public MyAttribute(MyDelegate d) { }
}

// Anonymous Method in Attribute declaration is not allowed.
[MyAttribute(delegate{/* anonymous Method in Attribute declaration */})]  // CS1706
class Program
{
}
```

コンパイラ エラー CS1708

エラー メッセージ
固定 バッファには、ローカルまたはフィールドをとおしてのみアクセスできます。

C# 2.0 の新しい機能として、**struct** 内でインライン配列を定義できる (つまり、構造体内に直接配列を定義できる) ことが挙げられます。これらの配列は、ローカル変数またはフィールドを介してのみアクセスでき、式の左辺で中間値として参照することはできません。また、**static** または **readonly** として宣言されたフィールドで、この配列にアクセスすることもできません。

このエラーを解決するには、配列変数を定義し、この変数にインライン配列を代入します。または、インライン配列を表すフィールドから、**static** 修飾子か **readonly** 修飾子を削除します。

使用例
次の例では CS1708 エラーが生成されます。

```
// CS1708.cs
// compile with: /unsafe
using System;

unsafe public struct Foo
{
    public fixed char name[10];
}

public unsafe class C
{
    public Foo UnsafeMethod()
    {
        Foo myFoo = new Foo();
        return myFoo;
    }

    static void Main()
    {
        C myC = new C();
        myC.UnsafeMethod().name[3] = 'a';  // CS1708
        // Uncomment the following 2 lines to resolve:
        // Foo myFoo = myC.UnsafeMethod();
        // myFoo.name[3] = 'a';

        // The field cannot be static.
        C._foo1.name[3] = 'a';  // CS1708

        // The field cannot be readonly.
        myC._foo2.name[3] = 'a';  // CS1708
    }

    static readonly Foo _foo1;
    public readonly Foo _foo2;
}
```

コンパイラ エラー CS1713

エラー メッセージ
型 'Typename1' のメタデータ名のビルド中に予期しないエラーが発生しました - '理由'

通常、このエラーは、内部コンパイル エラーが原因で発生します。メタデータの名前を短くするなど、コードにわずかな変更を加え、再コンパイルすることによって、このエラーを解決できる場合があります。

コンパイラ エラー CS1714

エラー メッセージ
'TypeName1' の基本クラスまたはインターフェイスは、解決することができなかったか、または無効です

TypeName1 の継承元のクラスまたはインターフェイスは、コンパイラが見つけることができないなどの理由により解決できないか無効です。このエラーを解決するには、このいずれの問題に該当するのかを調べ、型の場所を正しく指定するか、基本クラスでコンパイラ エラーを修正します。

コンパイラ エラー CS1715

エラー メッセージ
'Type1': オーバーライドされたメンバ 'MemberName' に対応するために、型は 'Type2' でなければなりません

このエラーは、コンパイラ エラー CS0508 とほぼ同じです。ただし、CS0508 が戻り値の型を持つメソッドに対してのみ適用されるのに対し、CS1715 は、"戻り値の型" ではなく "型" だけを持つプロパティおよびインデクサに適用されます。

使用例
次のコードでは、CS1715 エラーが生成されます。

```
// CS1715.cs
abstract public class Base
{
    abstract public int myProperty
    {
        get;
        set;
    }
}

public class Derived : Base
{
    int myField;
    public override double myProperty  // CS1715
    // try the following line instead
    // public override int myProperty
    {
        get { return myField; }
        set { myField;= value; }
    }

    public static void Main()
    {
        Derived d = new Derived();
        d.myProperty = 5;
    }
}
```

コンパイラ エラー CS1716

エラー メッセージ
'System.Runtime.CompilerServices.FixedBuffer' 属性を使用しません。'fixed' フィールド修飾子を使用してください。

このエラーは、フィールド宣言など、固定サイズの配列宣言が記述されたアンセーフ コード セクションで発生します。この属性は使用しないでください。代わりに、**fixed** キーワードを使用します。

使用例
次の例では CS1716 エラーが生成されます。

```
// CS1716.cs
// compile with: /unsafe
using System;
using System.Runtime.CompilerServices;

public struct UnsafeStruct
{
    [FixedBuffer(typeof(int), 4)]  // CS1716
    unsafe public int aField;
    // Use this single line instead of the above two lines.
    // unsafe public fixed int aField[4];
}

public class TestUnsafe
{
    static int Main()
    {
        UnsafeStruct us = new UnsafeStruct();
        unsafe
        {
            if (us.aField[0] == 0)
                return us.aField[1];
            else
                return us.aField[2];
        }
    }
}
```

コンパイラ エラー CS1719

エラー メッセージ
Win32 リソース ファイル 'ファイル名' を読み込み中にエラーが発生しました -- '理由'

エラー メッセージが示している理由 ("ファイルが見つかりません" や "アクセスが拒否されました" など) により、Win32 リソース ファイルの読み取りに失敗しました。このエラーは、理由として記述されている問題を訂正することによって解決できます。

コンパイラ エラー CS1721

エラー メッセージ
クラス 'クラス' に複数の基本クラスを指定することはできません: 'class_1' および 'class_2'

このエラー メッセージの最も一般的な原因としては、多重継承の使用があります。C# では、複数のクラスを継承することはできません。クラス宣言で、1 つの基本クラスを指定したら、その後に続けて指定できる型はインターフェイスに限られます。

使用例

次の例では CS1721 エラーが生成されます。

```
// CS1721.cs
public class A {}
public class B {}
public class MyClass : A, B {}   // CS1721
```

コンパイラ エラー CS1722

エラー メッセージ
基本クラス 'クラス' は、他のインターフェイスの前に指定されなければなりません。

継承するクラスや実装するインターフェイスを指定する場合は、クラス名を最初に指定する必要があります。

使用例
次の例では CS1722 エラーが生成されます。

```
// CS1722.cs
// compile with: /target:library
public class A {}
interface I {}

public class MyClass : I, A {}   // CS1722
public class MyClass2 : A, I {}   // OK
```

コンパイラ エラー CS1724

エラー メッセージ
引数が 'System.Runtime.InteropServices.DefaultCharSetAttribute' に指定された値は無効です。

このエラーは、DefaultCharSetAttribute クラスに無効な引数が指定されると生成されます。

使用例
次の例では CS1724 エラーが生成されます。

```
// CS1724.cs
using System.Runtime.InteropServices;
// To resolve, replace 42 with a valid CharSet value.
 [module:DefaultCharSetAttribute((CharSet)42)]   // CS1724
class C {

[DllImport("F.Dll")]
extern static void FW1Named();

static void Main() {}
}
```

コンパイラ エラー CS1728

エラー メッセージ
型' のメンバであるため、デリゲートを 'メンバ' にバインドできません。

デリゲートを **Nullable** 型のメンバにバインドできません。

使用例
次の例では CS1728 エラーが生成されます。

```
// CS1728.cs
// compile with: /W:2
class Test
{
   delegate T GetT<T>();
   delegate T GetT1<T>(T t);

   delegate bool E(object o);
   delegate int I();
   delegate string S();

   static void Main()
   {
      int? x = null;
      int? y = 5;

      GetT<int> d1 = x.GetValueOrDefault;   // CS1728
      GetT<int> d2 = y.GetValueOrDefault;   // CS1728
      GetT1<int> d3 = x.GetValueOrDefault;   // CS1728
      GetT1<int> d4 = y.GetValueOrDefault;   // CS1728
   }
}
```

コンパイラ エラー CS1900

エラー メッセージ
警告レベルの範囲は 0-4 です。

/warn コンパイラ オプションに使用できる値は、5 つの値 (0、1、2、3、または 4) のうち 1 つだけです。その他の値を **/warn** に渡すと、CS1900 エラーが生成されます。

次の例では CS1900 エラーが生成されます。

```
// CS1900.cs
// compile with: /W:5
// CS1900 expected
class x
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1902

エラー メッセージ
/debug のオプション 'option' が無効です。full か pdbonly を指定してください。

無効なオプションが /debug コンパイラ オプションに渡されました。

次の例では CS1902 エラーが生成されます。

```
// CS1902.cs
// compile with: /debug:x
// CS1902 expected
class x
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS1906

エラー メッセージ
無効なオプション 'オプション' です。リソースの参照可能範囲は 'public' または 'private' でなければなりません。

このエラーは、/resource (出力へのリソース ファイルの埋め込み) か /linkresource (.NET Framework リソースへのリンク) のいずれかのコマンド ライン オプションが無効であることを示します。**/resource** か **/linkresource** のコマンド ライン オプションの構文をチェックし、アクセシビリティ修飾子が **public** または **private** として宣言されていることを確認してください。

コンパイラ エラー CS1908

エラー メッセージ
DefaultValue 属性への引数の型は、パラメータ型と一致しなければなりません

このエラーは、DefaultValueAttribute 属性値に不適切な引数を使用したときに生成されます。パラメータの型に一致する値を使用します。

使用例
次の例では CS1908 エラーが生成されます。

```
// CS1908.cs
// compile with: /target:library
using System.Runtime.InteropServices;

public interface ISomeInterface
{
   void Bad([Optional] [DefaultParameterValue("true")] bool b);   // CS1908

   void Good([Optional] [DefaultParameterValue(true)] bool b);   // OK
}
```

コンパイラ エラー CS1909

エラー メッセージ
DefaultValue 属性は型 '型' のパラメータ上で適用できません

CS1909 は、このパラメータの型で適用できない `DefaultValue` 属性を使用したときに生成されます。

使用例
次の例では CS1909 エラーが生成されます。

```
// CS1909.cs
// compile with: /target:library
using System.Runtime.InteropServices;

public interface ISomeInterface
{
   void Test1([DefaultParameterValue(new int[] {1, 2})] int[] arr1);   // CS1909

   void Test2([DefaultParameterValue("Test String")] string s);   // OK
}
```

コンパイラ エラー CS1910

エラー メッセージ
型 '型' の引数は DefaultValue 属性に適用できません

パラメータの型がオブジェクトである場合、DefaultParameterValueAttribute の引数は **null**、整数型、浮動小数点、**bool**、**string**、**enum**、**char** のいずれかにする必要があります。Type 型や配列型の引数は使用できません。

使用例
次の例では CS1910 エラーが生成されます。

```
// CS1910.cs
// compile with: /target:library
using System.Runtime.InteropServices;

public interface MyI
{
   void Test([DefaultParameterValue(typeof(object))] object o);   // CS1910
}
```

コンパイラ エラー CS2000

エラー メッセージ
コンパイラの初期化に失敗しました。

このエラーは、初期化エラーを示します。

セットアップを使用して Visual Studio または .NET Framework SDK を再インストールするか、既存のインストールを修復します。

それでもエラーが解消しない場合は、製品サポートまでお問い合わせください。

コンパイラ エラー CS2001

エラー メッセージ
ソース ファイル 'file' が見つかりませんでした。

ソース ファイル名がコンパイラに渡されましたが、見つかりませんでした。ファイル名のスペルおよびファイルの場所を確認します。

コンパイラ エラー CS2003

エラー メッセージ
応答ファイル 'file' が複数回含まれました。

応答ファイルがコンパイラに複数回渡されました。コンパイラに対しては、出力ファイルごとに 1 回しか応答ファイルを渡すことができません。

応答ファイルの詳細については、「@ (応答ファイルの指定)」を参照してください。

コンパイラ エラー CS2005

エラー メッセージ
'オプション' オプションのファイルが指定されていません。

コンパイラ オプションの指定が不完全です。

たとえば、/recurse を使用する場合は、**/recurse:***filename***.cs** のように、検索するファイルを指定する必要があります。

使用例
次の例では CS2005 エラーが生成されます。

```
// CS2005.cs
// compile with: /recurse:
// CS2005 expected
class x
{
   public static void Main() {}
}
```

コンパイラ エラー CS2006

エラー メッセージ
コマンドライン構文エラー: 'オプション' オプションの ':テキスト' がありません。

option の構文にテキストを追加する必要があります。詳細については、「コンパイラ オプション」を参照してください。

コンパイラ エラー CS2007

エラー メッセージ
認識できないコマンド ライン オプション : 'option'

スラッシュ (/) で始まる文字列がコンパイラに渡されましたが、コンパイラ オプションではありませんでした。

次の例では CS2007 エラーが生成されます。

```
// CS2007.cs
// compile with: /recur
// CS2007 expected
class x
{
   public static void Main() {}
}
```

コンパイラ エラー CS2008

エラー メッセージ
入力が指定されていません。

コンパイラが起動され、コンパイラ オプションが指定されましたが、ソース コード ファイルが渡されませんでした。

コンパイラ エラー CS2011

エラー メッセージ
応答ファイル 'file' を開けません。

コンパイルで応答ファイルが指定されましたが、コンパイラではそのファイルを見つけて開くことができませんでした。

応答ファイルの詳細については、「@ (応答ファイルの指定)」を参照してください。

コンパイラ エラー CS2012

エラー メッセージ
書き込みモードで 'ファイル' を開けません

/bugreport:*file* コンパイラ オプションの使用時に、ファイルを開いて書き込むことができませんでした。指定したファイル名が有効であり、ファイルが読み取り専用でないことを確認してください。

コンパイラ エラー CS2013

エラー メッセージ
イメージの基数 'value' が無効です。

無効な値 (数値以外) が、/baseaddress コンパイラ オプションに渡されました。

次の例では CS2013 エラーが生成されます。

```
// CS2013.cs
// compile with: /target:library /baseaddress:x
// CS2013 expected
class MyClass
{
}
```

コンパイラ エラー CS2015

エラー メッセージ
'ファイル' はテキスト ファイルではなく、バイナリ ファイルです。

コンパイラに渡されたファイルがバイナリ ファイルでした。コンパイラではソース コード ファイルを予期しています。

コンパイラ エラー CS2016

エラー メッセージ
コード ページ 'codepage' は無効か、インストールされていません。

/codepage コンパイラ オプションに無効な値が渡されました。

次の例では CS2016 エラーが生成されます。

```
// CS2016.cs
// compile with: /codepage:x
// CS2016 expected
class MyClass
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS2017

エラー メッセージ
モジュールまたはライブラリをビルドする場合は /main を指定できません。

/target:library の作成時には、メイン エントリ ポイントを指定できません。

次の例では CS2017 エラーが生成されます。

```
// CS2017.cs
// compile with: /main:MyClass /target:library
// CS2017 expected
class MyClass
{
   public static void Main()
   {
   }
}
```

コンパイラ エラー CS2018

エラー メッセージ
メッセージ ファイル 'cscmsgs.dll' が見つかりません。

コンパイラのエラー メッセージと警告メッセージが格納されている .dll ファイルが見つかりませんでした。このファイルは、ほかのコンパイラ サポート ファイルと同じディレクトリに存在する必要があります。

コンパイラ エラー CS2019

エラー メッセージ
/target のターゲットの種類が無効です : exe、winexe、library または module のいずれかを指定してください。

/target コンパイラ オプションを使用しましたが、無効なパラメータが渡されました。このエラーを解決するには、出力ファイルに適した形式の **/target** オプションを使用してプログラムを再コンパイルします。

次の例では CS2017 エラーが生成されます。

```
// CS2019.cs
// compile with: /target:libra
// CS2019 expected
class MyClass
{
}
```

コンパイラ エラー CS2020

エラー メッセージ
入力ファイルの最初のセットのみが、module 以外のターゲットをビルドすることができます。

複数出力のコンパイルでは、最初の出力ファイルが /target:exe、/target:winexe、または /target:library で作成される必要があります。それ以降の出力ファイルは、/target:module で作成される必要があります。

コンパイラ エラー CS2021

エラー メッセージ
ファイル名 'file' が長すぎるか無効です。

C# コンパイラに渡されるすべてのファイル名の長さは、Windows のヘッダー ファイルで定義されている **_MAX_PATH** を超えないようにする必要があります。コンパイラは、次の場合にこのエラーを生成します。

- ファイル名 (パスを含む) が **_MAX_PATH** よりも長い。
- ファイル名に無効な文字が含まれている。
- ワイルドカードが許可されていないファイル名 (リソース ファイル名など) に、ワイルドカードが含まれている。

コンパイラ エラー CS2022

エラー メッセージ
ソース ファイル名の前にオプション /out および /target を指定する必要があります。

コマンド ラインでは、/out (出力ファイル名の設定) コンパイラ オプションと /target (出力ファイル形式の指定) コンパイラ オプションをソース コード ファイルの前に置く必要があります。

コンパイラ エラー CS2024

エラー メッセージ
ファイル セクション アライメント番号 '#' が無効です。

無効な値が /filealign コンパイラ オプションに渡されました。

次の例では CS2024 エラーが生成されます。

```
// CS2024.cs
// compile with: /filealign:ex
// CS2024 expected
class MyClass
{
   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 1) CS2029

エラー メッセージ
'/定義' の値が無効です。'identifier' は有効な識別子ではありません。

この警告は、/define オプションで使用されている値に無効な文字が含まれている場合に発生します。

コンパイラ エラー CS2032

エラー メッセージ
文字 '文字' は、コマンド ラインまたは応答ファイルに許可されていません

csc.exe の応答ファイルおよびコマンド ライン オプションに、下位 ASCII 制御文字 (0 ～ 31) やパイプ (|) 文字を使用することはできません。統合開発環境 (IDE: Integrated Development Environment) と同様、無効な文字はコマンド ライン プロセッサによって排除されるため、通常、コマンド ラインからこのエラーが直接生成されることはありません。このエラーは、次のように応答ファイルを使用して生成できます。

このエラーを生成するには

1. [マイ ドキュメント] に`/target:exe /out:cs|2032.exe cs2032.cs` という行を含むファイルを CS2032.rsp というファイル名で作成します。
2. [マイ ドキュメント] に、なんらかのデータが含まれた適当なファイルを cs2032.cs というファイル名で作成します。
3. [スタート] ボタンをクリックし、[すべてのプログラム] をポイントします。次に [Microsoft Visual Studio 2005] をポイントし、[Visual Studio Tools] をポイントして、[Visual Studio 2005 コマンド プロンプト] をクリックします。

   [Visual Studio 2005 コマンド プロンプト] ウィンドウが表示されます。
4. [Visual Studio 2005 コマンド プロンプト] ウィンドウで、現在のディレクトリを [マイ ドキュメント] に変更します。
5. [Visual Studio 2005 コマンド プロンプト] から「`csc @cs2032.rsp`」を実行します。
6. CS2032 エラー メッセージが表示されます。
7.

1.

コンパイラ エラー CS2033

エラー メッセージ
同じ短いファイル名を使用している長いファイル名が既に存在するとき、短いファイル名 'filename' を作成することはできません。

名前が 8 文字を超える任意の C# ファイルをコンパイルします。次に、最初に作成したファイルの名前から先頭の数文字を使って (先頭の 6 文字に "~1" を付けるなど)、別のファイルをコンパイルします。2 回目のコンパイルで、このエラーが発生します。

このエラーを解決するには、短い方のファイル名を変更して、長い方のファイル名と競合しないようにします。

コンパイラ エラー CS2034

エラー メッセージ
extern エイリアスを宣言する /reference オプションにはファイル名が 1 つだけ指定できます。複数のエイリアスまたはファイル名を指定するには、複数の /reference オプションを使用してください。

2 つのエイリアスまたはファイル名を指定するには、次のように、**/reference** オプションを 2 つ指定します。

使用例
次のコードでは、CS2034 エラーが生成されます。

```
// CS2034.cs
// compile with: /r:A1=cs2034a1.dll;A2=cs2034a2.dll
// to fix, compile with: /r:A1=cs2034a1.dll /r:A2=cs2034a2.dll
// CS2034
extern alias A1;
extern alias A2;
using System;
```

コンパイラ エラー CS2035

エラー メッセージ
コマンド ライン構文エラー: 'compiler_option' オプションには ':<number>' が必要です

一部のコンパイラ オプションには値が必要です。

使用例
次の例では CS2035 エラーが生成されます。

```
// CS2035.cs
// compile with: /baseaddress
// CS2035 expected
```

コンパイラの警告 (レベル 1) CS3027

エラー メッセージ
基本インターフェイス 'type_2' が CLS 準拠でないため、'type_1' は CLS 準拠ではありません。

非 CLS 準拠型は、CLS 準拠型の基本型にはなれません。

使用例
次の例では、その型を非 CLS 準拠型にすることにより、シグネチャに非 CLS 準拠型を使用するメソッドを含むインターフェイスを示します。

```
// CS3027.cs
// compile with: /target:library
public interface IBase
{
   void IMethod(uint i);
}
```

次の例では CS3027 エラーが生成されます。

```
// CS3027_b.cs
// compile with: /reference:CS3027.dll /target:library /W:1
[assembly:System.CLSCompliant(true)]
public interface IDerived : IBase {}
```

コンパイラ エラー CS5001

エラー メッセージ
プログラム 'プログラム' は、エントリ ポイントに適切な静的 'Main' メソッドを含んでいません

このエラーは、実行可能ファイルとして生成されるコードに、Main メソッドが見つからない場合に発生します。このエラーは、エントリ ポイントとして使用される **Main** 関数が小文字で (**main** のように) 入力されていたり、**`Main`** が `static` として宣言されていない場合にも発生します。

使用例
次の例では、`CS5001` エラーが生成されます。

```
// CS5001.cs
// CS5001 expected
public class a
{
   // Uncomment the following line to resolve.
   // public static void Main() {}
}
```

コンパイラ エラー CS0025

エラー メッセージ
標準ライブラリ ファイル 'file' が見つかりません。

必要なファイルが見つかりませんでした。パスが正しいことと、ファイルが存在することを確認してください。

ファイルが Visual Studio のシステム ファイルである場合は、Visual Studio の修復インストールを試みるか、再インストールする必要があります。

コンパイラの警告 (レベル 1) CS0183

エラー メッセージ
式は常に指定された型 ('type') です。

条件付きステートメントが常に **true** になる場合、条件付きステートメントは不要です。この警告は、型の評価に **is** 演算子を使用した場合に発生します。評価の対象が値型の場合、条件判定は不要です。

次の例では警告 CS0183 が生成されます。

```
// CS0183.cs
// compile with: /W:1
using System;
public class Test
{
   public static void F(Int32 i32, String str)
   {
      if (str is Object)          // OK
         Console.WriteLine( "str is an object" );
      else
         Console.WriteLine( "str is not an object" );

      if (i32 is Object)   // CS0183
         Console.WriteLine( "i32 is an object" );
      else
         Console.WriteLine( "i32 is not an object" ); // never reached
   }


   public static void Main()
   {

      F(0, "CS0183");
      F(120, null);
   }
}
```

コンパイラの警告 (レベル 1) CS0184

エラー メッセージ
式は指定された型 ('type') ではありません。

テストしている変数が ***type*** として宣言されることも ***type*** から派生することもないため、式は **true** になりません。

次の例では警告 CS0184 が生成されます。

```
// CS0184.cs
// compile with: /W:1
class MyClass
{
   public static void Main()
   {
      int i = 0;
      if (i is string)   // CS0184
         i++;
   }
}
```

コンパイラの警告 (レベル 1) CS0420

エラー メッセージ
'識別子': volatile フィールドへの参照は、volatile として扱われません。

通常、volatile フィールドを、**ref** パラメータや **out** パラメータで渡すことはしません。これらのパラメータは、関数のスコープ内では volatile として扱われないためです。ただし、インタロック API を呼び出す場合など、この規則には例外があります。意図的に volatile フィールドを参照パラメータとして使用する場合は、他の警告と同様、#pragma warning を使って警告を無効にできます。

次の例では CS0420 エラーが生成されます。

```
// CS0420.cs
// compile with: /W:1
using System;

class TestClass
{
   private volatile int i;

   public void TestVolatile(ref int ii)
   {
   }

   public static void Main()
   {
      TestClass x = new TestClass();
      x.TestVolatile(ref x.i);   // CS0420
   }
}
```

コンパイラの警告 (レベル 1) CS0465

エラー メッセージ
'Finalize' メソッドを導入すると、デストラクタの呼び出しに影響する可能性があります。デストラクタを宣言しようとしましたか?

この警告は、シグネチャが `public virtual void Finalize` のメソッドを使用してクラスを作成した場合に発生します。

このようなクラスを基本クラスとして使用し、派生クラスでデストラクタを定義した場合、デストラクタによって Finalize ではなく、基本クラスの `Finalize` メソッドがオーバーライドされます。

使用例
次の例では CS0465 警告が生成されます。

```
// CS0465.cs
// compile with: /target:library
class A
{
   public virtual void Finalize() {}   // CS0465
}

// OK
class B
{
   ~B() {}
}
```

コンパイラの警告 (レベル 1) CS0602

エラー メッセージ
'old_feature' 機能は使用しないでください。'new_feature' を代わりに使用してください。

コードで使用する言語機能 (*old_feature*) はまだサポートされていますが、今後のリリースではサポートされなくなる可能性があります。代わりに、推奨される構文 (*new_feature*) を使用してください。

コンパイラの警告 (レベル 1) CS0612

エラー メッセージ
'member' は古い形式です。

クラス デザイナによって、メンバが Obsolete 属性でマークされました。これは、クラスの今後のバージョンでこのメンバがサポートされない可能性があることを示します。

次の例では、旧式のメンバにアクセスしたときに警告 CS0612 が生成されるようすが示されています。

```
// CS0612.cs
// compile with: /W:1
using System;

class MyClass
{
   [Obsolete]
   public static void ObsoleteMethod()
   {
   }

   [Obsolete]
   public static int ObsoleteField;
}

class MainClass
{
   static public void Main()
   {
      MyClass.ObsoleteMethod();    // CS0612 here: method is deprecated
      MyClass.ObsoleteField = 0;   // CS0612 here: field is deprecated
   }
}
```

コンパイラの警告 (レベル 1) CS0626

エラー メッセージ
メソッド 'method' は外部に設定されていて属性を持っていません。外部の実装を指定できるよう DllImport 属性を追加してください。

extern としてマークされたメソッドは、DllImport などの属性でもマークされる必要があります。

属性はメソッドの実装場所を指定します。プログラムでは、実行時にこの情報が必要になります。

次の例では警告 CS0626 が生成されます。

```
// CS0626.cs
// compile with: /warnaserror
using System.Runtime.InteropServices;

public class MyClass
{
   static extern public void mf(); // CS0626
   // try the following line
   // [DllImport("mydll.dll")] static extern public void mf();

   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 1) CS0658

エラー メッセージ
'attribute modifier' は認識できる属性の場所ではありません。このブロック内の属性はすべて無視されます。

無効な属性の修飾子が指定されました。詳細については、「属性の対象」を参照してください。

次の例では CS0658 エラーが生成されます。

```
// CS0658.cs
using System;
public class TestAttribute : Attribute{}
[badAttributeLocation: Test]   // CS0658, badAttributeLocation is invalid
class ClassTest
{
   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 1) CS0672

エラー メッセージ
メンバ 'member1' は古い形式のメンバ 'member2 をオーバーライドします。Obsolete 属性を 'member1' に追加してください。

Obsolete としてマークされたメソッドに対する **override** が見つかりました。しかし、オーバーライドするメソッド自体は **Obsolete** としてマークされていませんでした。オーバーライドするメソッドは、呼び出されると CS0612 を生成します。

メソッドの宣言を確認し、メソッドとそのすべてのオーバーライドを **Obsolete** としてマークする必要があるかどうかを明示的に指定してください。

次の例では警告 CS0672 が生成されます。

```
// CS0672.cs
// compile with: /W:1
class MyClass
{
   [System.Obsolete]
   public virtual void ObsoleteMethod()
   {
   }
}

class MyClass2 : MyClass
{
   public override void ObsoleteMethod()   // CS0672
   {
   }
}

class MainClass
{
   static public void Main()
   {
   }
}
```

コンパイラの警告 (レベル 1) CS0684

エラー メッセージ
'interface' インターフェイスは、'CoClassAttribute' でマークされていますが、'ComImportAttribute' ではマークされていません。

インターフェイスに **CoClassAttribute** を指定した場合は、**ComImportAttribute** も指定する必要があります。

次の例では CS0684 エラーが生成されます。

```
// CS0684.cs
// compile with: /W:1
using System;
using System.Runtime.InteropServices;

[CoClass(typeof(C))] // CS0684
// try the following line instead
// [CoClass(typeof(C)), ComImport]
interface I
{
}

class C
{
   static void Main() {}
}
```

コンパイラの警告 (レベル 1) CS0688

エラー メッセージ
'method1' はリンク要求を含んでいますが、リンク要求を含んでいない 'method2' をオーバーライドまたは実装します。セキュリティに問題が発生する可能性があります。

派生クラスのメソッドで設定されたリンク確認要求は、基本クラスのメソッドを呼び出すことによって容易に迂回されてしまいます。万全なセキュリティを確保するには、基本クラスのメソッドについても、リンク確認要求を使用する必要があります。詳細については、「Demand と LinkDemand」を参照してください。

使用例

次の例では CS0688 エラーが生成されます。基本クラスに変更を加えずにこの警告を解決するには、オーバーライドするメソッドからセキュリティ属性を削除します。これで、セキュリティ上の問題が解決されるわけではありません。

```
// CS0688.cs
// compile with: /W:1
using System;
using System.Security.Permissions;

class Base
{
    //Uncomment the following line to close the security hole
    //[FileIOPermission(SecurityAction.LinkDemand, All=@"C:\\")]
    public virtual void DoScaryFileStuff()
    {
    }
}

class Derived: Base
{
    [FileIOPermission(SecurityAction.LinkDemand, All=@"C:\\")] // CS0688
    public override void DoScaryFileStuff()
    {
    }
    static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS1030

エラー メッセージ
#warning: 'text'

#warning ディレクティブで定義された警告のテキストを表示します。

次の例では、ユーザー定義の警告の作成方法を示しています。

```
// CS1030.cs
class Sample
{
   static void Main()
   {
      #warning Let's give a warning here   // CS1030
   }
}
```

コンパイラの警告 (レベル 1) CS1200

エラー メッセージ
機能 '無効な機能' は使用できません。'有効な機能' を使用してください。

使用しようとしている機能は、現在推奨されていません。有効な機能を使用するようにコードを変更してください。

コンパイラの警告 (レベル 1) CS1201

エラー メッセージ
機能 '無効な機能' は使用できません。代わりに '有効な機能' を使用してください。

使用しようとしている機能は、現在推奨されていません。有効な機能を使用するようにコードを変更してください。

コンパイラの警告 (レベル 1) CS1202

エラー メッセージ
機能 '無効な機能' は使用できません。'有効な機能' を使用してください。

使用しようとしている機能は、現在推奨されていません。有効な機能を使用するようにコードを変更してください。

コンパイラの警告 (レベル 1) CS1203

機能 '無効な機能' は使用できません。'有効な機能' を使用してください。

使用しようとしている機能は、現在推奨されていません。有効な機能を使用するようにコードを変更してください。

コンパイラの警告 (レベル 1) CS1522

エラー メッセージ
空の switch ブロックです。

case ステートメントまたは **default** ステートメントがない switch ブロックが検出されました。**switch** ブロックには、1 つ以上の **case** ステートメントまたは **default** ステートメントが必要です。

次の例では警告 CS1522 が生成されます。

```
// CS1522.cs
// compile with: /W:1
using System;
class x
{
   public static void Main()
   {
      int i = 6;

      switch(i)   // CS1522
      {
         // add something to the switch block, for example:
         /*
         case (5):
            Console.WriteLine("5");
            return;
         default:
            Console.WriteLine("not 5");
            return;
         */
      }
   }
}
```

コンパイラの警告 (レベル 1) CS1570

エラー メッセージ
'コンストラクト' の XML コメントの形式が正しくありません — '理由'

/doc を使用するときは、ソース コードに含まれるコメントを XML 形式にする必要があります。XML マークアップが付いたエラーがあると、警告 CS1570 が生成されます。次に例を示します。

- <exception> タグなどで **cref** に文字列を渡す場合は、その文字列を二重引用符で囲む必要があります。
- 終了タグのない <seealso> などのタグを使用する場合は、右山かっこの前にスラッシュを付ける必要があります。
- 記述するテキストの中で不等号 (より大) または不等号 (より小) を使用する場合は、それぞれ > および < で表す必要があります。
- <include> タグのファイル属性またはパス属性が見つからなかったか、または不適切な形式になっていました。

次の例では警告 CS1570 が生成されます。

```
// CS1570.cs
// compile with: /W:1
namespace ns
{
   // the following line generates CS1570
   /// <summary> returns true if < 5 </summary>
   // try this instead
   // /// <summary> returns true if &lt;5 </summary>

   public class MyClass
   {
      public static void Main ()
      {
      }
   }
}
```

コンパイラの警告 (レベル 1) CS1574

エラー メッセージ
'コンストラクト' の XML コメントに cref 属性 '項目' が指定されていますが、解決できません

たとえば **<exception>** タグ内などで **cref** タグに渡される文字列が、現在のビルド環境内では使用できないメンバを参照しています。**cref** タグに渡す文字列は、メンバまたはフィールドの構文的に正しい名前であることが必要です。

詳細については、「ドキュメント コメントとして推奨されるタグ」を参照してください。

次の例では警告 CS1574 が生成されます。

```
// CS1574.cs
// compile with: /W:1 /doc:x.xml
using System;

/// <exception cref="System.Console.WriteLin">An exception class.</exception>   // CS1574
// instead, uncomment and try the following line
// /// <exception cref="System.Console.WriteLine">An exception class.</exception>
class EClass : Exception
{
}

class TestClass
{
   public static void Main()
   {
      try
      {
      }
      catch(EClass)
      {
      }
   }
}
```

コンパイラの警告 (レベル 1) CS1580

エラー メッセージ
XML コメントの cref 属性で、パラメータ 'parameter number' の型が無効です。

メソッドのオーバーロード形式の参照時に、構文エラーが検出されました。このエラーは、通常、型ではなくパラメータ名が指定されたことを示します。生成される XML ファイルでは、不正な形式の行が表示されます。

次の例では警告 CS1580 が生成されます。

```
// CS1580.cs
// compile with: /W:1 /doc:x.xml
using System;

/// <seealso cref="Test(i)"/>   // CS1580
// try the following line instead
// /// <seealso cref="Test(int)"/>
public class MyClass
{
   /// <summary>help text</summary>
   public static void Main()
   {
   }

   /// <summary>help text</summary>
   public void Test(int i)
   {
   }

   /// <summary>help text</summary>
   public void Test(char i)
   {
   }
}
```

コンパイラの警告 (レベル 1) CS1581

エラー メッセージ
XML コメントの cref 属性の戻り値の型が無効です。

メソッドの参照時に、無効な戻り値の型が原因のエラーが検出されました。

使用例
次の例では警告 CS1581 が生成されます。

```
// CS1581.cs
// compile with: /W:1 /doc:x.xml

/// <summary>help text</summary>
public class MyClass
{
    /// <summary>help text</summary>
    public static void Main()
    {
    }

    /// <summary>help text</summary>
    public static explicit operator int(MyClass f)
    {
        return 0;
    }
}

/// <seealso cref="MyClass.explicit operator intt(MyClass)"/>  // CS1581
// try the following line instead
// /// <seealso cref="MyClass.explicit operator int(MyClass)"/>
public class MyClass2
{
}
```

コンパイラの警告 (レベル 1) CS1584

エラー メッセージ
'member' の XML コメントで、cref 属性 'invalid_syntax' の構文が正しくありません。

ドキュメント コメントのタグに渡されたパラメータの 1 つに、無効な構文がありました。詳細については、「ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS1584 エラーが生成されます。

```
// CS1584.cs
// compile with: /W:1 /doc:CS1584.xml
/// <remarks>Test class</remarks>
public class Test
{
   /// <remarks>Called in <see cref="Test.Mai()n"/>.</remarks>   // CS1584
   // try the following line instead
   // /// <remarks>Called in <see cref="Test.Main()"/>.</remarks>
   public static void Test2() {}

   /// <remarks>Main method</remarks>
   public static void Main() {}
}
```

コンパイラの警告 (レベル 1) CS1589

エラー メッセージ
ファイル 'file' の XML フラグメント 'fragment' を含めることができません -- reason

ファイル (*file*) を参照した、<include> タグの構文 (*fragment*) は、***reason*** に示された理由により無効です。

生成される XML ファイルには、不正な形式の行が配置されます。

次の例では警告 CS1589 が生成されます。

```
// CS1589.cs
// compile with: /W:1 /doc:CS1589_out.xml

/// <include file='CS1589.doc' path='MyDocs/MyMembers[@name="test"]/' />   // CS1589
// try the following line instead
// /// <include file='CS1589.doc' path='MyDocs/MyMembers[@name="test"]/*' />
class Test
{
   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 1) CS1590

エラー メッセージ
無効な XML のインクルード要素です -- ファイル属性がありません。

<include> タグに渡された **path** 属性または **doc** 属性が、不足しているか不完全です。

次の例では警告 CS1590 が生成されます。

```
// CS1590.cs
// compile with: /W:1 /doc:x.xml

/// <include path='MyDocs/MyMembers[@name="test"]/*' />   // CS1590
// try the following line instead
// /// <include file='x.doc' path='MyDocs/MyMembers[@name="test"]/*' />
class Test
{
   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 1) CS1592

エラー メッセージ
コメント ファイルの中の XML の形式が正しくありません -- 'reason'

<include> タグで指定されたファイルで問題が見つかりました。原因は ***reason*** で報告されています。

コンパイラの警告 (レベル 1) CS1598

エラー メッセージ
XML パーサーを以下の理由で読み込めませんでした: '理由'。XML ドキュメント ファイル 'file' は生成されません。

/doc オプションが指定されましたが、コンパイラでは msxml3.dll を見つけて読み込むことができませんでした。msxml3.dll ファイルがインストールおよび登録されているかどうかを確認してください。

コンパイラの警告 (レベル 1) CS1607

エラー メッセージ
アセンブリの生成 -- reason

コンパイルのアセンブリ作成フェーズで警告が発生しました。

64 ビット アプリケーションを 32 ビット プラットフォームで作成する場合、64 ビット バージョンのすべての参照アセンブリが対象プラットフォームにインストールされていることを確認する必要があります。

x86 固有のすべての共通言語ランタイム (CLR: Common Language Runtime) アセンブリには、対応する 64 ビットのアセンブリがあります (すべての CLR アセンブリが全プラットフォームに存在します)。このため、CLR アセンブリについての CS1607 警告は無視してかまいません。

詳細については、「AI.exe ツールのエラーと警告」を参照してください。

コンパイラの警告 (レベル 1) CS1616

エラー メッセージ
オプション 'option' はソース ファイルまたは追加されたモジュールで指定された属性 'attribute' をオーバーライドします

この警告は、ソース ファイル内のアセンブリ属性 AssemblyKeyFile または AssemblyKeyName が、コマンド ライン オプション /keyfile または /keycontainer (またはプロジェクトのプロパティで指定されたキー ファイル名またはキー コンテナ) と競合している場合に発生します。

次の例では、`cs1616.snk` というキー ファイルを想定しています。このファイルは、コマンド ラインで次のように入力して生成します。

```
sn -k CS1616.snk
```

次の例では CS1616 エラーが生成されます。

```
// CS1616.cs
// compile with: /keyfile:cs1616.snk
using System.Reflection;

// To fix the error, remove the next line
[assembly: AssemblyKeyFile("cs1616b.snk")]  // CS1616

class C
{
  public static void Main()
  {
  }
}
```

コンパイラの警告 (レベル 1) CS1633

エラー メッセージ
認識できない #pragma ディレクティブです。

C# コンパイラでサポートされていないプラグマが使用されています。このエラーを解決するには、サポートされているプラグマだけを使うようにします。

次の例では CS1633 エラーが生成されます。

```
// CS1633.cs
// compile with: /W:1
#pragma unknown  // CS1633

class C
{
   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 1) CS1634

エラー メッセージ
disable または restore を指定してください。

このエラーは、disable や restore が省略されているなど、#pragma warning 句が間違った形式で指定されている場合に発生します。詳細については、「#pragma 警告 (C# リファレンス)」を参照してください。

使用例
次の例では CS1634 エラーが生成されます。

```
// CS1634.cs
// compile with: /W:1

#pragma warning   // CS1634
// Try this instead:
// #pragma warning disable 0219

class MyClass
{
  public static void Main()
  {
  }
}
```

コンパイラの警告 (レベル 1) CS1635

エラー メッセージ
警告 '警告コード' はグローバルで無効にされたため、復元することはできません。

この警告は、**/nowarn** コマンド ライン オプションまたはプロジェクト設定で、コンパイル ユニット全体の警告が無効にされているにもかかわらず、`#pragma warning restore` を使用して警告を生成しようとした場合に発生します。このエラーを解決するには、/nowarn のコマンド ライン オプションまたはプロジェクト設定を使わないようにするか、コマンド ラインまたはプロジェクト設定で無効にするすべての警告について、`#pragma warning restore` を削除します。詳細については、「#pragma 警告 (C# リファレンス)」を参照してください。

次の例では CS1635 エラーが生成されます。

```
// CS1635.cs
// compile with: /w:1 /nowarn:162

enum MyEnum {one=1,two=2,three=3};

class MyClass
{
    public static void Main()
    {
#pragma warning disable 162

    if (MyEnum.three == MyEnum.two)
        System.Console.WriteLine("Duplicate");

#pragma warning restore 162
    }
}
```

コンパイラの警告 (レベル 1) CS1645

エラー メッセージ
機能 '機能' は標準 ISO C# 言語仕様ではありません。別のコンパイラでは受け入れられない可能性があります。

ISO 規格に準拠しない機能が使用されています。この機能を使用したコードは、他のコンパイラではコンパイルできない可能性があります。

```
// CS1645.cs
// compile with: /W:1 /t:module /langversion:ISO-1
[assembly:System.CLSCompliant(false)]
// To supress the warning use the switch: /nowarn:1645
[module:System.CLSCompliant(false)]   // CS1645
class Test
{
}
```

コンパイラの警告 (レベル 1) CS1658

エラー メッセージ
'警告テキスト' です。エラー 'エラー コード' を参照してください

この警告は、警告によってエラーがオーバーライドされる場合に生成されます。問題の詳細については、メッセージに示されているエラーを参照してください。Visual Studio IDE で該当するエラーを見つけるには、インデックスを使用します。たとえば、上記のテキストに "エラー 'CS1037' を参照してください。" のように示されている場合は、インデックスで CS1037 を検索します。

使用例
次の例では CS1658 警告が生成されます。

```
// CS1658.cs
// compile with: /doc:x.xml
// CS1584 expected
/// <summary>
/// </summary>
public class C
{
    /// <see cref="C.F(params object[])"/>  // CS1658
    public static void M()
    {
    }

    /// <summary>
    /// </summary>
    public void F(params object[] o)
    {
    }

    static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS1682

エラー メッセージ
型 '型' への参照では 'nested type' 内に入れ子にされていると指定されていますが、見つかりませんでした

このエラーは、他の参照または既存のコードと矛盾する参照をインポートした場合に発生します。このエラーは一般に、メタデータ内のクラスを参照するコードを記述した後に、そのクラスを削除したり、その定義を変更したりすると発生します。

使用例

```
// CS1682.cs
// compile with: /target:library /keyfile:mykey.snk
public class A {
   public class N1 {}
}
```

```
// CS1682_b.cs
// compile with: /target:library /reference:CS1682.dll
using System;
public class Ref {

   public static A A1() {
      return new A();
   }

   public static A.N1 N1() {
      return new A.N1();
   }
}
```

```
// CS1682_c.cs
// compile with: /target:library /keyfile:mykey.snk /out:CS1682.dll
public class A {
   public void M1() {}
}
```

次の例では CS1682 警告が生成されます。

```
// CS1682_d.cs
// compile with: /reference:CS1682.dll /reference:CS1682_b.dll /W:1
// CS1682 expected
class Tester {
   static void Main()
   {
      Ref.A1().M1();
   }
}
```

コンパイラの警告 (レベル 1) CS1683

エラー メッセージ
型 '型名' への参照では、このアセンブリで定義されていると指定されていますが、ソースまたは追加モジュール内では定義されていません

このエラーは、自分自身 (コンパイルの対象となるアセンブリ) を参照するアセンブリをインポートするとき、該当する参照が存在しない場合に発生します。たとえば、コンパイルするアセンブリに、インポート対象のアセンブリによって参照されるメンバが存在していたとします。その後、アセンブリを更新するときに、インポート対象のアセンブリによって参照されるメンバを誤って削除してしまった場合、この問題が発生します。

コンパイラの警告 (レベル 1) CS1684

エラー メッセージ
型 '型名' への参照では '名前空間' 内で定義されていると指定されていますが、見つかりませんでした

このエラーは、ある名前空間から、別の名前空間の型を参照するとき、後者の名前空間に目的の型が存在しない場合に発生します。たとえば、mydll.dll で、yourdll.dll の `A` という型を参照しようとしたところ、A が yourdll.dll に見つからないときに発生します。このエラーの原因としては、使用している yourdll.dll が、`A` が定義される前の古いバージョンであることが考えられます。

次の例では CS1684 エラーが生成されます。

使用例

```
// CS1684_a.cs
// compile with: /target:library /keyfile:CS1684.key
public class A {
   public void Test() {}
}

public class C2 {}
```

```
// CS1684_b.cs
// compile with: /target:library /r:cs1684_a.dll
// post-build command: del /f CS1684_a.dll
using System;
public class Ref
{
   public static A GetA() { return new A(); }
   public static C2 GetC() { return new C2(); }
}
```

今度は、最初のアセンブリをリビルドします。今度は、C2 クラスは定義されません。

```
// CS1684_c.cs
// compile with: /target:library /keyfile:CS1684.key /out:CS1684_a.dll
public class A {
   public void Test() {}
}
```

次のモジュールでは、`Ref` 識別子を使用して、2 つ目のモジュールを参照します。2 つ目のモジュールは、前の手順でコンパイルから除外された `C2` クラスを参照することとなり、コンパイルすると、CS1684 エラー メッセージが返されます。

```
// CS1684_d.cs
// compile with: /reference:cs1684_a.dll /reference:cs1684_b.dll
// CS1684 expected
class Tester
{
   public static void Main()
   {
      Ref.GetA().Test();
   }
}
```

コンパイラの警告 (レベル 1) CS1685

エラー メッセージ
定義済みの型 'System.type 名前' は、グローバル エイリアスの複数のアセンブリ内で定義されています。'ファイル名' からの定義を使用してください。

このエラーは、定義済みのシステム型 (System.int32 など) が 2 つのアセンブリで見つかった場合に発生します。これは、たとえば、.Net Framework のバージョン 1.0 とバージョン 1.1 を同時に実行しようとした場合など、mscorlib を 2 つの異なる場所から参照している場合が該当します。

コンパイラでは、いずれか一方のアセンブリの定義のみが使用されます。コンパイラによって検索されるのはグローバル エイリアスだけであり、**/reference** で定義されたライブラリは検索されません。**/nostdlib** が指定されている場合、コンパイラはまず Object を検索します。その後、**Object** を検出したファイルで、定義済みのすべての型を検索します。

コンパイラの警告 (レベル 1) CS1687

エラー メッセージ
ソース ファイルは、PDB 内で表せる 16,707,565 行の限界を超えているため、デバッグ情報は不正確になります

PDB およびデバッガには、ファイルのサイズに関して一定の制限があります。ソース ファイルが大きすぎると、ファイルの行数がこの制限を超え、デバッガが正常に動作しなくなります。このようなファイルに対しては、`#line hidden` などを使用してデバッグ情報が生成されないようにするか、ファイルを分割するなどしてファイル サイズを小さくする必要があります。たとえば、大きなクラスは、**partial** キーワードを使って分割することもできます。

コンパイラの警告 (レベル 1) CS1691

エラー メッセージ
'number' は有効な警告番号ではありません。

#pragma warning プリプロセッサ ディレクティブに渡された数値は、有効な警告番号ではありません。エラー番号が使用されていないか、また、無効な文字が使用されていないかなど、警告を表す番号が正しいことを確認してください。

使用例

次のコードは CS1691 を生成します。

```
// CS1691.cs
public class C
{
    int i = 1;
    public static void Main()
    {
        C myC = new C();
#pragma warning disable 151  // CS1691
// Try the following line instead:
// #pragma warning disable 1645
        myC.i++;
#pragma warning restore 151  // CS1691
// Try the following line instead:
// #pragma warning restore 1645
    }
}
```

コンパイラの警告 (レベル 1) CS1692

エラー メッセージ
無効な数字です。

#pragma や **#line** など、いくつかのプリプロセッサ ディレクティブでは、パラメータとして数値が使用されます。この警告は、無効 (大きすぎる、形式が間違っている、無効な文字が含まれるなど) な数値が指定されていることを示しています。このエラーを解決するには、正しい数値を指定します。

使用例
次の例では CS1692 エラーが生成されます。

```
// CS1692.cs

#pragma warning disable a  // CS1692
// Try this instad:
// #pragma warning disable 1691

class A
{
    static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS1694

エラー メッセージ
プリプロセッサ ディレクティブに対して無効なファイル名が指定されました。ファイル名は長すぎるか、または有効なファイル名ではありません。

この警告は、**#pragma checksum** プリプロセッサ ディレクティブを使用した場合に発生します。指定されたファイル名が 256 文字を超えています。この警告を解決するには、ファイル名を短くします。

使用例
次の例では CS1694 警告が生成されます。

```
// cs1694.cs
#pragma checksum "MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile123
4567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFi
le1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile123456789
0.txt" {00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F}   // CS1694
class MyClass {}
```

コンパイラの警告 (レベル 1) CS1695

エラー メッセージ
無効な #pragma チェックサム構文です。有効な #pragma チェックサムは、"ファイル名" "{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}" "XXXX..." です。

このエラーが発生するケースはきわめてまれです。通常、チェックサムは、Code DOM API を使用してコードを生成した場合に、実行時に挿入されるためです。

ただし、この **#pragma** ステートメントを直接入力し、GUID またはチェックサムを誤って入力した場合、このエラーが発生します。コンパイラによる構文チェックでは、手動で入力された GUID が正しいかどうかまでは検証されません。ただし、GUID に使用されている数値の桁数とデリミタが正しいこと、および、数値が 16 進数であるかどうかはチェックされます。同様に、チェックサムに使用されている数値の桁数が偶数であることと、16 進数が使用されていることがチェックされます。

使用例
次のコードは CS1695 を生成します。

```
// CS1695.cs

#pragma checksum "12345"  // CS1695

public class Test
{
    static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS1696

エラー メッセージ
単一行コメントか行の終わりが必要です。

プリプロセッサ ディレクティブの後に、行終端記号または単一行コメントを追加する必要があります。有効なプリプロセッサ ディレクティブの処理は完了しましたが、この構文の入力規則について、なんらかの違反が見つかりました。

使用例
次の例では CS1696 が生成されます。

```
// CS1696.cs
class Test
{
   public static void Main()
   {
      #pragma warning disable 1030;219   // CS1696
      #pragma warning disable 1030   // OK
   }
}
```

コンパイラの警告 (レベル 1) CS1697

エラー メッセージ
'ファイル名' に指定された異なるチェックサム値です。

1 つのファイルに複数のチェックサムが指定されています。プロジェクトに同じ名前の複数のファイルが存在する場合、デバッガはチェックサムの値で、デバッグするファイルを判断します。このエラーが発生するケースはきわめてまれですが、コードの生成を伴うアプリケーションを作成する場合、このエラーに遭遇することがあります。このエラーを解決するには、特定のコード ファイルに対してチェックサムが 1 回だけ生成されるようにします。

コンパイラの警告 (レベル 1) CS1699

エラー メッセージ
コマンド ライン オプション "compiler_option" を使用するか、"attribute_name" 以外の適切なプロジェクト設定を使用してください。

アセンブリに署名するには、キー ファイルを指定する必要があります。Microsoft Visual C# 2005 より前は、ソース コードの CLR 属性を使用してキー ファイルを指定しました。これらの属性は使用されなくなりました。

Microsoft Visual C# 2005 からは、プロジェクト デザイナの [署名] ページまたはアセンブリ リンカを使用してキー ファイルを指定します。

プロジェクト デザイナの [署名] ページを使用することをお勧めします。詳細については、「[署名] ページ (プロジェクト デザイナ)」および「アセンブリおよびマニフェストへの署名の管理」を参照してください。

「方法 : 厳密な名前でアセンブリに署名する」では、次のコンパイラ オプションを使用します。

- /keyfile (厳密名キー ファイルの指定) (C# コンパイラ オプション) (AssemblyKeyFileAttribute 属性の代わり)
- /keycontainer (厳密名キー コンテナの指定) (C# コンパイラ オプション) (`AssemblyKeyNameAttribute` の代わり)
- `/delaysign` (アセンブリの遅延署名) (`C#` コンパイラ オプション) (`AssemblyDelaySignAttribute` の代わり)

これらの属性が推奨されなくなった背景には、次のような経緯があります。

- コンパイラによって生成されたバイナリ ファイルに属性が埋め込まれる、というセキュリティ上の問題がありました。第三者にバイナリ ファイルを配布すると、結果的に、そこに保存されているキーも一緒に配布されてしまいます。
- 属性に指定するパスの形式から生じる利便性の問題もありました。これまでは、現在の作業ディレクトリ (統合開発環境 (`IDE`) 内で変更される可能性がある)、つまり出力ディレクトリに対する相対パスが使用されていたため、キー ファイルは `"..\\..\\mykey.snk"` のように指定していました。また、属性で指定することは、プロジェクト システムが適切にサテライト アセンブリの署名を行うことへの障害となります。属性の代わりにコンパイラ オプションを使った場合、キーに絶対パスとファイル名を使用でき、出力ファイルには何も埋め込む必要がありません。プロジェクト システムおよびソース コード管理システムは、プロジェクトが移動されたとしても、その完全パスを適切に扱うことができます。プロジェクト システムは、プロジェクトとキー ファイルの位置関係を確実に維持でき、完全パスをコンパイラに渡すことができます。他のビルド プログラムは、正しい属性の指定されたソース ファイルを生成する代わりに、適切なパスを直接コンパイラに渡すことによって、容易に出力ファイルに署名できます。
- フレンド アセンブリで属性を使用すると、コンパイラの効率に悪影響を及ぼすことがあります。属性を使用した場合、コンパイラは、フレンドシップを付与するかどうかの判断段階ではキーの存在を確認できず、推測に依存する必要がありました。コンパイルの最後に、キーの存在を知った段階で初めて、推測が正しかったかどうかを検証できます。キー ファイルをコンパイラ オプションで指定した場合、フレンドシップを付与する必要があるかどうかをコンパイラが直ちに判断できます。

使用例
次の例では `CS1699` 警告が生成されます。このエラーを解決するには、属性を削除し、**`/delaysign`** を使用してコンパイルします。

```
// CS1699.cs
// compile with: /target:library
[assembly:System.Reflection.AssemblyDelaySign(true)]   // CS1699
```

参照
処理手順
方法 : 厳密な名前でアセンブリに署名する
関連項目
[署名] ページ (プロジェクト デザイナ)
その他の技術情報
アセンブリおよびマニフェストへの署名の管理

コンパイラの警告 (レベル 1) CS1707

エラー メッセージ
新しい言語規則のために、'MethodName2' の代わりに 'MethodName1' にバインドされたデリゲート 'DelegateName' です。

C# 2.0 では、デリゲートのメソッドへのバインディングに新しい規則が導入されました。従来は無視されていた情報が考慮されるようになります。この警告は、デリゲートのバインド先メソッドのオーバーロードが、従来とは異なることを示しています。このデリゲートを本当に 'MethodName2' ではなく 'MethodName1' にバインドしてよいかを確認してください。

デリゲートのバインド先のメソッドをコンパイラがどのように判断するかについては、「デリゲートの共変性と反変性 (C# プログラミング ガイド)」を参照してください。

コンパイラの警告 (レベル 1) CS1709

エラー メッセージ
プリプロセッサ ディレクティブに指定されたファイル名は空です。

ファイル名を含むプリプロセッサ ディレクティブが指定されていますが、そのファイルが空です。この警告を解決するには、必要な情報をファイルに追加します。

使用例
次の例では CS1709 警告が生成されます。

```
// CS1709.cs
class Test
{
    static void Main()
    {
        #pragma checksum "" "{406EA660-64CF-4C82-B6F0-42D48172A799}" ""  // CS1709
    }
}
```

コンパイラの警告 (レベル 1) CS1720

エラー メッセージ
'一般的な型' の既定値が Null であるため、式は常に System.NullReferenceException になります

このエラーは、ジェネリック型に渡される変数の既定値 (クラスなどの参照型) を、式の中で使用した場合に発生します。次のような式があるとします。

```
default(T).ToString()
```

T は参照型であるため、既定値は null です。そのため、これに対して ToString メソッドを適用しようとすると、NullReferenceException がスローされます。

使用例

型パラメータである T にクラス参照制約を適用することによって、T を参照型としています。

次の例では CS1720 エラーが生成されます。

```
// CS1720.cs
using System;
public class Tester
{
    public static void GenericClass<T>(T t1) where T : class
    {
        Console.WriteLine(default(T).ToString());  // CS1720
    }
    public static void Main() {}
}
```

コンパイラの警告 (レベル 1) CS1723

エラー メッセージ
'param' 上の XML コメントに型パラメータを参照する cref 属性 'attribute' が指定されています。

このエラーは、XML コメントで型パラメータを参照したときに生成されます。

使用例
次の例では CS1723 警告が生成されます。

```
// CS1723.cs
// compile with: /t:library /doc:filename.XML
///<summary>A generic list class.</summary>
///<see cref="T" />   // CS1723
// To resolve comment the previous line.
public class List<T>
{
}
```

コンパイラの警告 (レベル 1) CS1911

エラー メッセージ
匿名メソッドまたは反復子から、'base' キーワードをとおしてメンバ 'メンバ' へアクセスすると、確認不可能なコードを生じさせます。

反復子のメソッド本体内または匿名メソッド内で **base** キーワードを使用して仮想関数を呼び出すと、検証不能なコードになります。検証不能なコードは、部分信頼環境では実行できません。

仮想関数呼び出しをヘルパー関数に移動すると、CS1911 警告を解決できます。

使用例
次の例では CS1911 警告が生成されます。

```
// CS1911.cs
// compile with: /W:1
using System;

delegate void D();
delegate D RetD();

class B {
   protected virtual void M() {
      Console.WriteLine("B.M");
   }
}

class Der : B {
   protected override void M() {
      Console.WriteLine("D.M");
   }

   void Test() { base.M(); }
   D Test2() { return new D(base.M); }

   public D CallBaseM() {
      return delegate () { base.M(); };   // CS1911

      // try the following line instead
      // return delegate () { Test(); };
   }

   public RetD DelToBaseM() {
      return delegate () { return new D(base.M); };   // CS1911

      // try the following line instead
      // return delegate () { return Test2(); };
   }
}

class Program {
   public static void Main() {
      Der der = new Der();
      D d = der.CallBaseM();
      d();
      RetD rd = der.DelToBaseM();
      rd()();
   }
}
```

コンパイラの警告 (レベル 1) CS2002

エラー メッセージ
ソース ファイル 'file' が複数回指定されました。

ソース ファイル名がコンパイラに複数回渡されました。出力ファイルを作成するコンパイラに対しては、1 つのファイルを 1 回しか指定できません。

次の例では CS2002 エラーが生成されます。

```
// CS2002.cs
// compile with: CS2002.cs
public class A
{
public static void Main(){}
}
```

エラーを生成するには、コマンド ラインで次のサンプルをコンパイルします。

```
csc CS2002.cs CS2002.cs
```

コンパイラの警告 (レベル 1) CS2014

エラー メッセージ
コンパイラ オプション 'old option' は古い形式です。'new option' を代わりに使用してください。

旧式のコンパイラ オプションを使用しています。詳細については、「C# コンパイラ オプション」を参照してください。

コンパイラの警告 (レベル 1) CS2023

エラー メッセージ
応答ファイルで指定されているため、/noconfig オプションを無視します。

/noconfig コンパイラ オプションが応答ファイルで指定されましたが、この処理は許可されていません。

コンパイラの警告 (レベル 1) CS3000

エラー メッセージ
可変個の引数を指定したメソッドは CLS に準拠していません。

メソッドで使用されている引数で、共通言語仕様 (CLS) に含まれない機能が公開されています。CLS 準拠の詳細については、「CLS 準拠コードの記述」を参照してください。

次の例では、CS3000 警告が生成されます。

```
// CS3000.cs
// compile with: /target:library
// CS3000 expected
[assembly:System.CLSCompliant(true)]

public class Test
{
   public void AddABunchOfInts( __arglist ) {}   // CS3000
}
```

コンパイラの警告 (レベル 1) CS3001

エラー メッセージ
引数の型 'type' は CLS に準拠していません。

public メソッド、protected メソッド、または **protected internal** メソッドには、共通言語仕様 (CLS: Common Language Specification) に準拠した型のパラメータを渡す必要があります。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では、CS3001 エラーが生成されます。

```
// CS3001.cs

[assembly:System.CLSCompliant(true)]
public class a
{
    public void bad(ushort i)   // CS3001
    {
    }

    private void OK(ushort i)   // OK, private method
    {
    }

    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3002

エラー メッセージ
'method' の戻り値の型は CLS に準拠していません。

public メソッド、protected メソッド、または **protected internal** メソッドは、型が共通言語仕様 (CLS: Common Language Specification) に準拠する値を返す必要があります。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では、CS3002 エラーが生成されます。

```
// CS3002.cs

[assembly:System.CLSCompliant(true)]
public class a
{
    public ushort bad()   // CS3002, public method
    {
        ushort a;
        a = ushort.MaxValue;
        return a;
    }

    private ushort OK()   // OK, private method
    {
        ushort a;
        a = ushort.MaxValue;
        return a;
    }

    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3003

エラー メッセージ
'variable' の型は CLS に準拠していません。

public 変数、protected 変数、または **protected internal** 変数の型は、共通言語仕様 (CLS: Common Language Specification) に準拠している必要があります。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では、CS3003 エラーが生成されます。

```
// CS3003.cs

[assembly:System.CLSCompliant(true)]
public class a
{
    public ushort a1;   // CS3003, public variable
    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3004

エラー メッセージ
混合、分解された Unicode 文字は CLS に準拠していません。

共通言語仕様 (CLS: Common Language Specification) に準拠するために public 識別子、protected 識別子、または **protected internal** 識別子で使用できるのは、構成済みの Unicode 文字だけです。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

コンパイラの警告 (レベル 1) CS3005

エラー メッセージ
大文字、小文字の違いのみの識別子 'identifier' は CLS に準拠していません。

他の **public** 識別子、**protected** 識別子、**protected internal** 識別子と大文字小文字だけが異なる public 識別子、protected 識別子、**protected internal** 識別子は、共通言語仕様 (CLS: Common Language Specification) に準拠していません。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では、CS3005 エラーが生成されます。

```
// CS3005.cs

using System;

[assembly:CLSCompliant(true)]
public class a
{
    public static int a1 = 0;
    public static int A1 = 1;   // CS3005

    public static void Main()
    {
        Console.WriteLine(a1);
        Console.WriteLine(A1);
    }
}
```

コンパイラの警告 (レベル 1) CS3006

エラー メッセージ
ref または out のみが異なるオーバーロードされたメソッド 'メソッド' は、CLS に準拠していません。

ref パラメータや out パラメータに基づいてオーバーロードしたメソッドは、共通言語仕様 (CLS: Common Language Specification) に準拠できません。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例

次の例では CS3006 警告が生成されます。この警告を解決するには、アセンブリ レベルの属性をコメント アウトするか、またはメソッドの定義を 1 つ削除します。

```
// CS3006.cs

using System;

[assembly: CLSCompliant(true)]
public class MyClass
{
    public void f(int i)
    {
    }

    public void f(ref int i)   // CS3006
    {
    }

    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3007

エラー メッセージ
名前のない配列型のみが異なるオーバーロードされたメソッド 'メソッド' は、CLS に準拠していません。

このエラーは、ジャグ配列を受け取る、オーバーロードされたメソッドが存在し、配列要素の型以外にメソッド シグネチャの違いがない場合に発生します。このエラーを回避するには、ジャグ配列ではなく四角形配列を使用する、パラメータを追加して関数呼び出しを明確にする、または、オーバーロードされたメソッドの名前を変更します。CLS 準拠でなくてもかまわない場合は、CLSCompliantAttribute 属性を削除します。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では、CS3007 エラーが生成されます。

```
// CS3007.cs
[assembly: System.CLSCompliant(true)]
public struct S
{
    public void F(int[][] array) { }
    public void F(byte[][] array) { }  // CS3007
    // Try this instead:
    // public void F1(int[][] array) {}
    // public void F2(byte[][] array) {}
    // or
    // public void F(int[,] array) {}
    // public void F(byte[,] array) {}
}
```

コンパイラの警告 (レベル 1) CS3008

エラー メッセージ
識別子 'identifier' は CLS に準拠していません。

public、protected、または **protected internal** の識別子をアンダースコア文字 (_) で始めることは、共通言語仕様 (CLS) に違反します。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では、CS3008 エラーが生成されます。

```
// CS3008.cs

using System;

[assembly:CLSCompliant(true)]
public class a
{
    public static int _a = 0;  // CS3008
    // OK, private
    // private static int _a1 = 0;

    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3009

エラー メッセージ
'型': 基本型 '型' は CLS に準拠していません。

共通言語仕様 (CLS: Common Language Specification) 準拠としてマークされているアセンブリで、基本型に CLS 準拠不要のマークが付けられました。アセンブリが CLS 準拠であることを指定する属性を削除するか、または型が CLS 準拠でないことを示す属性を削除してください。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では、CS3009 エラーが生成されます。

```
// CS3009.cs

using System;

[assembly:CLSCompliant(true)]
[CLSCompliant(false)]
public class B
{
}

public class C : B   // CS3009
{
    public static void Main () {}
}
```

コンパイラの警告 (レベル 1) CS3010

エラー メッセージ
'メンバ': CLS と互換性のあるインターフェイスには CLS と互換性のあるメンバだけを含む必要があります。

`[assembly:CLCSompliant(true)]` でマークされたアセンブリで、`[CLCSompliant(false)]` でマークされたメンバがインターフェイスに含まれています。共通言語仕様 (CLS: Common Language Specification) 準拠の属性を 1 つ削除してください。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では、CS3010 警告が生成されます。

```
// CS3010.cs

using System;

[assembly:CLSCompliant(true)]
public interface I
{
    [CLSCompliant(false)]
    int mf();   // CS3010
}

public class C : I
{
    public int mf()
    {
        return 1;
    }

    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3011

エラー メッセージ
'member' : CLS 準拠メンバだけが抽象になることができます。

クラス メンバに abstract と共通言語仕様 (CLS: Common Language Specification) 非準拠の両方を指定することはできません。CLS では、すべてのクラス メンバが実装されるように指定します。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例

次の例では、CS3011 エラーが生成されます。

```
// CS3011.cs

using System;


[assembly:CLSCompliant(true)]
public abstract class I
{
    [CLSCompliant(false)]
    public abstract int mf();   // CS3011

    // OK
    [CLSCompliant(false)]
    public void mf2()
    {
    }
}

public class C : I
{
    public override int mf()
    {
        return 1;
    }

    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3012

エラー メッセージ
CLS コンプライアンス チェックのためにモジュールではなく、アセンブリに CLSCompliant 属性を指定してください。

`[module:System.CLCSompliant(true)]` を使用してモジュールを共通言語仕様 (CLS: Common Language Specification) 準拠にするには、モジュールを /target:module コンパイラ オプションでビルドする必要があります。CLS の詳細については、「共通言語仕様」を参照してください。

使用例

次のサンプルを **/target:module** を使用せずにビルドすると、警告 CS3012 が生成されます。

```
// CS3012.cs
// compile with: /W:1

[module:System.CLSCompliant(true)]   // CS3012
public class C
{
    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3013

エラー メッセージ
追加されたモジュールは、アセンブリに一致するように CLSCompliant 属性と共に設定されなければなりません。

/target:module コンパイラ オプションを使用してコンパイルされたモジュールが、/addmodule によってコンパイルに追加されました。しかし、モジュールの共通言語仕様 (CLS: Common Language Specification) 準拠が、現在のコンパイルの CLS 状態と一致しません。

CLS 準拠は、モジュールの属性で示されます。たとえば、`[module:CLSCompliant(true)]` はモジュールが CLS 準拠であることを示し、`[module:CLSCompliant(false)]` はモジュールが CLS 準拠ではないことを示します。既定値は `[module:CLSCompliant(false)]` です。CLS の詳細については、「共通言語仕様」を参照してください。

コンパイラの警告 (レベル 1) CS3014

エラー メッセージ
アセンブリに CLSCompliant 属性が含まれていないため、'メンバ' を CLS 準拠として設定できません。

共通言語仕様 (CLS: Common Language Specification) 準拠を指定しなかったソース コード ファイルの構成要素が、CLS 準拠としてマークされました。これは認められていません。この警告を解決するには、アセンブリ レベルの CLS 準拠属性をファイルに追加します。次の例で、アセンブリ レベル属性が指定されている行のコメントを解除すると警告を回避できます。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では、CS3014 エラーが生成されます。

```
// CS3014.cs

using System;

// [assembly:CLSCompliant(true)]
public class I
{
    [CLSCompliant(true)]   // CS3014
    public void mf()
    {
    }

    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3015

エラー メッセージ
'method signature' は CLS 準拠型のみを使用するコンストラクタにアクセスできません。

共通言語仕様 (CLS: Common Language Specification) に準拠させる場合、属性クラスの引数リストに配列を含めることはできません。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では C3015 エラーが生成されます。

```
// CS3015.cs
// compile with: /target:library
using System;

[assembly:CLSCompliant(true)]
public class MyAttribute : Attribute
{
   public MyAttribute(int[] ai) {}   // CS3015
   // try the following line instead
   // public MyAttribute(int ai) {}
}
```

コンパイラの警告 (レベル 1) CS3016

エラー メッセージ
属性の引数としての配列は CLS 互換ではありません。

属性に配列を渡す指定は、共通言語仕様 (CLS: Common Language Specification) に準拠していません。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例

次の例では、CS3016 エラーが生成されます。

```
// CS3016.cs

using System;

[assembly : CLSCompliant(true)]
[C(new int[] {1, 2})]   // CS3016
// try the following line instead
// [C()]
class C : Attribute
{
    public C()
    {
    }

    public C(int[] a)
    {
    }

    public static void Main ()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3017

エラー メッセージ
アセンブリの CLSCompliant 属性と異なるモジュールの CLSCompliant 属性は指定できません

この警告は、アセンブリの CLSCompliant 属性がモジュールの CLSCompliant 属性と競合している場合に発生します。CLS 準拠として指定したアセンブリに、非 CLS 準拠のモジュールを含めることはできません。この警告を解決するには、アセンブリとモジュールの CLSCompliant 属性を true または false に統一するか、いずれかの CLSCompliant 属性を削除します。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例

次の例では、CS3017 警告が生成されます。

```
// CS3017.cs
// compile with: /target:module

using System;

[module: CLSCompliant(true)]
[assembly: CLSCompliant(false)]  // CS3017
// Try this line instead:
// [assembly: CLSCompliant(true)]
class C
{
    static void Main() {}
}
```

コンパイラの警告 (レベル 1) CS3018

エラー メッセージ
CLS に準拠していない型 '型' のメンバであるため、'型' を CLS 準拠として設定できません

この警告は、CLSCompliant 属性の **true** に設定された入れ子のクラスが、CLSCompliant 属性の **false** に設定されたクラスのメンバとして宣言されている場合に発生します。CLS 準拠ではない外部クラスのメンバである場合、そこで入れ子にされているクラスが CLS 準拠になることはできないため、このような記述は認められません。この警告を解決するには、入れ子にされているクラスから CLSCompliant 属性を削除するか、この属性の設定を **true** から **false** に変更します。CLS 準拠の詳細については、「CLS 準拠コードの記述」および「共通言語仕様」を参照してください。

使用例
次の例では CS3018 エラーが生成されます。

```
// CS3018.cs
// compile with: /target:library
using System;

[assembly: CLSCompliant(true)]
[CLSCompliant(false)]
public class Outer
{
   [CLSCompliant(true)]   // CS3018
   public class Nested {}

   // OK
   public class Nested2 {}

   [CLSCompliant(false)]
   public class Nested3 {}
}
```

コンパイラの警告 (レベル 1) CS3022

エラー メッセージ
CLSCompliant 属性は、パラメータに適用されても意味がありません。メソッドに適用してください。

メソッドのパラメータには、CLS に準拠しているかどうかのチェックは適用されません。CLS 準拠の規則は、メソッドおよび型の宣言に対して適用されます。

使用例
次の例では CS3022 エラーが生成されます。

```
// CS3022.cs
// compile with: /W:1

using System;

[assembly: CLSCompliant(true)]
[CLSCompliant(true)]
public class C
{
    public void F([CLSCompliant(true)] int i)
    {
    }

    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS3023

エラー メッセージ
CLSCompliant 属性は、戻り値の型に適用されても意味がありません。メソッドに適用してください。

関数の戻り値の型には、CLS に準拠しているかどうかのチェックは適用されません。CLS 準拠の規則は、メソッドおよび型の宣言に対して適用されます。

使用例
次の例では、CS3023 警告が生成されます。

```
// C3023.cs

[assembly:System.CLSCompliant(true)]
public class Test
{
    [return:System.CLSCompliant(true)]  // CS3023
    // Try this instead:
    // [method:System.CLSCompliant(true)]
    public static int Main()
    {
        return 0;
    }
}
```

コンパイラの警告 (レベル 1) CS3026

エラー メッセージ
CLS 準拠フィールド 'フィールド' を volatile にすることはできません

volatile 変数は CLS に準拠していません。

使用例
次の例では CS3026 警告が生成されます。

```
// CS3026.cs
[assembly:System.CLSCompliant(true)]
public class Test
{
public volatile int v0 =0;   // CS3026
      // To resolve remove the CLS-CComplient attribute.
public static void Main() { }
}
```

コンパイラの警告 (レベル 1) CS5000

エラー メッセージ
コンパイラ オプション '/option' が不明です。

無効なコンパイラ オプションが指定されました。

コンパイラの警告 (レベル 2) CS0108

エラー メッセージ
'member1' は継承メンバ 'member2' を隠します。非表示にする場合は、new キーワードを使用してください。

変数が、基本クラスの変数と同じ名前で宣言されました。しかし、new キーワードは使用されていません。この警告は **new** を使用する必要があることを示しています。変数は、宣言で **new** を使用している場合と同様に宣言されます。

次の例では警告 CS0108 が生成されます。

```
// CS0108.cs
// compile with: /W:2
using System;

namespace x
{
   public class clx
   {
      public int i = 1;
   }

   public class cly : clx
   {
      public static int i = 2;   // CS0108, use the new keyword
      // the compiler parses the previous line as if you had specified:
      // public static new int i = 2;

      public static void Main()
      {
         Console.WriteLine(i);
      }
   }
}
```

コンパイラの警告 (レベル 2) CS0114

エラー メッセージ
'function1' は継承されたメンバ 'function2' を隠します。現在のメンバでそのインプリメンテーションをオーバーライドするには、override キーワードを追加してください。オーバーライドしない場合は、new キーワードを追加してください。

クラス内の宣言が基本クラス内の宣言と矛盾しており、基本クラスのメンバが隠ぺいされます。

詳細については、「base」を参照してください。

次の例では警告 CS0114 が生成されます。

```
// CS0114.cs
// compile with: /W:2 /warnaserror
abstract public class clx
{
   public abstract void f();
}

public class cly : clx
{
   public void f() // CS0114, hides base class member
   // try the following line instead
   // override public void f()
   {
   }

   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 2) CS0162

エラー メッセージ
到達できないコードが検出されました。

実行されないコードが検出されました。

次の例では CS0162 エラーが生成されます。

```
// CS0162.cs
// compile with: /W:2
public class A
{
   public static void Main()
   {
      goto lab1;
      {
         // The following statements cannot be reached:
         int i = 9;   // CS0162
         i++;
      }
      lab1:
      {
      }
   }
}
```

コンパイラの警告 (レベル 2) CS0164

エラー メッセージ
このラベルは参照されていません。

ラベルが宣言されましたが使用されませんでした。

次の例では CS0164 エラーが生成されます。

```
// CS0164.cs
// compile with: /W:2
public class a
{
   public int i = 0;

   public static void Main()
   {
      int i = 0;   // CS0164
      l1: i++;
      // the following lines resolve this error
      // if(i < 10)
      //    goto l1;
   }
}
```

コンパイラの警告 (レベル 2) CS0251

エラー メッセージ
負のインデックスで配列します。配列は常にゼロからの開始を示します。

配列のインデックスに負の数を使用しないでください。

次の例では CS0251 エラーが生成されます。

```
// CS0251.cs
// compile with: /W:2
class MyClass
{
   public static void Main()
   {
      int[] myarray = new int[] {1,2,3};
      try
      {
         myarray[-1]++;   // CS0251
         // try the following line instead
         // myarray[1]++;
      }
      catch (System.IndexOutOfRangeException e)
      {
         System.Console.WriteLine("{0}", e);
      }
   }
}
```

コンパイラの警告 (レベル 2) CS0252

エラー メッセージ
予期しない参照比較です。比較値を取得するには型 'type' に左辺をキャストしてください。

コンパイラが参照比較を行っています。文字列の値を比較する場合は、式の左辺を *type* にキャストしてください。

次の例では警告 CS0252 が生成されます。

```
// CS0252.cs
// compile with: /W:2
using System;

class MyClass
{
   public static void Main()
   {
      string s = "11";
      object o = s + s;

      bool b = o == s;   // CS0252
      // try the following line instead
      // bool b = (string)o == s;
   }
}
```

コンパイラの警告 (レベル 2) CS0253

エラー メッセージ
予期しない参照比較です。比較値を取得するには型 'type' に右辺をキャストしてください。

コンパイラが参照比較を行っています。文字列の値を比較する場合は、式の右辺を *type* にキャストしてください。

次の例では警告 CS0253 が生成されます。

```
// CS0253.cs
// compile with: /W:2
using System;
class MyClass
{
   public static void Main()
   {
      string s = "11";
      object o = s + s;

      bool c = s == o;   // CS0253
      // try the following line instead
      // bool c = s == (string)o;
   }
}
```

コンパイラの警告 (レベル 2) CS0278

エラー メッセージ
'型' は、'パターン名' パターンを実装しません。'メソッド名' があいまいなため、'メソッド名' と混同します。

C# には、**foreach** や **using** など、あらかじめ定義されたパターンを使用するステートメントがいくつかあります。たとえば、**foreach** では、"列挙可能な" パターンを実装したコレクション クラスが使用されます。

CS0278 は、あいまいな部分があるために、コンパイラが適切な仲介処理を実行できない場合に発生する可能性があります。たとえば、"列挙可能な" パターンを処理するためには、**MoveNext** というメソッドが必要です。コンパイルの対象となるコードには、**MoveNext** メソッドが 2 つ存在している可能性があります。どちらのインターフェイスを使うべきか、コンパイラが検索を試みるためです。ただし、あいまいさの原因を突き止め、解決することをお勧めします。

詳細については、「方法 : foreach を使用してコレクション クラスにアクセスする (C# プログラミング ガイド)」を参照してください。

使用例
次の例では、CS0278 エラーが生成されます。

```
// CS0278.cs
using System.Collections.Generic;
public class myTest
{
   public static void TestForeach<W>(W w)
      where W: IEnumerable<int>, IEnumerable<string>
   {
      foreach (int i in w) {}   // CS0278
   }
}
```

コンパイラの警告 (レベル 2) CS0279

エラー メッセージ
型名' は、パターン名' パターンを実装しません。'メソッド名' は、スタティックであるか、またはパブリックではありません。

C# には、**foreach** や **using** など、あらかじめ定義されたパターンを使用するステートメントがいくつかあります。たとえば、**foreach** では、列挙可能なパターンを実装したコレクション クラスが使用されます。このエラーは、メソッドが、**static** で宣言されているか、**public** 以外のスコープで宣言されているために、コンパイラが適切な仲介処理を実行できない場合に発生します。パターン内のメソッドは、クラスのインスタンスとして利用できること、また、パブリックであることが必要です。

使用例
次の例では、CS0279 エラーが生成されます。

```
// CS0279.cs

using System;
using System.Collections;

public class myTest : IEnumerable
{
    IEnumerator IEnumerable.GetEnumerator()
    {
        yield return 0;
    }

    internal IEnumerator GetEnumerator()
    {
        yield return 0;
    }

    public static void Main()
    {
        foreach (int i in new myTest()) {}  // CS0279
    }
}
```

コンパイラの警告 (レベル 2) CS0280

エラー メッセージ
'型' は、'パターン名' パターンを実装しません。'メソッド名' には正しくないシグネチャが含まれます。

C# の **foreach** および **using** の 2 つのステートメントでは、それぞれ "コレクション" と "リソース" という、定義済みのパターンが使用されます。この警告は、メソッドのシグネチャが間違っているために、コンパイラが、いずれかのステートメントを定義済みのパターンに対応付けることができなかった場合に発生します。たとえば、"コレクション" のパターンでは、MoveNext と呼ばれる、パラメータを受け取らずに **boolean** を返すメソッドが存在していることが必要です。エラーの発生したコードには、パラメータを受け取るか、オブジェクトを返す **MoveNext** メソッドが含まれている可能性があります。

もう 1 つの例が、"リソース" のパターンが使用される **using** です。"リソース" のパターンでは、Dispose メソッドが必要です。同じ名前のプロパティを定義した場合、この警告が生成されます。

この警告を解決するには、型で使用しているメソッド シグネチャが、対応するパターンのメソッド シグネチャと一致していること、および、そのパターンで要求されるメソッドと同じ名前のプロパティが使われていないことを確認します。

使用例

次の例では、CS0280 エラーが生成されます。

```
// CS0280.cs
using System;
using System.Collections;

public class ValidBase: IEnumerable
{
   IEnumerator IEnumerable.GetEnumerator()
   {
      yield return 0;
   }

   internal IEnumerator GetEnumerator()
   {
      yield return 0;
   }
}

class Derived : ValidBase
{
   // field, not method
   new public int GetEnumerator;
}

public class Test
{
   public static void Main()
   {
      foreach (int i in new Derived()) {}   // CS0280
   }
}
```

コンパイラの警告 (レベル 2) CS0435

エラー メッセージ
'file_2' の名前空間 '名前空間' は、'file_1' のインポートされた型 '型' と競合します。名前空間を使用します。

この警告は、ソース ファイル (file_2) 内の名前空間が、インポートされたファイル (file _1) 内の型と競合している場合に発生します。この場合、コンパイルには、ソース ファイル内の型が使用されます。

次の例では、CS0435 エラーが生成されます。

最初に、次のファイルをコンパイルします。

```
// CS0435_1.cs
// compile with: /t:library
public class Util
{
   public class A { }
}
```

続けて、次のファイルをコンパイルします。

```
// CS0435_2.cs
// compile with: /r:CS0435_1.dll

using System;

namespace Util
{
   public class A { }
}

public class Test
{
   public static void Main()
   {
      Console.WriteLine(typeof(Util.A)); // CS0435
   }
}
```

コンパイラの警告 (レベル 2) CS0436

エラー メッセージ
file_2' の型 '型' は、インポートされた型 '型' と競合します。'file_2' にある型を使用します。

この警告は、ソース ファイル (file_2) 内の型が、インポートされたファイル (file _1) 内の型と競合している場合に発生します。この場合、コンパイルには、ソース ファイル内の型が使用されます。

使用例

```
// CS0436_a.cs
// compile with: /target:library
public class A {
   public void Test() {
      System.Console.WriteLine("CS0436_a");
   }
}
```

次の例では CS0436 警告が生成されます。

```
// CS0436_b.cs
// compile with: /reference:CS0436_a.dll
// CS0436 expected
public class A {
   public void Test() {
      System.Console.WriteLine("CS0436_b");
   }
}

public class Test
{
   public static void Main()
   {
      A x = new A();
      x.Test();
   }
}
```

コンパイラの警告 (レベル 2) CS0437

エラー メッセージ
'file_2' の型 '型' は、'file_1' のインポートされた名前空間 '名前空間' と競合します。型を使用します。

この警告は、ソース ファイル (file_2) 内の型が、file _1 内のインポートされた名前空間と競合している場合に発生します。この場合、コンパイルには、ソース ファイル内の型が使用されます。

使用例

```
// CS0437_a.cs
// compile with: /target:library
namespace Util
{
   public class A {
      public void Test() {
         System.Console.WriteLine("CS0437_a.cs");
      }
   }
}
```

次の例では CS0437 警告が生成されます。

```
// CS0437_b.cs
// compile with: /reference:CS0437_a.dll /W:2
// CS0437 expected
class Util
{
   public class A {
      public void Test() {
         System.Console.WriteLine("CS0437_b.cs");
      }
   }
}

public class Test
{
   public static void Main()
   {
      Util.A x = new Util.A();
      x.Test();
   }
}
```

コンパイラの警告 (レベル 2) CS0440

エラー メッセージ
'global::' はエイリアスではなく常にグローバル名前空間を参照するため、'global' という名前のエイリアスを定義することはお勧めしません。

この警告は、global という名前のエイリアスを定義した場合に発生します。

使用例
次の例では、CS0440 警告が生成されます。

```
// CS0440.cs
// Compile with: /W:1

using global = MyClass;   // CS0440
class MyClass
{
    static void Main()
    {
        // Note how global refers to the global namespace
        // even though it is redefined above.
        global::System.Console.WriteLine();
    }
}
```

コンパイラの警告 (レベル 2) CS0444

エラー メッセージ
定義済みの型 'type name 1' は、'System namespace 1' では見つかりませんでしたが、'System namespace 2' に見つかりました

Int32 などの定義済みのオブジェクトが、予期しない場所 ('System namespace 2') で見つかりました。

このエラーは、.NET Framework が正しくインストールされていないことを示します。これを解決するには、.NET Framework を再インストールします。

また、基本クラス ライブラリを独自に作成している場合にも、このエラーが発生する可能性があります。その場合は、mscorlib をビルドし直すことによってエラーを解決できます。

コンパイラの警告 (レベル 2) CS0458

エラー メッセージ
式の結果は常に型 '型名' の 'null' になります。

この警告は、null 許容式の結果が常に **null** になることを示しています。

次のコードでは、CS0458 警告が生成されます。

使用例
このエラーが発生する、null 許容型の演算の例を次に示します。

```
// CS0458.cs
using System;
public  class Test
{
    public static void Main()
    {
        int a = 5;
        int? b = a + null;    // CS0458
        int? qa = 15;
        b = qa + null;        // CS0458
        b -= null;            // CS0458
        int? qa2 = null;
        b = qa2 + null;       // CS0458
        qa2 -= null;          // CS0458
    }
}
```

コンパイラの警告 (レベル 2) CS0464

エラー メッセージ
型 't型' の null と比較すると、いつも 'false' を生成します。

この警告は、null 許容変数と、null との間で、== と != のいずれにも該当しない比較操作を実行した場合に発生します。このエラーを解決するには、値が **null** であることをチェックする必要があるかどうかを確認します。`i == null` のような比較の結果は true または false になります。`i > null` のような比較の結果は常に false になります。

使用例
次の例では CS0464 警告が生成されます。

```
// CS0464.cs
class MyClass
{
   public static void Main()
   {
      int? i = 0;
      if (i < null) ;   // CS0464

      i++;
   }
}
```

コンパイラの警告 (レベル 2) CS0467

エラー メッセージ
メソッド 'メソッド' とメソッド以外の 'non-method' があいまいです。メソッド グループを使用しています。

異なるインターフェイスから継承したメンバが同じシグネチャを持つ場合、あいまいさに関するエラーが発生します。

使用例
次の例では CS0467 警告が生成されます。

```
// CS0467.cs
interface IList
{
int Count { get; set; }
}
interface ICounter
{
void Count(int i);
}

interface IListCounter : IList, ICounter {}

class Driver
{
    void Test(IListCounter x)
    {
        x.Count = 1;
        x.Count(1);   // CS0467
        // To resolve change the method name "Count" to another name.
    }

    static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 2) CS0618

エラー メッセージ
'member' は古い形式です : 'text'

クラス メンバが Obsolete 属性でマークされました。これにより、クラス メンバが参照されると警告が発行されます。

次の例では警告 CS0618 が生成されます。

```
// CS0618.cs
// compile with: /W:2
using System;

public class C
{
   [Obsolete("Use newMethod instead", false)]   // warn if referenced
   public static void m2()
   {
   }

   public static void newMethod()
   {
   }
}

class MyClass
{
   public static void Main()
   {
      C.m2();  // CS0618
   }
}
```

コンパイラの警告 (レベル 2) CS0652

エラー メッセージ
整数定数への比較ができません。定数が型 'type' の範囲外です。

定数と変数の間の比較が検出されましたが、定数が変数の範囲外になっています。

次の例では CS0652 エラーが生成されます。

```
// CS0652.cs
// compile with: /W:2
public class Class1
{
   private static byte i = 0;
   public static void Main()
   {
      short j = 256;
      if (i == 256)   // CS0652, 256 is out of range for byte
         i = 0;
   }
}
```

コンパイラの警告 (レベル 2) CS0728

エラー メッセージ
using または lock ステートメントの引数であるローカルの '変数' への割り当てが間違っている可能性があります。

using ブロックまたは **lock** ブロックで一時的なリソース リークが発生する場合があります。次に例を示します。

```
thisType f = null;
using (f)
{
f = new thisType();
...
}
```

この場合、`thisType` 変数の元の値 (null など) は、**using** ブロックの実行が終了した時点で破棄されます。しかし、このブロック内で生成された `thisType` オブジェクトについては、最終的にガベージ コレクションが実行されるまで破棄されません。

このエラーを解決するには、次のようにします。

```
using (thisType f = new thisType())
{
...
}
```

これで、新たに割り当てられた `thisType` オブジェクトが破棄されるようになります。

使用例
次のコードでは、CS0728 警告が生成されます。

```
// CS0728.cs

using System;
public class ValidBase : IDisposable
{
    public void Dispose() {  }
}

public class Logger
{
    public static void dummy()
    {
        ValidBase vb = null;
        using (vb)
        {
            vb = null;  // CS0728
        }
        vb = null;
    }
    public static void Main() { }
}
```

コンパイラの警告 (レベル 2) CS1571

エラー メッセージ
'construct' の XML コメントで、'parameter' の param タグが重複しています。

/doc コンパイラ オプションの使用時に、同じメソッド パラメータに対する複数のコメントが見つかりました。重複する 1 行を削除してください。

次の例では CS1571 エラーが生成されます。

```
// CS1571.cs
// compile with: /W:2 /doc:x.xml

/// <summary>help text</summary>
public class MyClass
{
   /// <param name='Int1'>Used to indicate status.</param>
   /// <param name='Char1'>An initial.</param>
   /// <param name='Int1'>Used to indicate status.</param> // CS1571
   public static void MyMethod(int Int1, char Char1)
   {
   }

   /// <summary>help text</summary>
   public static void Main ()
   {
   }
}
```

コンパイラの警告 (レベル 2) CS1572

エラー メッセージ
'construct' の XML コメントで、'parameter' の param タグが存在しますが、その名前に相当するパラメータはありません。

/doc コンパイラ オプションの使用時に、メソッド用として指定されていないパラメータに対するコメントが見つかりました。名前属性に渡される値を変更するか、またはコメント行を 1 行削除してください。

次の例では警告 CS1572 が生成されます。

```
// CS1572.cs
// compile with: /W:2 /doc:x.xml

/// <summary>help text</summary>
public class MyClass
{
   /// <param name='Int1'>Used to indicate status.</param>
   /// <param name='Char1'>Used to indicate status.</param>
   /// <param name='Char2'>???</param> // CS1572
   public static void MyMethod(int Int1, char Char1)
   {
   }

   /// <summary>help text</summary>
   public static void Main ()
   {
   }
}
```

コンパイラの警告 (レベル 2) CS1587

エラー メッセージ
XML コメントが有効な言語要素の中にありません。

ドキュメント コメントの推奨タグは、すべての言語要素で使用できるわけではありません。たとえば、名前空間ではタグを使用できません。XML コメントの詳細については、「ドキュメント コメント用の推奨タグ (C# プログラミング ガイド)」を参照してください。

使用例
次の例では警告 CS1587 が生成されます。

```
// CS1587.cs
// compile with: /W:2 /doc:x.xml

/// <summary>test</summary>   // CS1587, tag not allowed on namespace
namespace MySpace
{
    class MyClass
    {
        public static void Main()
        {
        }
    }
}
```

コンパイラの警告 (レベル 2) CS1668

エラー メッセージ
'パス文字列' で指定された無効な検索パス 'パス' です -- 'システム エラー メッセージ'

コマンド ラインの /lib に指定されたパスが無効であるか、LIB 環境変数のパスが無効です。指定されたパスが存在し、アクセス可能であることを確認してください。エラー メッセージの単一引用符で囲まれた部分は、オペレーティング システムから返されるエラーです。

このエラーは、Visual Studio 2005 がインストールされているコンピュータから Visual Studio .NET 2002 または Visual Studio .NET 2003 をアンインストールした場合に発生することがあります。このエラーを修正するには、次の手順に従ってください。

1. [マイ コンピュータ] を右クリックし、[プロパティ] をクリックします。
2. [詳細] タブをクリックし、[環境変数] をクリックします。
3. 上部ペインの LIB エントリをクリックし、Version 2.0 の lib ファイルを指していることを確認します。

   Visual Studio 2005 を既定の場所にインストールした場合、完全パスは C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Lib になります。

コンパイラの警告 (レベル 2) CS1698

エラー メッセージ
循環アセンブリ参照 'AssemblyName1' は、出力アセンブリ名 'AssemblyName2' に対応していません。'AssemblyName1' への参照を追加するか、または出力アセンブリ名が一致するように変更してください。

CS1698 警告は、アセンブリ参照が不適切な場合に発生します。これは、参照アセンブリが再コンパイルされると発生する場合があります。解決するには、それ自体が、参照先のアセンブリの依存関係となっているアセンブリを置換しないでください。

使用例

```
// CS1698_a.cs
// compile with: /target:library /keyfile:mykey.snk
[assembly:System.Reflection.AssemblyVersion("2")]
public class CS1698_a {}
```

```
// CS1698_b.cs
// compile with: /target:library /reference:CS1698_a.dll /keyfile:mykey.snk
public class CS1698_b : CS1698_a {}
```

次の例では CS1698 警告が生成されます。

```
// CS1698_c.cs
// compile with: /target:library /out:cs1698_a.dll /reference:cs1698_b.dll /keyfile:mykey.s
nk
// CS1698 expected
[assembly:System.Reflection.AssemblyVersion("3")]
public class CS1698_c : CS1698_b {}
public class CS1698_a {}
```

コンパイラの警告 (レベル 2) CS1701

エラー メッセージ
参照 "アセンブリ名 #1" は "アセンブリ名 #2" と一致していると仮定して、実行時ポリシーを指定する必要がある可能性があります。

2 つのアセンブリ間で、リリース番号またはバージョン番号が異なります。正しく統合するためには、以下に示す例に従って、アプリケーションの .config ファイルでディレクティブを指定し、アセンブリの厳密名を正しく指定する必要があります。

使用例

次のマルチファイルの例では、2 つの異なる外部エイリアスを使ってアセンブリを参照しています。この最初の例では、CS1701_d アセンブリを生成する古いバージョンのコードを作成します。

```
// CS1701_a.cs
// compile with: /target:library /out:cs1701_d.dll /keyfile:mykey.snk
using System.Reflection;
[assembly:AssemblyVersion("1.0")]
public class A {
   public void M1() {}
}

public class C1 {}
```

これは、CS1701_d アセンブリの新しいバージョンを生成するコードです。古いバージョンとは異なるディレクトリにコンパイルされることに注意してください。これは、出力ファイルが同じ名前であるために必要な処理です。

```
// CS1701_b.cs
// compile with: /target:library /out:c:\\cs1701_d.dll /keyfile:mykey.snk
using System.Reflection;
[assembly:AssemblyVersion("2.0")]
public class A {
   public void M2() {}
   public void M1() {}
}

public class C2 {}
public class C1 {}
```

この例では、外部エイリアス A1 および A2 が設定されます。

```
// CS1701_c.cs
// compile with: /target:library /reference:A2=c:\\cs1701_d.dll /reference:A1=cs1701_d.dll

extern alias A1;
extern alias A2;
// using System;
using a1 = A1::A;
using a2 = A2::A;

public class Ref {
   public static a1 A1() { return new a1(); }
   public static a2 A2() { return new a2(); }

   public static A1::C1 M1() { return new A1::C1(); }
   public static A2::C2 M2() { return new A2::C2(); }
}
```

この例では、A の 2 つの異なるエイリアスを使用してメソッドが呼び出されます。次の例では C1701 警告が生成されます。

```
// CS1701_d.cs
// compile with: /reference:c:\\CS1701_d.dll /reference:CS1701_c.dll
// CS1701 expected
```

```
class Tester {
   public static void Main() {
      Ref.A1().M1();
      Ref.A2().M2();
   }
}
```

コンパイラの警告 (レベル 2) CS1710

エラー メッセージ
'型' 上の XML コメントには 'パラメータ' の重複する typeparam タグがあります。

ジェネリック型に対するドキュメント コメントで、型パラメータ用のタグが重複しています。

使用例
次のコードでは、CS1710 警告が表示されます。

```
// CS1710.cs
// To resolve this warning, delete one of the duplicate <typeparam>'s.
using System;
class Stack<ItemType>
{
}

/// <typeparam name="MyType">can be an int</typeparam>
/// <typeparam name="MyType">can be an int</typeparam>
class MyStackWrapper<MyType>
{
    // Open constructed type Stack<MyType>.
    Stack<MyType> stack;
    public MyStackWrapper(Stack<MyType> s)
    {
        stack = s;
    }
}

class CMain
{
    public static void Main()
    {
        // Closed constructed type Stack<int>.
        Stack<int> stackInt = new Stack<int>();
        MyStackWrapper<int> MyStackWrapperInt =
            new MyStackWrapper<int>(stackInt);
    }
}
```

コンパイラの警告 (レベル 2) CS1711

エラー メッセージ
'型' の XML コメントは 'パラメータ' の typeparam タグを含みますが、その名前の型パラメータはありません

ジェネリック型に対するドキュメント コメントで、タグに指定された型パラメータの名前に誤りがあります。

使用例
次のコードでは、CS1711 警告が生成されます。

```
// cs1711.cs
// compile with: /doc:cs1711.xml
// CS1711 expected
using System;
///<typeparam name="WrongName">can be an int</typeparam>
class CMain
{
    public static void Main() { }
}
```

コンパイラの警告 (レベル 2) CS3019

エラー メッセージ
このアセンブリの外から認識できないため、CLS 準拠の確認は '型' で実行されません

この警告は、CLSCompliantAttribute 属性を持つ型またはメンバを、別のアセンブリから参照できない場合に発生します。この警告を解決するには、他のアセンブリから参照できないクラスまたはメンバの属性を削除するか、他のアセンブリがその型またはメンバを参照できるように修正します。CLS 準拠の詳細については、「CLS 準拠コードの記述」を参照してください。

使用例
次の例では CS3019 警告が生成されます。

```
// CS3019.cs
// compile with: /W:2

using System;

[assembly: CLSCompliant(true)]

// To fix the error, remove the next line
[CLSCompliant(true)]  // CS3019
class C
{
    [CLSCompliant(false)]  // CS3019
    void Foo()
    {
    }

    static void Main()
    {
    }
}
```

参照
概念
共通言語仕様

コンパイラの警告 (レベル 2) CS3021

エラー メッセージ
アセンブリに CLSCompliant 属性が含まれていないため、'型' には CLSCompliant 属性が必要ありません。

この警告は、アセンブリ レベルの CLSCompliant 属性が true (`[assembly: CLSCompliant(true)]`) に設定されていないアセンブリのクラスに `[CLSCompliant(false)]` が見つかった場合に発生します。アセンブリ自体が CLS 準拠として宣言されていないため、そのアセンブリに含まれるコードは暗黙的に非 CLS 準拠として見なされます。明示的に宣言する必要はありません。CLS 準拠の詳細については、「CLS 準拠コードの記述」を参照してください。

この警告が表示されないようにするには、該当する属性を削除するか、アセンブリ レベルの属性を追加します。

使用例
次の例では、CS3021 エラーが生成されます。

```
// CS3021.cs
using System;
// Uncomment the following line to declare the assembly CLS Compliant,
// and avoid the warning without removing the attribute on the class.
//[assembly: CLSCompliant(true)]

// Remove the next line to avoid the warning.
[CLSCompliant(false)]               // CS3021
public class C
{
    public static void Main()
    {
    }
}
```

参照
概念
共通言語仕様

コンパイラの警告 (レベル 3) CS0067

エラー メッセージ
イベント 'event' は使用されませんでした。

イベントが宣言されましたが、宣言されたクラス内で使用されませんでした。

次の例では CS0067 エラーが生成されます。

```
// CS0067.cs
// compile with: /W:3
using System;
delegate void MyDelegate();

class MyClass
{
   public event MyDelegate evt;   // CS0067
   // uncomment TestMethod to resolve this CS0067
/*
   private void TestMethod()
   {
      if (evt != null)
         evt();
   }
*/
   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 3) CS0105

エラー メッセージ
'namespace' の using ディレクティブは、この名前空間で既に使用されています。

1 回しか宣言できない名前空間が複数回宣言されました。重複する名前空間の宣言をすべて削除してください。

次の例では CS0105 エラーが生成されます。

```
// CS0105.cs
// compile with: /W:3
using System;
using System;   // CS0105

public class a
{
   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 3) CS0168

エラー メッセージ
変数 'var' は宣言されていますが、使用されませんでした。

変数を宣言しても使用しない場合に、この警告が生成されます。

次の例では警告 CS0168 が 2 回発生します。

```
// CS0168.cs
// compile with: /W:3
public class clx
{
   public int i;
}

public class clz
{
   public static void Main()
   {
      int j = 0;   // CS0168, uncomment the following line
      // j++;
      clx a;       // CS0168, try the following line instead
      // clx a = new clx();
   }
}
```

コンパイラの警告 (レベル 3) CS0169

エラー メッセージ
private フィールド 'class member' が一度も使用されませんでした。

プライベート変数が宣言されましたが、参照されませんでした。この警告は、通常、クラスのプライベート メンバを宣言し、そのメンバを使用しない場合に生成されます。

次の例では警告 CS0169 が生成されます。

```
// compile with: /W:3
using System;
public class ClassX
{
   int i;   // CS0169, i is not used anywhere

   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 3) CS0219

エラー メッセージ
変数 'variable' は割り当てられていますが、その値が使用されていません。

変数を宣言して代入するときにレベル 3 の警告が表示されますが、その変数は使用されません。

次の例では警告 CS0219 が生成されます。

```
// CS0219.cs
// compile with: /W:3
public class MyClass
{
   public static void Main()
   {
      int a = 0;   // CS0219
   }
}
```

コンパイラの警告 (レベル 3) CS0282

エラー メッセージ
部分クラスまたは構造体 '型' の複数の宣言内にあるフィールド間に、定義された順序がありません。順序を指定するには、すべてのインスタンス フィールドが同じ宣言内になければなりません。

このエラーを解決するには、すべてのメンバ変数を 1 つの部分クラス定義にまとめます。

このエラーが発生する一般的な原因として、**struct** が分割定義されているとき、そのメンバ変数が別々の場所に定義されていることが考えられます。

次のコードでは、CS0282 エラーが生成されます。

使用例
次のコードには、**struct** が 1 つ定義されています。以下の 2 つのモジュールを、次のコマンドを使って、1 回の手順でコンパイルします。

```
csc /targt:library cs0282_1.cs cs0282_2.cs
```

```
partial struct A
{
    int i;
}
```

次のコードには、同じ **struct** の競合する定義が含まれています。

```
partial struct A
{
    int j;
}
```

コンパイラの警告 (レベル 3) CS0414

エラー メッセージ
private フィールド 'フィールド' は割り当てられていますが、その値が使用されていません

この警告は、値の代入されたプライベートなフィールドが型に存在するが、その後、値がまったく読み取られない場合に発生します。不要な代入が、パフォーマンスに悪影響を及ぼす場合があります。

次の例では CS0414 エラーが生成されます。

```
// CS0414
// compile with: /W3
class C
{
   private int i = 1;  // CS0414

   public static void Main()
   { }
}
```

コンパイラの警告 (レベル 3) CS0419

エラー メッセージ
'メソッド名 1' は cref 属性内の不適切な参照です。'Method Name2' を仮定しますが、'Method Name3' を含む別のオーバーロードに一致した可能性もあります。

コード内の XML ドキュメント コメントで、参照を解決できませんでした。このエラーは、メソッドがオーバーロードされているなど、同じ名前の重複する識別子が見つかった場合に発生することがあります。この警告を解決するには、修飾名を使って参照を明確に指定するか、オーバーロードされたメソッドのパラメータをかっこで囲って指定します。

次の例では、CS0419 エラーが生成されます。

```
// cs0419.cs
// compile with: /doc:x.xml /W:3
interface I
{
   /// text for F(void)
   void F();
   /// text for F(int)
   void F(int i);
}
/// text for class MyClass
public class MyClass
{
   /// <see cref="I.F"/>
   public static void MyMethod(int i)
   {
   }
/* Try this instead:
   /// <see cref="I.F(int)"/>
   public static void MyMethod(int i)
   {
   }
*/
   /// text for Main
   public static void Main ()
   {
   }
}
```

コンパイラの警告 (レベル 2) CS0469

エラー メッセージ
goto case' 値は型 '型' に暗黙的に変換できません。

goto case を使用する場合は、goto case の値からスイッチの型への暗黙の型変換が必要です。

使用例
次の例では CS0469 エラーが生成されます。

```
// CS0469.cs
// compile with: /W:2
class Test
{
   static void Main()
   {
      char c = (char)180;
      switch (c)
      {
         case (char)127:
            break;

         case (char)180:
            goto case 127;   // CS0469
            // try the following line instead
            // goto case (char) 127;
      }
   }
}
```

コンパイラの警告 (レベル 3) CS0642

エラー メッセージ
empty ステートメントが間違っている可能性があります。

条件付きステートメントの後にセミコロン (;) があると、コードが予測どおりに実行されないことがあります。

/nowarn コンパイラ オプションまたは **#pragmas warning** を使用すると、この警告を無効にできます。詳細については、「/nowarn (指定した警告の非表示) (C# コンパイラ オプション)」または「#pragma 警告 (C# リファレンス)」を参照してください。

次の例では警告 CS0642 が生成されます。

```
// CS0642.cs
// compile with: /W:3
class MyClass
{
   public static void Main()
   {
      int i;

      for (i = 0; i < 10; i += 1);   // CS0642 semicolon intentional?
      {
         System.Console.WriteLine (i);
      }
   }
}
```

コンパイラの警告 (レベル 3) CS0659

エラー メッセージ
'class' は Object.Equals(object o) をオーバーライドしますが、Object.GetHashCode() をオーバーライドしません。

Equals 関数のオーバーライドが検出されましたが、**GetHashCode** のオーバーライドがありません。**Equals** のオーバーライドは、**GetHashCode** もオーバーライドすることを示します。

詳細については、次のトピックを参照してください。

- Hashtable.
- Equals および等値演算子 (==) 実装のガイドライン
- Equals メソッドの実装
- GetHashCode

次の例では警告 CS0659 が生成されます。

```
// CS0659.cs
// compile with: /W:3 /target:library
class Test
{
   public override bool Equals(object o) { return true; }   // CS0659
}

// OK
class Test2
{
   public override bool Equals(object o) { return true; }
   public override int GetHashCode() { return 0; }
}
```

コンパイラの警告 (レベル 3) CS0660

エラー メッセージ
'クラス' は 演算子 == または演算子 != を定義しますが、Object.Equals(object o) をオーバーライドしません。

ユーザー定義の等値演算子または非等値演算子が検出されましたが、**Equals** 関数のオーバーライドがありません。ユーザー定義の等値演算子または非等値演算子は、**Equals** 関数もオーバーライドすることを示します。

次の例では警告 CS0660 が生成されます。

```
// CS0660.cs
// compile with: /W:3 /warnaserror
class Test   // CS0660
{
   public static bool operator == (object o, Test t)
   {
      return true;
   }

   // uncomment the Equals function to resolve
   // public override bool Equals(object o)
   // {
   //    return true;
   // }

   public override int GetHashCode()
   {
      return 0;
   }

   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 3) CS0661

エラー メッセージ
'class' は演算子 == または演算子! = を定義しますが、Object.GetHashCode() をオーバーライドしません。

ユーザー定義の等値演算子または非等値演算子が検出されましたが、**GetHashCode** 関数のオーバーライドがありません。ユーザー定義の等値演算子または非等値演算子は、**GetHashCode** 関数もオーバーライドすることを示します。

次の例では警告 CS0661 が生成されます。

```
// CS0661.cs
// compile with: /W:3
class Test   // CS0661
{
   public static bool operator == (object o, Test t)
   {
      return true;
   }

   public static bool operator != (object o, Test t)
   {
      return true;
   }

   public override bool Equals(object o)
   {
      return true;
   }

   // uncomment the GetHashCode function to resolve
   // public override int GetHashCode()
   // {
   //    return 0;
   // }

   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 3) CS0665

エラー メッセージ
条件式の割り当ては常に定数です。== を使用するつもりで = を使用しましたか?

条件式で == 演算子ではなく = 演算子が使用されました。

次の例では CS0665 エラーが生成されます。

```
// CS0665.cs
// compile with: /W:3
class Test
{
   public static void Main()
   {
      bool i = false;

      if (i = true)   // CS0665
      // try the following line instead
      // if (i == true)
      {
      }

      System.Console.WriteLine(i);
   }
}
```

コンパイラの警告 (レベル 3) CS0675

エラー メッセージ
Bitwise-or 演算子が sign-extended 演算子で使用されています。まず、小さい符号なしの型をキャストしてみてください。

コンパイラで変数を暗黙に拡張して符号拡張した後に、結果値をビットごとの OR 演算で使用しました。これにより、予測不可能な動作を起こすことがあります。

次の例では警告 CS0675 が生成されます。

```
// CS0675.cs
// compile with: /W:3
using System;

public class sign
{
   public static void Main()
   {
      int hi = 1;
      int lo = 1;
      long value = (((long)hi) << 32) | lo;              // CS0675
      // try the following line instead
      // long value = (((long)hi) << 32) | ((uint)lo);   // correct
   }
}
```

コンパイラの警告 (レベル 3) CS0693

エラー メッセージ
型パラメータ '型パラメータ' は、外の型 '型!' からの型パラメータと同じ名前です

内部クラスと外部クラスとで同じ名前の型パラメータが使用されています。この状況を回避するには、いずれかの型パラメータの名前を変更します。

使用例
次の例では CS0693 エラーが生成されます。

```
// CS0693.cs
// compile with: /W:3 /target:library
class Outer<T>
{
   class Inner<T> {}   // CS0693
   // try the following line instead
   // class Inner<U> {}
}
```

コンパイラの警告 (レベル 3) CS1700

エラー メッセージ
アセンブリ参照 'アセンブリ名' は無効なため、解決できません。

この警告は、InternalsVisibleToAttribute などの属性が正しく指定されていないことを示します。

詳細については、「フレンド アセンブリ (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS1700 警告が生成されます。

```
// CS1700.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:InternalsVisibleTo("app2, Retargetable=f")]   // CS1700
[assembly:InternalsVisibleTo("app2")]   // OK
```

コンパイラの警告 (レベル 3) CS1702

エラー メッセージ
参照 "アセンブリ名 #1" は "アセンブリ名 #2" と一致していると仮定して、実行時ポリシーを指定する必要がある可能性があります。

2 つのアセンブリ参照が、異なるビルド番号/リビジョン番号を持つため、自動的には統合できません。アプリケーションの .config ファイル内で、強制的に統合するように指定することにより、実行時ポリシーを作成する必要があります。

コンパイラの警告 (レベル 3) CS1717

エラー メッセージ
同じ変数に割り当てられました。他の変数に割り当てますか?

この警告は、`a = a` のように、同じ変数どうしで代入しようとした場合に発生します。

この警告が発生する一般的な原因としては、次のようなケースが考えられます。

- **if** ステートメントで `a = a` のような条件を記述した (例 : `if (a = a)`)。実際は `if (a == a)` と記述しようとした可能性があります。ただし、この条件は常に true を返すため、より簡潔に `if (true)` と記述することもできます。
- タイプミス。`a = b` と記述すべきところを間違って入力したケースです。
- コンストラクタでパラメータ名とフィールド名が競合するにもかかわらず、**this** キーワードを使用しなかった (正しくは `this.a = a`)。

使用例

次の例では CS1717 エラーが生成されます。

```
// CS1717.cs
// compile with: /W:3
public class Test
{
   public static void Main()
   {
      int x = 0;
      x = x;   // CS1717
   }
}
```

コンパイラの警告 (レベル 3) CS1718

エラー メッセージ
同じ変数と比較されました。他の変数と比較しますか?

単に比較対象を間違って指定した場合は、ステートメントを正しく修正してください。

または、`if (a == a) (true)` や `if (a < a) (false)` などのステートメントで、true か false かを判定している場合も考えられます。その場合は、単に `if (true)` または `if (false)` のように記述することをお勧めします。これには 2 つの理由があります。

- コードが簡潔になります。表現を簡潔にした方が、コードが明快になります。
- 混乱を防ぐことができます。C# 2.0 の新機能として、null 許容値型を使用できます。これは、SQL Server で使用されているプログラミング言語 T-SQL における **null** 値とよく似ています。T-SQL の使用経験のある開発者が `if (a == a)` のような式を見ると、三項条件のロジックを想像し、null 許容型の影響について誤解を招く可能性があります。**true** または **false** を使用することにより、このような誤解を防ぐことができます。

使用例
次のコードでは、CS1718 警告が生成されます。

```
// CS1718.cs
using System;
public class Tester
{
    public static void Main()
    {
        int i = 0;
        if (i == i) { // CS1718.cs
        //if (true) {
            i++;
        }
    }
}
```

コンパイラの警告 (レベル 4) CS0028

エラー メッセージ
'関数宣言' で間違った認証が使われています。エントリ ポイントとして使用することはできません。

無効なシグネチャを使用して宣言されているため、**Main** のメソッド宣言は無効です。`Main` は静的として宣言され、int または void を返す必要があります。詳細については、「Main() とコマンド ライン引数 (C# プログラミング ガイド)」を参照してください。

次の例では警告 CS0028 が生成されます。

```
// CS0028.cs
// compile with: /W:4 /warnaserror
public class a
{
    public static double Main(int i)   // CS0028
    // Try the following line instead:
    // public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 4) CS0078

エラー メッセージ
'l' と数字の '1' との混同を避けるため、'L' を使用してください。

この警告は、大文字の L ではなく小文字の l を使用した **long** 型へのキャストがコンパイラで検出されると発生します。

次の例では警告 CS0078 が生成されます。

```
// CS0078.cs
// compile with: /W:4
class test {
   public static void TestL(long i)
   {
   }

   public static void TestL(int i)
   {
   }

   public static void Main()
   {
      TestL(25l);   // CS0078
      // try the following line instead
      // TestL(25L);
   }
}
```

コンパイラの警告 (レベル 4) CS0109

エラー メッセージ
メンバ 'member' は継承メンバを隠しません。キーワード new は必要ありません。

基本クラス内の既存の宣言をオーバーライドしないクラス宣言に、new キーワードが含まれていました。**new** キーワードは削除できます。

次の例では警告 CS0109 が生成されます。

```
// CS0109.cs
// compile with: /W:4
namespace x
{
   public class a
   {
      public int i;
   }

   public class b : a
   {
      public new int i;
      public new int j;   // CS0109
      public static void Main()
      {
      }
   }
}
```

コンパイラの警告 (レベル 4) CS0402

エラー メッセージ
'識別子': エントリ ポイントがジェネリックになったり、ジェネリック型の中に存在したりすることはできません。

ジェネリック型にエントリ ポイントが存在しています。この警告を解決するには、ジェネリック型以外のクラスまたは構造体に Main を実装します。

```
// CS0402.cs
// compile with: /W:4
class C<T>
{
   public static void Main()  // CS0402
   {

   }
}

class CMain
{
   public static void Main() {}
}
```

コンパイラの警告 (レベル 4) CS0422

エラー メッセージ
/incremental オプションは現在サポートされていません。

インクリメンタル コンパイル (**/incr** または **/incremental**) は Visual C# 2005 ではサポートされていません。

コンパイラの警告 (レベル 4) CS0429

エラー メッセージ
到達できない式コードが検出されました。

このエラーは、制御の渡らない式がコード中に存在する場合に発生します。次の例では、`false && myTest()` という条件式がこれに該当します。`&&` 演算子の左辺は常に false を返すため、`myTest()` メソッドが評価されることはありません。`&&` 演算子は、**false** として評価されるステートメントが見つかると、そこで評価を終了します。その結果、右辺の式が評価されることはありません。

使用例
次のコードでは、CS0429 エラーが生成されます。

```
// CS0429.cs
public class cs0429
{
    public static void Main()
    {
        if (false && myTest())  // CS0429
        // Try the following line instead:
        // if (true && myTest())
        {
        }
        else
        {
            int i = 0;
            i++;
        }
    }

    static bool myTest() { return true; }
}
```

コンパイラの警告 (レベル 4) CS0628

エラー メッセージ
'member' : 新規のプロテクト メンバがシール クラスで宣言されました。

シールクラスを継承してプロテクトメンバを使用できるクラスは他にないため、シール クラスではプロテクト メンバを使用できません。

次の例では CS0628 エラーが生成されます。

```
// CS0628.cs
// compile with: /W:4
sealed class C
{
   protected int i;   // CS0628
}

class MyClass
{
   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 4) CS0649

エラー メッセージ
フィールド 'field' は割り当てられません。常に既定値 'value' を使用します。

値が代入されていない、初期化されないプライベート フィールドまたは内部フィールドの宣言が検出されました。

次の例では警告 CS0649 が生成されます。

```
// CS0649.cs
// compile with: /W:4
using System.Collections;

class MyClass
{
   Hashtable table;  // CS0649
   // You may have intended to initialize the variable to null
   // Hashtable table = null;

   // Or you may have meant to create an object here
   // Hashtable table = new Hashtable();

   public void Func(object o, string p)
   {
      // Or here
      // table = new Hashtable();
      table[p] = o;
   }

   public static void Main()
   {
   }
}
```

コンパイラの警告 (レベル 4) CS1573

エラー メッセージ
パラメータ 'パラメータ' には 'パラメータ' の XML コメント内に対応する param タグがありませんが、他のパラメータにはあります。

/doc コンパイラ オプションの使用時に、メソッド内のすべてのパラメータではなく、一部のパラメータに対するコメントが見つかりました。パラメータのコメントを入力し忘れた可能性があります。

次の例では警告 CS1573 が生成されます。

```
// CS1573.cs
// compile with: /W:4
public class MyClass
{
   /// <param name='Int1'>Used to indicate status.</param>
   // enter a comment for Char1?
   public static void MyMethod(int Int1, char Char1)
   {
   }

   public static void Main ()
   {
   }
}
```

コンパイラの警告 (レベル 4) CS1591

エラー メッセージ
公開されている型またはメンバ 'Type_or_Member' の XML コメントがありません。

/doc コンパイラ オプションが指定されましたが、構成要素にはコメントがありませんでした。

次の例では警告 CS1591 が生成されます。

```
// CS1591.cs
// compile with: /W:4 /doc:x.xml

/// text
public class Test
{
   // /// text
   public static void Main()   // CS1591, remove "//" from previous line
   {
   }
}
```

コンパイラの警告 (レベル 4) CS1610

エラー メッセージ
既定の Win32 リソースに使用される一時ファイル 'file' を削除できません -- resource

/win32res コンパイラ オプションを使用するときに **%TEMP%** ディレクトリに DELETE アクセス許可がない場合、この警告はコンパイラが作成した一時ファイルを削除できなかったことを示します。

%TEMP% ディレクトリに対する読み取り/書き込み/削除のアクセス許可があることを確認してください。

必要に応じて、これらのファイルを手動で削除できます。この処理で、C# やその他のプログラムに影響を与えることはありません。

コンパイラの警告 (レベル 4) CS1712

エラー メッセージ
型パラメータ '型パラメータ' は、対応する typeparam タグが '型' の XML コメントにありませんが、他の型パラメータはあります

ジェネリック型に対するドキュメントの **typeparam** タグが指定されていません。詳細については、「<typeparam> (C# プログラミング ガイド)」を参照してください。

使用例

次のコードでは、CS1712 警告が生成されます。このエラーを解決するには、型パラメータである S 用の **typeparam** タグを追加します。

```
// CS1712.cs
// compile with: /doc:cs1712.xml
using System;
class Test
{
    public static void Main() {}
    /// <param name="j"> This is the j parameter.</param>
    /// <typeparam name="T"> This is the T type parameter.</typeparam>
    public void bar<T,S>(int j) {}  // CS1712
}
```

コンパイラ エラー CS1725

エラー メッセージ
フレンド アセンブリ参照 '参照' は無効です。InternalsVisibleTo 宣言にバージョン、カルチャ、公開キー トークン、またはプロセッサ属性を指定することはできません。

フレンド アセンブリの参照にはバージョンのカルチャを追加できません。フレンド アセンブリから部分クラスを参照できるようにする必要があります。

使用例
次の例では CS1725 エラーが生成されます。

```
// CS1725.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:InternalsVisibleTo("partial01,version=1.1.0.0")]   // CS1725
// try the following line instead
// [assembly:InternalsVisibleTo("partial01")]

partial class TestClass
{
   public static string strBar = "my string";
}
```

コンパイラ エラー CS1726

エラー メッセージ
フレンド アセンブリ参照 '参照' は無効です。厳密な名前の署名つきアセンブリはその InternalsVisibleTo 宣言内で公開キーを指定しなければなりません。

厳密な名前で署名されたアセンブリでは、他の厳密な名前で署名されたアセンブリに対して、InternalsVisibleToAttribute で作成されたフレンド アセンブリのアクセス権のみを許可します。

CS1726 を解決するには、フレンド アクセスを許可するアセンブリに署名する (厳密な名前を付ける) か、フレンド アクセスを許可しないようにします。

詳細については、「フレンド アセンブリ (C# プログラミング ガイド)」を参照してください。

使用例
次の例では CS1726 エラーが生成されます。

```
// CS1726.cs
// compile with: /keyfile:CS1726.key /target:library
using System.Runtime.CompilerServices;
[assembly:InternalsVisibleTo("UnsignedAssembly")]   // CS1726
// try the following line instead
// [assembly:InternalsVisibleTo("SignedAssembly, PublicKey=0024000004800000940000000602000000240000525341310004000001000100031d7b6f3abc16c7de526fd67ec2926fe68ed2f9901afbc5f1b6b428bf6cd9086021a0b38b76bc340dc6ab27b65e4a593fa0e60689ac98dd71a12248ca025751d135df7b98c5f9d09172f7b62dabdd302b2a1ae688731ff3fc7a6ab9e8cf39fb73c60667e1b071ef7da5838dc009ae0119a9cbff2c581fc0f2d966b77114b2c4")]

class A {}
```

コンパイラの警告 (レベル 2) CS0472

エラー メッセージ
型 'value2' の値が型 'value3' の 'null' に等しくなることはないので、式の結果は常に 'value1' になります。

定数の null 値が指定された演算子を使用すると、この警告が生成されます。

使用例
次の例では CS0472 エラーが生成されます。

```
public class Test
{
    public static int Main()
    {
        int i = 5;
        int counter = 0;

        // Comparison:
        if (i == null)  // CS0472
        // To resolve, use a valid value for i.
            counter++;
        return counter;
    }
}
```

コンパイラ エラー CS1727

エラー メッセージ
承認なしにエラー報告を自動的に送信することはできません。エラー報告送信の承認を受けるには、'' を参照してください。

エラー テキストに記載されている Web サイトに、Visual Studio 2005 コマンド ライン ツールで自動的なエラー レポートを有効にする方法が説明されています。

使用例
次の例では CS1727 エラーが生成されます。

```
// CS1727.cs
// compile with: /errorreport:send
// CS1727 expected
class Test {}
```

コンパイラ エラー CS0735

エラー メッセージ
無効な型が TypeForwardedTo 属性の引数として指定されました

次の例では CS0735 エラーが生成されます。

```
// CS735.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:TypeForwardedTo(typeof(int[]))]   // CS0735
[assembly:TypeForwardedTo(typeof(string))]   // OK
```

コンパイラ エラー CS1057

エラー メッセージ
'メンバ': 静的クラスにプロテクト メンバを含むことはできません。

このエラーは、静的クラス内でプロテクト メンバが宣言されたときに生成されます。

使用例
次の例では CS1057 エラーが生成されます。

```
// CS1057.cs
using System;

static class Class1
{
    protected static int x;   // CS1057
    public static void Main()
    {
    }
}
```

コンパイラの警告 (レベル 1) CS1058

エラー メッセージ
前の catch 句は例外のすべてを既にキャッチしました。スローされる例外以外のものはすべて System.Runtime.CompilerServices.RuntimeWrappedException にラップされます。

`catch()` ブロックで、`catch (System.Exception e)` ブロックの後ろに例外型が指定されていない場合、この属性はエラー CS1058 を引き起こします。`catch()` ブロックが例外をキャッチしないという警告が表示されます。

`catch (System.Exception e)` ブロックの後ろの `catch()` ブロックは、AssemblyInfo.cs ファイル内で **RuntimeCompatibilityAttribute** が false に設定されている場合 (`[assembly: RuntimeCompatibilityAttribute(WrapNonExceptionThrows = false)]`)、非 CLS 準拠の例外をキャッチできます。この属性が明示的に false に設定されていない場合、スローされた非 CLS 準拠の例外はすべて Exceptions としてラップされ、`catch (System.Exception e)` ブロックでキャッチされます。詳細については、「方法 : CLS 準拠でない例外をキャッチする」を参照してください。

使用例
次の例では CS1058 警告が生成されます。

```
// CS1058.cs
// CS1058 expected
using System.Runtime.CompilerServices;

// the following attribute is set to true by default in the C# compiler
// set to false in your source code to resolve CS1058
[assembly: RuntimeCompatibilityAttribute(WrapNonExceptionThrows = false)]

class TestClass
{
   static void Main()
   {
      try {}

      catch (System.Exception e) {
         System. Console.WriteLine("Caught exception {0}", e);
      }

      catch {}   // CS1058. This line will never be reached.
   }
}
```

C# 用語集

IDE

統合開発環境 (Integrated Development Environment) の略語。コンパイラ、デバッガ、コード エディタ、デザイナなどの各種開発ツールの統合ユーザー インターフェイスを提供するアプリケーションです。

アクセサ [accessor]

プロパティに関連付けられているプライベート データ メンバの値を設定または取得するメソッド。読み書き可能なプロパティには、**get** アクセサと **set** アクセサがあります。読み取り専用であるプロパティのアクセサは、**get** だけです。詳細については、「プロパティ」を参照してください。

アクセス可能なメンバ [accessible member]

特定の型からアクセスできるメンバ。ある型でアクセス可能なメンバが、他の型からもアクセスできるとは限りません。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」および「フレンド アセンブリ (C# プログラミング ガイド)」を参照してください。

アクセス修飾子 [access modifier]

型または型のメンバへのアクセスを制限するキーワード (private、protected、internal、public など)。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

アクセス不可のメンバ [inaccessible member]

特定の型でアクセスできないメンバ。ある型ではアクセス不可のメンバが、他の型からもアクセスできないとは限りません。詳細については、「アクセス修飾子 (C# プログラミング ガイド)」を参照してください。

値型 [value type]

値型は、ヒープに割り当てられる参照型とは違って、スタックに割り当てられるデータ型です。組み込み型は、数値型、構造体型、および null 許容型を含め、すべて値型です。クラス型と文字列型は参照型です。詳細については、「値型 (C# リファレンス)」を参照してください。

イベント [event]

クラスまたは構造体のメンバ。変更通知を送信します。詳細については、「イベント (C# プログラミング ガイド)」を参照してください。

入れ子にされた型 [nested type]

別の型の宣言の中で宣言されている型。

インターフェイス [interface]

パブリック メソッド、イベント、およびデリゲートのシグネチャのみを含む型。インターフェイスを継承するオブジェクトは、インターフェイスで定義されているメソッドとイベントをすべて実装する必要があります。クラスや構造体は、任意の数のインターフェイスを継承できます。詳細については、「インターフェイス (C# プログラミング ガイド)」を参照してください。

オブジェクト [object]

クラスのインスタンス。オブジェクトはメモリ内に存在し、データとデータを操作するメソッドを含みます。詳細については、「オブジェクト、クラス、および構造体 (C# プログラミング ガイド)」を参照してください。

基本クラス [base class]

他のクラス (派生クラス) に継承されるクラス。詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

クラス [class]

オブジェクトを記述するデータ型。クラスは、データと、データを操作するメソッドの両方を含みます。詳細については、「クラス」を参照してください。

継承 [inheritance]

C# では継承がサポートされているので、別のクラス (基本クラス) から派生したクラスは、基本クラスと同じメソッドとプロパティを継承します。継承は、基本クラスと派生クラスに関係します。詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

構造体 [struct]

通常、一定の論理的な関係を持つ少数の変数を格納するために使用する複合データ型。構造体は、メソッドやイベントを格納することもできます。構造体は、継承はサポートしませんが、インターフェイスはサポートします。構造体は値型ですが、クラスは参照型です。詳細については、「構造体 (C# プログラミング ガイド)」を参照してください。

コンストラクタ [constructor]

クラスまた構造体の特殊なメソッド。同じ型のオブジェクトを初期化します。詳細については、「コンストラクタ (C# プログラミング ガイド)」を参照してください。

参照型 [reference type]

データ型の 1 つ。参照型として宣言された変数は、データ格納先の位置を指します。詳細については、「参照型 (C# リファレンス)」を参照してください。

ジェネリック [generics]

ジェネリックでは、型パラメータで定義されているクラスまたはメソッドを定義できます。クライアント コードで型をインスタンス化するときは、特定の型を引数として指定します。詳細については、「ジェネリック (C# プログラミング ガイド)」を参照してください。

静的 [static]

静的として宣言されたクラスやメソッドは、**new** キーワードを使用してあらかじめインスタンス化しなくても存在します。`Main()` は静的メソッドです。詳細については、「静的クラスと静的クラス メンバ (C# プログラミング ガイド)」を参照してください。

デストラクタ [destructor]

クラスまたは構造体の特殊なメソッド。システムによってインスタンスを破棄できるようにします。詳細については、「デストラクタ (C# プログラミング ガイド)」を参照してください。

デリゲート [delegate]

デリゲートは、メソッドを参照する型です。デリゲートにメソッドを代入すると、デリゲートはそのメソッドとまったく同じように動作します。詳細については、「デリゲート (C# プログラミング ガイド)」を参照してください。

匿名メソッド [anonymous method]

匿名メソッドは、デリゲートにパラメータとして渡されるコード ブロックです。詳細については、「匿名メソッド (C# プログラミング ガイド)」を参照してください。

派生クラス [derived class]

継承を使用して、他のクラス (基本クラス) の動作とデータを取得、追加、または変更するクラス。詳細については、「継承 (C# プログラミング ガイド)」を参照してください。

反復子 [iterator]

反復子は、コレクションまたは配列を含むクラスのコンシューマが、foreach、in (C# リファレンス) を使用してそのコレクションや配列を反復処理できるようにするメソッドです。

フィールド [field]

直接アクセスできるクラスまたは構造体のデータ メンバ。

プロパティ [property]

アクセサによってアクセスされるデータ メンバ。詳細については、「プロパティ」を参照してください。

変更可能な型 [mutable type]

インスタンスの作成後、インスタンス データ (フィールドおよびプロパティ) を変更できる型。ほとんどの参照型は、変更可能な型です。

変更不可の型 [immutable type]

インスタンスの作成後、インスタンス データ (フィールドとプロパティ) を変更できない型。ほとんどの値型は、変更不可の型です。

メソッド [method]

クラスまたは構造体の動作を示す名前付きコード ブロック。

メンバ [member]

クラスまたは構造体で宣言されているフィールド、プロパティ、メソッド、またはイベント。

呼び出し履歴 [call stack]

実行時における、プログラムを開始してから現在実行されているステートメントまでの一連のメソッド呼び出しの履歴。

リファクタリング [refactoring]

以前に入力したコードを再利用すること。Visual C# Express コード エディタでは、強調表示したコード ブロックをメソッドに変換する場合など、コードの書式をインテリジェントに再設定できます。詳細については、「リファクタリング」を参照してください。

参照

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

C# 言語仕様

Version 1.2 および 2.0 の C# 言語仕様は、C# の文法と構文に関する信頼性のある情報源です。これには、Visual C# 製品ドキュメントで取り上げられていない、C# 言語に関する詳細な情報が全面的に網羅されています。

1.2 仕様では、Visual C# 2005 より前に言語に追加された機能について説明し、2.0 仕様では、Visual C# 2005 に追加された機能について説明しています。

C# 言語仕様は、Microsoft Word 形式で次の場所から入手できます。

- MSDN オンラインの Visual C# Developer Center
- Visual Studio (Microsoft Visual Studio 2005 インストール ディレクトリの VC#\Specifications\1033\ ディレクトリ)

Microsoft Word がコンピュータにインストールされていない場合、無料の Word Viewer 2003 を使用して Word 形式の仕様を表示、コピー、および印刷できます。

C# 言語仕様は、Addison Wesley 発行の書籍としても入手できます。

参照

概念

C# プログラミング ガイド

その他の技術情報

C# リファレンス

Visual C# のサンプル

ここに記載されているサンプルの抜粋から、サンプル コードにアクセスできます。各抜粋にはサンプルのファイルを開いたりコピーしたりするためのリンクが含まれています。また、.NET Framework SDK には、.NET Framework の機能と Visual C# のコードの例を示す、テクノロジのサンプル、アプリケーションのサンプル、およびクイック スタート チュートリアルが用意されています。

クイック スタート チュートリアルは、.NET Framework テクノロジが開発者に提供する機能を理解するための最も効率の良い方法です。クイック スタートには、一連のサンプルの他、構文、アーキテクチャ、および Visual Studio と .NET Framework の機能に簡単に慣れるためにデザインされたサポート ドキュメントが用意されています。クイック スタート チュートリアルには、.NET Framework テクノロジの最も重要な機能を示す多数のアプリケーションだけでなく、ASP.NET アプケーションと Windows フォーム アプリケーションのサンプルも含まれています。

クイック スタートを使用するには、[開始] ボタンをクリックし、[プログラム] をポイントして、[Microsoft .NET Framework SDK v2.0] をポイントします。次に、[Quickstart Tutorials] をクリックします。"Microsoft .NET Framework SDK クイック スタート チュートリアル" アプリケーションの Web ページが表示されます。このページの指示に従ってクイック スタートを実行すると、サンプル データベースが構築され、インストールが終了します。詳細については、「サンプルとクイック スタート」を参照してください。

メモ：
Visual C# Express では、これらのサンプルに対して個別に Visual Studio ソリューション (.sln) ファイルを開こうとすると、"ソリューション フォルダはこのバージョンの Visual Studio ではサポートされていません。ソリューション フォルダ 'Solution Items' は使用できないものとして表示されます。" というメッセージが表示されます。このフォルダは Visual C# Express では使用できませんが、プロジェクトの構築と実行に影響はありません。

このセクションの内容

入門用のサンプル

匿名デリゲートのサンプル	名前のないデリゲートを使用してアプリケーションの複雑さを低減する方法を示します。
配列のサンプル	配列の使い方を示します。
コレクション クラスのサンプル	**foreach** ステートメントで使用できる非ジェネリック コレクション クラスの作成方法を示します。
ジェネリックのサンプル (C#)	**foreach** ステートメントで使用できるジェネリック コレクション クラスの作成方法を示します。
コマンド ライン パラメータのサンプル	単純なコマンド ライン処理と、配列にインデックスを付ける例を示します。
条件付きメソッドのサンプル	条件付きメソッドの例を示します。条件付きメソッドは、シンボルが定義されているかどうかに応じてメソッドへの呼び出しを挿入または省略できる強力な機構を提供します。
デリゲートのサンプル	デリゲートの宣言、割り当て、および結合の方法を示します。
イベントのサンプル	C# でのイベントの使い方を示します。
明示的なインターフェイス実装のサンプル	インターフェイス メンバの明示的な実装方法を示します。
Hello World サンプル	Hello World アプリケーションを示します。
インデクサのサンプル	配列表記を使用してオブジェクトにアクセスする方法を示します。
インデックス付きプロパティのサンプル	インデックス付きプロパティを使用するクラスの実装方法を示します。インデックス付きプロパティにより、複数の異なる種類の項目を持つ、配列と同様のコレクションを表すクラスを使用できます。
プロパティのサンプル	プロパティの宣言と使用の方法を示します。抽象プロパティの例も示します。
構造体のサンプル	C# での**構造体**の使い方を示します。

演算子のオーバーロードのサンプル	ユーザー定義クラスで演算子をオーバーロードする方法を示します。
ユーザー定義変換のサンプル	ユーザー定義型との間の変換を定義する方法を示します。
バージョン管理のサンプル	**override** キーワードと **new** キーワードを使用した C# でのバージョン管理の例を示します。
yield のサンプル	yield キーワードを使用してコレクション内の項目をフィルタ処理する方法を示します。

中級用および上級用のサンプル

属性のサンプル	カスタム属性クラスを作成し、そのクラスをコードで使用し、リフレクションを通じて照会する方法を示します。
COM 相互運用性 (第 1 部) サンプル	C# を使用して COM オブジェクトと相互運用する方法を示します。
COM 相互運用性 (第 2 部) サンプル	C# サーバーを C++ COM クライアントと共に使用する方法を示します。
ライブラリのサンプル	コンパイラ オプションを使用して複数のソース ファイルから DLL を作成する方法を示します。ライブラリをほかのプログラムで使用する方法も示します。
null 許容のサンプル	null に設定できる値型について説明します。
OLE DB のサンプル	Microsoft Access データベースを C# から使用する方法を示します。データセットを作成し、データベースからデータセットにテーブルを追加する方法を示します。
部分型のサンプル	クラスおよび構造体を複数の C# ソース コード ファイルで定義する方法を示します。
プラットフォーム呼び出しのサンプル	エクスポートされた DLL 関数を C# から呼び出す方法を示します。
セキュリティのサンプル	.NET Framework セキュリティについて説明し、C# でセキュリティのアクセス許可を変更する 2 とおりの方法 (アクセス許可クラスおよびアクセス許可属性) を示します。
スレッド処理のサンプル	スレッドの作成と実行、スレッドの同期化、スレッド間の対話、スレッド プールの使用などのさまざまなスレッド処理の例を示します。
アンセーフ コードのサンプル	ポインタの使い方を示します。
XML ドキュメントのサンプル	XML を使用してコードを文書化する方法を示します。

関連するセクション

サンプルとクイック スタート | Visual C# のチュートリアル

Hello World サンプル

Download sample

このサンプルでは、C# の複数バージョンの Hello World プログラムを示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを変更するか、[次へ] を再度クリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、プロジェクトのソリューションが信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で Hello World のサンプルをビルドして実行するには

1. ソリューション (HelloWorld.sln) を開きます。
2. ソリューション エクスプローラで、HelloWorld1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
3. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
4. 任意のキーを押して HelloWorld1 を閉じます。
5. ソリューション エクスプローラで、HelloWorld2 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
6. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
7. 任意のキーを押して HelloWorld2 を閉じます。
8. ソリューション エクスプローラで、HelloWorld3 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
9. ソリューション エクスプローラで、HelloWorld3 プロジェクトを右クリックし、[プロパティ] をクリックします。
10. [構成プロパティ] フォルダを開き、[デバッグ] をクリックします。
11. [コマンド ライン引数] プロパティに「**A B C D**」と入力し、[OK] をクリックします。
12. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
13. 任意のキーを押して HelloWorld3 を閉じます。
14. ソリューション エクスプローラで、HelloWorld4 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
15. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
16. 任意のキーを押して HelloWorld4 を閉じます。

コマンド ラインから Hello World のサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、HelloWorld ディレクトリに移動します。

2. 次のように入力します。

```
cd HelloWorld1
csc Hello1.cs
Hello1
```

3. 次のように入力します。

```
cd ..\HelloWorld2
csc Hello2.cs
Hello2
```

4. 次のように入力します。

```
cd ..\HelloWorld3
csc Hello3.cs
Hello3 A B C D
```

5. 次のように入力します。

```
cd ..\HelloWorld4
csc Hello4.cs
Hello4
```

参照

概念

Hello World -- 最初のプログラム (C# プログラミング ガイド)

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

コマンド ライン パラメータのサンプル

Download sample

このサンプルでは、コマンド ラインにアクセスする方法、およびコマンド ライン パラメータの配列にアクセスする 2 とおりの方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でコマンド ライン パラメータのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、CmdLine1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. ソリューション エクスプローラで、プロジェクトを右クリックし、[プロパティ] をクリックします。
3. [構成プロパティ] フォルダを開き、[デバッグ] をクリックします。
4. [コマンド ライン引数] プロパティにコマンド ライン パラメータを入力し、[OK] をクリックします。例については、チュートリアルを参照してください。
5. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
6. CmdLine2 について前の手順を繰り返します。

コマンド ラインからコマンド ライン パラメータのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、CmdLine1 ディレクトリに移動します。
2. 次のように入力します。

```
csc cmdline1.cs
cmdline1 A B C
```

3. Change Directory コマンドを使用して、CmdLine2 ディレクトリに移動します。
4. 次のように入力します。

```
csc cmdline2.cs
cmdline2 John Paul Mary
```

参照

処理手順
方法 : コマンド ライン引数を表示する (C# プログラミング ガイド)
方法 : foreach を使用してコマンド ライン引数にアクセスする (C# プログラミング ガイド)
概念
C# プログラミング ガイド
その他の技術情報
Visual C# のサンプル

配列のサンプル

Download sample

このサンプルでは、C# での配列の動作について説明します。詳細については、「配列 (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で配列のサンプルをビルドして実行するには

- [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインから配列のサンプルをビルドして実行するには

- コマンド プロンプトで次のように入力します。

```
csc arrays.cs
arrays
```

参照

関連項目

パラメータとしての配列の受け渡し (C# プログラミング ガイド)
ref と out を使用した配列の引き渡し (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

プロパティのサンプル

Download sample

このサンプルでは、C# プログラミング言語でプロパティがどのように重要な部分であるかを示します。プロパティの宣言と使用の方法について示します。詳細については、「プロパティ (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でプロパティのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、Person プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. shapetest について前の手順を繰り返します。

コマンド ラインからプロパティのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、person ディレクトリに移動します。
2. 次のように入力します。

```
csc person.cs
person
```

3. Change Directory コマンドを使用して、shapetest ディレクトリに移動します。
4. 次のように入力します。

```
csc abstractshape.cs shapes.cs shapetest.cs
shapetest
```

参照

処理手順

方法 : 読み取り/書き込みプロパティを宣言および使用する (C# プログラミング ガイド)
方法 : 抽象プロパティを定義する (C# プログラミング ガイド)

関連項目

プロパティとインデクサの比較 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

ライブラリのサンプル

Download sample

このサンプルでは、C# での DLL の作成と使用の方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でライブラリのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、FunctionTest プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. ソリューション エクスプローラで、FunctionTest プロジェクトを右クリックし、[プロパティ] をクリックします。
3. [構成プロパティ] フォルダを開き、[デバッグ] をクリックします。
4. [コマンド ライン引数] プロパティに、「`3 5 10`」と入力します。
5. [OK] をクリックします。
6. [デバッグ] メニューの [デバッグなしで開始] をクリックします。これにより、Functions にライブラリが自動的に作成され、プログラムが実行されます。

コマンド ラインからライブラリのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Functions ディレクトリに移動します。
2. 次のように入力します。

```
csc /target:library /out:Functions.dll Factorial.cs DigitCounter.cs
```

3. Change Directory コマンドを使用して、FunctionTest ディレクトリに移動します。
4. 次のように入力します。

```
copy ..\Functions\Functions.dll .
csc /out:FunctionTest.exe /R:Functions.DLL FunctionClient.cs
FunctionTest 3 5 10
```

参照

処理手順
方法 : C# DLL を作成して使用する (C# プログラミング ガイド)
概念
C# プログラミング ガイド
その他の技術情報
Visual C# のサンプル

バージョン管理のサンプル

Download sample

このサンプルでは、**override** キーワードと **new** キーワードを使用した、C# でのバージョン管理の例を示します。バージョン管理により、クラスを拡張するときに基本クラスと派生クラスの間の互換性が維持されます。詳細については、
「Override キーワードと New キーワードによるバージョン管理 (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でバージョン管理のサンプルをビルドして実行するには

- [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインからバージョン管理のサンプルをビルドして実行するには

- コマンド プロンプトで次のように入力します。

```
csc versioning.cs
versioning
```

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

コレクション クラスのサンプル

Download sample

このサンプルでは、**foreach** ステートメントで使用できるコレクション クラスの実装方法を示します。詳細については、「コレクション クラス (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ :
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でコレクション クラスのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、CollectionClasses1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. CollectionClasses2 について前の手順を繰り返します。

コマンド ラインからコレクション クラスのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、CollectionClasses1 ディレクトリに移動します。
2. 次のように入力します。

```
csc tokens.cs
tokens
```

3. Change Directory コマンドを使用して、CollectionClasses2 ディレクトリに移動します。
4. 次のように入力します。

```
csc tokens2.cs
tokens2
```

参照

処理手順

方法 : foreach を使用してコレクション クラスにアクセスする (C# プログラミング ガイド)

関連項目

System.Collections.Generic

概念

構造体のサンプル

Download sample

このサンプルでは、構造体の構文と使い方を示します。クラスと構造体の重要な違いについても説明します。詳細については、「オブジェクト、クラス、および構造体 (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で構造体のサンプルをビルドして実行するには

1. ソリューション エクスプローラで、Struct1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. Struct2 について前の手順を繰り返します。

コマンド ラインから構造体のサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Struct1 ディレクトリに移動します。
2. 次のように入力します。

```
csc struct1.cs
struct1
```

3. Change Directory コマンドを使用して、Struct2 ディレクトリに移動します。
4. 次のように入力します。

```
csc struct2.cs
struct2
```

参照

処理手順

方法 : メソッドに構造体を渡すこととクラス参照を渡すことの違いを理解する (C# プログラミング ガイド)

関連項目

struct (C# リファレンス)

概念

C# プログラミング ガイド
その他の技術情報
Visual C# のサンプル

インデクサのサンプル

Download sample

このサンプルでは、C# クラスでインデクサを宣言し、配列と同様にクラスにアクセスする方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でインデクサのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、indexers プロジェクトを右クリックし、[プロパティ] をクリックします。
2. [構成プロパティ] フォルダを開き、[デバッグ] をクリックします。
3. [コマンド ライン引数] プロパティに、「**..\..\Test.txt**」と入力します。
4. [OK] をクリックします。
5. [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインからインデクサのサンプルをビルドして実行するには

1. サンプル プログラムをコンパイルするには、コマンド プロンプトで次のように入力します。

   ```
   csc indexer.cs
   ```

2. サンプル プログラムは、コマンド ライン引数で指定されたファイル内のバイトを反転します。たとえば、Test.txt 内のバイトを反転し、その結果を参照するには、次のコマンドを実行します。

   ```
   indexers Test.txt
   type Test.txt
   ```

3. 反転したファイルを元に戻すには、同じファイルでプログラムを再度実行します。

参照

関連項目

インデクサの使用 (C# プログラミング ガイド)
インターフェイスのインデクサ (C# プログラミング ガイド)
プロパティとインデクサの比較 (C# プログラミング ガイド)

概念

C# プログラミング ガイド
その他の技術情報
Visual C# のサンプル

インデックス付きプロパティのサンプル

Download sample

このサンプルでは、C# クラスでインデックス付きプロパティを宣言し、異なる種類の項目を持つ、配列と同様のコレクションを表す方法を示します。詳細については、「プロパティ (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でインデックス付きプロパティのサンプルをビルドして実行するには

- [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインからインデックス付きプロパティのサンプルをビルドして実行するには

- コマンド プロンプトで次のように入力します。

```
csc indexedproperty.cs
indexedproperty
```

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

ユーザー定義変換のサンプル

Download sample

このサンプルでは、クラスまたは構造体の間での変換を定義する方法と、このような変換の使い方を示します。詳細については、「変換演算子 (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でユーザー定義変換のサンプルをビルドして実行するには

1. ソリューション エクスプローラで、Conversion1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. Conversion2 について前の手順を繰り返します。

コマンド ラインからユーザー定義変換のサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Conversion1 ディレクトリに移動します。
2. 次のように入力します。

```
csc conversion.cs
conversion
```

3. Change Directory コマンドを使用して、Conversion2 ディレクトリに移動します。
4. 次のように入力します。

```
csc structconversion.cs
structconversion
```

参照

処理手順

方法 : 構造体間にユーザー定義の変換を実装する (C# プログラミング ガイド)

関連項目

暗黙的な数値変換の一覧表 (C# リファレンス)

明示的な数値変換の一覧表 (C# リファレンス)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

ジェネリックのサンプル (C#)

Download sample

このサンプルでは、型パラメータを 1 つ使用してカスタムのジェネリック List クラスを作成する方法と、List のコンテンツで **foreach** の繰り返しを有効にする **IEnumerable<T>** を実装する方法を示します。また、クライアントのコードで型引数を指定して、クラスのインスタンスを作成する方法と、型パラメータを制限することによって、型引数に対して実行できる操作を増やす方法についても示します。

反復子ブロックを実装するジェネリック コレクション クラスの例については、
「方法 : ジェネリック リストの反復子ブロックを作成する (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ :
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でジェネリックのサンプルをビルドして実行するには

- [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインからジェネリックのサンプルをビルドして実行するには

- コマンド プロンプトで次のように入力します。

```
csc generics.cs
generics
```

説明

このサンプルは、使い方を示すために用意されたものであり、製品コードに使用する場合は修正が必要です。製品レベルのコードにするには、可能であれば System.Collections.Generic 名前空間でコレクション クラスを使用することを強くお勧めします。

参照

関連項目

System.Collections.Generic

概念

ジェネリック (C# プログラミング ガイド)

その他の技術情報

Visual C# のサンプル

演算子のオーバーロードのサンプル

Download sample

このサンプルでは、ユーザー定義クラスで演算子をオーバーロードする方法を示します。詳細については、「C# の演算子」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で演算子のオーバーロードのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、Complex プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. Dbbool について前の手順を繰り返します。

コマンド ラインから演算子のオーバーロードのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Complex ディレクトリに移動します。
2. 次のように入力します。

```
csc complex.cs
complex
```

3. Change Directory コマンドを使用して、Dbbool ディレクトリに移動します。
4. 次のように入力します。

```
csc dbbool.cs
dbbool
```

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

デリゲートのサンプル

Download sample

このサンプルでは、デリゲートの種類を示します。デリゲートを静的メソッドとインスタンス メソッドに割り当てる方法、およびそれらを結合してマルチキャスト デリゲートを作成する方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でデリゲートのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、Delegates1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. Delegates2 について前の手順を繰り返します。

コマンド ラインからデリゲートのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Delegates1 ディレクトリに移動します。
2. 次のように入力します。

```
csc bookstore.cs
bookstore
```

3. Change Directory コマンドを使用して、Delegates2 ディレクトリに移動します。
4. 次のように入力します。

```
csc compose.cs
compose
```

参照

処理手順

方法 : デリゲートを結合する (マルチキャスト デリゲート) (C# プログラミング ガイド)

関連項目

デリゲートの使用 (C# プログラミング ガイド)

概念

デリゲート (C# プログラミング ガイド)
C# プログラミング ガイド
その他の技術情報
Visual C# のサンプル

イベントのサンプル

Download sample

このサンプルでは、C# でのイベントの宣言、起動、および構成の方法を示します。詳細については、「イベント (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でイベントのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、Events1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. Events2 について前の手順を繰り返します。

コマンド ラインからイベントのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Events1 ディレクトリに移動します。
2. 次のように入力します。

```
csc events1.cs
events1
```

3. Change Directory コマンドを使用して、Events2 ディレクトリに移動します。
4. 次のように入力します。

```
csc events2.cs
events2
```

参照

処理手順

方法 : イベント サブスクリプションとサブスクリプションの解除 (C# プログラミング ガイド)
方法 : .NET Framework ガイドラインに準拠したイベントを発行する (C# プログラミング ガイド)
方法 : 派生クラスから基本クラス イベントを発生させる (C# プログラミング ガイド)
方法 : インターフェイス イベントを実装する (C# プログラミング ガイド)

方法 : ディクショナリを使用してイベント インスタンスを格納する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

明示的なインターフェイス実装のサンプル

Download sample

このサンプルでは、インターフェイス メンバを明示的に実装する方法、およびそれらのメンバにインターフェイス インスタンスからアクセスする方法を示します。詳細については、「インターフェイス (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で明示的なインターフェイス実装のサンプルをビルドして実行するには

1. ソリューション エクスプローラで、ExplicitInterface1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. ExplicitInterface2 について前の手順を繰り返します。

コマンド ラインから明示的なインターフェイス実装のサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、ExplicitInterface1 ディレクトリに移動します。
2. 次のように入力します。

```
csc explicit1.cs
explicit1
```

3. Change Directory コマンドを使用して、ExplicitInterface2 ディレクトリに移動します。
4. 次のように入力します。

```
csc explicit2.cs
explicit2
```

参照

処理手順

方法 : インターフェイス メンバを明示的に実装する (C# プログラミング ガイド)
方法 : 継承を使用してインターフェイス メンバを明示的に実装する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

条件付きメソッドのサンプル

Download sample

このサンプルでは、条件付きメソッドの例を示します。条件付きメソッドは、シンボルが定義されているかどうかに応じてメソッドの呼び出しを挿入または省略できる強力な機構を提供します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で条件付きメソッドのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、プロジェクトを右クリックし、[プロパティ] をクリックします。
2. [構成プロパティ] フォルダを開き、[Debug] をクリックします。
3. [コマンド ライン引数] プロパティを「**A B C**」に設定します。
4. [構成プロパティ] フォルダで、[ビルド] をクリックします。
5. [条件付きコンパイル定数] プロパティを変更します。たとえば、DEBUG を追加または削除します。[OK] をクリックします。
6. [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインから条件付きメソッドのサンプルをビルドして実行するには

- 条件付きメソッドを挿入するには、コマンド プロンプトで次のように入力して、サンプル プログラムをコンパイルおよび実行します。

```
csc CondMethod.cs tracetest.cs /d:DEBUG
tracetest A B C
```

参照

関連項目

Conditional (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

XML ドキュメントのサンプル

Download sample

このサンプルでは、XML を使用してコードをドキュメント化する方法を示します。詳細については、「XML ドキュメント コメント (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で XML ドキュメントのサンプルをビルドするには

1. ソリューション エクスプローラで、プロジェクトを右クリックし、[プロパティ] をクリックします。
2. [構成プロパティ] フォルダを開き、[ビルド] をクリックします。
3. [XML ドキュメント ファイル] プロパティを XMLsample.xml に設定します。
4. [ビルド] メニューの [ビルド] をクリックします。XML 出力ファイルが debug ディレクトリに作成されます。

コマンド ラインから XML ドキュメントのサンプルをビルドするには

1. サンプルの XML ドキュメントを生成するには、コマンド プロンプトで次のように入力します。

```
csc XMLsample.cs /doc:XMLsample.xml
```

2. 生成された XML を参照するには、次のコマンドを実行します。

```
type XMLsample.xml
```

参照

処理手順

方法 : XML ドキュメント機能を使用する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

プラットフォーム呼び出しのサンプル

Download sample

このサンプルでは、プラットフォーム呼び出し (エクスポートされた DLL 関数) を C# から実行する方法を示します。詳細については、「相互運用性 (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でプラットフォーム呼び出しのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、PinvokeTest プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. Marshal および Pinvoke について前の手順を繰り返します。

コマンド ラインからプラットフォーム呼び出しのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、PinvokeTest ディレクトリに移動します。
2. 次のように入力します。

```
csc PinvokeTest.cs
PinvokeTest
```

3. Change Directory コマンドを使用して、Marshal ディレクトリに移動します。
4. 次のように入力します。

```
csc Marshal.cs
Marshal
```

5. Change Directory コマンドを使用して、Pinvoke ディレクトリに移動します。
6. 次のように入力します。

```
csc logfont.cs pinvoke.cs
pinvoke
```

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

COM 相互運用性 (第 1 部) サンプル

Download sample

このサンプルでは、C# プログラムをアンマネージ COM コンポーネントと相互運用する方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出 ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で COM 相互運用性 (第 1 部) サンプルをビルドして実行するには

1. ソリューション エクスプローラで、Interop1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. ソリューション エクスプローラで、Interop1 プロジェクトを右クリックし、[プロパティ] をクリックします。
3. [構成プロパティ] フォルダを開き、[デバッグ] をクリックします。
4. [コマンド ライン引数] プロパティに、「`c:\winnt\clock.avi`」などの AVI ファイルを入力します。
5. [OK] をクリックします。
6. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
7. Interop2 について前の手順を繰り返します。

Interop1 に含まれている QuartzTypeLib.dll が古い場合

1. ソリューション エクスプローラで、Interop1 の [参照設定] を開きます。
2. [QuartzTypeLib] を右クリックし、[削除] をクリックします。
3. [参照設定] を右クリックし、[参照の追加] をクリックします。
4. [COM] タブで、"ActiveMovie control type library" という名前のコンポーネントを選択します。
5. 選択して、[OK] をクリックします。
6. Interop1 を再ビルドします。

メモ：
コンポーネントに参照を追加すると、コマンド ラインで tlbimp を呼び出して QuartzTypeLib.dll を作成する場合と同じ処理が行われます (次の手順を参照)。

コマンド ラインから COM 相互運用性 (第 1 部) サンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Interop1 ディレクトリに移動します。
2. 次のように入力します。

```
tlbimp %windir%\system32\quartz.dll /out:QuartzTypeLib.dll
csc /r:QuartzTypeLib.dll interop1.cs
interop1 %windir%\clock.avi
```

3. Change Directory コマンドを使用して、Interop2 ディレクトリに移動します。
4. 次のように入力します。

```
csc interop2.cs
interop2 %windir%\clock.avi
```

参照

関連項目

相互運用性 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

COM 相互運用性 (第 2 部) サンプル

Download sample

このサンプルでは、C# サーバーを C++ COM クライアントと共に使用する方法を示します。

メモ：
このサンプルをコンパイルするには、Visual C++ をインストールする必要があります。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で COM 相互運用性 (第 2 部) サンプルをビルドして実行するには

1. ソリューション エクスプローラで、COMClient プロジェクトを右クリックし、[プロパティ] をクリックします。
2. [構成プロパティ] フォルダを開き、[デバッグ] をクリックします。
3. [コマンド ライン引数] プロパティに名前を入力します。
4. [OK] をクリックします。
5. [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインから COM 相互運用性 (第 2 部) サンプルをビルドして実行するには

1. Change Directory コマンドを使用して、COMInteropPart2\COMClient ディレクトリに移動します。
2. C# サーバー コードを COMClient ディレクトリにコピーします。

   ```
   copy ..\CSharpServer\CSharpServer.cs
   ```

3. サーバーをコンパイルします。

   ```
   csc /target:library CSharpServer.cs
   regasm CSharpServer.dll /tlb:CSharpServer.tlb
   ```

4. クライアントをコンパイルします。パスと環境変数が vcvars32.bat で正しく設定されていることを確認します。

   ```
   cl COMClient.cpp
   ```

5. クライアントを実行します。

```
COMClient friend
```

参照

関連項目

相互運用性 (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

属性のサンプル

Download sample

このサンプルでは、カスタム属性クラスの作成、そのクラスのコードでの使用、およびリフレクションを通じてそのクラスを照会する方法を示します。属性の詳細については、「属性 (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で属性のサンプルをビルドして実行するには

- [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインから属性のサンプルをビルドして実行するには

- コマンド プロンプトで次のように入力します。

```
csc AttributesTutorial.cs
AttributesTutorial
```

参照

処理手順

方法 : 属性を使用して C/C++ の共用体を作成する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

セキュリティのサンプル

Download sample

このサンプルでは、アクセス許可クラスとアクセス許可属性を通じてセキュリティのアクセス許可を変更する方法を示します。詳細については、「セキュリティ (C# プログラミング ガイド)」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でセキュリティのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、Security1 プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. Security2 および Security3 について前の手順を繰り返します。

コマンド ラインからセキュリティのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Security1 ディレクトリに移動します。
2. 次のように入力します。

```
csc ImperativeSecurity.cs
ImperativeSecurity
```

3. Change Directory コマンドを使用して、Security2 ディレクトリに移動します。
4. 次のように入力します。

```
csc DeclarativeSecurity.cs
DeclarativeSecurity
```

5. Change Directory コマンドを使用して、Security3 ディレクトリに移動します。
6. 次のように入力します。

```
csc SuppressSecurity.cs
SuppressSecurity
```

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

スレッド処理のサンプル

Download sample

このサンプルでは、次のスレッド処理手法を使用します。詳細については、「スレッド処理 (C# プログラミング ガイド)」を参照してください。

- スレッドの作成、起動、終了
- スレッド プールの使用
- スレッドの同期と対話

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でスレッド処理のサンプルをビルドして実行するには

1. ソリューション エクスプローラで、ThreadStartStop プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. ThreadPool および ThreadSync について前の手順を繰り返します。

コマンド ラインからスレッド処理のサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Threads ディレクトリに移動します。
2. 次のように入力します。

```
cd ThreadStartStop
csc ThreadStartStop.cs
ThreadStartStop
```

3. 次のように入力します。

```
cd ..\ThreadPool
csc ThreadPool.cs
ThreadPool
```

4. 次のように入力します。

```
cd ..\ThreadSync
csc ThreadSync.cs
ThreadSync
```

参照

処理手順

方法 : スレッドを作成および終了する (C# プログラミング ガイド)
方法 : スレッド プールを使用する (C# プログラミング ガイド)
方法 : producer スレッドと consumer スレッドを同期する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

アンセーフ コードのサンプル

Download sample

このサンプルでは、C# でのアンマネージ コード (ポインタを使用するコード) の使い方を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio でアンセーフ コードのサンプルをビルドして実行するには

1. ソリューション エクスプローラで、FastCopy プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
3. ソリューション エクスプローラで、ReadFile プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
4. ソリューション エクスプローラで、ReadFile プロジェクトを右クリックし、[プロパティ] をクリックします。
5. [構成プロパティ] フォルダを開き、[デバッグ] をクリックします。
6. [コマンド ライン引数] プロパティに、「`..\..\ReadFile.cs`」と入力します。
7. [OK] をクリックします。
8. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
9. ソリューション エクスプローラで、PrintVersion プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
10. [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインからアンセーフ コードのサンプルをビルドして実行するには

1. Change Directory コマンドを使用して、Unsafe ディレクトリに移動します。
2. 次のように入力します。

```
cd FastCopy
csc FastCopy.cs /unsafe
FastCopy
```

3. 次のように入力します。

```
cd ..\ReadFile
csc ReadFile.cs /unsafe
ReadFile ReadFile.cs
```

4. 次のように入力します。

```
cd ..\PrintVersion
csc PrintVersion.cs /unsafe
PrintVersion
```

参照

処理手順

方法 : ポインタを使用してバイトの配列をコピーする (C# プログラミング ガイド)

方法 : Windows の ReadFile 関数を使用する (C# プログラミング ガイド)

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

OLE DB のサンプル

Download sample

このサンプルでは、Microsoft Access データベースを C# から使用する方法を示します。データセットを作成し、データベースからデータセットにテーブルを追加する方法を示します。サンプル プログラムで使用する BugTypes.mdb データベースは、Microsoft Access 2000.mdb ファイルです。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で OLE DB のサンプルをビルドして実行するには

- [デバッグ] メニューの [デバッグなしで開始] をクリックします。

メモ：
リリース モードでソリューションを構築している場合、BugTypes.mdb を \bin\release フォルダにコピーします。

コマンド ラインから OLE DB のサンプルをビルドして実行するには

- コマンド プロンプトで次のように入力します。

```
csc oledbsample.cs
oledbsample
```

参照

概念

C# プログラミング ガイド

その他の技術情報

Visual C# のサンプル

データへのアクセス (Visual Studio)

yield のサンプル

Download sample

このサンプルでは、**IEnumerable<int>** と **yield** キーワードを実装し、List クラスを作成する方法を示します。これによって List のコンテンツで **foreach** を繰り返すことができるようになります。2 つのプロパティが定義されます。1 つは奇数を返し、もう 1 つは偶数を返します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で yield のコード例をビルドして実行するには

- [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインから yield のコード例をビルドして実行するには

1. **Change Directory (cd)** コマンドを使用して、**Yield** ディレクトリに変更します。
2. 次のように入力します。

```
csc Yield.cs
Yield
```

参照

関連項目

yield (C# リファレンス)

概念

反復子 (C# プログラミング ガイド)

その他の技術情報

Visual C# のサンプル

ジェネリックのサンプル (C#)

匿名デリゲートのサンプル

Download sample

このサンプルでは、匿名デリゲートを使用して、従業員のボーナスを計算する方法を示します。匿名デリゲートを使用すると、独立したメソッドを定義する必要がないため、プログラムが単純になります。

各従業員のデータは、1 つのオブジェクトに保存されています。このオブジェクトには、個人の詳細情報と、ボーナスの計算に必要なアルゴリズムを参照するデリゲートが含まれています。デリゲートを利用したアルゴリズムを定義することで、実際の計算方法とは関係なく、ボーナスの計算に同じメソッドを使用できます。また、ローカル変数の multiplier は、デリゲートの計算メソッドから参照されるため、キャプチャされた外部変数になります。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で AnonymousDelegates コードのサンプルをビルドして実行するには

- [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインから AnonymousDelegates コードのサンプルをビルドして実行するには

1. Change Directory (cd) コマンドを使用して、AnonymousDelegates ディレクトリに移動します。
2. 次のように入力します。

```
csc AnonymousDelegates.cs
AnonymousDelegates
```

参照

関連項目

匿名メソッド (C# プログラミング ガイド)

概念

デリゲート (C# プログラミング ガイド)

その他の技術情報

Visual C# のサンプル

C# リファレンス

部分型のサンプル

Download sample

このサンプルでは、部分型の使い方を示します。部分型により、クラスや構造体を複数の C# ファイルで定義できます。これにより、複数のプログラマが同じクラスの異なる部分に対して並行して作業を行うことができ、複雑なクラスの異なるファセットを別々のファイルに保存できます。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で PartialTypes コード サンプルをビルドして実行するには

- [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインから PartialTypes コード サンプルをビルドして実行するには

1. **Change Directory (cd)** コマンドを使用して、**PartialTypes** ディレクトリに変更します。
2. 次のように入力します。

```
csc PartialTypes.cs
PartialTypes
```

参照

関連項目

partial (C# リファレンス)
部分クラス定義 (C# プログラミング ガイド)

その他の技術情報

Visual C# のサンプル

null 許容のサンプル

Download sample

このサンプルでは、Null 許容型の使用例を示します。この機能を使用すると、参照型を **null** に設定する場合と同様に、値型を初期化前の状態にするか、空にできます。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないため、アプリケーションまたは Web サイトでは使用しないでください。想定した用途以外でサンプル コードを使用した場合でも、Microsoft は付随的損害または間接的損害の責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

Visual Studio で null 許容コードのサンプルをビルドして実行するには

1. Windows エクスプローラでダブルクリックするか、[ファイル] メニューの [開く] をクリックして、ソリューション ファイルの Nullable.sln を開きます。
2. [デバッグ] メニューの [デバッグなしで開始] をクリックします。

コマンド ラインから Null 許容コード例をビルドして実行するには

1. **Change Directory (cd)** コマンドを使用して、Nullable ディレクトリのサブディレクトリである \Boxing、\Basics、または \Operator に変更を加えます。
2. Boxing の場合、次のように入力します。

   ```
   csc Boxing.cs
   Boxing
   ```

   Basics の場合、次のように入力します。

   ```
   csc Basics.cs
   Basics
   ```

   Operator の場合、次のように入力します。

   ```
   csc Operator.cs
   Operator
   ```

参照
概念

null 許容型 (C# プログラミング ガイド)

その他の技術情報

Visual C# のサンプル