ASP.NET 4 と Visual Studio 2010 による Web 開発の概要

.NET Framework 4 では、ASP.NET 向けに興味深い変更が数多く導入されています。このドキュメントでは、リリース予定の .NET Framework 4 および Visual Studio 2010 に含まれるさまざまな新機能の概要について説明します。

目次

コア サービス

ASP.NET 4 では、出力のキャッシュやセッション状態の保存など、ASP.NET のコア サービスを強化するさまざまな機能が導入されています。

Web.config ファイルの縮小

これまで数回の .NET Framework のリリースにわたり AJAX、ルーティング、IIS 7 との統合などの新機能が追加されてきたため、Web アプリケーションの構成を保持する Web.config ファイルの内容が大幅に増加してしまいました。そのため、Visual Studio のようなツールを使用しないと、新しい Web アプリケーションの構成や起動が難しくなってきています。.NET Framework 4 では、構成の主な要素を machine.config ファイルに移動し、アプリケーションはこれらの設定を継承するようになりました。したがって、ASP.NET 4 アプリケーションの Web.config ファイルは、空になるか、アプリケーションが対象とするフレームワークの Visual Studio バージョンを指定する次の行のみで構成されます。

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation targetFramework="4.0" />
  </system.web>
</configuration>
```

拡張可能な出力キャッシュ

ASP.NET 1.0 のリリースから、開発者は出力キャッシュを利用して、生成したページ、コントロール、および HTTP 応答の出力をメモリに保存できるようになりました。キャッシュ後の Web 要求では、出力を再生成しないで、メモリから生成済みの出力を取得することで、コンテンツの提供時間を短縮できます。ただし、この方法には制限事項があります。つまり、生成済みのコンテンツを常にメモリ内に保持しておく必要があるため、トラフィックが多いサーバーでは、出力キャッシュが使用するメモリと、Web アプリケーションの他のコンポーネントに必要なメモリが競合する可能性があります。

ASP.NET 4 では、出力キャッシュに拡張ポイントが追加され、1 つまたは複数のカスタム出力キャッシュ プロバイダーを構成できるようになります。出力キャッシュ プロバイダーで HTML コンテンツを保持する場合、任意の記憶域メカニズムを使用できます。したがって、ローカル ディスクやリモート ディスク、クラウド ストレージ、分散キャッシュ エンジンなど、各種データ保持メカニズム用のカスタム出力キャッシュ プロバイダーを作成できます。

カスタム出力キャッシュ プロバイダーは、新しい System.Web.Caching.OutputCacheProvider 型の派生クラスとして作成します。次に、outputCache 要素の新しいサブセクション providers を使用して、Web.config ファイルでプロバイダーを構成します。

```
<caching>
 <outputCache defaultProvider="AspNetInternalProvider">
  <providers>
   <add name="DiskCache"
     type="Test.OutputCacheEx.DiskOutputCacheProvider, DiskCacheProvider"/>
  </providers>
 </outputCache>
</caching>
```

ASP.NET 4 の既定では、すべての HTTP 応答、表示ページ、およびコントロールは、メモリ内の出力キャッシュを使用します。そのため、上記の例では、defaultProvider 属性が AspNetInternalProvider に設定されています。defaultProvider に別のプロバイダー名を指定すれば、Web アプリケーションに使用する既定の出力キャッシュ プロバイダーを変更できます。

さらに、コントロールや要求ごとに、異なる出力キャッシュ プロバイダーを指定できます。Web ユーザー コントロールごとに異なる出力キャッシュ プロバイダーを指定する最も簡単な方法は、次のように、ページまたはコントロールのディレクティブで新しい providerName 属性を宣言します。

```
<%@ OutputCache Duration="60" VaryByParam="None" providerName="DiskCache" %>
```

HTTP 要求に異なる出力キャッシュ プロバイダーを指定する場合は、さらに少し処理が必要です。特定の要求に使用するプロバイダーを、宣言ではなくプログラムから指定するには、Global.asax ファイル内の新しい GetOuputCacheProviderName GetOuputCacheProviderName メソッドをオーバーライドします。この例を次に示します。

```
public override string GetOutputCacheProviderName(HttpContext context)
{
  if (context.Request.Path.EndsWith("Advanced.aspx"))
    return "DiskCache";
  else
    return base.GetOutputCacheProviderName(context);
}
```

ASP.NET 4 に出力キャッシュ プロバイダーの拡張ポイントが追加されたことで、Web サイトにより大胆でインテリジェントな出力キャッシュ方針を立てられるようになります。たとえば、サイトの "トップ 10" ページはメモリ内にキャッシュし、トラフィックが少ないページはディスクにキャッシュします。また、ある表示ページのバリエーションをすべてキャッシュし、分散キャッシュを使用すれば、フロントエンドの Web サーバーのメモリ使用負荷を軽減できます。

自動開始 Web アプリケーション

Web アプリケーションの中には、最初の要求を処理する前に、大量のデータを読み込んだり、負荷の高い初期化処理を実行したりする必要があるものがあります。以前のバージョンの ASP.NET では、このような場合、ASP.NET アプリケーションを "起動" してから、Global.asax ファイルの Application_Load メソッド内で初期化コードを実行する方法を独自に工夫しなければなりませんでした。

ASP.NET 4 が Windows Server 2008 R2 の IIS 7.5 で実行されるときは、このようなシナリオに直接対応する "自動開始" という新しいスケーラビリティ機能を利用できます。この自動開始機能により、アプリケーション プールの開始、ASP.NET アプリケーションの初期化、および HTTP 要求の受け取りを管理できます。

IIS 7.5 用アプリケーション 自動開始 (Warm-Up) モジュール

IIS チームは、IIS 7.5 用のアプリケーション 自動開始モジュールの最初の Beta バージョンをリリースしました。これにより、いままでよりもさらに簡単にアプリケーションを自動開始することができます。

カスタム コードを記述する代わりに、Web アプリケーションがネットワークからのリクエストを受け入れる前に実行するリソースの Url を指定します。

この自動開始は IIS サービスの開始時　(IIS アプリケーション プールを AlwaysRunning として構成している場合)、IIS ワーカー プロセスがリサイクルさ

れる際に発生します。リサイクルする間、古いワーカー プロセスはアプリケーションが中断しないように、または準備ができていないキャッシュにより他の問題が発生しないよう、新しいワーカープロセスが生成され完全に自動開始されるまでリクエストをし続けます。

なお、このモジュールは、ASP.NET version 2.0 以上で動作します。

詳細については、IIS.net の Web サイト上の Application Warm-Up を参照してください。自動開始機能を使用する方法を示すチュートリアルについては、IIS.net の Web サイト上の Managing and Maintaining IIS 7 を参照してください。

自動開始機能を使用するには、IIS 管理者が、applicationHost.config ファイルで次の構成を使用して、IIS 7.5 のアプリケーション プールが自動的に開始されるように設定します。

```
<applicationPools>
  <add name="MyApplicationPool" startMode="AlwaysRunning" />
</applicationPools>
```

1 つのアプリケーション プールに複数のアプリケーションが含まれる可能性があるため applicationHost.config ファイルでは次の構成を使用して、個々のアプリケーションが自動的に開始されるようにします。

```
<sites>
 <site name="MySite" id="1">
  <application path="/"
    serviceAutoStartEnabled="true"
    serviceAutoStartProvider="PrewarmMyCache" >
    <!-- その他のコンテンツ -->
  </application>
 </site>
</sites>

<!-- その他のコンテンツ -->

<serviceAutoStartProviders>
  <add name="PrewarmMyCache"
    type="MyNamespace.CustomInitialization, MyLibrary" />
</serviceAutoStartProviders>
```

IIS 7.5 サーバーがコールドスタートされるか、個別のアプリケーション プールがリサイクルされるときは、IIS 7.5 が applicationHost.config ファイル内の情報を使用して、自動開始が必要な Web アプリケーションを特定します。自動開始に設定されているアプリケーションごとに、IIS7.5 が要求を ASP.NET 4 に送り、アプリケーションは一時的に HTTP 要求を受け取らない状態で起動されるようにします。アプリケーションがこの状態のときに、ASP.NET は serviceAutoStartProvider 属性で定義される型 (前の例を参照) のインスタンスを作成し、パブリック エントリ ポイントを呼び出します。

必要なエントリ ポイントを指定した自動開始のマネージ型を作成する場合は、次に示すように、IProcessHostPreloadClient インターフェイスを実装します。

```
public class CustomInitialization : System.Web.Hosting.IProcessHostPreloadClient
{
  public void Preload(string[] parameters)
  {
    // 初期化を実行します
  }
}
```

初期化コードから Preload メソッドが実行され、その後メソッドから制御が戻ると、ASP.NET アプリケーションが要求を処理できる状態になります。

IIS 7.5 および ASP.NET 4 に自動開始機能が追加されたことで、最初の HTTP 要求の処理に先立つ、負荷の高い初期化を実行する方法を適切に定義できるようになります。たとえば、新しい自動開始機能を使用してアプリケーションを初期化し、アプリケーションの初期化が完了し、HTTP トラフィックを受け取れる状態になった時点でロード バランサーに通知できます。

ページの完全なリダイレクト

Web アプリケーションでは、時間の経過と共にページや他のコンテンツが移動することがよくあります。そのため、検索エンジンに古くなったリンクが蓄積されることになります。ASP.NET では、これまで、古い URL に対する要求を Response.Redirect メソッドを使用して新しい URL に転送することで対処してきました。しかし、Redirect メソッドは "HTTP 302

Found" (一時的なリダイレクト) 応答を発行するため、ユーザーが古い URL にアクセスすると、余分な HTTP ラウンド トリップが発生します。

ASP.NET 4 からは新たに RedirectPermanent ヘルパー メソッドが追加されます。このメソッドを使用すると、次のように、"HTTP 301 Moved Permanently" 応答の発行が容易になります。

```
RedirectPermanent("/newpath/foroldcontent.aspx");
```

検索エンジンや他のユーザー エージェントが完全なリダイレクトを認識すれば、コンテンツに関連付けられている新しい URL が保存されます。その結果、一時的なリダイレクトのためにブラウザーが生成する不要なラウンド トリップがなくなります。

セッション状態サイズの大幅な縮小

ASP.NET には、Web ファーム全体のセッション状態を保存する既定のオプションが 2 つあります。1 つはアウトプロセスのセッション状態サーバーを呼び出すセッション状態プロバイダー、もう 1 つは Microsoft SQL Server データベースにデータを保存するセッション状態プロバイダーです。どちらも状態情報を Web アプリケーションのワーカー プロセス外に保存するため、リモートの記憶域に送信する前にセッション状態をシリアル化する必要があります。セッション状態に保持する情報量によっては、シリアル化後のデータ サイズがかなり大きくなります。

ASP.NET 4 では、どちらのアウトプロセスのセッション状態プロバイダーにも新たな圧縮オプションが導入されます。次の例のように compressionEnabled 構成オプションを true に設定すると、ASP.NET は .NET Framework の System.IO.Compression.GZipStream クラスを使用して、シリアル化済みのセッション状態を圧縮 (および圧縮解除) します。

```
<sessionState
   mode="SqlServer"
   sqlConnectionString="data source=dbserver;Initial Catalog=aspnetstate"
   allowCustomSqlDatabase="true"
   compressionEnabled="true"
/>
```

Web.config ファイルにこの新しい属性を追加するだけで、Web サーバーでの CPU サイクルに余裕のあるアプリケーションは、シリアル化するセッション状態データのサイズを大幅に削減できます。

使用可能な URL 範囲の拡大

ASP.NET 4 では、アプリケーションの URL を長くすることができる新しいオプションが導入されます。以前のバージョンの ASP.NET では、(NTFS ファイル パスの制限により) URL パスの長さは最大 260 文字に制限されていました。ASP.NET 4 では、2 つの新しい httpRuntime 構成属性を使用することで、この制限をアプリケーションに合わせて増減できます。これらの新しい属性の使用例を次に示します。

```
<httpRuntime maxRequestPathLength="260" maxQueryStringLength="2048" />
```

パス (URL のうち、プロトコル、サーバー名、およびクエリ文字列を含まない部分) のサイズを増減するには、maxRequestPathLength 属性を変更します。クエリ文字列のサイズを増減するには、maxQueryStringLength 属性の値を変更します。

ASP.NET 4 では、URL の文字チェックの対象となる文字を構成することも可能です。ASP.NET は、URL のパス部分で無効な文字を検出すると、要求を拒否して HTTP 400 エラーを発行します。以前のバージョンの ASP.NET では、URL の文字チェックは、固定の文字セットに限定されていました。ASP.NET 4 では、次のように httpRuntime 構成要素の新しい requestPathInvalidChars 属性を使用して、有効な文字セットをカスタマイズできます。

```
<httpRuntime requestPathInvalidChars="&lt;,&gt;,*,%,&amp;,:,\" />
```

既定では、requestPathInvalidChars 属性により 7 種の文字が無効文字に定義されています (Web.config ファイルは XML ファイルのため、requestPathInvalidChars に割り当てられる既定の文字のうち、小なり記号 (<)、大なり記号 (>)、アンパサンド (&) はエンコードします)。この無効文字のセットは、必要に応じてカスタマイズできます。

注 ASP.NET 4 では常に、ASCII の 0x00 ～ 0x1F の範囲の文字を含む URL パスは拒否されます。この範囲内の文字は、IETF の RFC 2396 において無効 URL 文字とされているためです (http://www.ietf.org/rfc/rfc2396.txt、英語)。IIS 6 以上を実行する Windows Server 上では、これらの文字を含む URL は、http.sys プロトコル デバイス ドライバーによって自動的に拒否されます。

拡張可能な要求の検証

ASP.NET の要求の検証では、受信 HTTP 要求データ内を検索し、クロスサイト スクリプト (XSS) 攻撃でよく使用される文字列が含まれていないかどうかを確認します。XSS の可能性がある文字列が見つかると、疑わしい文字列にフラグを付けて、エラーを返します。組み込みの要求の検証では、XSS 攻撃で最もよく使用される文字列が検出された場合にしかエラーが返されません。これまでもより厳しい XSS 検証が試みられましたが、あまりにも多くの誤検知が生まれることになりました。しかし、より厳しい要求の検証が必要な場合や、逆に、特定のページまたは特定の種類の要求については XSS チェックを敢えて緩くしたい場合があります。

ASP.NET 4 では、要求の検証機能を拡張できるため、要求の検証に独自のロジックを使用できます。要求の検証を拡張するには、新しい System.Web.Util.RequestValidator 型から派生したクラスを作成し、Web.config ファイルの httpRuntime セクションで、アプリケーションからこのカスタム型を使用するように構成します。要求の検証のカスタム クラスを構成する例を次に示します。

```
<httpRuntime requestValidationType="Samples.MyValidator, Samples" />
```

新しい requestValidationType 属性には、要求のカスタム検証を提供するクラスを指定する標準の .NET Framework 型識別子文字列が必要です。ASP.NET は、要求ごとにカスタム型を呼び出して、受信 HTTP 要求データの各部分を処理します。次のような要求のカスタム検証クラスを使用すると、受信 URL、すべての HTTP ヘッダー (cookie およびカスタム ヘッダー)、エンティティ ボディのすべてを検査できます。

```
public class CustomRequestValidation : RequestValidator
```

```
{
  protected override bool IsValidRequestString(
      HttpContext context, string value,
      RequestValidationSource requestValidationSource,
         string collectionKey,
      out int validationFailureIndex)
  {...}
}
```

受信 HTTP データの各部分を検査しない場合は、base.IsValidRequestString を呼び出すだけで、この要求の検査クラスではなく、ASP.NET の既定の要求の検証を実行できます。

オブジェクト キャッシュとオブジェクト キャッシュの拡張性

ASP.NET は最初にリリースしたときから、強力なメモリ内オブジェクト キャッシュ (System.Web.Caching.Cache) を実装してきました。このキャッシュの実装は非常に人気が高く、Web アプリケーション以外でも使用されています。しかし、ASP.NET オブジェクト キャッシュを使用することだけが目的で、Windows フォームアプリケーションや WPF アプリケーションに System.Web.dll への参照を組み込むのは面倒です。

すべてのアプリケーションからキャッシュを利用できるようにするため、.NET Framework 4 では、新しいアセンブリ、新しい名前空間、いくつかの基本型、および具象キャッシュ実装を導入しました。新しい System.Runtime.Caching.dll アセンブリの System.Runtime.Caching 名前空間には、新しいキャッシュ API があります。この名前空間には、2 つの主要なクラス セットがあります。

- あらゆる種類のカスタム キャッシュ実装を作成する基盤となる抽象型
- メモリ内オブジェクト キャッシュの具象実装 (System.Runtime.Caching.MemoryCache クラス)

新しい MemoryCache クラスは、ASP.NET キャッシュを忠実にモデル化しており、内部キャッシュ エンジン ロジックの多くを ASP.NET と共有しています。System.Runtime.Caching のパブリック キャッシュ API は、カスタム キャッシュをサポートするように更新されています

が、ASP.NET の Cache オブジェクトを使用したことがあれば、新しい API になじみのある概念が使われていることがわかるでしょう。

新しい MemoryCache クラスと、それをサポートする基本 API の詳細を説明するには、それだけで 1 つのドキュメントが必要です。しかし、次の例を見れば、新しいキャッシュ API がどのように機能するかがわかります。この例は、Windows フォーム アプリケーション用に作成したもので、System.Web.dll には依存しません。

```
private void btnGet_Click(object sender, EventArgs e)
{
    //既定の MemoryCache インスタンスへの参照を取得します。
    //単一のアプリケーション内に複数の MemoryCache を
    //作成できることに注意してください。
    ObjectCache cache = MemoryCache.Default;

    //この例では、キャッシュにファイルの内容を保存します。
    string fileContents = cache["filecontents"] as string;

    //ファイルの内容が現在キャッシュ内になければ、
    //ディスクからその内容が読み取られ、キャッシュに格納されます。
    if (fileContents == null)
    {
        //CacheItemPolicy オブジェクトには、1 つのキャッシュ エントリ
        //に関するキャッシュの依存関係およびキャッシュの有効期限の
        //メタデータがすべて保持されます。
        CacheItemPolicy policy = new CacheItemPolicy();

        //ファイルの依存関係の作成に必要な情報を集めます。
        //この場合は、ディスク上のファイルのファイル パスのみが必要です。
        List<string> filePaths = new List<string>();
        filePaths.Add("c:\\data.txt");

        //新しいキャッシュ API では、依存関係を "変更モニター" と呼びます。
        //この例では、ディスク上の内容が変更されたら、キャッシュ エントリが
        //自動的に失効するようにします。この機能は、HostFileChangeMonitor
        //により提供されます。
        policy.ChangeMonitors.Add(new HostFileChangeMonitor(filePaths));

        //ファイルの内容をフェッチします。
        fileContents = File.ReadAllText("c:\\data.txt");

        //ファイルの内容をキャッシュに格納します。
        cache.Set("filecontents", fileContents, policy);
```

```
    }

    MessageBox.Show(fileContents);
}
```

拡張可能な HTML、URL、および HTTP ヘッダー エンコード

ASP.NET 4 では、HTML エンコード、URL エンコード、HTML 属性のエンコード、および送信 HTTP ヘッダーのエンコードという一般的なテキスト エンコード作業用に、カスタム エンコード ルーチンを作成できます。カスタム エンコーダーを作成するには、次のように、新しい System.Web.Util.HttpEncoder 型から派生し、Web.config ファイルの httpRuntime セクションを指定して ASP.NET からこの型が使用されるように構成します。

```
<httpRuntime encoderType="Samples.MyCustomEncoder, Samples"  />
```

カスタム エンコーダーを構成すると、System.Web.HttpUtility クラスまたは System.Web.HttpServerUtility クラスのパブリック エンコード メソッドが呼び出されるたびに、ASP.NET が自動的にこのカスタム エンコード実装を呼び出します。このため、Web 開発チームのある部門では積極的な文字エンコードを実装するカスタム エンコーダーを作成し、他の部門では引き続きパブリック ASP.NET エンコード API を使用するような構成が可能です。httpRuntime 要素でカスタム エンコーダーを一元的に構成することで、パブリック ASP.NET エンコード API からのテキスト エンコード呼び出しが、確実にすべてカスタム エンコーダーを経由するようにもできます。

同一のワーカー プロセスで実行される個別のアプリケーションのパフォーマンスの監視

1 台のサーバーでホストできる Web サイト数を増やすため、多くのホストは複数の ASP.NET アプリケーションを同一のワーカー プロセスで実行します。しかし、複数のアプリケーションが 1 つの共有ワーカー プロセスを使用すると、問題が発生しているアプリケーションを個別に特定することは困難です。

ASP.NET 4 では、CLR によって導入される新しいリソース監視機能を利用します。この機能を有効にするには、次のような XML 構成を aspnet.config 構成ファイルに追加します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
 <runtime>
  <appDomainResourceMonitoring enabled="true"/>
 </runtime>
</configuration>
```

> **注** aspnet.config ファイルは、.NET Framework のインストール先ディレクトリにあります。Web.config ファイルとは異なります。

appDomainResourceMonitoring 機能を有効にすると、% Managed Processor Time と Managed Memory Used という 2 つのパフォーマンス カウンターが "ASP.NET Applications" パフォーマンス カテゴリに追加されます。どちらのパフォーマンス カウンターも新しい CLR アプリケーション ドメイン リソース管理機能を使用して、個別の ASP.NET アプリケーションの推定 CPU 時間とマネージ メモリの使用率を追跡します。その結果、ASP.NET 4 では、同一のワーカー プロセス内で実行される個別のアプリケーションのリソース使用率を詳細に把握できるようになります。

複数バージョンへの対応

.NET Framework の特定のバージョンに対応するアプリケーションを作成できます。ASP.NET 4 では、Web.config ファイルの compilation 要素の新しい属性を使用して、.NET Framework 4 以降を対象バージョンに指定できます。.NET Framework 4 を明示的に対象バージョンに指定し、system.codedom のエントリなど Web.config ファイルの省略可能な要素を指定する場合、それらの要素は .NET Framework 4 に適したものにする必要があります (.NET Framework 4 を明示的に指定しなければ、Web.config ファイル内の未指定のエントリから対象フレームワークを推論します)。

次に、Web.config ファイルの compilation 要素の targetFramework 属性の使用例を示します。

```
<compilation targetFramework="4.0"/>
```

.NET Framework の特定のバージョンを対象に指定する場合は、次の点に注意してください。

- .NET Framework 4 のアプリケーション プールでは、Web.config ファイルで targetFramework 属性が指定されていないか、Web.config ファイルが見つからない場合、ASP.NET ビルド システムは .NET Framework 4 が対応バージョンであると見なします (.NET Framework 4 で実行できるように、アプリケーション コードの変更が必要になることがあります)。
- targetFramework 属性を指定しないで、system.codeDom 要素を Web.config ファイルで定義する場合、この構成ファイルには .NET Framework 4 に適したエントリを含める必要があります。
- aspnet_compiler コマンドを使用してアプリケーション (ビルド環境など) をプリコンパイルする場合、対象のフレームワークに適したバージョンの aspnet_compiler コマンドを使用する必要があります。.NET Framework 3.5 以前のバージョンを対象にコンパイルする場合は、.NET Framework 2.0 (%WINDIR%\Microsoft.NET\Framework\v2.0.50727) に付属のコンパイラーを使用してください。.NET Framework 4 以降のバージョンを対象に作成されたアプリケーションをコンパイルする場合は、.NET Framework 4 に付属のコンパイラーを使用してください。
- 実行時に、コンパイラーはコンピューター (したがって GAC) にインストールされている最新のフレームワーク アセンブリを使用します。後日フレームワークが更新された場合 (たとえば、バージョン 4.1 がインストールされた場合)、targetFramework 属性で以前のバージョン (4.0 など) が対応バージョンに指定されていても、新しいバージョンのフレームワークの機能を使用できます (ただし、Visual Studio 2010 でのデザイン時、または aspnet_compiler コマンドを使用するときは、フレームワークの新しい機能を使用するとコンパイラー エラーが発生します)。

Ajax

Web From と MVC に含まれる jQuery

Web フォームと MVC の両方の Visual Studio テンプレートには、オープン ソースの jQuery のライブラリが含まれます。新しい web サイトまたはプロジェクトを作成すると、次の 3 つのファイルを含む Script フォルダーが作成されます。

- jQuery-1.4.1.js – 人間が判読可能な、jQuery ライブラリの軽量化されていない (unminified) バージョン
- jQuery-14.1.min.js – jQuery ライブラリの軽量化 (minified) バージョン
- jQuery-1.4.1-vsdoc.js – jQuery ライブラリのためのドキュメンテーション ファイル

アプリケーションの開発中は、jQuery の軽量化されていないバージョンを含んでください。プロダクション用のアプリケーションでは、jQuery の軽量化されたバージョンを含んでください。

例えば、以下の Web フォーム ページは、ASP.NET TextBox コントロールにフォーカスがあるときに、それらの背景色を黄色に変えるのにどう jQuery を使用できるかを示してしいます。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ShowjQuery.aspx.cs"
Inherits="ShowjQuery" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Show jQuery</title>
</head>
```

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="txtFirstName" runat="server" />
        <br />
        <asp:TextBox ID="txtLastName" runat="server" />
    </div>
    </form>
    <script src="Scripts/jquery-1.4.1.js" type="text/javascript"></script>
    <script type="text/javascript">

        $("input").focus( function() { $(this).css("background-color", "yellow");
        });

    </script>
</body>
</html>
```

コンテンツ配信ネットワーク のサポート

Microsoft Ajax コンテンツ配信ネットワーク (CDN) は、ASP.NET Ajax と jQuery スクリプトを Web アプリケーションに追加するのを簡単にします。例えば、以下のようにページに Ajax.microsoft.com を示す <script> タグを追加するだけで jQuery ライブラリを使用し始めることができます。

<script src="http://ajax.microsoft.com/ajax/jquery/jquery-1.4.2.js" type="text/javascript"></script>

Microsoft Ajax CDN を利用することで、Ajax アプリケーションのパフォーマンスを大幅に向上することができます。

Microsoft Ajax CDN の内容は、世界中にあるサーバーにキャッシュされています。さらに、Microsoft Ajax CDN はブラウザー キャッシュされた JavaScript ファイルを、別のドメインにある Web サイトのために再利用することができます。

セキュリティで保護されたソケット レイヤーを使用して Web ページを提供する場合に Microsoft Ajax コンテンツ配信ネットワークは SSL (HTTPS) をサポートします。

Microsoft Ajax CDN の詳細は、次の web サイトを参照してください。

http://www.asp.net/ajaxlibrary/CDN.ashx

ASP.NET の ScriptManager コントロールは、 Microsoft Ajax CDN をサポートします。EnableCdn プロパティを指定するだけで、ASP.NET Framework のすべての JavaScript ファイルを CDN から取得するようにすることができます。

```
<asp:ScriptManager ID="sm1" EnableCdn="true" runat="server" />
```

EnableCdn プロパティに true がセットされると、ASP.NET Framework は、 CDN が含んでいる JavaScript ファイルから、validation や UpdatePanel が使用するすべての ASP.NET Framework JavaScript を取得します。このプロパティの設定は、Web アプリケーションの性能にダイナミックな影響力を持ちます。

WebResource プロパティを使用すると、独自の JavaScript ファイルを CDN のパスに設定することができます。

新しい CdnPath プロパティの CDN へのパスは、EnableCdn プロパティが true のときに使用されます。

```
[assembly: WebResource("Foo.js", "application/x-javascript",
CdnPath = "http://foo.com/foo/bar/foo.js")]
```

ScriptManager の分割スクリプト指定

以前の ASP.NET の ScriptManger では、ASP.NET Ajax Library は一枚岩的な構造となっているため、使用する際には全体をロードする必要がありました。

新しい ScriptManager.AjaxFrameworkMode プロパティを利用すると、ASP.NET Ajax ライブラリのどのコンポーネントがロードされるかを制御して、必要とする ASP.NET Ajax ライブラリのコンポーネントだけをロードすることができます。

ScriptManager.AjaxFrameworkMode プロパティには、次の値が設定できます。

- Enabled ― すべての Microsoft AJAX スクリプトが組み込まれます (従来の動作)。これが既定値です。
- Explicit ― 分割スクリプト ファイルをそれぞれ明示的に追加する必要があります。相互に依存関係があるすべてのスクリプトを開発者が責任をもって組み込む必要があります。
- Disabled ― すべての Microsoft ASP.NET AJAX スクリプト機能が無効になり、ScriptManager コントロールはどのスクリプトも自動的に参照しません。

たとえば、AjaxFrameworkMode を Explicit に設定し、部分的なレンダリングなしで

ASP.NET AJAX のブラウザーの履歴機能を使用するには、次の例のように ScriptManager

コントロールを構成する必要があります。

```
<asp:ScriptManager ID="sm1" AjaxFrameworkMode="Explicit" runat="server">
<Scripts>
    <asp:ScriptReference Name="MicrosoftAjaxCore.js" />
    <asp:ScriptReference Name="MicrosoftAjaxComponentModel.js" />

    <asp:ScriptReference Name="MicrosoftAjaxSerialization.js" />
    <asp:ScriptReference Name="MicrosoftAjaxNetwork.js" />
</Scripts>
</asp:ScriptManager>
```

Web フォーム

Web フォームは、ASP.NET 1.0 のリリース以来、ASP.NET のコア機能の 1 つです。ASP.NET 4 では、以下を含め、Web フォームにさまざまな機能強化が施されています。

- meta タグを設定する機能
- ビュー ステートの制御機能の強化
- ブラウザー機能の簡単な操作
- Web フォームでの ASP.NET ルーティングの使用のサポート
- 生成済み ID の制御機能の強化
- 選択した行をデータ コントロールに保持する機能
- FormView コントロールと ListView コントロールでの HTML レンダリング制御機能の強化
- データ ソース コントロールのフィルター処理のサポート

Page.MetaKeywords プロパティと Page.MetaDescription プロパティによる Meta タグの設定

ASP.NET 4 Web フォームのちょっとした新機能 1 つは、Page クラスに MetaKeywords と MetaDescription という 2 つのプロパティが追加されたことです。この 2 つのプロパティは、次のように、ページ内の対応する meta タグを表します。

```
<head id="Head1" runat="server">
 <title>Untitled Page</title>
 <meta name="keywords" content="These, are, my, keywords" />
 <meta name="description" content="This is the description of my page" />
</head>
```

この 2 つのプロパティは、ページの Title プロパティと同じように機能します。

MetaKeywords プロパティと MetaDescription プロパティは、次の規則に従います。

1. これらのプロパティ名に一致する meta タグが head 要素内にない場合 (つまり、Page.MetaKeywords の場合は name="keywords"、Page.MetaDescription の場合は name="description" がなく、これらのプロパティが設定されていない場合)、レンダリング時に meta タグがページに追加されます。

2. これらのプロパティ名が meta タグに既に設定されている場合は、これらのプロパティは既存のタグのコンテンツの get および set メソッドとして動作します。

MetaKeywords と MetaDescription は実行時に設定できるため、データベースやその他のソースからコンテンツを取得したり、動的にタグを設定してある特定のページの目的を説明したりできます。

また、Keywords プロパティと Description プロパティを Web フォーム ページのマークアップの先頭にある @ Page ディレクティブに設定することもできます。

```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeFile="Default.aspx.cs"
  Inherits="_Default"
  Keywords="These, are, my, keywords"
  Description="This is a description" %>
```

これにより、ページ内で既に宣言されている meta タグのコンテンツが (あれば)、オーバーライドされます。

description meta タグのコンテンツは、Google での検索リスティングのプレビューを改善するために使用されます (詳細については、Google Webmaster Central ブログの記事「Improve snippets with a meta description makeover (メタ説明の変更によるスニペットの改良)」(英語) を参照してください)。Google および Windows Live Search はいかなる目的でも keywords のコンテンツを使用しませんが、他の検索エンジンでは使用される可能性があります。詳細については、Search Engine Guide Web サイトの「Meta Keywords Advice (メタ キーワードについてのアドバイス)」(英語) を参照してください。

これらの新しいプロパティは簡単な機能ですが、手動で追加したり、meta タグを作成するカスタム コードを記述したりする手間を省くことができます。

個別のコントロールのビュー ステートの有効化

既定では、ページのビュー ステートは有効なので、アプリケーションで必要としなくても、ページ上の各コントロールがビュー ステートを格納する可能性があります。ビュー ステート データはページの HTML に含まれるため、クライアントへのページ送信とポストバックに必要な時間が増大します。格納されるビュー ステートが必要以上に増えると、パフォーマンスが大幅に低下する原因になります。以前のバージョンの ASP.NET では、コントロールごとにビュー ステートを無効にしてページサイズを削減できましたが、各コントロールでそれぞれ明示的に無効にする必要がありました。ASP.NET 4 では、Web サーバー コントロールに ViewStateMode プロパティが追加されました。このプロパティを使用して、既定ではビュー ステートを無効にしておき、ページ内のビュー ステートが必要なコントロールでのみ有効にすることができます。

ViewStateMode プロパティは、Enabled、Disabled、および Inherit という 3 つの値から成る列挙値を使用します。Enabled では、対象のコントロールと、Inherit が設定されているか何も設定されていないその子コントロールのビュー ステートが有効になります。Disabled ではビュー ステートが無効になり、Inherit では親コントロールの ViewStateMode 設定を使用します。

ViewStateMode プロパティがどのように機能するかを次の例で示します。ページのコントロールのマークアップには、ViewStateMode 値が含まれています。

```
<form id="form1" runat="server">
 <script runat="server">
    protected override void OnLoad(EventArgs e) {
       if (!IsPostBack) {
          label1.Text = label2.Text = "[DynamicValue]";
       }
       base.OnLoad(e);
    }
 </script>
 <asp:PlaceHolder ID="PlaceHolder1" runat="server" ViewStateMode="Disabled">
    Disabled: <asp:Label ID="label1" runat="server" Text="[DeclaredValue]" /><br />
  <asp:PlaceHolder ID="PlaceHolder2" runat="server" ViewStateMode="Enabled">
       Enabled: <asp:Label ID="label2" runat="server" Text="[DeclaredValue]" />
```

```
  </asp:PlaceHolder>
 </asp:PlaceHolder>
 <hr />
 <asp:button ID="Button1" runat="server" Text="Postback" />
 <%-- 他のマークアップはここから記載します --%>
```

ご覧のとおり、このコードでは、PlaceHolder1 コントロールのビュー ステートを無効にしています。子コントロールの label1 は、このプロパティ値を継承 (Inherit はコントロールの ViewStateMode の既定値) するため、ビュー ステートを格納しません。PlaceHolder2 コントロールでは、ViewStateMode が Enabled に設定されているため、label2 はこのプロパティを継承し、ビュー ステートを格納します。ページが初めて読み込まれるときに、両方の Label コントロールの Text プロパティは文字列 "[DynamicValue]" に設定されます。

これらの設定の結果として、ページが最初に読み込まれるときに、次の出力がブラウザーに表示されます。

```
Disabled: [DynamicValue]
Enabled:  [DynamicValue]
```

ただし、ポストバック後は、次の出力が表示されます。

```
Disabled: [DeclaredValue]
Enabled: [DynamicValue]
```

おそらく予想されているとおり、label1 (ViewStateMode 値が Disabled に設定されている) は、コード内でこれに設定されている値を保存していません。ただし、label2 (ViewStateMode 値が Enabled に設定されている) は、そのビュー ステートを保存しています。

また、ViewStateMode は次のように @ Page ディレクティブに設定することもできます。

```
<%@ Page Language="C#" AutoEventWireup="true"
   CodeBehind="Default.aspx.cs"
   Inherits="WebApplication1._Default"
   ViewStateMode="Disabled" %>
```

Page クラスも 1 つのコントロールに過ぎません。このコントロールは、ページ内の他のすべてのコントロールの親コントロールの役割を果たします。Page オブジェクトの

ViewStateMode の既定値は Enabled です。コントロールの既定値は Inherit なので、ページまたはコントロールのレベルで ViewStateMode を設定しない限り、Enabled が継承されます。

EnableViewState プロパティが true に設定されている場合に限り、ViewStateMode の値によってビュー ステートが維持されるかどうかが決まります。EnableViewState プロパティが false に設定されていると、ViewStateMode が Enabled に設定されていても、ビュー ステートは維持されません。

この機能は、マスター ページの ContentPlaceHolder コントロールで使用すると便利です。このコントロールでマスター ページについては ViewStateMode を Disabled に設定したうえで、ビュー ステートが必要なコントロールを保持する ContentPlaceHolder コントロールについて個別に ViewStateMode を 有効にします。

ブラウザー機能に対する変更

ASP.NET では "ブラウザー機能" により、ユーザーがサイトの参照に使用しているブラウザーの機能を特定できます。ブラウザー機能は、Request.Browser プロパティによって公開される HttpBrowserCapabilities オブジェクトで表されます。たとえば、HttpBrowserCapabilities オブジェクトを使用して、現在の種類とバージョンのブラウザーで、特定バージョンの JavaScript をサポートしているかどうかを判断できます。また、HttpBrowserCapabilities オブジェクトを使用して、要求がモバイル デバイスから送られたものかどうかも判断できます。

HttpBrowserCapabilities オブジェクトは、一連のブラウザー定義ファイルによって処理されます。これらのファイルには、特定のブラウザーの機能についての情報が保持されています。ASP.NET 4 では、ブラウザー定義ファイルが更新されて、Google Chrome、Research in Motion の BlackBerry スマートフォン、アップルの iPhone など、最近登場したブラウザーやデバイスの情報が追加されました。

新しいブラウザー定義ファイルは、次のとおりです。

- blackberry.browser
- chrome.browser
- Default.browser

- firefox.browser
- gateway.browser
- generic.browser
- ie.browser
- iemobile.browser
- iphone.browser
- opera.browser
- safari.browser

ブラウザー機能プロバイダーの使用

ASP.NET 3.5 Service Pack 1 では、次の方法でブラウザーの機能を定義できました。

- コンピューター レベルでは、次のフォルダーの .browser XML ファイルを作成または更新する

  \Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\Browsers

  ブラウザー機能を定義したら、Visual Studio コマンド プロンプトから次のコマンドを実行して、ブラウザー機能アセンブリをリビルドし、GAC に追加します。

  ```
  aspnet_regbrowsers.exe –I c
  ```

- 個別のアプリケーションでは、アプリケーションの App_Browsers フォルダーに .browser ファイルを作成する

上記の方法では XML ファイルを変更する必要があり、コンピューター レベルの変更の場合は、aspnet_regbrowsers.exe プロセスを実行した後にアプリケーションを再起動する必要があります。

ASP.NET 4 では、"ブラウザー機能プロバイダー" という機能が追加されました。名前が示すように、この機能ではプロバイダーを作成して、そのプロバイダーにより、独自のコードを使用してブラウザーの機能を特定することができます。

実際には、多くの場合、開発者は独自のブラウザー機能を定義しません。ブラウザーのファイルを更新するのは難しく、その更新プロセスはかなり複雑です。また、.browser ファイルの XML 構文は複雑で、使用も定義も難しい場合があります。このプロセスを大幅に簡素化

するには、共通のブラウザー定義構文、または最新のブラウザー定義を保持するデータベース、さらにはそのようなデータベース Web サービスを用意できるかどうかにかかっています。新機能のブラウザー機能プロバイダーを使用すると、このようなシナリオを実現できるため、サードパーティの開発者にとって便利です。

新機能の ASP.NET 4 ブラウザー機能プロバイダーを使用する主な方法としては、ASP.NET ブラウザー機能定義の機能を拡張する方法と、完全にこれを置き換える方法の 2 とおりがあります。これ以降では、まず、機能を置き換える方法を説明してから、これを拡張する方法について説明します。

ASP.NET ブラウザー機能の置き換え

ASP.NET ブラウザー機能定義を完全に置き換えるには、次の手順を実行します。

1. HttpCapabilitiesProvider から派生し、GetBrowserCapabilities メソッドをオーバーライドするプロバイダー クラスを作成します。

```
public class CustomProvider : HttpCapabilitiesProvider
{
  public override HttpBrowserCapabilities
       GetBrowserCapabilities(HttpRequest request)
  {
    HttpBrowserCapabilities browserCaps = new HttpBrowserCapabilities();
    Hashtable values = new Hashtable(180, StringComparer.OrdinalIgnoreCase);
    values[String.Empty] = request.UserAgent;
    values["browser"] = "MyCustomBrowser";
    browserCaps.Capabilities = values;
    return browserCaps;
  }
}
```

   この例のコードでは、新しい HttpBrowserCapabilities オブジェクトを作成し、browser という機能のみを指定して、その機能を MyCustomBrowser に設定しています。

2. プロバイダーをアプリケーションに登録します。

アプリケーションでプロバイダーを使用するには、provider 属性を Web.config ファイルか Machine.config ファイルの browserCaps セクションに追加する必要があります (provider 属性は、特定のモバイル デバイス用のフォルダーなど、アプリケーションの特定のディレクトリの location 要素に定義することもできます)。構成ファイルに provider プロパティを設定する例を次に示します。

```
<system.web>
<browserCaps provider="ClassLibrary2.CustomProvider, ClassLibrary2,
  Version=1.0.0.0, Culture=neutral" />
</system.web>
```

新しいブラウザー機能定義は、コードを使用しても登録できます。

```
void Application_Start(object sender, EventArgs e)
{
  HttpCapabilitiesBase.BrowserCapabilitiesProvider =
    new ClassLibrary2.CustomProvider();
  // ...
}
```

このコードは、Global.asax ファイルの Application_Start イベント内で実行される必要があります。BrowserCapabilitiesProvider クラスに対するすべての変更は、解決された HttpCapabilitiesBase オブジェクトのキャッシュの状態が有効に保たれるように、アプリケーションのコードが実行される前に行われる必要があります。

HttpBrowserCapabilities オブジェクトのキャッシュ

上記の例では問題が 1 つあります。HttpBrowserCapabilities オブジェクトを取得するために、カスタム プロバイダーが呼び出されるたびにコードが実行されることです。これは、要求ごとに何度も繰り返される可能性があります。この例では、プロバイダーのコードは長くありません。しかし、カスタム プロバイダーのコードが、HttpBrowserCapabilities オブジェクトを取得するためにかなりの処理を実行する場合は、大きなオーバーヘッドになります。これを防ぐため、次のように HttpBrowserCapabilities オブジェクトをキャッシュできます。

1. 次の例のように、HttpCapabilitiesProvider から派生するクラスを作成します。

```
public class CustomProvider : HttpCapabilitiesProvider
```

```
{
  public override HttpBrowserCapabilities
    GetBrowserCapabilities(HttpRequest request)
  {
    string cacheKey = BuildCacheKey();
    int cacheTime = GetCacheTime();
    HttpBrowserCapabilities browserCaps =
      HttpContext.Current.Cache[cacheKey] as
      HttpBrowserCapabilities;
    if (browserCaps == null)
    {
      HttpBrowserCapabilities browserCaps = new
        HttpBrowserCapabilities();
      Hashtable values = new Hashtable(180,
        StringComparer.OrdinalIgnoreCase);
      values[String.Empty] = request.UserAgent;
      values["browser"] = "MyCustomBrowser";
      browserCaps.Capabilities = values;
      HttpContext.Current.Cache.Insert(cacheKey,
        browserCaps, null, DateTime.MaxValue,
        TimeSpan.FromSeconds(cacheTime));
    }
    return browserCaps;
  }
}
```

この例では、コードはカスタムの BuildCacheKey メソッドを呼び出してキャッシュ キーを生成し、カスタムの GetCacheTime メソッドを呼び出してキャッシュにかかる時間を取得しています。次に、解決済みの HttpBrowserCapabilities オブジェクトをキャッシュに追加します。カスタム プロバイダーを必要とするこれ以降の要求では、このオブジェクトをキャッシュから取得して再利用できます。

2. 前述の手順に従って、プロバイダーをアプリケーションに登録します。

ASP.NET ブラウザー機能の拡張

これまでに、ASP.NET 4 で新しい HttpBrowserCapabilities オブジェクトを作成する方法を説明しました。ASP.NET のブラウザー機能は、ASP.NET で既に用意されているブラウザー機能定義に新しい定義を追加することで、機能を拡張することもできます。その場合、XML ブラウザー定義を使用する必要はありません。その方法を次の手順で示します。

1. 次のように、HttpCapabilitiesEvaluator から派生し、GetBrowserCapabilities メソッドをオーバーライドするクラスを作成します。

```
public class CustomProvider : HttpCapabilitiesEvaluator
{
  public override HttpBrowserCapabilities
    GetBrowserCapabilities(HttpRequest request)
  {
    HttpBrowserCapabilities browserCaps =
      base.GetHttpBrowserCapabilities(request);
    if (browserCaps.Browser == "Unknown")
    {
      browserCaps = MyBrowserCapabilitiesEvaulator(request);
    }
    return browserCaps;
  }
}
```

このコードでは、まず ASP.NET ブラウザー機能を使用して、ブラウザーを識別しています。ただし、要求で定義されている情報を基にブラウザーを識別できなければ (つまり、HttpBrowserCapabilities オブジェクトの Browser プロパティが "Unknown" という文字列であれば)、カスタム プロバイダー (MyBrowserCapabilitiesEvaluator) を呼び出してブラウザーを識別します。

2. 前述の例に従って、プロバイダーをアプリケーションに登録します。

既存の機能定義への新しい機能の追加によるブラウザー機能の拡張

カスタムのブラウザー定義プロバイダーを作成して、動的に新しいブラウザー定義を作成するだけでなく、既存のブラウザー定義に機能を追加して拡張することもできます。このため、目的の定義に近く、必要な機能が少し不足している定義を利用できます。それには、次の手順を実行します。

1. 次のように、HttpCapabilitiesEvaluator から派生し、GetBrowserCapabilities メソッドをオーバーライドするクラスを作成します。

```
public class CustomProvider : HttpCapabilitiesEvaluator
{
```

```
  public override HttpBrowserCapabilities
    GetBrowserCapabilities(HttpRequest request)
  {
    HttpBrowserCapabilities browserCaps =
      base.GetHttpBrowserCapabilities(request);
    browserCaps.Frames = true;
    browserCaps.Capabilities["MultiTouch"] = "true";
    return browserCaps;
  }
}
```

この例のコードでは、次のコードを使用することで、既存の ASP.NET HttpCapabilitiesEvaluator クラスを拡張し、現在の要求の定義に一致した HttpBrowserCapabilities オブジェクトを取得しています。

```
HttpBrowserCapabilities browserCaps =
  base.GetHttpBrowserCapabilities(request);
```

それが完了すると、このブラウザーの機能を追加または変更できます。新しいブラウザー機能を指定するには、次の 2 つの方法があります。

- キーと値の組を、HttpCapabilitiesBase オブジェクトの Capabilities プロパティによって公開される IDictionary オブジェクトに追加します。上記の例のコードでは、MultiTouch という機能を、値を true に設定して追加しています。
- HttpCapabilitiesBase オブジェクトの既存のプロパティを設定します。上記の例のコードでは、Frames プロパティを true に設定しています。このプロパティは、単に、Capabilities プロパティによって公開される IDictionary オブジェクトのアクセサーです。

  注 このモデルは、コントロール アダプターも含め、HttpBrowserCapabilities のどのプロパティにも当てはまります。

2. 前述の手順に従って、プロバイダーをアプリケーションに登録します。

ASP.NET 4 でのルーティング

ASP.NET 4 では、Web フォームでルーティングを使用するためのサポートが組み込まれました。ルーティングを使用すると、物理ファイルにマップされない要求 URL をアプリケーションが受け取るように構成できます。ルーティングを使用することで、ユーザーにとってわかりやすく、アプリケーションでの検索エンジンの最適化 (SEO) を図るうえで有効な URL を定義できます。たとえば、既存のアプリケーションで、製品カテゴリを表示するページの URL が次のように設定されているとします。

http://website/products.aspx?categoryid=12

ルーティングを使用することで、次の URL を受け取って、同じ情報をレンダリングするようにアプリケーションを構成できます。

http://website/products/software

ルーティングは、ASP.NET 3.5 SP1 から利用できるようになりました (ASP.NET 3.5 SP1 でのルーティングの使用例については、Phil Haack のブログ記事「Using Routing With WebForms (Web フォームでのルーティングの使用)」(英語) を参照してください)。ただし、ASP.NET 4 には、次のような、ルーティングを使いやすくするための機能がいくつか追加されています。

- PageRouteHandler クラス。ルートを定義するときに使用する簡単な HTTP ハンドラーです。このクラスは、要求のルーティング先のページにデータを渡します。
- 新しいプロパティ HttpRequest.RequestContext および Page.RouteData (HttpRequest.RequestContext.RouteData オブジェクトのプロキシ)。これらのプロパティにより、ルートから渡される情報に簡単にアクセスできます。
- System.Web.Compilation.RouteUrlExpressionBuilder および System.Web.Compilation.RouteValueExpressionBuilder に定義される次の新しい式ビルダー。
    - RouteUrl。ASP.NET サーバー コントロール内のルーティング URL 形式に対応する URL を簡単に作成できます。
    - RouteValue。RouteContext オブジェクトから情報を簡単に抽出できます。

- RouteParameter クラス。RouteContext オブジェクトに保持されているデータを、データ ソース コントロールのクエリに渡しやすくします (FormParameter に似ています)。

Web フォーム ページのルーティング

Route クラスの新しい MapPageRoute メソッドを使用して、Web フォームのルーティングを定義する例を次に示します。

```
public class Global : System.Web.HttpApplication
{
  void Application_Start(object sender, EventArgs e)
  {
    RouteTable.Routes.MapPageRoute("SearchRoute",
      "search/{searchterm}", "~/search.aspx");
    RouteTable.Routes.MapPageRoute("UserRoute",
      "users/{username}", "~/users.aspx");
  }
}
```

ASP.NET 4 では、MapPageRoute メソッドが導入されました。次の例は、前の例の SearchRoute 定義と内容は同じですが、PageRouteHandler クラスを使用しています。

```
RouteTable.Routes.Add("SearchRoute", new Route("search/{searchterm}",
  new PageRouteHandler("~/search.aspx")));
```

この例のコードでは、ルーティング先が物理ページにマップされています (最初のルーティング先は ~/search.aspx)。最初のルーティング定義では、searchterm というパラメーターを URL から抽出して、ページに渡すようにも指定されています。

MapPageRoute メソッドでは、次のメソッド オーバーロードをサポートします。

- *MapPageRoute(string routeName, string routeUrl, string physicalFile, bool checkPhysicalUrlAccess)*
- *MapPageRoute(string routeName, string routeUrl, string physicalFile, bool checkPhysicalUrlAccess, RouteValueDictionary defaults)*
- *MapPageRoute(string routeName, string routeUrl, string physicalFile, bool checkPhysicalUrlAccess, RouteValueDictionary defaults, RouteValueDictionary constraints)*

checkPhysicalUrlAccess パラメーターは、ルーティング先の物理ページ (ここでは search.aspx) のセキュリティ アクセス許可と、受信 URL (ここでは search/{searchterm}) のアクセス許可を確認

するかどうかを指定します。checkPhysicalUrlAccess の値が false の場合、受信 URL のアクセス許可のみが確認されます。これらのアクセス許可は、次のような設定を使用して、Web.config ファイルで定義されます。

```
<configuration>
 <location path="search.aspx">
  <system.web>
   <authorization>
    <allow roles="admin"/>
    <deny users="*"/>
   </authorization>
  </system.web>
 </location>
 <location path="search">
  <system.web>
   <authorization>
    <allow users="*"/>
   </authorization>
  </system.web>
 </location>
</configuration>
```

この構成例では、admin の役割に含まれているユーザー以外は、すべてのユーザーによる物理ページ search.aspx へのアクセスが拒否されます。checkPhysicalUrlAccess パラメーターを true (既定値) に設定すると、物理ページ search.aspx には admin の役割のユーザーしかアクセスできないため、URL /search/{searchterm} へのアクセスが admin ユーザーにしか許可されません。checkPhysicalUrlAccess が false に設定され、サイトが上記の例のように構成されている場合、すべての認証済みユーザーに URL /search/{searchterm} へのアクセスが許可されます。

Web フォーム ページのルーティング情報の読み取り

Web フォームの物理ページのコードでは、2 つの新しいプロパティ HttpRequest.RequestContext と Page.RouteData を使用することで、ルーティングによって URL から抽出された情報 (または他のオブジェクトが RouteData オブジェクトに追加したその他の情報) にアクセスできます (Page.RouteData は HttpRequest.RequestContext.RouteData をラップします)。Page.RouteData の使用例を次に示します。

```
protected void Page_Load(object sender, EventArgs e)
{
   string searchterm = Page.RouteData.Values["searchterm"] as string;
   label1.Text = searchterm;
}
```

このコードでは、前述のルーティングのサンプルで定義されている、searchterm パラメーターに渡された値を抽出します。次の要求 URL について考えてみましょう。

```
http://localhost/search/scott/
```

この要求が行われると、"scott" という語が search.aspx ページにレンダリングされます。

マークアップ内のルーティング情報へのアクセス

Web フォーム ページのコード内にあるルーティング データを取得する方法は既に説明しましたが、同じ情報にアクセスする式をマークアップ内で使用することもできます。(Phil Haack のブログ記事「Express Yourself With Custom Expression Builders」(カスタム式ビルダーで自身を表現する) を参照)。式ビルダーは、宣言コードを扱うための強力で洗練された手段であるため、あまり知られていないのが残念です。

ASP.NET 4 では、Web フォーム ルーティング用に 2 つの新しい式ビルダーが追加されています。これらの使用例を次に示します。

```
<asp:HyperLink ID="HyperLink1" runat="server"
   NavigateUrl="<%$RouteUrl:SearchTerm=scott%>">Search for Scott</asp:HyperLink>
```

この例では、RouteUrl 式を使用して、ルーティング パラメーターに基づく URL を定義しています。このため、完全な URL をマークアップにハード コーディングする必要がなく、また、このリンクを変更しなくても、後で URL 構造を変更できます。

このマークアップでは、以前定義したルーティングを基に、次の URL が生成されます。

```
http://localhost/search/scott
```

ASP.NET によって、入力パラメーターを基に自動的に正しいルーティングが組み立てられます (つまり、正しい URL が生成されます)。また、式にルーティング名を含めることもできるため、使用するルーティングを指定できます。

RouteValue 式の使用例を次に示します。

```
<asp:Label ID="Label1" runat="server" Text="<%$RouteValue:SearchTerm%>" />
```

このコントロールを含むページが実行されると、値 "scott" がラベルに表示されます。

RouteValue 式を使用することで、非常に簡単にマークアップ内にルーティング データを使用できます。マークアップで複雑な Page.RouteData["x"] 構文を処理する必要がありません。

データ ソース コントロール パラメーターでのルーティング データの使用

RouteParameter クラスを使用すると、データ ソース コントロール内のクエリのパラメーター値として、ルーティング データを指定できます。RouteParameter の機能は、FormParameter と非常に似ています。この RouteParameter の例を次に示します。

```
<asp:sqldatasource id="SqlDataSource1" runat="server"
   connectionstring="<%$ ConnectionStrings:MyNorthwind %>"
   selectcommand="SELECT CompanyName,ShipperID FROM Shippers where
      CompanyName=@companyname"
 <selectparameters>
   <asp:routeparameter name="companyname" RouteKey="searchterm" />
 </selectparameters>
</asp:sqldatasource>
```

この例では、ルーティング パラメーターの値 searchterm が、Select ステートメントの @companyname パラメーターに使用されます。

クライアント ID の設定

新しい ClientIDMode プロパティは、コントロールがレンダリング時に要素の id 属性をどのように作成するかという、.ASP.NET の長年の問題に対処しています。レンダリングされた要素の id 属性を把握することは、アプリケーションにこれらの要素を参照するクライアント スクリプトが含まれている場合、重要です。

Web サーバー コントロールにレンダリングされる HTML の id 属性は、そのコントロールの ClientID プロパティを基に生成されます。ClientID プロパティから id 属性を生成するためのこれまでのアルゴリズムでは、名前付けコンテナーがある場合はそれと ID を連結し、繰り

返し使用されるコントロール (データ コントロールなど) の場合は、プレフィックスと連番を追加する方法がとられてきました。このアルゴリズムでは、ページ内のコントロールの ID は必ず一意になりますが、予想できないコントロール ID が生成されることになるため、クライアント スクリプトから参照することが困難でした。

新しい ClientIDMode プロパティを使用すると、コントロールのクライアント ID の生成方法をより厳密に指定できます。ClientIDMode プロパティは、ページのコントロールも含め、どのコントロールにも設定できます。使用できる設定は、次のとおりです。

- AutoID – ASP.NET の以前のバージョンで使用されていた ClientID プロパティ値を生成するアルゴリズムと同じです。
- Static – ClientID 値が親の名前付けコンテナーの ID と連結されていない ID と同じであることを示します。Web ユーザー コントロールで使用すると便利です。Web ユーザー コントロールは、別のページの別のコンテナー コントロールに配置されている可能性があるため、AutoID アルゴリズムを使用するコントロールについては、どのような ID 値になるか予測できないことから、クライアント スクリプトの作成が難しくなります。
- Predictable – 主に、繰り返しテンプレートを使用するデータ コントロールで使用されます。コントロールの名前付けコンテナーの ID プロパティを連結しますが、生成された ClientID 値には、"ctlxxx" のような文字列は含まれません。この設定は、コントロールの ClientIDRowSuffix プロパティと連携して機能します。ClientIDRowSuffix プロパティをデータ フィールドの名前に設定すると、そのフィールドの値が、生成される ClientID 値のサフィックスとして使用されます。通常、データ レコードの主キーを ClientIDRowSuffix 値に使用します。
- Inherit – コントロールの既定の動作です。コントロールの ID 生成方法は、親と同じになることを示します。

ClientIDMode プロパティは、ページ レベルで設定できます。この設定により、現在のページ内のすべてのコントロールに対して、既定の ClientIDMode 値を定義できます。

ページ レベルの ClientIDMode の既定値は AutoID で、コントロール レベルの ClientIDMode の既定値は Inherit です。このため、このプロパティをコード内のどこにも設定しなかった場合、すべてのコントロールに対して、AutoID アルゴリズムが既定で使用されます。

ページ レベルの値は @ Page ディレクティブに設定します。

```
<%@ Page Language="C#" AutoEventWireup="true"
 CodeFile="Default.aspx.cs"
 Inherits="_Default"
 ClientIDMode="Predictable" %>
```

また、ClientIDMode 値は、コンピューター レベルまたはアプリケーション レベルで、構成ファイルでも設定できます。この設定により、アプリケーション内のすべてのページに対して、既定の ClientIDMode 設定を定義できます。この値をコンピューター レベルで設定すると、そのコンピューター上のすべての Web サイトに対する既定の ClientIDMode 設定が定義されます。構成ファイルでの ClientIDMode 設定の例を次に示します。

```
<system.web>
  <pages clientIDMode="Predictable"></pages>
</system.web>
```

前述のとおり、ClientID プロパティの値は、コントロールの親の名前付けコンテナーから導き出されます。マスター ページを使用する場合など、状況によっては、次のレンダリング済み HTML の ID のような ID がコントロールに対して生成される場合があります。

```
<div id="ctl00_ContentPlaceHolder1_ParentPanel">
 <div id="ctl00_ContentPlaceHolder1_ParentPanel_NamingPanel1">
  <input name="ctl00$ContentPlaceHolder1$ParentPanel$NamingPanel1$TextBox1"
    type="text" value="Hello!"
    id="ctl00_ContentPlaceHolder1_ParentPanel_NamingPanel1_TextBox1" />
</div>
```

このマークアップ内の input 要素 (TextBox コントロール) の深さは、ページに含まれる 2 つの名前付けコンテナー (入れ子になっている ContentPlaceholder コントロール) 分でしかありませんが、マスター ページの処理方法のために、最終的なコントロール ID は次のようになります。

```
ctl00_ContentPlaceHolder1_ParentPanel_NamingPanel1_TextBox1
```

これは、長い ID です。ページ内では一意になりますが、ほとんどの用途では不要な長さです。レンダリング後の ID の長さを短縮し、ID が生成される方法を調整 (たとえば "ctlxxx" プレフィックスを省略する) とします。この最も簡単な方法は、ClientIDMode プロパティを設定することです。

```
<tc:NamingPanel runat="server" ID="ParentPanel" ClientIDMode="Static">
 <tc:NamingPanel runat="server" ID="NamingPanel1" ClientIDMode="Predictable">
   <asp:TextBox ID="TextBox1" runat="server" Text="Hello!"></asp:TextBox>
 </tc:NamingPanel>
</tc:NamingPanel>
```

この例では、ClientIDMode プロパティが、一番外側の NamingPanel 要素では Static に、内側の NamingControl 要素では Predictable に設定されています。これらの設定の結果、次のようなマークアップが生成されます (ページの他の要素とマスター ページは、前の例と同じだとします)。

```
<div id="ParentPanel">
 <div id="ParentPanel_NamingPanel1">
  <input name="ctl00$ContentPlaceHolder1$ParentPanel$NamingPanel1$TextBox1"
    type="text" value="Hello!" id="ParentPanel_NamingPanel1_TextBox1" />
</div>
```

Static 設定によって、一番外側の NamingPanel 要素内のコントロールの名前付け階層がリセットされ、生成された ID から ContentPlaceHolder と MasterPage ID が省略されています (レンダリング済み要素の name 属性は変わらず、イベントやビュー ステートなどの通常の ASP.NET 機能は維持されていることに注意してください)。また、名前付け階層がリセットされたことで、NamingPanel 要素のマークアップを別の ContentPlaceholder コントロールに移動しても、レンダリング済みクライアント ID は同じになる点も便利です。

> **注** 開発者が責任を持って、レンダリング済みのコントロール ID が一意になるようにする必要があります。一意でないと、クライアントの document.getElementById 関数など、個々の HTML 要素に一意の ID が必要な機能が正常に動作できなくなる可能性があります。

データ バインドされたコントロールでの予測可能なクライアント ID の作成

データ バインドされたリスト コントロール内のコントロールに対して、レガシー アルゴリズムによって生成された ClientID 値は、長くなり、予測可能であるとは言えません。ClientIDMode 機能を使用することで、これらの ID の生成方法をより詳細に制御できます。

ListView コントロールが含まれる次のマークアップを参照してください。

```
<asp:ListView ID="ListView1" runat="server" DataSourceID="SqlDataSource1"
    OnSelectedIndexChanged="ListView1_SelectedIndexChanged"
    ClientIDMode="Predictable"
    RowClientIDRowSuffix="ProductID">
</asp:ListView>
```

この例では、ClientIDMode プロパティと RowClientIDRowSuffix プロパティが、マークアップ内に設定されています。ClientIDRowSuffix プロパティは、データ バインドされたコントロールでのみ使用でき、その動作は、使用しているコントロールによって異なります。その違いを次に説明します。

- GridView コントロール – データ ソース内の 1 つ以上の列の名前を指定できます。この列名が実行時に連結されて、クライアント ID が作成されます。たとえば、RowClientIDRowSuffix を "ProductName, ProductId" に設定すると、レンダリング済み要素のコントロール ID は、次のような形式になります。

  rootPanel_GridView1_ProductNameLabel_Chai_1

- ListView コントロール – クライアント ID に追加されるデータ ソース内の 1 列を指定できます。たとえば、ClientIDRowSuffix を "ProductName" に設定すると、レンダリング済みのコントロール ID は、次のような形式になります。

  rootPanel_ListView1_ProductNameLabel_1

  この場合、末尾の 1 は、現在のデータ項目の製品 ID から取得されます。

- Repeater コントロール– このコントロールは ClientIDRowSuffix プロパティをサポートしません。Repeater コントロールでは、現在行のインデックスが使用されます。

ClientIDMode="Predictable" を Repeater コントロールに使用すると、次のような形式のクライアント ID が生成されます。

Repeater1_ProductNameLabel_0

末尾の 0 は、現在行のインデックスです。

FormView コントロールと DetailsView コントロールは複数の行を表示しないため、ClientIDRowSuffix プロパティをサポートしません。

データ コントロールでの行選択の保持

GridView コントロールと ListView コントロールでは、ユーザーが行を選択できます。以前のバージョンの ASP.NET では、ページ上の行インデックスを基に選択が行われていました。たとえば、ページ 1 の 3 番目の項目を選択してページ 2 に移動した場合、ページ 2 でも 3 番目の項目が選択されます。

選択内容の保持は、もともと .NET Framework 3.5 SP1 の Dynamic Data プロジェクトでのみサポートされていた新機能です。この機能を有効にすると、現在の選択項目は、その項目のデータ キーを基に選択されます。つまり、ページ 1 で 3 行目を選択してページ 2 に移動しても、ページ 2 ではどの項目も選択されません。ページ 1 に戻ると、3 行目が選択されたままになっています。これは、以前のバージョンの ASP.NET での動作よりも、はるかに自然な選択方法です。選択内容の保持機能は、EnablePersistedSelection プロパティを使用することで、すべてのプロジェクトの GridView コントロールと ListView コントロールでサポートされるようになりました。

```
<asp:GridView id="GridView2" runat="server" EnablePersistedSelection="true">
</asp:GridView>
```

ASP.NET Chart コントロール

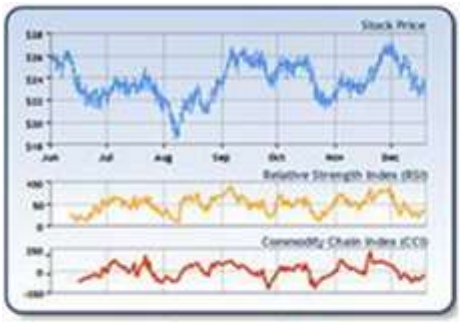
ASP.NET Chart コントロールは、.NET Framework のデータ表示機能を拡張します。Chart コントロールを使用することで、複雑な統計分析や金融分析についての直感的で視覚に訴えるグ

ラフを含む ASP.NET ページを簡単に作成できます。ASP.NET Chart コントロールは、.NET Framework 3.5 SP1 リリースのアドオンとして導入されたもので、.NET Framework 4 に組み込まれました。

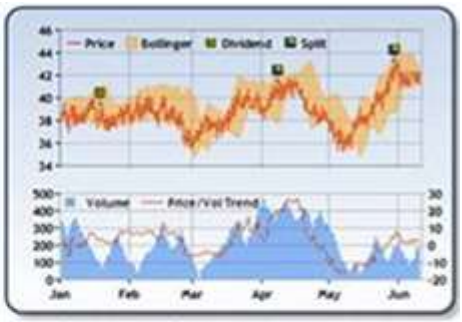
このコントロールの主な機能は、次のとおりです。

- 35 種類のグラフ
- グラフ エリア数、タイトル数、凡例数、注釈数の制限なし
- すべてのグラフ要素に使用できるさまざまな表示設定
- ほとんどのグラフでの 3-D サポート
- データ ポイントに合わせて自動的に配置を調整できるスマート データ ラベル
- 背景の縞模様、目盛省略線、対数目盛
- データ分析とデータ変換のための 50 を超える金融数式および統計数式
- グラフ データの簡単なバインドと操作
- 日付、時刻、通貨など、共通のデータ形式のサポート
- AJAX を使用したクライアント クリック イベントなど、双方向機能およびイベント駆動型のカスタマイズのサポート
- 状態管理
- バイナリ ストリーミング

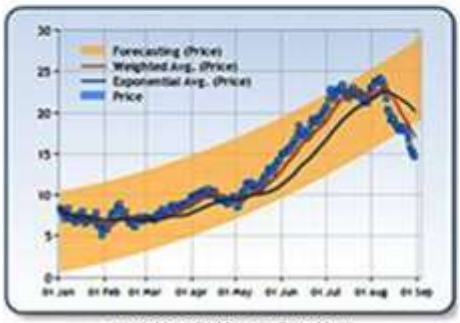
次の図は、ASP.NET Chart コントロールによって生成された金融グラフの例です。



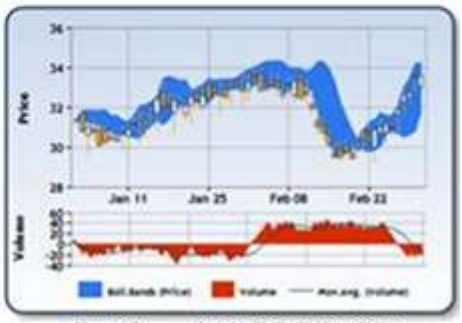
複数の価格指標



ボリンジャー バンドと出来高指標の使用



平均値の計算と予測の使用



ボリンジャー バンドと移動平均の使用

その他の ASP.NET Chart コントロールの使用例については、MSDN Web サイトの「Samples Environment for Microsoft Chart Controls (Microsoft Chart コントロールのサンプル環境)」(英語) からサンプル コードをダウンロードしてください。Chart コントロールのフォーラム (英語) では、コミュニティ コンテンツとして他にもサンプルが提供されています。

ASP.NET ページへの Chart コントロールの追加

マークアップを使用して、ASP.NET ページに Chart コントロールを追加する例を次に示します。この例では、Chart コントロールにより、静的データ ポイントの棒グラフが生成されます。

```
<asp:Chart ID="Chart1" runat="server">
 <Series>
  <asp:Series Name="Series1" ChartType="Column">
   <Points>
    <asp:DataPoint AxisLabel="Product A" YValues="345"/>
    <asp:DataPoint AxisLabel="Product B" YValues="456"/>
    <asp:DataPoint AxisLabel="Product C" YValues="125"/>
    <asp:DataPoint AxisLabel="Product D" YValues="957"/>
   </Points>
  </asp:Series>
 </Series>
 <ChartAreas>
```

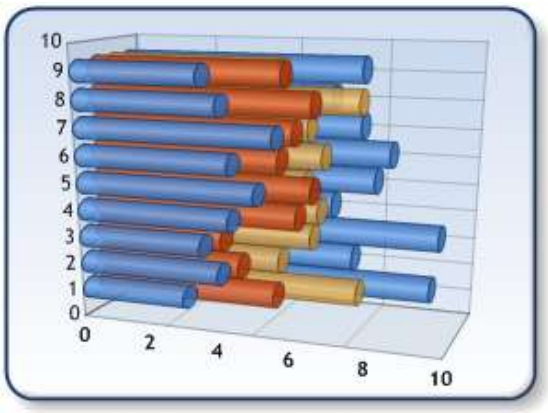
```
  <asp:ChartArea Name="ChartArea1">
    <AxisY IsLogarithmic="True" />
  </asp:ChartArea>
 </ChartAreas>
 <Legends>
  <asp:Legend Name="Legend1" Title="Product Sales" />
 </Legends>
</asp:Chart>
```

3-D グラフの使用

Chart コントロールには ChartAreas コレクションが含まれています。このコレクションには、グラフ エリアの特徴を定義する ChartArea オブジェクトを含めることができます。たとえば、グラフ エリアに 3-D を使用するには、Area3DStyle プロパティを使用します。

```
<asp:ChartArea Name="ChartArea1">
 <area3dstyle
    Rotation="10"
    Perspective="10"
    Enable3D="True"
    Inclination="15"
    IsRightAngleAxes="False"
    WallWidth="0"
    IsClustered="False" />
<%-- 他のマークアップはここから記載します --%>
</asp:ChartArea>
```

以下の図は、4 つのデータ系列がある、グラフの種類が Bar の 3-D グラフです。

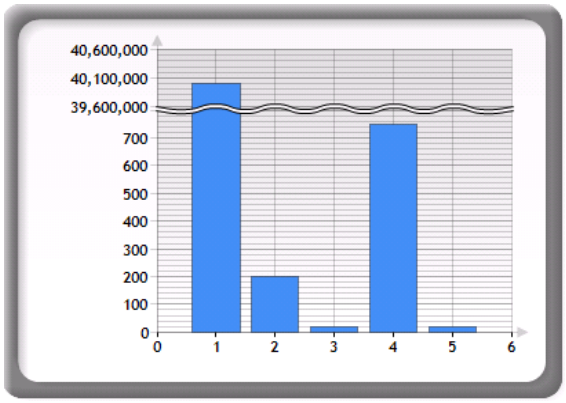


目盛省略線と対数目盛の使用

目盛省略線と対数目盛は、グラフをさらに洗練できる 2 種類の追加機能です。これらの機能は、グラフ エリアの各軸に固有の機能です。たとえば、グラフ エリアの主 Y 軸にこれらの機能を使用するには、AxisY.IsLogarithmic プロパティと ScaleBreakStyle プロパティを ChartArea オブジェクトに使用します。次のコードでは、主 Y 軸に目盛省略線を使用する例を示します。

```
<asp:ChartArea Name="ChartArea1">

 <axisy>
  <ScaleBreakStyle
     BreakLineStyle="Wave"
     CollapsibleSpaceThreshold="40"
     Enabled="True" />
 </axisy>

<%-- 他のマークアップはここから記載します --%>
</asp:ChartArea>
```

次の図は、Y 軸の目盛省略線が有効なグラフです。



QueryExtender コントロールを使用したデータのフィルター処理

データ駆動型 Web ページを作成する開発者が非常によく行う作業の 1 つはデータのフィルター処理です。これは、従来、データ ソース コントロールに Where 句を作成することで実行されてきました。この方法は複雑になりがちで、場合によっては Where 構文を使用するために、基盤のデータベースの完全な機能を利用できません。

ASP.NET 4 では、フィルター処理を容易にするため、QueryExtender コントロールが追加されました。このコントロールは、EntityDataSource コントロールまたは LinqDataSource コントロールに追加して、これらのコントロールから返されるデータのフィルター処理に使用できます。QueryExtender コントロールは LINQ に依存しているため、データがページに送信される前に、データベース サーバー上でフィルターが適用されます。その結果、操作効率が非常に高くなります。

QueryExtender コントロールは、さまざまなフィルター オプションをサポートします。これらのオプションについて説明し、その使用例を紹介します。

検索

検索オプションの場合、QueryExtender コントロールは、指定されたテキストを使用して、指定されたフィールドでレコードを検索します。次の例では、QueryExtender コントロールは、TextBoxSearch コントロールに入力されたテキストを使用して、LinqDataSource コントロールから返されたデータの ProductName 列と Supplier.CompanyName 列でコンテンツを検索します。

```
<asp:LinqDataSource ID="dataSource" runat="server"> TableName="Products">
</asp:LinqDataSource>
<asp:QueryExtender TargetControlID="dataSource" runat="server">
 <asp:SearchExpression DataFields="ProductName, Supplier.CompanyName"
     SearchType="StartsWith">
  <asp:ControlParameter ControlID="TextBoxSearch" />
 </asp:SearchExpression>
</asp:QueryExtender>
```

範囲

範囲オプションは検索オプションと似ていますが、範囲を定義する 1 組の値を指定します。次の例では、QueryExtender コントロールは、LinqDataSource コントロールから返されたデータの UnitPrice 列を検索します。範囲は、ページ上の TextBoxFrom コントロールと TextBoxTo コントロールから読み取られます。

```
<asp:LinqDataSource ID="dataSource" runat="server"> TableName="Products">
</asp:LinqDataSource>
<asp:QueryExtender TargetControlID="dataSource" runat="server">
```

```
 <asp:RangeExpression DataField="UnitPrice" MinType="Inclusive"
     MaxType="Inclusive">
   <asp:ControlParameter ControlID="TextBoxFrom" />
   <asp:ControlParameter ControlID="TexBoxTo" />
 </asp:RangeExpression>
</asp:QueryExtender>
```

プロパティ式

プロパティ式オプションを使用すると、任意のプロパティ値に対する比較を定義できます。式の評価が true になると、確認されたデータが返されます。次の例では、QueryExtender コントロールは、Discontinued 列のデータとページ上の CheckBoxDiscontinued コントロールの値とを比較して、データをフィルター処理しています。

```
<asp:LinqDataSource ID="dataSource" runat="server" TableName="Products">
</asp:LinqDataSource>
<asp:QueryExtender TargetControlID="dataSource" runat="server">
  <asp:PropertyExpression>
    <asp:ControlParameter ControlID="CheckBoxDiscontinued" Name="Discontinued" />
  </asp:PropertyExpression>
</asp:QueryExtender>
```

カスタム式

QueryExtender コントロールで使用するカスタム式を指定できます。このオプションを使用すると、カスタム フィルター ロジックを定義する関数をページで呼び出すことができます。QueryExtender コントロールのカスタム式を宣言により指定する例を次に示します。

```
<asp:LinqDataSource ID="dataSource" runat="server" TableName="Products">
</asp:LinqDataSource>
<asp:QueryExtender TargetControlID="dataSource" runat="server">
  <asp:CustomExpression OnQuerying="FilterProducts" />
</asp:QueryExtender>
```

QueryExtender コントロールによって呼び出されるカスタム関数の例を次に示します。この場合、Where 句を含むデータベース クエリを使用する代わりに、コードで LINQ クエリを使用してデータをフィルター処理しています。

```
protected void FilterProducts(object sender, CustomExpressionEventArgs e)
{
```

```
    e.Query = from p in e.Query.Cast<Product>()
              where p.UnitPrice >= 10
              select p;
}
```

上記の例では、QueryExtender コントロールで一度に使用される式は 1 つしかありません。ただし、QueryExtender コントロール内には、複数の式を含めることができます。

HTML エンコードされたコード式

ASP.NET サイト (特に ASP.NET MVC を使用しているサイト) の中には、<%= expression %> (通称 "コード ナゲット") を多用して、応答に対するテキストを記述しているものがあります。コード ナゲットを使用するときに、テキストの HTML エンコード処理を忘れがちです。テキストがユーザー入力であれば、ページに XSS (クロス サイト スクリプティング) 攻撃の脆弱性が生じることになります。

ASP.NET 4 では、コード式に次の新しい構文が導入されました。

```
<%: expression %>
```

この構文では、応答への書き込み時に、既定で HTML エンコードが使用されます。この新しい式は、実質的には次のように変換されます。

```
<%= HttpUtility.HtmlEncode(expression) %>
```

たとえば、<%: Request["UserInput"] %> では、Request["UserInput"] の値が HTML エンコード処理されます。

この機能の目的は、古い構文のすべてのインスタンスを新しい構文に置き換えて、どれを使用すべきか逐一判断しなくても済むようにすることです。ただし、出力されるテキストが HTML になるべきものであったり、既にエンコードされている場合があります。後者の場合は、二重にエンコード処理が行われる可能性があります。

そのような場合を考慮し、ASP.NET 4 では新しいインターフェイス IHtmlString と、具象実装の HtmlString を併せて導入しています。これらのインスタンスを使用すると、戻り値が既に、HTML で表示するために適切にエンコードされているため (エンコードされていない場合は

検証されます)、再び HTML エンコード処理をしないように通知できます。たとえば、次の場合は HTML エンコード処理されてはいけません (実際、処理されません)。

```
<%: new HtmlString("<strong>HTML that is not encoded</strong>") %>
```

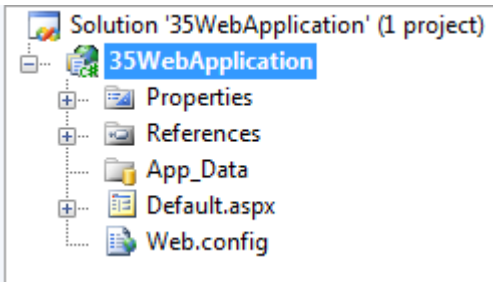
ASP.NET MVC 2 ヘルパー メソッドは、この新しい構文を処理して、二重エンコードを行わないように更新されていますが、これは ASP.NET 4 を実行している場合のみです。この新しい構文は、ASP.NET 3.5 SP1 を使用するアプリケーションを実行する場合は、機能しません。

この構文によって、確実に XSS 攻撃を防げるわけではないことに注意してください。たとえば、二重引用符で囲まれていない属性値を使用する HTML には、依然として脆弱なユーザー入力が含まれる可能性があります。ASP.NET コントロールと ASP.NET MVC ヘルパーの出力に含まれる属性値は、必ず、推奨方法である二重引用符で囲まれた形式になります。

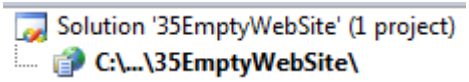
同様に、この構文では、ユーザー入力を基に JavaScript 文字列を作成する場合などに、JavaScript エンコードが実行されません。

プロジェクト テンプレートの変更

以前のバージョンの ASP.NET では、Visual Studio を使用して新しい Web サイト プロジェクトまたは Web アプリケーション プロジェクトを作成する場合、作成されたプロジェクトには Default.aspx ページ、既定の Web.config ファイル、および App_Data フォルダーしか含まれていません (下図参照)。



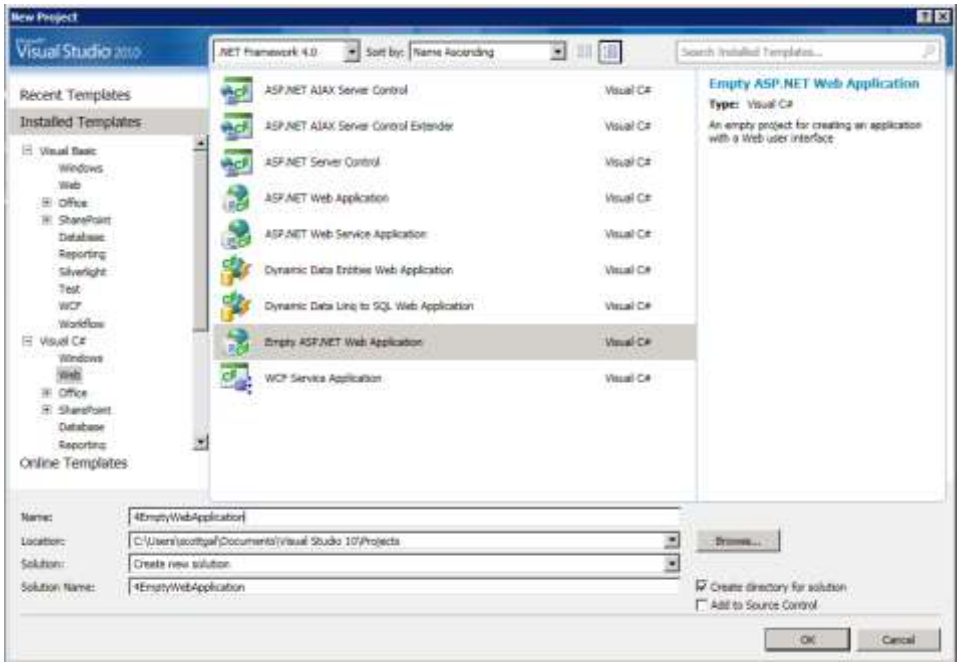
Visual Studio は空の Web サイト プロジェクトもサポートしており、このプロジェクトにはファイルがまったくありません (下図参照)。



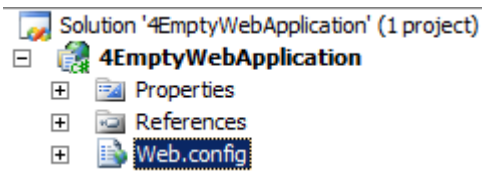
このため、初心者にとっては、運用 Web アプリケーションを構築するための手がかりがほとんどありません。したがって、ASP.NET 4 では、空の Web アプリケーション プロジェクト用に 1 つ、Web アプリケーション用に 1 つ、Web サイト プロジェクト用に 1 つ、合計 3 種類の新しいテンプレートが導入されています。

空の Web アプリケーション テンプレート

名前が示すように、空の Web アプリケーション テンプレートは、各種ファイルが取り除かれている Web アプリケーション プロジェクトです。このプロジェクト テンプレートは、Visual Studio の [新しいプロジェクト] ダイアログ ボックスから選択します (下図参照)。



空の ASP.NET Web アプリケーションを作成すると、Visual Studio は次のようなフォルダー レイアウトを作成します。



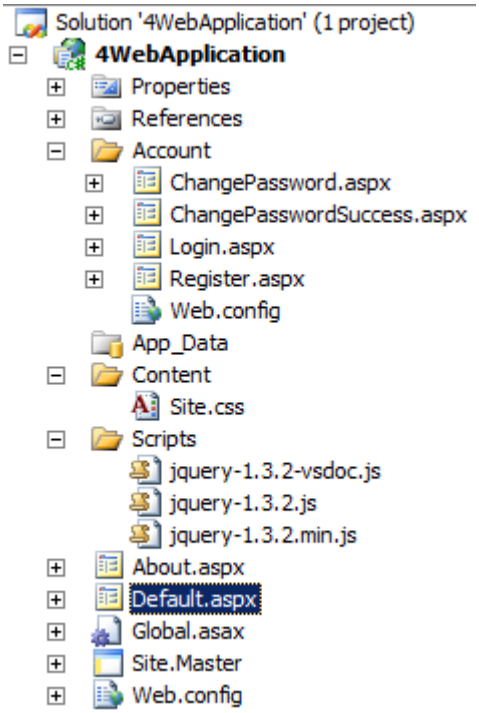
これは、1 つを除いて、以前のバージョンの ASP.NET の空の Web サイトのレイアウトに似ています。Visual Studio 2010 の空の Web アプリケーションおよび空の Web サイト プロジェクトには、次のような、プロジェクトの対応バージョンのフレームワークを特定するために Visual Studio が使用する情報を保持した最小限の Web.config ファイルが含まれています。

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <compilation debug="true" targetFramework="4.0" />
    </system.web>
</configuration>
```

この targetFramework プロパティがないと、古いアプリケーションを開く際に互換性を維持するため、Visual Studio は既定で .NET Framework 2.0 を対応バージョンに設定します。

Web アプリケーション テンプレートと Web サイト プロジェクト テンプレート

Visual Studio 2010 に付属する他の新しい 2 つのプロジェクト テンプレートは、大幅に変更されています。次の図は、新しい Web アプリケーション プロジェクトを作成する際に作成されるプロジェクト レイアウトです(Web サイト プロジェクトのレイアウトは、事実上これと同じです)。



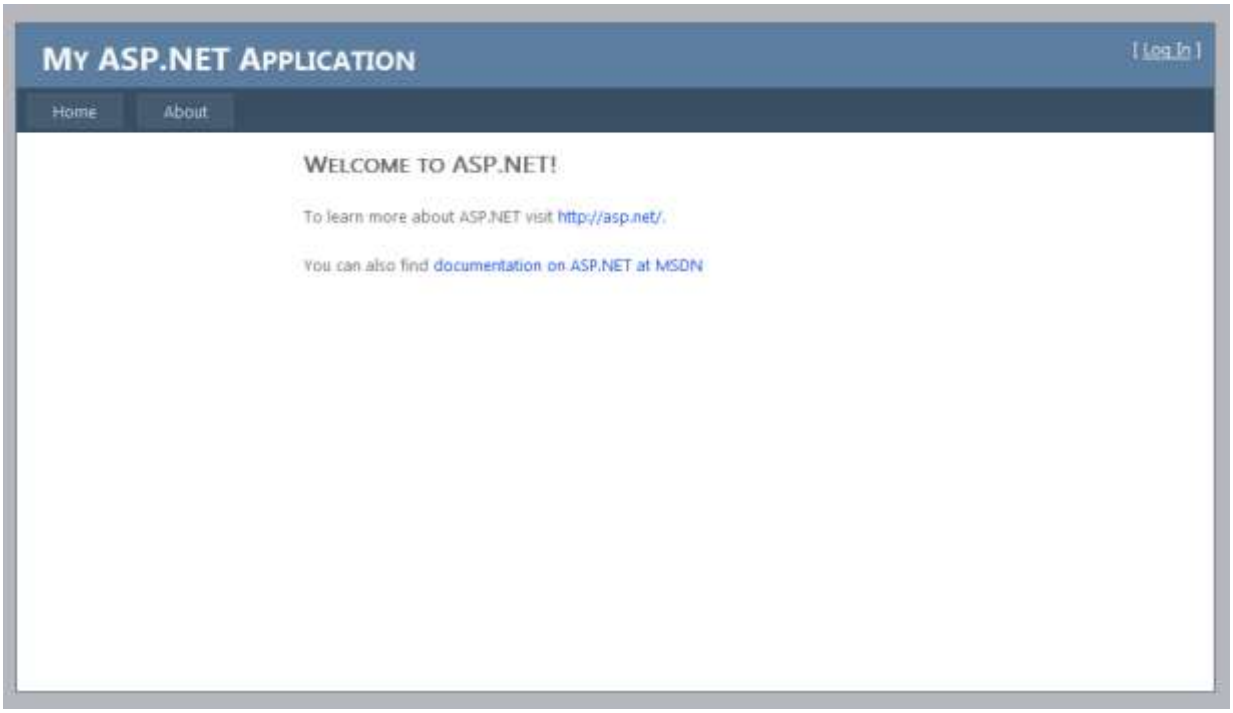
プロジェクトには、以前のバージョンでは作成されなかったさまざまなファイルが含まれています。また、新しい Web アプリケーション プロジェクトには、基本のメンバーシップ機能が構成されているため、すぐに新しいアプリケーションへのアクセスをセキュリティ保護する作業に取り掛かることができます。この機能が追加されたことで、新しいプロジェクトの Web.config ファイルには、メンバーシップ、役割、およびプロファイルの構成に使用されるエントリが追加されています。新しい Web アプリケーション プロジェクトの Web.config ファイルの例を次に示します(この例では、roleManager が無効にされています)。

```
<?xml version="1.0"?>
<configuration>
    <connectionStrings>
      <add name="ApplicationServices" connectionString="data source=.\SQLEXPRESS;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;Use
Instance=true" providerName="System.Data.SqlClient"/>
    </connectionStrings>

    <system.web>
      <compilation debug="true" targetFramework="4.0" />

      <authentication mode="Forms">
        <forms loginUrl="~/Account/Login.aspx"/>
      </authentication>

      <membership>
        <providers>
          <clear/>
          <add name="AspNetSqlMembershipProvider" type="System.Web.Security.SqlMembershipProvider, System.Web, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a" connectionStringName="ApplicationServices" enablePasswordRetrieval="false" enablePasswordReset="true"
requiresQuestionAndAnswer="false" requiresUniqueEmail="false" passwordFormat="Hashed" maxInvalidPasswordAttempts="5" minRequiredPasswordLength="6"
minRequiredNonalphanumericCharacters="0" passwordAttemptWindow="10" passwordStrengthRegularExpression="" applicationName="/"/>
        </providers>
      </membership>

      <profile>
        <providers>
          <clear/>
          <add name="AspNetSqlProfileProvider" type="System.Web.Profile.SqlProfileProvider, System.Web, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a" connectionStringName="ApplicationServices" applicationName="/"/>
        </providers>
      </profile>

      <roleManager enabled="false">
        <providers>
          <clear/>
          <add connectionStringName="ApplicationServices" applicationName="/" name="AspNetSqlRoleProvider" type="System.Web.Security.SqlRoleProvider,
System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
          <add applicationName="/" name="AspNetWindowsTokenRoleProvider" type="System.Web.Security.WindowsTokenRoleProvider, System.Web, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
        </providers>
      </roleManager>

      <pages clientIDMode="Predictable">
      </pages>
    </system.web>
</configuration>
```

プロジェクトには、Account ディレクトリに 2 つ目の Web.config ファイルも含まれています。2 つ目の構成ファイルでは、ログインしていないユーザーによる ChangePassword.aspx ページへのアクセスをセキュリティ保護する手段を提供します。2 つ目の Web.config ファイル内容の例を次に示します。

```
<?xml version="1.0"?>
<configuration>
    <location path="ChangePassword.aspx">
      <system.web>
        <authorization>
          <deny users="?"/>
        </authorization>
      </system.web>
    </location>
</configuration>
```

新しいプロジェクト テンプレートで既定で作成されるページの内容も、以前のバージョンに比べて増えています。プロジェクトには既定のマスター ページと CSS ファイルが含まれ、既定のページ (Default.aspx) がマスター ページを使用するように既定で構成されています。したがって、Web アプリケーションまたは Web サイトの初回実行時に、既定の (ホーム) ページが既に使用できる状態になっています。実際、このページは、新しい MVC アプリケーションを起動するときに表示される既定のページに似ています。



プロジェクト テンプレートにこのような変更を行ったのは、新しい Web アプリケーションを構築する際の手がかりを提供するためです。セマンティクスが正しく、厳密に XHTML 1.0 に準拠したマークアップと、CSS を使用して指定されたレイアウトを使用しているテンプレートのページは、ASP.NET 4 Web アプリケーションを構築する際のベスト プラクティスとし

て参考にできます。また、既定のページは、簡単にカスタマイズできる 2 列のレイアウトになっています。

たとえば、新しい Web アプリケーションでは、いくつかの色を変更し、My ASP.NET Application ロゴの代わりに自社のロゴを挿入するとします。それには、Content の下に新しいディレクトリを作成して、自社のロゴ イメージを保存します。

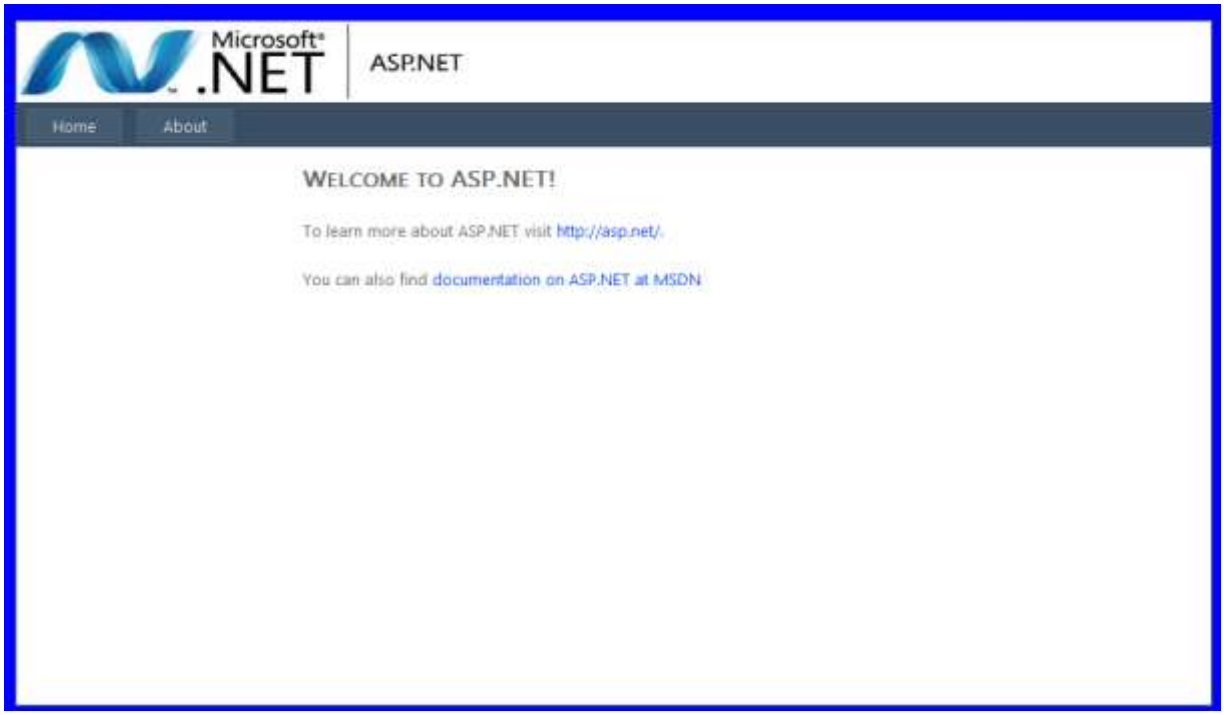
ページにイメージを追加するには、Site.Master ファイルを開き、My ASP.NET Application というテキストが定義されている場所を探して、src 属性に新しいロゴ イメージを設定した image 要素でこのテキストを置き換えます。

```
<div class="title">
    <img src="Content/Images/ASPNETLogo.jpg" width="376px" height="62px" border="0" alt="ASP.NET Logo" />
</div>
```

次に、Site.css ファイルを開いて、CSS クラス定義を変更し、ページの背景ととヘッダーの色を変更します。

```
.header
{
    position: relative;
    margin: 0px;
    padding: 0px;
    background: white;
}
```

これらの変更の結果、わずかな手間で、次のようなカスタマイズしたホーム ページを表示できます。



CSS 関連の強化

ASP.NET 4 で重点的に取り組まれた分野の 1 つは、最新の HTML 標準に準拠した HTML のレンダリングに関するものです。これには、ASP.NET Web サーバー コントロールが CSS スタイルを使用する方法の変更も含まれます。

レンダリングの互換性設定

既定では、Web アプリケーションまたは Web サイトの対応バージョンが .NET Framework 4 の場合、pages 要素の controlRenderingCompatibilityVersion 属性が "4.0" に設定されます。この要素はコンピューター レベルの Web.config ファイルで定義され、既定ではすべての ASP.NET 4 アプリケーションに適用されます。

```
<system.web>
  <pages controlRenderingCompatibilityVersion="3.5|4.0"/>
</system.web>
```

controlRenderingCompatibility の値は文字列なので、将来のリリースでバージョン定義が新しくなっても対応できます。現在のリリースでは、次の値がサポートされています。

- “3.5”。レガシーのレンダリングとマークアップを示します。コントロールによってレンダリングされるマークアップは 100% 下位互換性が保たれ、xhtmlConformance プロパティの設定が適用されます。
- “4.0”。ASP.NET Web サーバー コントロールは次のことを行います。
    - xhtmlConformance プロパティは、常に “Strict” として扱われます。その結果、コントロールは XHTML 1.0 Strict マークアップをレンダリングします。
    - 無効なスタイルはレンダリングしない非入力コントロールを無効にします。
    - 隠しフィールドを囲む div 要素がスタイル付きになったため、ユーザーが作成した CSS ルールに干渉しません。
    - Menu コントロールは、セマンティクスが正しく、アクセシビリティ ガイドラインに従ったマークアップをレンダリングします。
    - 検証コントロールは、インライン スタイルをレンダリングしません。
    - これまで border="0" をレンダリングしていたコントロール (ASP.NET Table コントロールの派生コントロールと ASP.NET Image コントロール) で、この属性がレンダリングされなくなりました。

コントロールの無効化

ASP.NET 3.5 SP1 以前のバージョンでは、Enabled プロパティが false に設定されている HTML マークアップでは、disabled 属性がレンダリングされます。ただし、HTML 4.01 の仕様に従うと、この値を設定できるのは input 要素だけです。

ASP.NET 4 では、controlRenderingCompatabilityVersion プロパティを "3.5" に設定できます。

```
<system.web>
  <pages controlRenderingCompatibilityVersion="3.5"/>
</system.web>
```

次のように Label コントロールのマークアップを作成すると、コントロールは無効になります。

```
<asp:Label id="Label" runat="server" Text="Test" Enabled="false">
```

Label コントロールは、次の HTML をレンダリングします。

```
<span id="Label1" disabled="disabled">Test</span>
```

ASP.NET 4 では、controlRenderingCompatabilityVersion を "4.0" に設定できます。この設定では、input 要素をレンダリングするコントロールの Enabled プロパティが false に設定されている場合のみ、disabled 属性をレンダリングします。HTML input 要素をレンダリングしないコントロールは、コントロールの無効時の表示の定義に使用できる CSS を参照する class 属性をレンダリングします。たとえば、上記の例の Label コントロールは、次のようなマークアップを生成します。

```
<span id="Label1" class="aspNetDisabled">Test</span>
```

このコントロールに指定されているクラスの既定の値は "aspNetDisabled" です。ただし、WebControl クラスの DisabledCssClass 静的プロパティを設定することで、この既定値は変更できます。コントロールを開発する場合、特定のコントロールに使用する動作も、SupportsDisabledAttribute プロパティを使用して定義できます。

隠しフィールドを囲む div 要素の非表示化

ASP.NET 2.0 以降のバージョンでは、システム固有の隠しフィールド (ビュー ステート情報の保存に使用される hidden 要素など) は、XHTML 標準に準拠するために、div 要素内でレンダリングされます。ただし、これは、CSS ルールがページ上の div 要素に影響する場合、問題が生じる可能性があります。たとえば、ページ上で隠し div 要素の周囲に 1 ピクセルの線が表示されてしまう可能性があります。ASP.NET 4 では、ASP.NET によって生成された隠しフィールドを囲む div 要素は、次のような CSS クラス参照を追加します。

```
<div class="aspNetHidden">...</div>
```

そのうえで、ASP.NET によって生成される hidden 要素にのみ適用される CSS クラスを定義できます。

```
<style type="text/css">
  DIV# aspNetHidden {border:0;}
</style>
```

テンプレート コントロールの外部テーブルのレンダリング

既定では、テンプレートをサポートする次の ASP.NET Web サーバー コントロールは、インライン スタイルの適用に使用される外部テーブル内にラップされます。

- *FormView*
- *Login*
- *PasswordRecovery*
- *ChangePassword*
- *Wizard*
- *CreateUserWizard*

これらのコントロールには、RenderOuterTable という新しいプロパティが追加され、これにより外部テーブルをマークアップから削除できます。たとえば、次の FormView コントロールの例を考えてみましょう。

```
<asp:FormView ID="FormView1" runat="server">
 <ItemTemplate>
  Content
 </ItemTemplate>
</asp:FormView>
```

このマークアップは、HTML テーブルを含む次の出力をページにレンダリングします。

```
<table cellspacing="0" border="0" id="FormView1" style="border-collapse:collapse;">
 <tr>
  <td colspan="2">
   Content
  </td>
 </tr>
</table>
```

テーブルがレンダリングされないようにするには、FormView コントロールの RenderOuterTable プロパティを設定します。

```
<asp:FormView ID="FormView1" runat="server" RenderOuterTable="false">
```

上記の例では、table、tr、および td 要素を含まない次の出力がレンダリングされます。

```
  Content
```

この機能強化により、コントロールによって予期しないタグがレンダリングされないため、CSS を使用してコントロールのコンテンツにスタイルを設定しやすくなります。

注 この変更により、Visual Studio 2010 デザイナーの自動フォーマット機能のサポートが無効になります。これは、自動フォーマット オプションによって生成されるスタイル属性をホストできる table 要素がなくなるためです。

ListView コントロールの機能強化

ListView コントロールは、ASP.NET 4 で使いやすくなっています。以前のバージョンの ListView コントロールでは、既知の ID を設定したサーバー コントロールを含むレイアウト テンプレートを指定する必要がありました。次のマークアップは、ASP.NET 3.5 での ListView コントロールの典型的な使用例です。

```
<asp:ListView ID="ListView1" runat="server">
  <LayoutTemplate>
    <asp:PlaceHolder ID="ItemPlaceHolder" runat="server"></asp:PlaceHolder>
  </LayoutTemplate>
  <ItemTemplate>
    <% Eval("LastName")%>
  </ItemTemplate>
</asp:ListView>
```

ASP.NET 4 では、ListView コントロールはレイアウト テンプレートを必要としません。上記のマークアップは、次のマークアップに置き換えることができます。

```
<asp:ListView ID="ListView1" runat="server">
  <ItemTemplate>
    <% Eval("LastName")%>
  </ItemTemplate>
</asp:ListView>
```

CheckBoxList コントロールと RadioButtonList コントロールの機能強化

ASP.N ET 3.5 では、次の 2 つの設定を使用することで、CheckBoxList と RadioButtonList のレイアウトを指定できました。

- Flow: span 要素をレンダリングしてコンテンツを保持します。
- Table: table 要素をレンダリングしてコンテンツを保持します。

これらの各コントロールのマークアップの例を次に示します。

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server" RepeatLayout="Flow">
 <asp:ListItem Text="CheckBoxList" Value="cbl" />
</asp:CheckBoxList>

<asp:RadioButtonList runat="server" RepeatLayout="Table">
 <asp:ListItem Text="RadioButtonList" Value="rbl" />
</asp:RadioButtonList>
```

既定では、次のような HTML がレンダリングされます。

```
<span id="CheckBoxList1"><input id="CheckBoxList1_0" type="checkbox" name="CheckBoxList1$0" /><label
for="CheckBoxList1_0">CheckBoxList</label></span>

<table id="RadioButtonList1" border="0">
 <tr>
  <td><input id="RadioButtonList1_0" type="radio" name="RadioButtonList1" value="rbl" /><label
for="RadioButtonList1_0">RadioButtonList</label></td>
 </tr>
</table>
```

これらのコントロールには項目のリストが含まれるため、正しいセマンティクスの HTML をレンダリングするには、HTML list 要素を使用してコンテンツをレンダリングする必要があります。これは、支援テクノロジを使用して Web ページを読むユーザーにとって扱いやすく、CSS を使用してコントロールのスタイルを設定しやすくします。

ASP.NET 4 では、CheckBoxList コントロールと RadioButtonList コントロールは、RepeatLayout プロパティに次の新しい値をサポートします。

- OrderedList – ol 要素内の li 要素としてレンダリングされます。
- UnorderedList – ul 要素内の li 要素としてレンダリングされます。

上記の新しい値の使用例を次に示します。

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server"
    RepeatLayout="OrderedList">
 <asp:ListItem Text="CheckBoxList" Value="cbl" />
```

```
</asp:CheckBoxList>

<asp:RadioButtonList ID="RadioButtonList1" runat="server"
    RepeatLayout="UnorderedList">
 <asp:ListItem Text="RadioButtonList" Value="rbl" />
</asp:RadioButtonList>
```

上記のマークアップでは、次の HTML が生成されます。

```
<ol id="CheckBoxList1">
 <li><input id="CheckBoxList1_0" type="checkbox" name="CheckBoxList1$0" value="cbl" /><label
for="CheckBoxList1_0">CheckBoxList</label></li>
</ol>

<ul id="RadioButtonList1">
 <li><input id="RadioButtonList1_0" type="radio" name="RadioButtonList1" value="rbl" /><label
for="RadioButtonList1_0">RadioButtonList</label></li>
</ul>
```

注 RepeatLayout を OrderedList または UnorderedList に設定したすると RepeatDirection プロパティは使用できなくなり、マークアップまたはコード内に RepeatDirection プロパティが設定されていると実行時に例外がスローされます。これらのコントロールの表示レイアウトは CSS を使用して定義されるため、RepeatDirection プロパティには値が含まれません。

Menu コントロール関連の機能強化

ASP.NET 4 以前は、Menu コントロールは一連の HTML テーブルをレンダリングしていました。これは、インライン プロパティを設定せずに CSS スタイルを適用することをさらに難しくしているだけでなく、アクセシビリティ標準にも準拠していませんでした。

ASP.NET 4 では、Menu コントロールは、順序指定されていないリストとリスト要素から成るセマンティック マークアップを使用して、HTML をレンダリングするようになりました。ASP.NET ページ内の Menu コントロールのマークアップの例を次に示します。

```
<asp:Menu ID="Menu1" runat="server">
  <Items>
    <asp:MenuItem Text="Home" Value="Home" />
    <asp:MenuItem Text="About" Value="About" />
```

```
  </Items>
</asp:Menu>
```

ページのレンダリング時に、コントロールは次の HTML を生成します (わかりやすくするため onclick コードは省略しています)。

```
<div id="Menu1">
  <ul>
    <li><a href="#" onclick="...">Home</a></li>
    <li><a href="#" onclick="...">About</a></li>
  </ul>
</div>
<script type="text/javascript">
 new Sys.WebForms.Menu('Menu1');
</script>
```

レンダリングの改善だけでなく、メニューのキーボード操作もフォーカスの管理を使用することで改善されています。Menu コントロールにフォーカスがあれば、矢印キーを使用して要素間を移動できます。また、Menu コントロールには、メニューの ARIA ガイドライン (英語) に従って、ARIA (Accessible Rich Internet Application) のロールと属性が追加され、アクセシビリティが向上しています。

Menu コントロールのスタイルは、レンダリングされた HTML 要素を使用してインラインでレンダリングされるのではなく、ページの先頭の style ブロック内でレンダリングされます。Menu コントロールのスタイル設定を完全に制御するのであれば、新しい IncludeStyleBlock プロパティを false に設定します。そうすると、style ブロックが生成されません。このプロパティを使用する方法の 1 つとしては、Visual Studio デザイナーの自動フォーマット機能を使用して、メニューの外観を設定します。次に、ページを実行して、ページのソースを開き、レンダリングされた style ブロックを外部の CSS ファイルにコピーします。Visual Studio で、スタイルを取り消し、IncludeStyleBlock を false に設定します。これにより、外部のスタイルシートのスタイルを使用して、メニューの外観が定義されます。

Wizard コントロールと CreateUserWizard コントロール

ASP.NET の Wizard コントロールと CreateUserWizard コントロールは、これらのコントロールがレンダリングする HTML を定義できるテンプレートをサポートします(CreateUserWizard は Wizard の派生クラスです)。完全にテンプレート化された CreateUserWizard コントロールのマークアップの例を次に示します。

```
<asp:CreateUserWizard ID="CreateUserWizard1" runat="server" ActiveStepIndex="0">
 <HeaderTemplate>
 </HeaderTemplate>

 <SideBarTemplate>
 </SideBarTemplate>

 <StepNavigationTemplate>
 </StepNavigationTemplate>

 <StartNavigationTemplate>
 </StartNavigationTemplate>

 <FinishNavigationTemplate>
 </FinishNavigationTemplate>

 <WizardSteps>
  <asp:CreateUserWizardStep ID="CreateUserWizardStep1" runat="server">
   <ContentTemplate>
   </ContentTemplate>

   <CustomNavigationTemplate>
   </CustomNavigationTemplate>
 </asp:CreateUserWizardStep>

  <asp:CompleteWizardStep ID="CompleteWizardStep1" runat="server">
   <ContentTemplate>
   </ContentTemplate>
  </asp:CompleteWizardStep>

 </WizardSteps>
</asp:CreateUserWizard>
```

このコントロールでは、次のような HTML が生成されます (テンプレートのコンテンツではなく、コントロールが生成するマークアップを示しています)。

```
<table cellspacing="0" cellpadding="0" border="0" id="CreateUserWizard1"
   style="border-collapse:collapse;">
 <tr>
  <td>Header</td>
 </tr>
 <tr style="height:100%;">
  <td>
   <table cellspacing="0" cellpadding="0" border="0"
     style="height:100%;width:100%;border-collapse:collapse;">
    <tr>
     <td style="height:100%;width:100%;"></td>
    </tr>
          </table>
  </td>
 </tr>
 <tr>
  <td align="right"></td>
 </tr>
</table>
```

ASP.NET 3.5 SP1 では、テンプレートのコンテンツを変更できますが、Wizard コントロールの出力の制御には制限があります。ASP.NET 4 では、LayoutTemplate テンプレートを作成し、(予約済みの名前を使用して) PlaceHolder コントロールを挿入し、Wizard control のレンダリング方法を指定できます。

```
<asp:CreateUserWizard ID="CreateUserWizard1" runat="server" ActiveStepIndex="1">
 <LayoutTemplate>
  <asp:PlaceHolder ID="headerPlaceholder" runat="server" />
  <asp:PlaceHolder ID="sideBarPlaceholder" runat="server" />
  <asp:PlaceHolder id="wizardStepPlaceholder" runat="server" />
  <asp:PlaceHolder id="navigationPlaceholder" runat="server" />
 </LayoutTemplate>
 <HeaderTemplate>
  Header
 </HeaderTemplate>
 <WizardSteps>
  <asp:CreateUserWizardStep runat="server">
   <ContentTemplate>
   </ContentTemplate>
  </asp:CreateUserWizardStep>
  <asp:CompleteWizardStep runat="server">
   <ContentTemplate>
   </ContentTemplate>
  </asp:CreateUserWizardStep>
```

```
 </WizardSteps>
</asp:CreateUserWizard>
```

この例では、LayoutTemplate 要素内に、次の名前付きプレースホルダーが含まれています。

- headerPlaceholder –実行時に、HeaderTemplate 要素のコンテンツに置き換えられます。
- sideBarPlaceholder –実行時に、SideBarTemplate 要素のコンテンツに置き換えられます。
- wizardStepPlaceHolder –実行時に、WizardStepTemplate 要素のコンテンツに置き換えられます。
- navigationPlaceholder –実行時に、定義済みのナビゲーション テンプレートに置き換えられます。

この例のプレースホルダーを使用するマークアップでは、次の HTML がレンダリングされます (テンプレートに実際に定義されているコンテンツは含まれていません)。

```
<span>
</span>
```

現在ユーザー定義でない HTML は span 要素のみです (span 要素がレンダリングされないとしても、将来のリリースではユーザー定義になると思われます)。したがって、Wizard コントロールによって生成される実質上すべてのコンテンツを完全に制御できます。

ASP.NET MVC

ASP.NET MVC は、2009 年の 3 月に ASP.NET 3.5 SP1 のアドオン フレームワークとして導入されました。Visual Studio 2010 のリリース時には、ASP.NET MVC 2 の RTM バージョンが含まれる予定です。ASP.NET 4 に含まれる ASP.NET MVC 2 のバージョンには新機能が追加されています。

エリアのサポート

エリアを使用すると、コントローラーやビューを大規模アプリケーションのセクションにグループ分けして、他のセクションから相対的に分離できます。各エリアは個別の ASP.NET MVC プロジェクトとして実装でき、メインのアプリケーションから参照できます。これに

より、大規模アプリケーション構築に伴う複雑さに対応し、1 つのアプリケーションを複数のチームによって開発しやすくしています。

DataAnnotations 属性の検証のサポート

DataAnnotations 属性を使用すると、メタデータ属性を使用して、モデルに検証ロジックを関連付けることができます。DataAnnotations 属性は、ASP.NET 3.5 SP1 において ASP.NET Dynamic Data に導入されました。DataAnnotations 属性は、既定のモデル バインダーに統合されており、メタデータによるユーザー入力の検証機能を提供します。

テンプレート ヘルパー

テンプレート ヘルパーを使用すると、編集テンプレートと表示テンプレートにデータ型を自動的に関連付けることができます。たとえば、テンプレート ヘルパーを使用して、日付の選択 UI 要素が System.DateTime 値に対して自動的にレンダリングされるように指定できます。これは、ASP.NET Dynamic Data のフィールド テンプレートに似ています。

Html.EditorFor ヘルパー メソッドと Html.DisplayFor ヘルパー メソッドには、標準のデータ型と、複数のプロパティがある複雑なオブジェクトのレンダリングのサポートが組み込まれています。また、DisplayName や ScaffoldColumn などの data-annotation 属性を ViewModel オブジェクトに適用できるようにすることで、レンダリングの基本的なカスタマイズもサポートしています。

UI ヘルパーからの出力をさらにカスタマイズして、生成される内容を完全に制御することをよく考えます。Html.EditorFor ヘルパー メソッドと Html.DisplayFor ヘルパー メソッドは、テンプレート メカニズムを使用して、このニーズをサポートしています。このメカニズムを利用すると、レンダリングされる出力をオーバーライドして制御できる外部テンプレートを定義できます。テンプレートは、クラス単位でレンダリングできます。

Dynamic Data

Dynamic Data は、2008 年半ばの .NET Framework 3.5 SP1 リリースから導入されました。この機能は、データ駆動型アプリケーションを作成するため、次のようなさまざまな機能強化が行われています。

- データ駆動型 Web サイトをすばやく構築するための RAD エクスペリエンス
- データ モデルに定義されている制約に基づく自動検証
- GridView コントロールおよび DetailsView コントロールによって生成されるマークアップを、Dynamic Data プロジェクトに含まれるフィールド テンプレートを使用して簡単に変更できる機能

> **注** 詳細については、MSDN ライブラリの Dynamic Data についてのドキュメントを参照してください。

ASP.NET4 では、データ駆動型 Web サイトをすばやく構築するための機能がさらに強化されています。

既存のプロジェクトでの Dynamic Data の有効化

.NET Framework 3.5 SP1 に搭載された Dynamic Data 機能により、次のような新機能が提供されました。

- フィールド テンプレート – データ バインド コントロール用のデータ型に基づくテンプレートを提供します。フィールド テンプレートを使用すると、フィールドごとにテンプレート フィールドを使用するよりも簡単に、データ コントロールの外観をカスタマイズできます。
- 検証 – Dynamic Data では、データ クラスに属性を使用して、必須フィールド、範囲チェック、型チェック、正規表現を使用したパターン マッチング、カスタム検証などの一般的なシナリオに使用する検証を指定できます。検証は、データ コントロールによって適用されます。

ただし、これらの機能には、次の要件がありました。

- データ アクセス層は、Entity Framework または LINQ to SQL を基盤にする必要がありました。
- これらの機能にサポートされるデータ ソース コントロールは、EntityDataSource コントロールまたは LinqDataSource コントロールだけでした。
- この機能のサポートに必要なファイルを揃えるには、Dynamic Data テンプレートまたは Dynamic Data エンティティ テンプレートを使用して作成された Web プロジェクトが必要でした。

ASP.NET 4 での Dynamic Data のサポートの主な目標は、あらゆる ASP.NET アプリケーションで Dynamic Data の新しい機能が利用できるようにすることです。既存のページで Dynamic Data 機能を利用できるコントロールのマークアップの例を次に示します。

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="True"
   DataKeyNames="ProductID" DataSourceID="LinqDataSource1">
</asp:GridView>
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
   ContextTypeName="DataClassesDataContext" EnableDelete="True" EnableInsert="True"
   EnableUpdate="True" TableName="Products">
</asp:LinqDataSource>
```

これらのコントロールで Dynamic Data サポートを有効にするには、次のコードをページのコードに追加する必要があります。

```
GridView1.EnableDynamicData(typeof(Product));
```

GridView コントロールが編集モードのときは、Dynamic Data は入力されたデータが適切な形式かどうかを自動的に検証します。適切な形式でなければエラー メッセージが表示されます。

この機能には、挿入モードの既定値を指定できるなど、他にもメリットがあります。Dynamic Data を使用しないでフィールドの既定値を実装するには、イベントを関連付け、コントロールを検索し (FindControl を使用)、値を設定しなければなりません。ASP.NET 4 では、

EnableDynamicData 呼び出しで 2 つ目のパラメーターがサポートされます。このパラメーターを使用すると、オブジェクトの任意のフィールドに既定値を渡すことができます。

```
DetailsView1.EnableDynamicData(typeof(Product), new { ProductName = "DefaultName" });
```

DynamicDataManager コントロールの宣言構文

DynamicDataManager コントロールは、コード内でしか構成できないのではなく、ASP.NET のほとんどのコントロールと同様、宣言によって構成できるように改善されています。DynamicDataManager コントロールのマークアップは、次の例のようになります。

```
<asp:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true">
 <DataControls>
   <asp:DataControlReference ControlID="GridView1" />
 </DataControls>
</asp:DynamicDataManager>

<asp:GridView id="GridView1" runat="server"
</asp:GridView>
```

このマークアップでは、DynamicDataManager コントロールの DataControls セクションで参照されている GridView1 コントロールでの Dynamic Data の動作を有効にします。

エンティティ テンプレート

エンティティ テンプレートは、カスタム ページを作成しないで、データのレイアウトをカスタマイズできる新しい手段です。ページ テンプレートは、FormView コントロール (以前のバージョンの Dynamic Data では、DetailsView コントロールがページ テンプレートに使用されていました) と DynamicEntity コントロールを使用して、エンティティ テンプレートをレンダリングします。これにより、Dynamic Data によってレンダリングされるマークアップをより細かく制御できます。

以下は、エンティティ テンプレートを保持するプロジェクト ディレクトリのレイアウトです。

```
\DynamicData\EntityTemplates
```

\DynamicData\EntityTemplates\Default.ascx

\DynamicData\EntityTemplates\Default_Edit.ascx

\DynamicData\EntityTemplates\Default_Insert.ascx

EntityTemplate ディレクトリには、データ モデル オブジェクトの表示方法を指定するテンプレートが保持されます。既定では、オブジェクトは Default.ascx テンプレートを使用してレンダリングされます。このテンプレートは、ASP.NET 3.5 SP1 の Dynamic Data が使用する DetailsView コントロールによって作成されるマークアップとまったく同じようなマークアップを提供します。Default.ascx コントロールのマークアップの例を次に示します。

```
<asp:EntityTemplate runat="server" ID="TemplateContainer1">
 <ItemTemplate>
  <tr
   <td>
    <asp:Label ID="Label1" runat="server" OnInit="Label_Init" />
   </td>
   <td>
    <asp:DynamicControl runat="server" OnInit="DynamicControl_Init" />
   </td>
  </tr>
 </ItemTemplate>
</asp:EntityTemplate>
```

既定のテンプレートを編集して、サイト全体の外観を変更できます。表示操作、編集操作、および挿入操作用のテンプレートがあります。新しいテンプレートをデータ オブジェクトの名前を基に追加して、ある種類のオブジェクトのみの外観を変更することができます。たとえば、次のテンプレートを追加できます。

\DynamicData\EntityTemplates\Products.aspx

このテンプレートには、次のマークアップが含まれる可能性があります。

```
<tr>
 <td>Name</td>
 <td><asp:DynamicControl runat="server" DataField="ProductName" /></td>
 <td>Category</td>
 <td><asp:DynamicControl runat="server" DataField="Category" /></td>
</tr>
```

新しいエンティティ テンプレートは、新しい DynamicEntity コントロールを使用して、ページ上に表示されます。このコントロールは、実行時にエンティティ テンプレートのコンテンツに置き換えられます。次のマークアップは、エンティティ テンプレートを使用する Detail.aspx ページ テンプレートの FormView コントロールです。このマークアップには、DynamicEntity 要素が含まれています。

```
<asp:FormView runat="server" ID="FormView1"
   DataSourceID="DetailsDataSource"
   OnItemDeleted="FormView1_ItemDeleted">
 <ItemTemplate>
   <table class="DDDetailsTable" cellpadding="6">
    <asp:DynamicEntity runat="server" />
    <tr class="td">
     <td colspan="2">
      <asp:DynamicHyperLink ID="EditHyperLink" runat="server"
         Action="Edit" Text="Edit" />
      <asp:LinkButton ID="DeleteLinkButton" runat="server"
         CommandName="Delete"
         CausesValidation="false"
         OnClientClick='return confirm("Are you sure you want to delete this item?");'
         Text="Delete" />
     </td>
    </tr>
   </table>
 </ItemTemplate>
</asp:FormView>
```

URL および電子メール アドレス用の新しいフィールド テンプレート

ASP.NET 4 では、EmailAddress.ascx と Url.ascx という新しい組み込みのフィールド テンプレートが導入されています。これらのテンプレートは、DataType 属性が EmailAddress または Url に設定されているフィールドに使用されます。EmailAddress オブジェクトの場合、フィールドは、mailto: プロトコルを使用して作成されるハイパーリンクとして表示されます。ユーザーがそのリンクをクリックすると、ユーザーの電子メール クライアントが開き、スケルトン メッセージが作成されます。オブジェクトの種類が Url の場合は、通常のハイパーリンクとして表示されます。

フィールドの設定例を次に示します。

```
[DataType(DataType.EmailAddress)]
public object HomeEmail { get; set; }

[DataType(DataType.Url)]
public object Website { get; set; }
```

DynamicHyperLink コントロールによるリンクの作成

Dynamic Data は、.NET Framework 3.5 SP1 で追加された新しいルーティング機能を使用して、エンド ユーザーが Web サイトにアクセスしたときに表示される URL を制御します。新しい DynamicHyperLink コントロールを使用すると、Dynamic Data サイトのページへのリンクを簡単に作成できます。DynamicHyperLink コントロールの使用例を次に示します。

```
<asp:DynamicHyperLink ID="ListHyperLink" runat="server"
   Action="List" TableName="Products">
 Show all products
</asp:DynamicHyperLink>
```

このマークアップでは、Global.asax ファイルに定義されているルーティングを基に、Products テーブルの List ページを指すリンクが作成されます。このコントロールは、Dynamic Data ページの基盤となっている既定のテーブル名を自動的に使用します。

データ モデルでの継承のサポート

Entity Framework でも、LINQ to SQL でも、データ モデルでの継承がサポートされます。たとえば、"保険ポリシー (InsurancePolicy)" テーブルがあるデータベースを考えます。このデータベースには、"保険ポリシー (InsurancePolicy)" と同じフィールドを持ち、他のフィールドが追加されている "車両保険ポリシー (CarPolicy)" テーブルや "住宅保険ポリシー (HousePolicy)" テーブルもあると考えられます。Dynamic Data は、データ モデルでの継承オブジェクトを把握し、継承されるテーブルの基盤構築をサポートするように変更されています。

多対多リレーションシップのサポート (Entity Framework のみ)

Entity Framework は、テーブル間の多対多リレーションシップに豊富なサポートを提供しています。これは、Entity オブジェクトのコレクションとしてリレーションシップとして公開

することで実装されます。多対多リレーションシップでのデータ表示およびデータ編集をサポートするために、新しい ManyToMany.ascx および ManyToMany_Edit.ascx フィールド テンプレートが追加されています。

表示制御と列挙値サポートを目的とする新しい属性

フィールドの表示方法をさらに細かく制御できるよう、DisplayAttribute が追加されています。以前のバージョンの Dynamic Data の DisplayName 属性では、フィールドのキャプションに使用される名前を変更できました。新しい DisplayAttribute クラスでは、フィールドの表示順序やフィルターとしてフィールドが使用されるかどうかなど、フィールドを表示するためオプションを指定できます。また、この属性を使用すると、GridView コントロールでラベルに使用される名前、DetailsView コントロールで使用される名前、フィールドのヘルプ テキスト、フィールドに使用されるウォーターマーク (フィールドがテキスト入力を受け付ける場合) を個別に制御することもできます。

EnumDataTypeAttribute クラスは、フィールドを列挙値にマップできるようにするために追加されました。この属性をフィールドに適用するときに、列挙型を指定できます。Dynamic Data は、新しい Enumeration.ascx フィールド テンプレートを使用して、列挙値を表示、編集するための UI を作成します。テンプレートは、データベースからの取得値と列挙値内の名前とをマップします。

フィルター サポートの強化

Dynamic Data 1.0 には、ブール型列と外部キー列用の組み込みのフィルターが付属していました。これらのフィルターでは、フィルターを表示するかどうかや、表示の順序を指定できませんでした。新しい DisplayAttribute 属性では、列をフィルターとして表示するかどうかや、表示順序を制御できるようにし、これらの問題を解決しています。

その他の強化として、フィルター処理サポートが、Web フォームの新しい QueryExtender 機能を使用するよう書き直されています。これにより、フィルターが使用されるデータ ソー

ス コントロールについての情報がなくても、フィルターを作成できます。これらの機能拡張のほか、フィルターはテンプレート コントロールに変えられ、新しいフィルターを追加できるようになっています。また、前述の DisplayAttribute クラスを使用すると、UIHint により列の既定のフィールド テンプレートをオーバーライドするのと同じ方法で、既定のフィルターをオーバーライドできます。

Visual Studio 2010 Web デザイナー関連の機能強化

Visual Studio 2010 の Web ページ デザイナーは、CSS の互換性を高め、HTML や ASP.NET マークアップのサポートを強化し、JScript 向けにデザインし直された IntelliSense を実装します。

CSS の互換性の強化

Visual Studio 2010 の Visual Web Developer デザイナーは、より忠実に CSS 2.1 標準に準拠するように更新されています。以前のバージョンの Visual Studio に比べて、HTML ソースの整合性が高まり、全体的な堅牢性が向上しています。内部のアーキテクチャも強化され、レンダリング、レイアウト、およびサービス性についての将来的な機能強化をより実現しやすくなっています。

HTML スニペットと JScript スニペット

HTML エディターでは、IntelliSense によるタグ名のオートコンプリート機能が提供されます。IntelliSense スニペット機能では、タグ全体などがオートコンプリートにより入力されます。Visual Studio 2010 では、以前のバージョンの Visual Studio でもサポートされていた C# や Visual Basic 以外に、Jscript でも IntelliSense スニペットがサポートされます。

Visual Studio 2010 には、必須属性 (runat="server" など) やタグ固有の共通属性 (ID、DataSourceID、ControlToValidate、Text など) を含め、よく使用される ASP.NET タグや HTML タグのオートコンプリートを支援するスニペットが 200 以上も含まれています。

追加のスニペットをダウンロードしたり、自身やチームでよく行う作業に使用するマークアップ ブロックをカプセル化した独自のスニペットを作成したりできます。

JScript IntelliSense の機能強化

Visual 2010 では、JScript IntelliSense はデザインが見直され、さらに便利に編集できるようになりました。IntelliSense は、registerNamespace などのメソッドや、その他の JavaScript フレームワークが使用する同様の手法によって動的に生成されたオブジェクトを認識できるようになります。パフォーマンスも向上し、処理遅延がまったくないか、ごくわずかな遅延のみで、大規模スクリプト ライブラリの分析や IntelliSense の表示を処理できます。互換性が大幅に向上し、ほとんどすべてのサードパーティのライブラリと、各種コーディング スタイルをサポートします。ドキュメント コメントは入力と同時に解析され、IntelliSense によって直ちに利用されるようになりました。

Visual Studio 2010 による Web アプリケーション配置

現状の Web アプリケーションの配置は、思うほど簡単ではありません。ASP.NET 開発者は、気が付くと次のような問題によく直面します。

- 共有ホスティング サイトの配置に、低速になりがちな FTP などのテクノロジーが必要。また、SQL スクリプトを実行してデータベースを構成するなどの作業を手動で実行する必要があるほか、アプリケーションとして仮想ディレクトリ フォルダーを構成するなど IIS 設定の変更が必要。
- エンタープライズ環境では、Web アプリケーション ファイルの配置に加えて、ASP.NET 構成ファイルと IIS 設定を変更しなければならないことが多い。データベース管理者は、アプリケーション データベースを実行するために、一連の SQL スクリプトの実行が必要。このようなインストールは手間がかかり、完了までに数時間かかることが多く、慎重な文書化が必要。

Visual Studio 2010 には、このような問題に対応し、シームレスに Web アプリケーションを配置できるようにするテクノロジが追加されていてます。このようなテクノロジの 1 つが、IIS Web 配置ツール (MsDeploy.exe) です。

Visual Studio 2010 の Web 配置機能は、次のカテゴリに大別できます。

- Web パッケージ
- Web.config の変換
- データベースの配置
- Web アプリケーションのワンクリック発行

これ以降、これらの各機能について詳しく説明します。

Web パッケージ

Visual Studio 2010 は MSDeploy ツールを使用して、アプリケーションの圧縮 (.zip) ファイルを作成します。これを "Web パッケージ" と呼びます。パッケージ ファイルには、アプリケーションについてのメタデータと、次のコンテンツが含まれます。

- IIS 設定。アプリケーション プールの設定やエラー ページの設定などを含みます。
- 実際の Web コンテンツ。Web ページ、ユーザー コントロール、静的コンテンツ (イメージや HTML ファイル) などです。
- SQL Server のデータベース スキーマとデータ。
- セキュリティ証明書、GAC にインストールされるコンポーネント、レジストリ設定など。

Web パッケージは、任意のサーバーにコピーし、IIS マネージャーを使用して手動でインストールできます。自動配置を行う場合は、コマンド ライン コマンドを使用するか、配置 API を使用してパッケージをインストールできます。

VS 2010 には、Web パッケージを作成するための、組み込みの MSBuild タスクとターゲットが用意されています。詳細については、Vishal Joshi のブログ記事「10 + 20 reasons why you

should create a Web Package (Web パッケージを作成すべき 10 + 20 の理由)」(英語) を参照してください。

Web.config の変換

Web アプリケーションの配置に、Visual Studio 2010 では XML Document Transform (XDT) (英語) を導入しています。XDT は、Web.config ファイルを配置設定から運用設定に変換できる機能です。変換設定は、web.debug.config、web.release.config などの名前を付けた変換ファイルに指定します(これらのファイル名は、MSBuild 構成と一致します)。変換ファイルには、配置される Web.config ファイルに必要な変更のみが含まれます。この変更は、簡単な構文を使用して指定します。

リリース構成の配置用に生成される可能性がある web.release.config ファイルの一部を、次の例に示します。この例の Replace キーワードは、配置中に Web.config ファイルの connectionString ノードが、例に指定されている値に置き換えられることを示します。

```
<connectionStrings xdt:Transform="Replace">
  <add name="BlogDB" connectionString="connection string detail]" />
</connectionStrings>
```

詳細については、Vishal Joshi のブログ記事「Web Deployment: Web.Config Transformation (Web 配置: Web.Config の変換)」(英語) を参照してください。

データベースの配置

Visual Studio 2010 の配置パッケージには、SQL Server データベースへの依存関係を含めることができます。パッケージ定義の一部として、ソース データベースの接続文字列を指定します。Web パッケージの作成時に、Visual Studio 2010 によって、データベース スキーマと任意でデータのための SQL スクリプトが作成され、これらがパッケージに追加されます。また、カスタム SQL スクリプトを用意して、サーバー上でこれらが実行される順序を指定することもできます。配置時に、ターゲット サーバーに適切な接続文字列を指定すると、配置

プロセスがこの接続文字列を使用して、データベース スキーマを作成するスクリプトとデータを追加するスクリプトを実行します。

また、ワンクリック発行を使用することで、アプリケーションがリモートの共有ホストに発行されるときに、直接データベースを発行するように配置を構成できます。詳細については、Vishal Joshi のブログ記事「Database Deployment with VS 2010 (VS 2010 を使用したデータベース配置)」(英語) を参照してください。

Web アプリケーションのワンクリック発行

Visual Studio 2010 では、IIS のリモート管理サービスを使用して、Web アプリケーションをリモート サーバーに発行できます。ホスト アカウント用、またはテスト サーバーやステージング サーバー用の発行プロファイルを作成します。各プロファイルには、適切な資格情報を安全に保存できます。その後、Web ワンクリック発行ツールバーを使用して、1 回のクリックで任意のターゲット サーバーに配置を行うことができます。Visual Studio 2010 では、MSBuild コマンド ラインを使用しても発行できます。これにより、継続的な統合モデルに発行を含めるように、チーム ビルド環境を構成できます。

詳細については、MSDN Web サイト上の How to: Deploy a Web Application Project Using One-Click Publish and Web Deploy と Vishal Joshi のブログ記事「Web 1-Click Publish with VS 2010 (VS 2010 を使用した Web の 1 クリック発行)」(英語) を参照してください。Visual Studio 2010 での Web アプリケーション配置のビデオ プレゼンテーションを見るには、Vishal Joshi のブログの「VS 2010 for Web Developer Previews (Web 開発者のための VS 2010 プレビュー)」(英語) を参照してください。

関連情報

ASP.NET 4 と Visual Studio 2010 の追加情報については、次の Web サイトを参照してください。

- ASP.NET 4 — MSDN Web サイトの ASP.NET 4 のオフィシャルドキュメント
- http://www.asp.net/ (英語) — ASP.NET 開発者のためのメインの Web リソース。

- http://www.asp.net/dynamicdata/ (英語) と ASP.NET Dynamic Data Content Map (英語)ー ASP.NET Dynamic Data のメインの Web リソース。
- http://www.asp.net/ajax/ (英語) ー ASP.NET AJAX のメインの Web リソ ース。
- http://blogs.msdn.com/webdevtools/ (英語) ー Visual Web Developer チームのブログ。Visual Studio 2010 の機能についての情報が含まれています。
- http://www.codeplex.com/aspnet (英語) ー ASP.NET のプレビュー リリースのメインの Web リソース。

免責事項

このドキュメントは暫定版であり、このソフトウェアの最終的な製品版の発売時に実質的に変更されることがあります。

このドキュメントに記載されている情報は、このドキュメントの発行時点におけるマイクロソフトの見解を反映したものです。変化する市場状況に対応する必要があるため、このドキュメントは、記載された内容の実現に関するマイクロソフトの確約とはみなされないものとします。また、発行以降に発表される情報の正確性に関して、マイクロソフトはいかなる保証もいたしません。

このドキュメントに記載された内容は情報の提供のみを目的としており、明示、黙示または法律の規定にかかわらず、これらの情報についてマイクロソフトはいかなる責任も負わないものとします。

お客様ご自身の責任において、適用されるすべての著作権関連法規に従ったご使用を願います。このドキュメントのいかなる部分も、米国 Microsoft Corporation の書面による許諾を受けることなく、その目的を問わず、どのような形態であっても、複製または譲渡することは禁じられています。ここでいう形態とは、複写や記録など、電子的な、または物理的なすべての手段を含みます。ただしこれは、著作権法上のお客様の権利を制限するものではありません。

マイクロソフトは、このドキュメントに記載されている内容に関し、特許、特許申請、商標、著作権、またはその他の無体財産権を有する場合があります。別途マイクロソフトのライセンス契約上に明示の規定のない限り、このドキュメントはこれらの特許、商標、著作権、またはその他の無体財産権に関する権利をお客様に許諾するものではありません。

別途記載されていない場合、このソフトウェアおよび関連するドキュメントで使用している会社、組織、製品、ドメイン名、電子メール アドレス、ロゴ、人物、場所、出来事などの名称は架空のものです。実在する商品名、団体名、個人名などとは一切関係ありません。