# Razor 構文と ASP.NET Web ページ

Microsoft® ASP.NET Web ページ はインターネット向けの Web サイトを構築する Web 開発者にとって世界で最良の体験を提供する無料の Web 開発技術です。本書は、Razor 構文とともに、ASP.NET Web ページを使うダイナミック Web コンテンツの作成方法についての概要を提供します。

## 内容


Razor 構文と ASP.NET Web ページ ........ 1
第 1 章 はじめての WebMatrix と ASP.NET Web ページ ........ 7
WebMatrix とは何か ........ 7
WebMatrix のインストール ........ 7
はじめての WebMatrix ........ 8
Web ページの作成 ........ 10
管理ツールを使ったヘルパーのインストール ........ 12
ASP.NET Web ページでコードを使う ........ 15
Visual Studio で、ASP.NET Razor ページをプログラミングする ........ 17
第 2 章 Razor 構文を使った ASP.NET Web プログラミングの紹介 ........ 20
8 つのプログラミング Tips ........ 20
1. @文字を使ってページにコードを追加する ........ 20
2. 中カッコでコードブロックを囲む ........ 21
3. ブロックの中で、各コード文をセミコロンで終わらせる ........ 22
4. 値を保存するために変数を使う ........ 22
5. ダブル クォーテーション記号でリテラル文字列値を囲む ........ 22
6. コードは大小文字を区別する ........ 23
7. コードのほとんどが関係するオブジェクト ........ 24
8. 意思決定用のコードを書く ........ 25
単純なコード サンプル ........ 26
基本的なプログラミングの概念 ........ 28
Razor 構文、サーバーコード、そして ASP.NET ........ 28
ASP.NET はどこで取り込むのか ........ 28
言語と構文 ........ 29
基本的な構文 ........ 30
変数 ........ 33
データ型の変換とテスト ........ 34

演算子 ....................................................................................................35
コード中でファイルやフォルダー パスを扱う ..............................................................36
条件文と繰り返し..........................................................................................38
オブジェクトとコレクション ...............................................................................43
パラメーター付きでメソッドを呼び出す ..................................................................46
エラー処理 .................................................................................................47
その他のリソース ...........................................................................................49
第３章　一貫性のある外観の作成 ..........................................................................50
再利用できるコンテンツ ブロックの作成 ...................................................................50
レイアウト ページを使った一貫性のある外観の作成......................................................53
複数のコンテンツ セクションを持つレイアウト ページのデザイン ......................................56
コンテンツ セクションをオプションにする .................................................................59
レイアウト ページにデータを渡す...........................................................................61
基本的なヘルパーの作成と使用 ............................................................................65
その他のリソース ...........................................................................................66
第４章　フォームを扱う.....................................................................................67
シンプルな HTML フォームを作成する.....................................................................67
フォームからユーザー入力を読み取る.......................................................................68
ユーザー入力の検証........................................................................................70
ポストバック後のフォームの値を復元する ..................................................................72
その他のリソース ...........................................................................................74
第５章　データを扱う........................................................................................75
データベースの紹介.........................................................................................75
データベースの作成.........................................................................................76
データベースへのデータの追加 .............................................................................77
データベースのデータの表示 ...............................................................................78
データベースのデータの挿入 ...............................................................................81
データベースのデータの更新 ...............................................................................84
データベースのデータの削除 ...............................................................................89
その他のリソース ...........................................................................................94
第６章　データをグリッドで表示する.......................................................................95
WebGrid ヘルパー .........................................................................................95
WebGrid ヘルパーを使ってデータを表示する...............................................................95
表示する列の指定と書式化..................................................................................97
グリッド全体をスタイル化する ..............................................................................99
データをページングする.....................................................................................101
その他のリソース ...........................................................................................102
第７章　データをグラフで表示する.........................................................................103

Chart ヘルパー .......... 103
グラフの要素 .......... 104
データからグラフを作成する .......... 104
配列を使う .......... 104
グラフのデータとしてデータベースの問い合わせを使う .......... 106
XML データを使う .......... 107
Web ページ中にグラフを表示する .......... 111
グラフのスタイル化 .......... 112
グラフの保存 .......... 113
グラフのキャッシュ .......... 114
グラフをイメージファイルとして保存する .......... 116
グラフを XML ファイルとして保存する .......... 117
その他のリソース .......... 119
第８章　ファイルを扱う .......... 120
テキストファイルの作成とデータの書き込み .......... 120
既存ファイルへのデータの追加 .......... 123
ファイルからのデータの読み込みと表示 .......... 124
ファイルの削除 .......... 126
ユーザーにファイルをアップロードさせる .......... 128
ユーザーに複数のファイルをアップロードさせる .......... 131
その他のリソース .......... 133
第９章　イメージを扱う .......... 134
Web ページに動的にイメージを追加する .......... 134
イメージをアップロードする .......... 136
イメージのサイズを変更する .......... 139
イメージを回転、反転させる .......... 141
イメージにウォーターマークを追加する .......... 142
イメージをウォーターマークとして使う .......... 144
その他のリソース .......... 145
第１０章　ビデオを扱う .......... 146
ビデオ プレーヤーを選ぶ .......... 146
Flash プレイヤー .......... 146
MediaPlayer プレイヤー .......... 147
Silverlight プレイヤー .......... 147
Flash（.swf）ビデオを再生する .......... 148
MediaPlayer（.wmv）ビデオを再生する .......... 150
Silverlight ビデオを再生する .......... 152
その他のリソース .......... 153

第１１章　　Web サイトに電子メールを追加する ......154
Web サイトから電子メール メッセージを送信する ......154
電子メールを使ってファイルを送信する ......157
その他のリソース ......159
第１２章　　Web サイトに検索を追加する ......160
Web サイトを検索する ......160
その他のリソース ......163
第１３章　　Web サイトにソーシャル ネットワーキングを追加する ......164
ソーシャル ネットワーキング サイトに Web サイトをリンクする ......164
Twitter フィードを追加する ......165
Gravatar イメージをレンダリングする ......167
XBOX ゲーマー カードを表示する ......168
Facebook の"Like" ボタンを表示する ......169
その他のリソース ......171
第１４章　トラフィックの解析 ......172
ビジター情報の追跡（解析） ......172
第１５章　　Web サイトのパフォーマンスを改良するためのキャッシング ......175
Web サイトの応答を改良するためのキャッシング ......175
その他のリソース ......177
第１６章　セキュリティとメンバーシップの追加 ......178
Web サイト メンバーシップの紹介 ......178
登録とログインページを持つ Web サイトの作成 ......179
メンバー限定ページの作成 ......183
ユーザー（ロール）のグループに対するセキュリティの作成 ......184
パスワード変更ページの作成 ......186
ユーザーに新しいパスワードを生成させる ......187
Web サイトに参加する自動プログラムを防止する ......192
その他のリソース ......194
第１７章　デバッグの紹介 ......195
サーバー情報を表示するために ServerInfo ヘルパーを使う ......195
ページの値を表示するために出力式を埋め込む ......197
オブジェクトの値を表示するために ObjectInfo ヘルパーを使う ......200
デバッグ ツールを使う ......202
Internet Explorer Developer Tools ......202
Firebug ......203
その他のリソース ......204
第１８章　サイト全体の動作をカスタマイズする ......205
Web サイトのスタートアップ コードを追加する ......205

Web サイトのグローバル値を設定する ........................................................................206
ヘルパーの値を設定する ........................................................................207
フォルダーのファイルの前後にコードを実行する ........................................................................209
フォルダー中のすべてのページのために初期化コードを実行する ........................................................................210
エラーを処理するために_PageStart.cshtml を使う ........................................................................212
フォルダーへのアクセスを制限するために_PageStart.cshtml を使う ........................................................................213
読みやすく検索しやすい URL を作成する ........................................................................214
ルーティングについて ........................................................................214
どのようにルーティングが働くか ........................................................................215
その他のリソース ........................................................................216
付録 – ASP.NET クイック API リファレンス ........................................................................217
クラス ........................................................................217
データ ........................................................................223
ヘルパー ........................................................................224
付録 - ASP.NET Web ページ Visual Basic ........................................................................231
8 つのプログラミング Tips ........................................................................231
ページには、@文字を使ってコードを追加する ........................................................................231
Code…End Code でコード ブロックを囲む ........................................................................232
ブロックの中で、各コード文を改行で終わらせる ........................................................................233
値を保存するために変数を使う ........................................................................233
ダブル クォーテーション記号でリテラル文字列値を囲む ........................................................................234
コードは大小文字を区別しない ........................................................................235
コードのほとんどが関係するオブジェクト ........................................................................235
意思決定用のコードを書く ........................................................................236
単純なコード サンプル ........................................................................237
Visual Basic 言語と構文 ........................................................................239
基本的な構文 ........................................................................239
変数 ........................................................................242
データ型の変換とテスト ........................................................................243
演算子 ........................................................................245
コード中でファイルやフォルダー パスを扱う ........................................................................246
条件文と繰り返し ........................................................................247
オブジェクトとコレクション ........................................................................251
パラメーター付きでメソッドを呼び出す ........................................................................254
エラー処理 ........................................................................255
その他のリソース ........................................................................257
付録 – Visual Studio における ASP.NET Web ページのプログラミング ........................................................................258
なぜ Visual Studio を使うのか ........................................................................258

ASP.NET Razor Tools のインストール ....... 258
Visual Studio 用 ASP.NET Razor Tools を使う ....... 259
IntelliSense を使う ....... 260
デバッガーを使う ....... 261
免責 ....... 264

# 第１章 はじめての WebMatrix と ASP.NET Web ページ

本章では、Web 開発者にとって世界で最良の Web 体験を提供する無料の Web 開発技術、Microsoft WebMatrix を紹介します。

ここでは次のことを学びます。

- WebMatrix とは何か
- WebMatrix のインストール方法
- WebMatrix を使った単純な Web サイトの作成方法
- WebMatrix を使ったダイナミックな Web ページの作成方法

## WebMatrix とは何か

WebMatrix は無料で提供される、Web サイトを構築するもっとも容易な方法を提供する Web 開発ツールの軽量なセットです。これには IIS Express（開発用 Web サーバー）、ASP.NET（Web フレームワーク）、SQL Server Compact（組み込みデータベース）が含まれます。また、Web サイト開発を合理化し、人気のあるオープンソースアプリケーションから Web サイトをはじめやすくする単純なツールも含みます。WebMatrix を使うスキルや開発したコードは、スムーズに Visual Studio や SQL Server に移行できます。

WebMatrix を使って作成した Web ページは動的、つまりコンテンツやスタイルをユーザー入力や、データベース情報のような他の情報に基づいて置き換えることができます。動的な Web ページをプログラムするために、Razor 構文や、C#または Visual Basic 言語とともに ASP.NET を使います。

すでに好みのプログラミングツールを持っていれば、ASP.NET を使った Web サイトを作成するために、WebMatrix ツールを試すことも、独自のツールを使うこともできます。

## WebMatrix のインストール

WebMatrix をインストールするために、Microsoft Web プラットフォーム インストーラー（Web PI）が使えます。これは、Web 関連技術のインストールや設定を容易にする無料のアプリケーションです。

1. Web プラットフォーム インストーラーを持っていない場合は、以下の URL からダウンロードしてください。
   http://go.microsoft.com/fwlink/?LinkID=205867

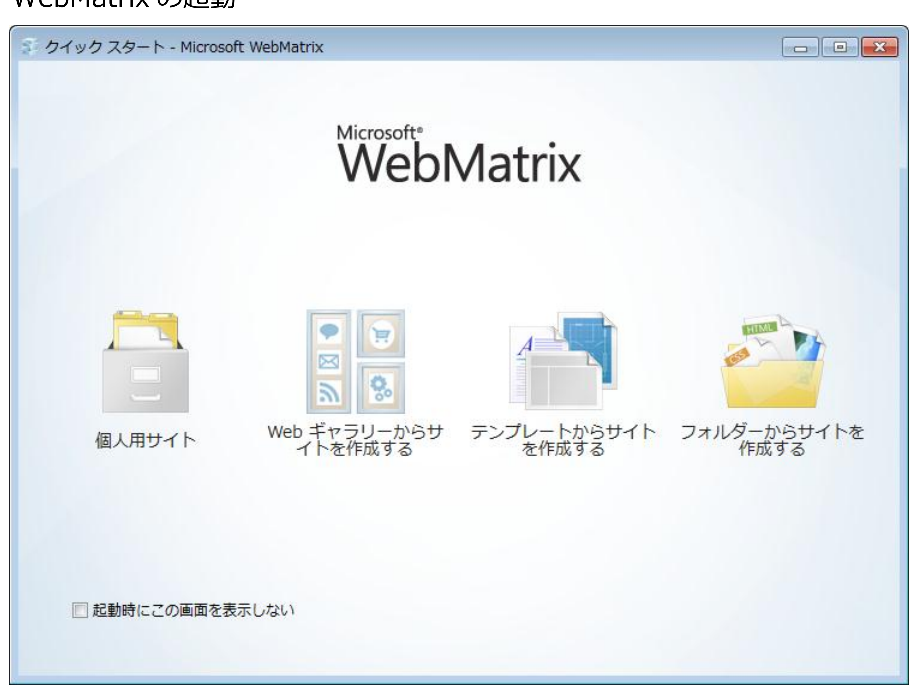
2. インストーラーを実行して、WebMatrix をインストールしてください。

**注意**　もし、ベータ版をインストールしている場合は、Web プラットフォーム インストーラーは、WebMatrix 1.0 にアップグレードします。ただし、WebMatrix を最初に開いたときに、ベータ版で作成した Web サイトが「個人用サイト」リストには表示されないかもしれません。以前作成したサイトを開くためには、「フォルダーからサイトを作成する」アイコンをクリックして、サイトを閲覧して開いてください。次に WebMatrix を開いた時には、そのサイトが「個人用サイト」リストに表示されます。

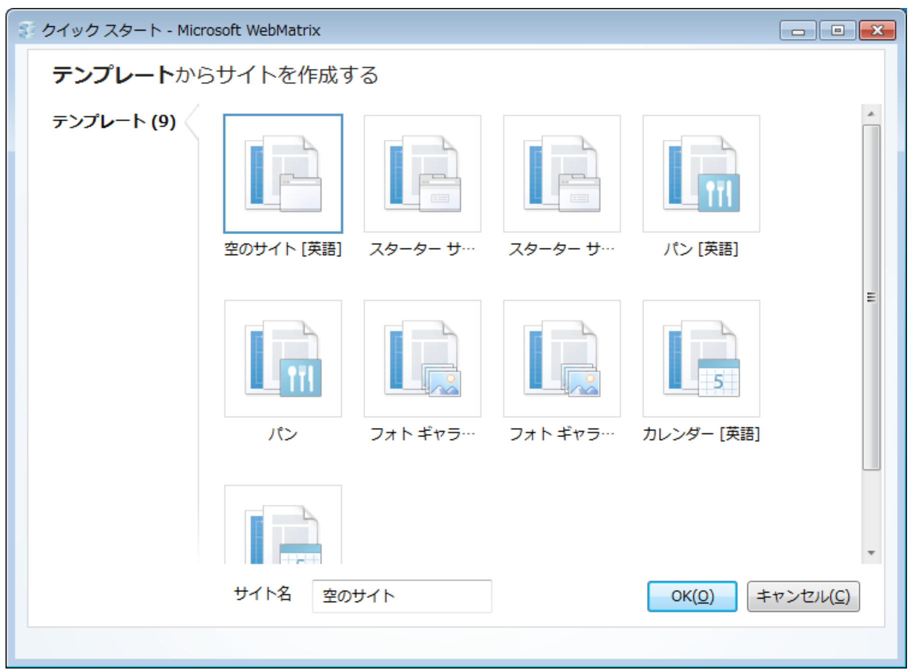
## はじめての WebMatrix

はじめに、新しい Web サイトと単純な Web ページを作成します。

1. WebMatrix の起動

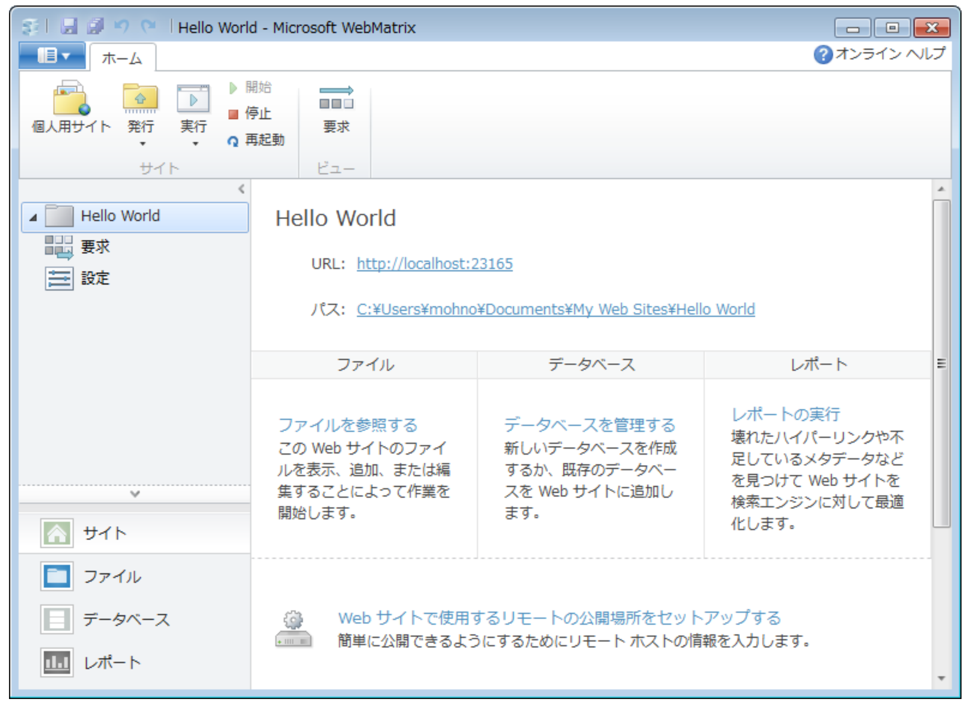


2. 「テンプレートからサイトを作成する」をクリックします。テンプレートには、いくつかのタイプの Web サイト用にあらかじめ作成されたファイルとページが含まれています。



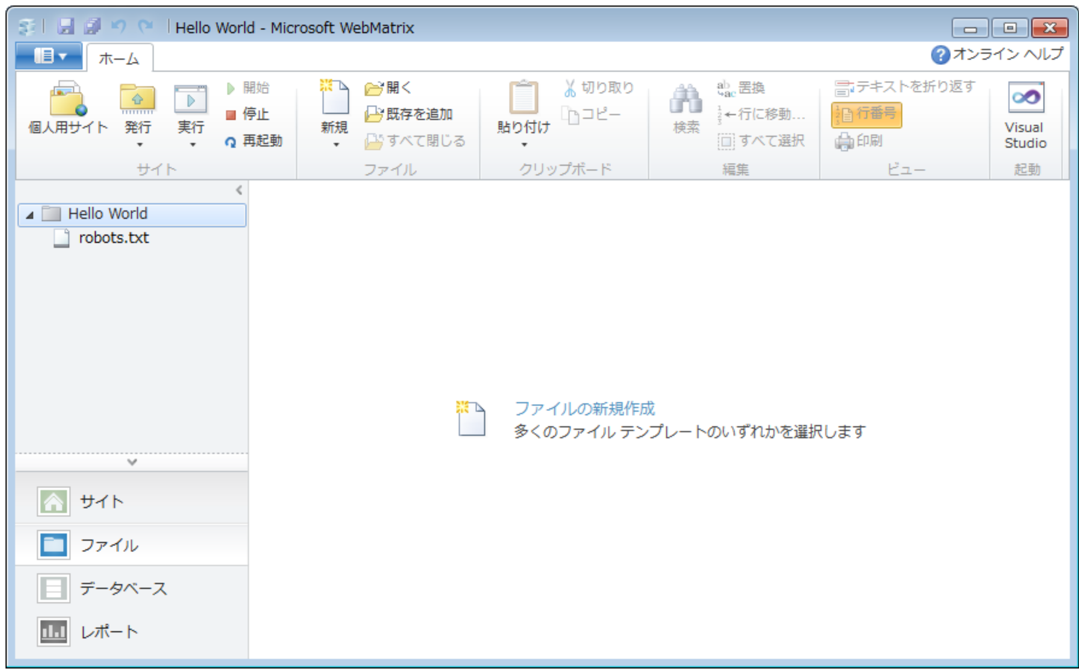

3. 「空のサイト[英語]」を選んで、新しいサイト名として「Hello World」と名前を付けます。
4. [OK] をクリックします。WebMatrix は新しいサイトを作成して、開きます。

上部には、Microsoft Office 2010 のようなクイック アクセス ツールバーとリボンが表示されています。左下部には、左ペインの上に何を表示するかを切り替えるためのボタンが置かれたワークスペース セレクターがあります。右側はコンテンツ ページで、レポートや編集ファイルなどを表示します。下部には、必要に応じてメッセージを表示する通知用バーがあらわれます。

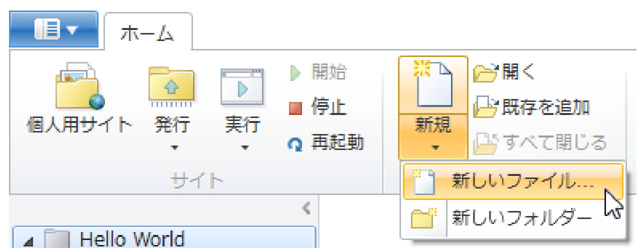


## Web ページの作成

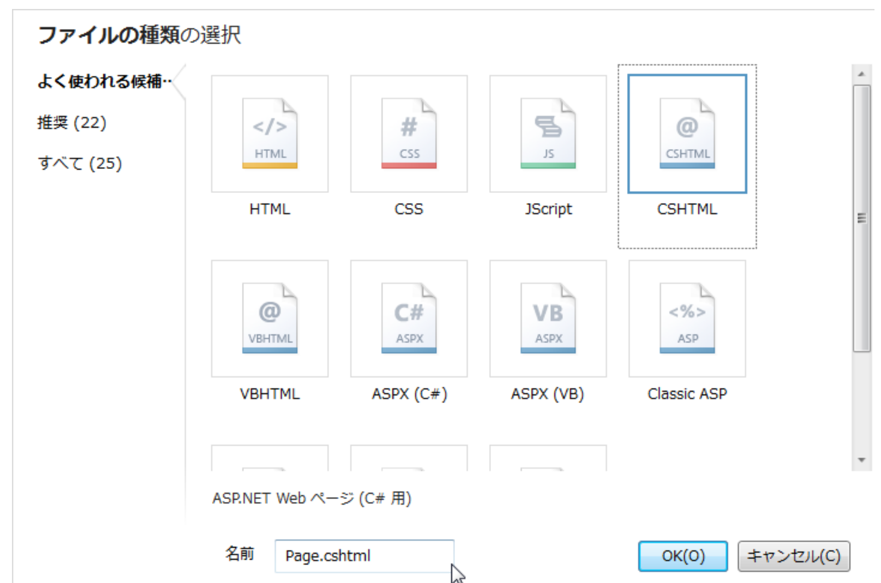
1. WebMatrix で、「ファイル」ワークスペースを選びます。このワークスペースは、ファイルやフォルダーに関する作業のための場所です。左ペインは、サイトのファイル構造を表示します。




2. リボンで、[新規] をクリックし、さらに [新しいファイル] をクリックします。




WebMatrix は、ファイルの種類の一覧を表示します。



3. [CSHTML] を選び、[名前] 入力ボックスで「default.cshtml」と入力します。CSHTML ページは、WebMatrix における特別な種類のページで、HTML や JavaScript コードのような Web ページの通常のコンテンツを含むことも、プログラミングされた Web ページのためのコードを含むこともできます。（CSHTML ファイルの詳細は、後述します。）
4. [OK] をクリックします。WebMatrix は次のようなページを作成し、エディターで開きます。

default.cshtml ×

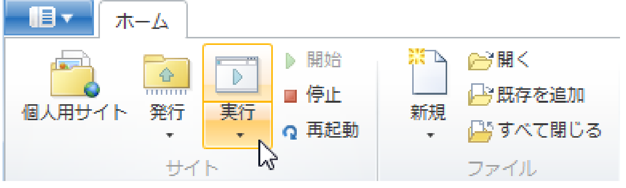
```
1  <!DOCTYPE html>
2
3  <html lang="en">
4      <head>
5          <meta charset="utf-8" />
6          <title></title>
7      </head>
8      <body>
9
10     </body>
11 </html>
12
```

5. このページに、次のようにタイトル、ヘッダー、パラグラフ コンテンツを追加します。

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Hello World Page</title>
    </head>
    <body>
    <h1>Hello World Page</h1>
    <p>Hello World!</p>
    </body>
</html>
```

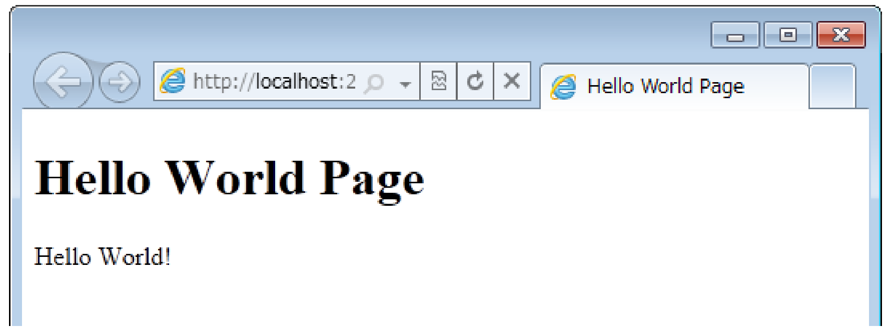
6. クイック アクセス ツールバーで、[保存] をクリックします。

7. リボンで、[実行] ボタンをクリックします。




**注意** [実行]する前に、「ファイル」ワークスペースのナビゲーションペインで選択されているのが、実行したい Web ページであるかどうかを確認してください。WebMatrix は、異なるページを編集しているときでも、選択されたページを実行します。どのページも選択されていないときは、WebMatrix はサイトのデフォルト ページ（default.cshtml）を実行しようとし、デフォルト ページがない場合は、ブラウザーはエラーを表示します。

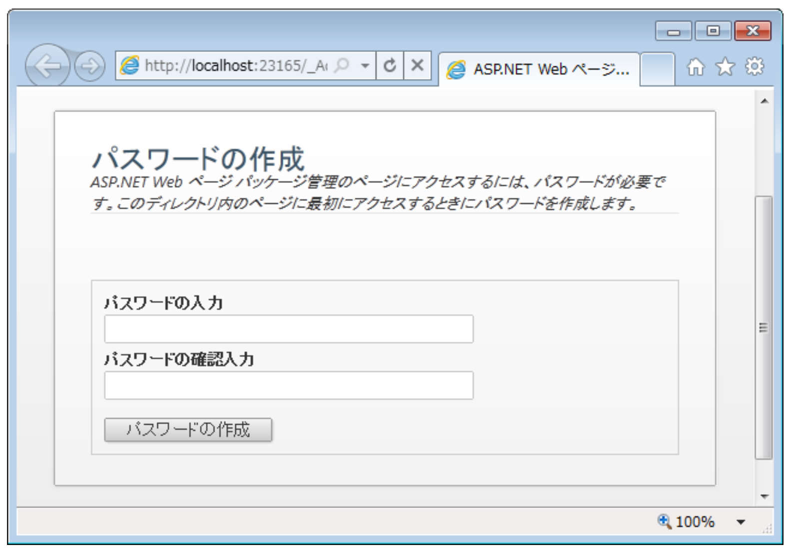
WebMatrix は、コンピューター上でページをテストできるよう Web サーバー（IIS Express）を起動します。ページは、既定のブラウザーで表示されます。




## 管理ツールを使ったヘルパーのインストール

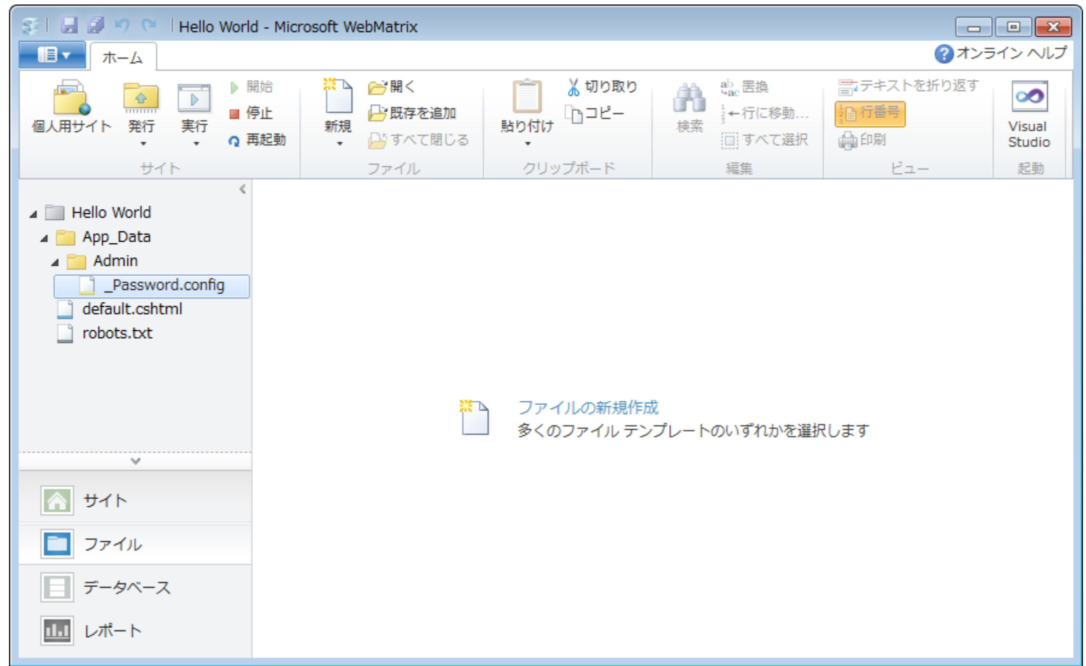
WebMatrix のインストールと、サイトを作りました。ここで、ASP.NET Web ページの管理ツールとヘルパーをインストールするためのパッケージ マネージャーについて学んでみましょう。WebMatrix には、汎用的なプログラミング処理を単純化するヘルパー（コンポーネント）が含まれており、この入門書を通じて使います。（いくつかのヘルパーはすでに WebMatrix に含まれていますが、他のものは必要に応じてインストールします。） 使えるヘルパーは附録に一覧が記載されています。以下の手続きは、管理ツールを使って ASP.NET Web ヘルパー ライブラリをインストールする手順を示したものです。この入門書や同じシリーズの他の入門書でこうしたヘルパーが使われます。

1. WebMatrix で、「サイト」ワークスペースをクリックします。
2. コンテンツ ペインで、「ASP.NET Web ページの管理」をクリックします。これにより、ブラウザーに管理ページが読み込まれます。最初に呼び出すときは、管理ページにログインするためのパスワードを作成するよう求められます。
3. パスワードを作成します。

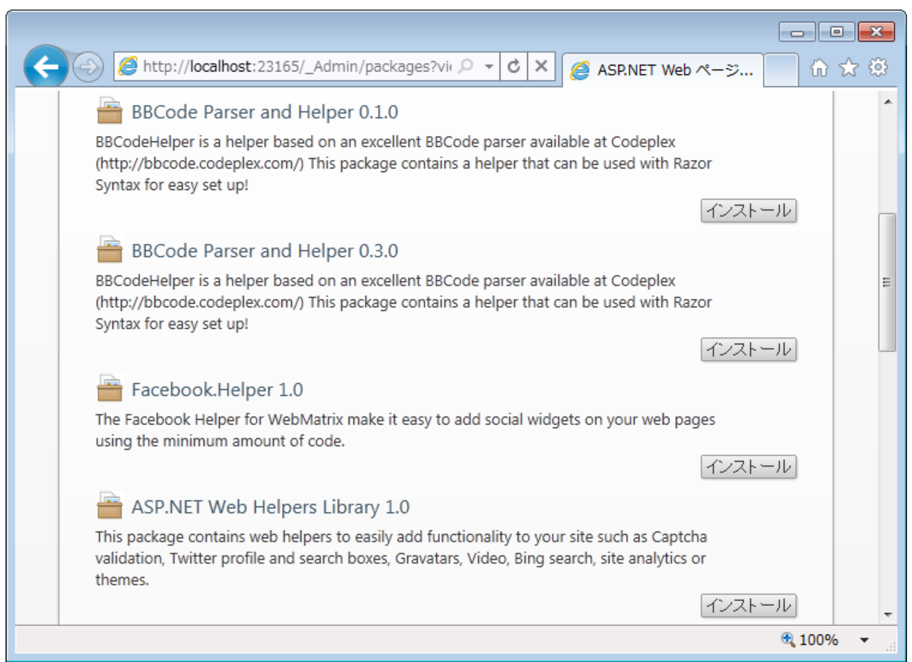



[パスワードの作成] をクリックすると、以下の画面のようなセキュリティ チェック ページが表示され、セキュリティのためにパスワード ファイルの名前を変更するように促されます。これは最初に呼び出したときに表示されるページで、まだファイル名は変更しません。次のステップに進んで、本書の指示に従ってください。

4. ブラウザーでセキュリティ チェック ページを表示したまま、WebMatrix に戻り、「ファイル」ワークスペースをクリックします。
5. 「Hello World」フォルダーを右クリックして、[最新の情報に更新] メニューを選びます。すると、ファイルとフォルダーの一覧に「App_Data」フォルダーが表示されます。これを開くと「Admin」フォルダーが表示されます。「./App_Data/Admin/」フォルダーには、新たに作成されたパスワード ファイル(_Password.config)が表示されます。以下の図は更新されたファイル構造とパスワード ファイルが選択された様子を示しています。

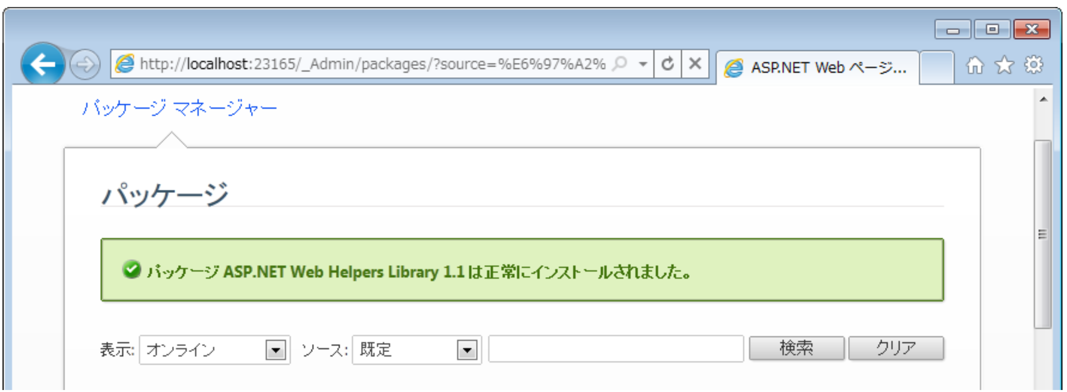



6. ファイル名から先頭のアンダースコア(_)文字を削除して、「Password.config」に変更します。
7. ブラウザーのセキュリティ チェック ページに戻り、パスワード ファイルの名前を変更するメッセージの最後の方にある「ここをクリック」リンクをクリックします。
8. 作成したパスワードを使って、管理ページにログインします。このページは、アドオン パッケージの一覧を含むパッケージ マネージャーを表示します。




他のフィード場所を表示したい場合は、[パッケージ ソースの管理] リンクを使って、フィードを追加、変更、削除できます。

9. 「ASP.NET Web Helpers Library 1.1」パッケージを探して、[インストール] ボタンを押し、指示に従ってこのパッケージをインストールしてください。「ASP.NET Web Helpers Library」をインストールすると、パッケージ マネージャーは合わせて「FaceBook.Helper 1.0 Library」もインストールします。このチュートリアルや後述において、これらのライブラリにあるヘルパーを使います。パッケージがインストールされると、パッケージマネージャーは次のように表示します。




このページでは、パッケージをアンインストールできます。また、新しいバージョンが使えるようになったときにパッケージを更新するために、このページを使うこともできます。「表示」ドロップダウン リストで、「インストール済み」を選ぶとインストールされているパッケージを表示し、「更新プログラム」を選ぶとインストール済みのパッケージで更新版があるものが表示されます。

次のセクションでは、動的なページを作成するために default.cshtml ページへコードを追加することが、いかに容易であるかを示します。

## ASP.NET Web ページでコードを使う

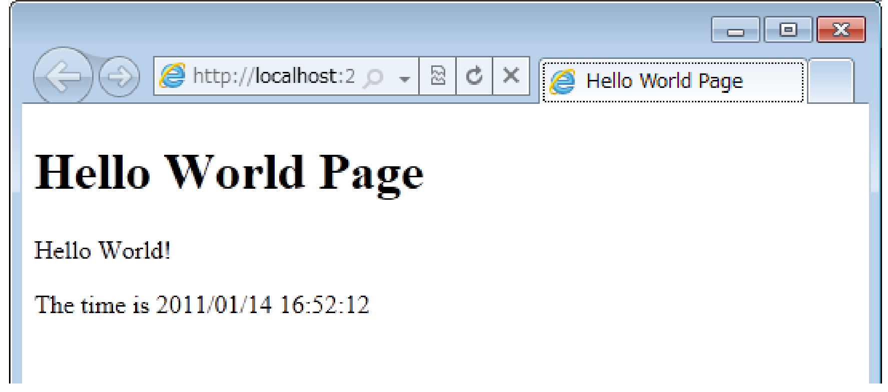
ここでは、ページ上にサーバーの日付と時間を表示する簡単なコードを使うページを作成します。この例は、ASP.NET Web ページの HTML 中にコードを埋め込む Razor 構文を紹介しています。(詳細は、次の章で説明します。) このコードは、本章で前述したヘルパーのひとつを使っています。

1. default.cshtml を開きます。
2. 以下の例のように、ページ中にマークアップを追加します。

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Hello World Page</title>
    </head>
    <body>
    <h1>Hello World Page</h1>
    <p>Hello World!</p>
    <p>The time is @DateTime.Now</p>
    </body>
</html>
```

このページは、通常の HTML マークアップに加えて、@文字で示される ASP.NET プログラムコードを含んでいます。

3. ページを保存して、ブラウザーを実行します。ページ上に現在の日付と時刻が表示されます。




ここで追加した 1 行が、サーバー上の現在時刻を決定し、表示用に書式化し、ブラウザーに送るというすべての作業をしてくれます。（オプションで書式を指定することもできます。これは、ただのデフォルトです。）

より複雑なことを考えるため、選択した Twitter ユーザーのツイートをスクロール リストに表示してみましょう。このために前述のヘルパーが使えます。ヘルパーは、汎用的な処理を単純化してくれるコンポーネントです。ここでは、処理すべきことは Twitter フィードを取得して表示するだけです。

1. 新しい CSHTML ファイルを作成し、TwitterFeed.cshtml という名前にします。
2. ページ上で、既存のコードを以下のコードで置き換えます。

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Twitter Feed</title>
    </head>
    <body>
        <h1>Twitter Feed</h1>
        <form action="" method="POST">
        <div>
            Enter the name of another Twitter feed to display:
             
            <input type="text" name="TwitterUser" value=""/>
             
            <input type="submit" value="Submit" />
        </div>
        <div>
            @if (Request["TwitterUser"].IsEmpty()) {
                @Twitter.Search("microsoft")
            }
            else {
                @Twitter.Profile(Request["TwitterUser"])
            }
        </div>
    </form>
    </body>
</html>
```

この HTML は、ユーザー名を入力するためのテキストボックスと、[Submit] ボタンと、[Submit] ボタンを表示します。これらは最初の <div> タグの間に含まれています。

2 つめの <div> タグの間には、いくらかのコードがあります。（以前示したとおり、ASP.NET Web ページにおけるコードを印付けるには、@文字を使います。） このページが最初に表示されたか、ユーザーがテキストボックスを空にしたまま[Submit] をクリックしたかどうかは、条件式 Request["TwitterUser"].IsEmpty() が真かどうかで判断します。ユーザーが指定されていない場合は、このページは “microsoft” という言葉を検索して、Twitter のフィードを表示します。そうでなければ、テキストボックスに入力されたユーザー名の Twitter フィードを表示します。

3. ブラウザーでページを実行すると、Twitter フィードには、”microsoft”という言葉を持つツイートが表示されます。

4. Twitter ユーザー名を入力して、[Submit] ボタンをクリックしてください。新しいフィードが表示されます。（存在しない名前を入力すると、Twitter フィードは表示されますが、空のままです。）

この例では、WebMatrix の使い方と、Razor 構文を使った単純な ASP.NET コードにより動的な Web ページをどのようにプログラムするかについて示しました。次章では、さまざまな種類の Web サイトの処理についてどのようにコードを使うかを示します。

## Visual Studio で、ASP.NET Razor ページをプログラミングする

ASP.NET Razor ページをプログラムするためには WebMatrix を使う以外に、Visual Studio 2010 を使うこともできます。無料の Visual Web Developer Express を含め、どのエディションでもかまいません。ASP.NET Razor ページを編集するために Visual Studio や Visual Web Developer を使う場合、生産性を高める 2 つのプログラミング ツール、IntelliSense とデバッガーが活用できます。たとえば、HTML 要素を入力する際、IntelliSense は要素が持てる属性の一覧を表示し、その属性にどんな値を設定できるかを表示します。IntelliSense は、HTML、JavaScript、C#、Visual Basic（ASP.NET Razor ページで使うプログラミング言語）のために機能します。

デバッガーは、実行中のプログラムを停止できます。変数の値を調べたり、プログラムを 1 行ずつステップ実行して実行の様子を確かめたりすることができます。

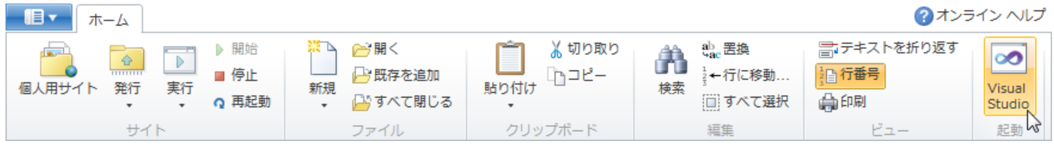
Visual Studio で ASP.NET Razor ページを使う場合は、コンピューターに以下のソフトウェアをインストールする必要があります。

- Visual Studio 2010 または Visual Web Developer 2010 Express
- ASP.NET MVC 3 RTM

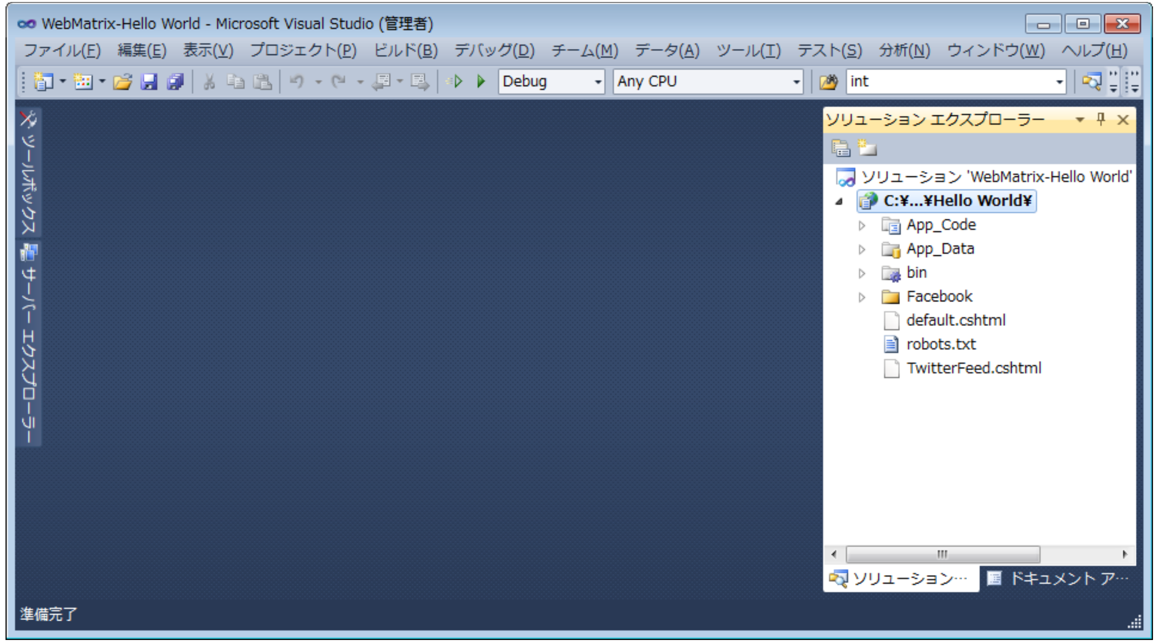
**注意**　Visual Web Developer 2010 Express と ASP.NET MVC 3 は、どちらも Web Platform インストーラーを使ってインストールできます。

Visual Studio をインストール済みのときは、WebMatrix で Web サイトを編集するときに、IntelliSense とデバッガーの利点を活用するために、Visual Studio 中でサイトを起動できます。

1. この章で作成したサイトを開き、「ファイル」ワークスペースをクリックします。
2. リボンで、[Visual Studio 起動] ボタンをクリックします。




Visual Studio でサイトを開いたら、「ソリューション エクスプローラー」ペインに Visual Studio におけるサイトの構造が表示されます。以下の図は、Visual Studio 2010 で Web サイトが開かれた様子です。




Visual Studio で、ASP.NET Razor ページのために IntelliSense とデバッガーを使う方法の概要については、「付録 – Visual Studio における ASP.NET Web ページのプログラミング」を参照してください。

**好みのテキスト エディターを使って ASP.NET ページを作成し、テストするには**

ASP.NET Web ページを作成したり、テストするために WebMatrix エディターを使う必要はありません。ページを作成するためには、メモ帳のような任意のテキスト エディターが使えます。ファイル名の拡張子として.cshtml（Visual Basic を使いたいときは.vbhtml）を使ってページを保存してください。

.cshtml ページをテストするもっとも簡単な方法は、WebMatrix の ［実行］ ボタンを使って Web サーバー（IIS Express）を起動することです。しかし、WebMatrix ツールを使いたくないのであれば、コマンドラインから Web サーバーを実行し、特定のポート番号に関連づけることができます。ブラウザーからは、そのポート番号を使って.cshtml ファイルを要求します。

Windows では、管理者としてコマンド プロンプトを開き、以下のフォルダーを変更します。

C:¥Program Files¥IIS Express

ただし、64 ビット システムの場合は、次のフォルダーを使ってください。

C:¥Program Files (x86)¥IIS Express

サイトの実際のパスにおいて、以下のコマンドを使います。

iisexpress.exe /port:35896 /path:C:¥BasicWebSite

ここで、他のプロセスで使われていない限り、どのポート番号を使うかは、あまり問題ではありません。（通常、1024 以上のポート番号は自由に使えます。）

パスの値としては、テストしたい.cshtml ファイルが置かれているパスを使います。

このコマンドを実行したら、ブラウザーを開き、次のように.cshtml ファイルを呼び出します。

http://localhost:35896/default.cshtml

IIS Express のコマンドライン オプションについては、コマンドラインで「iisexpress.exe /?」と入力してください。

# 第２章　Razor 構文を使った ASP.NET Web プログラミングの紹介

本章では、Razor 構文を使った ASP.NET Web ページにおけるプログラミングの概要を解説します。ASP.NET は、Web サーバーで動的な Web ページを運用するためのマイクロソフトの技術です。

ここでは次のことを学びます。

- Razor 構文を使った ASP.NET Web ページのプログラミングをはじめるための上位 8 つのプログラミング Tips
- 本書において必要な基本的なプログラミングの概念
- ASP.NET サーバーコードと Razor 構文に関する全体像

## 8 つのプログラミング Tips

このセクションでは、Razor 構文を使って ASP.NET サーバーコードを記述し始めるために知っておかなければならないいくつかの Tips を列挙しています。

> **注意**　Razor 構文は、本書の全体を通じて C#プログラミング言語を使っています。しかし、Razor 構文は Visual Basic 言語もサポートしており、本書で解説されるすべての内容は Visual Basic でも同じように実現できます。詳細については「付録 – ASP.NET Web ページ Visual Basic」を参照してください。

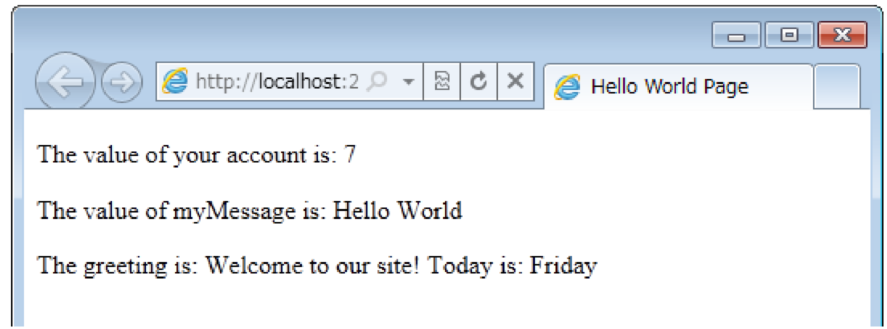
本章の後半では、これらのプログラミング テクニックについての詳細を解説しています。

### 1.　@文字を使ってページにコードを追加する

@文字は、インライン式、単一文のブロック、複文のブロックの始まりを表します。

```
<!-- Single statement blocks -->
@{ var total = 7; }
@{ var myMessage = "Hello World"; }

<!-- Inline expressions -->
<p>The value of your account is: @total </p>
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + " Today is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

このページをブラウザーで実行すると、これらの文は次のように表示されます。




**HTML のエンコード**

前述のように@文字を使ってページ中にコンテンツを表示するとき、ASP.NET は出力を HTML エンコードします。これは、予約された HTML 文字（<、>、&など）を、HTML タグやエンティティとしてでなく、それらの文字が Web ページ中の文字として表示されるコードに置き換えるということです。HTML エンコードをしないと、サーバーからの出力を正しく表示することはできず、セキュリティ リスクを持つことになりかねません。

マークアップのタグとして解釈させるための HTML マークアップを出力したいのであれば（たとえば、パラグラフのための<p></p>やテキストを強調するための<em></em>）、本章の後半にある「コード ブロックにおけるテキスト、マークアップ、コードの連結」セクションを参照してください。

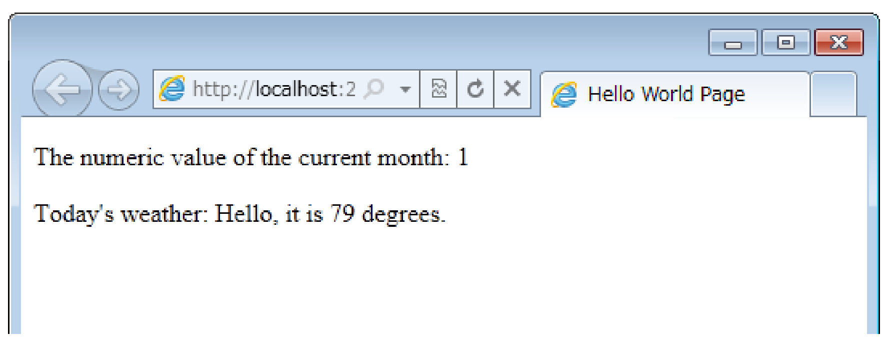
HTML エンコードについての詳細は、「第 4 章　フォームを扱う」を参照してください。

## 2.　中カッコでコードブロックを囲む

コードブロックは 1 つ、または複数の文を含むもので、中カッコで囲まれます。

```
<!-- Single statement block. -->
@{ var theMonth = DateTime.Now.Month; }
<p>The numeric value of the current month: @theMonth</p>

<!-- Multi-statement block. -->
@{
var outsideTemp = 79;
var weatherMessage = "Hello, it is " + outsideTemp + " degrees.";
}
<p>Today's weather: @weatherMessage</p>
```

ブラウザーでは、次のように表示されます。



## 3.　ブロックの中で、各コード文をセミコロンで終わらせる

コードブロックの中では、それぞれの完結するコード文はセミコロンで終わらせます。インライン式は、セミコロンで終わりません。

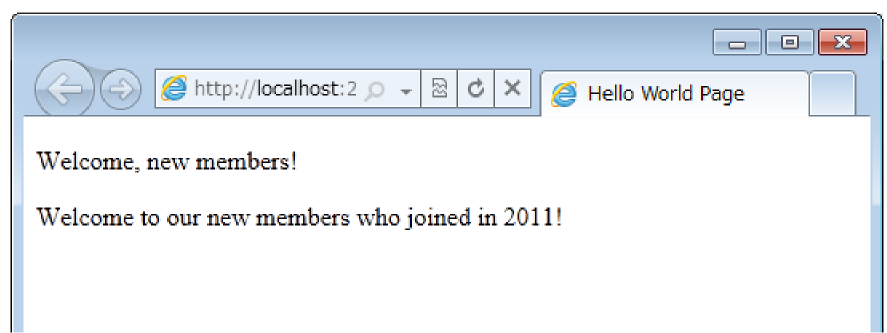
```
<!-- Single-statement block -->
@{ var theMonth = DateTime.Now.Month; }

<!-- Multi-statement block -->
@{
var outsideTemp = 79;
var weatherMessage = "Hello, it is " + outsideTemp + " degrees.";
}

<!-- Inline expression, so no semicolon -->
<p>Today's weather: @weatherMessage</p>
```

## 4.　値を保存するために変数を使う

変数を使って値を保存できます。これには文字列、数値、日付などが含まれます。新しい変数は予約語 var を使って作成します。変数は @ を使って直接ページに挿入できます。

```
<!-- Storing a string -->
@{ var welcomeMessage = "Welcome, new members!"; }
<p>@welcomeMessage</p>

<!-- Storing a date -->
@{ var year = DateTime.Now.Year; }

<!-- Displaying a variable -->
<p>Welcome to our new members who joined in @year!</p>
```

ブラウザーでは、次のように表示されます。




## 5.　ダブル クォーテーション記号でリテラル文字列値を囲む

文字列は、テキストとして扱われる文字のつながりです。文字列を指定するには、ダブル クォーテーション記号で囲みます。

```
@{ var myString = "This is a string literal"; }
```

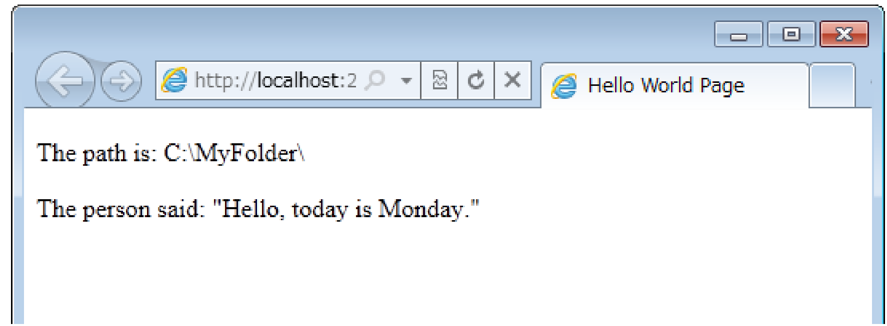
表示したい文字列がバックスラッシュ文字（＼）やダブル クォーテーション記号を含む場合は、@演算子を先頭に付けた置き換え用の文字列リテラルを使います。（C#では、置き換え用の文字列リテラルを使わない限り、＼文字は特別な意味を持ちます。）

```
<!-- Embedding a backslash in a string -->
@{ var myFilePath = @"C:¥MyFolder¥"; }
<p>The path is: @myFilePath</p>
```

ダブル クォーテーション記号を埋め込む場合は、置き換え用の文字列リテラルとクォーテーション記号の繰り返しを使います。

```
<!-- Embedding double quotation marks in a string -->
@{ var myQuote = @"The person said: ""Hello, today is Monday."""; }
<p>@myQuote</p>
```

結果は、次のようにブラウザーに表示されます。




> **注意**　@文字は、C#における置き換え用文字列リテラルの開始と、ASP.NET ページにおけるコードをあらわす両方で使われます。

## 6.　コードは大小文字を区別する

C#では、予約語（var、true、if など）や変数名は大文字と小文字を区別します。以下のコードは、lastName と LastName という 2 つの異なる変数を作成します。

```
@{
var lastName = "Smith";
var LastName = "Jones";
}
```

もし、「var lastName = “Smith”;」と変数を宣言し、ページ中でこの変数を「@LastName」で参照しようとすると、LastName が認識されずにエラーになります。

## 7. コードのほとんどが関係するオブジェクト

オブジェクトは、プログラムで扱うページ、テキストボックス、ファイル、イメージ、Web リクエスト、電子メールのメッセージ、顧客レコード（データベースの行）といったものをあらわします。オブジェクトは、その特徴をあらわすプロパティを持ちます。たとえば、テキストボックス オブジェクトは、Text プロパティを、リクエスト オブジェクトは Url プロパティを、電子メールのメッセージは From プロパティを、顧客オブジェクトは FirstName プロパティなどを持っています。また、オブジェクトは実行できる「動詞」としてメソッドを持ちます。たとえば、ファイル オブジェクトは Save メソッドを、イメージ オブジェクトは Rotate メソッドを、電子メール オブジェクトは Send メソッドなどを持っています。

ページ上のフォーム フィールドの値（テキストボックスなど）、リクエストを発生したブラウザーの種類、ページの URL、ユーザー アイディンティティなどの情報を得るために、しばしば Request オブジェクトを使います。次の例は、Request オブジェクトにアクセスする方法と、サーバー上のページの絶対パスを得るために Request オブジェクトの MapPath メソッドを呼び出す方法を示しています。

```
<table border="1">
<tr>
    <td>Requested URL</td>
    <td>Relative Path</td>
    <td>Full Path</td>
    <td>HTTP Request Type</td>
</tr>
<tr>
    <td>@Request.Url</td>
    <td>@Request.FilePath</td>
    <td>@Request.MapPath(Request.FilePath)</td>
    <td>@Request.RequestType</td>
</tr>
</table>
```

ブラウザーには次のように表示されます。

| Requested URL | Relative Path | Full Path | HTTP Request Type |
|---|---|---|---|
| http://localhost:23165/default.cshtml | /default.cshtml | C:\Users\[illegible]\Documents\My Web Sites\Hello World\default.cshtml | GET |

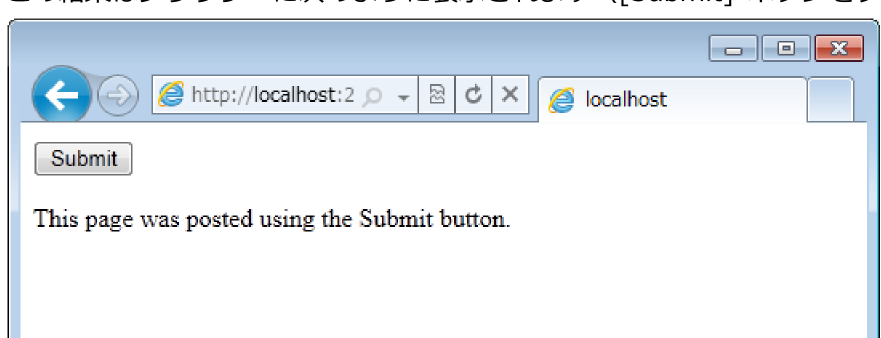
## 8.　意思決定用のコードを書く

動的な Web ページの重要な機能は、条件に応じて何をするかを決められるようにすることです。この処理のもっとも典型的な方法は、if 文（必要に応じて else 文）を使います。

```
@{
    var result = "";
    if(IsPost)
    {
        result = "This page was posted using the Submit button.";
    }
    else
    {
        result = "This was the first request for this page.";
    }
}
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
<body>
<form method="POST" action="" >
    <input type="Submit" name="Submit" value="Submit"/>
    <p>@result</p>
</form>
</body>
</html>
```

「if(IsPost)」という文は、「if(IsPost == true)」という記述の短縮版です。If 文には、条件を判断するいくつかの方法やコードブロックの繰り返しなどがあります。これらは本章で後述します。

この結果はブラウザーに次のように表示されます（[Submit] ボタンをクリックした後）。




**HTTP の GET と POST メソッドと IsPost プロパティ**

Web ページで使われるプロトコル（HTTP）は、サーバーにリクエストを送るために限られた数のメソッド（動詞）しかサポートしていません。もっとも典型的な 2 つが、ページを読むために使われる GET と、ページを送出するための POST です。一般に、ユーザーが最初にページをリクエストするとき、そのページは GET を使ってリクエストされます。ユーザーがフォームを入力して Submit をクリックすると、ブラウザーはサーバーに POST リクエストを送ります。

Web プログラミングでは、ページが GET としてまたは POST としてリクエストされたかどうかを判別することは便利で、ページをどのように処理するかを知ることができます。ASP.NET Web ページでは、IsPost プロパティを使ってリクエストが GET か POST かを区別できます。リクエストが POST であれば、IsPost プロパティは true を返し、フォーム上のテキストボックスの値を読むといった処理を実行できます。本書の多くの例は、IsPost の値に応じたページの処理方法について示しています。

## 単純なコード サンプル

以下の手順は、基本的なプログラミング テクニックを示すページを作成する方法です。この例では、ユーザーに 2 つの数字を入力させ、それらを加算した結果を表示します。

1. エディターで、ファイルを作成し、AddNumbers.cshtml という名前を付けます。
2. 以下のコードとマークアップをページ中にコピーして、ページの既存の内容と置き換えます。

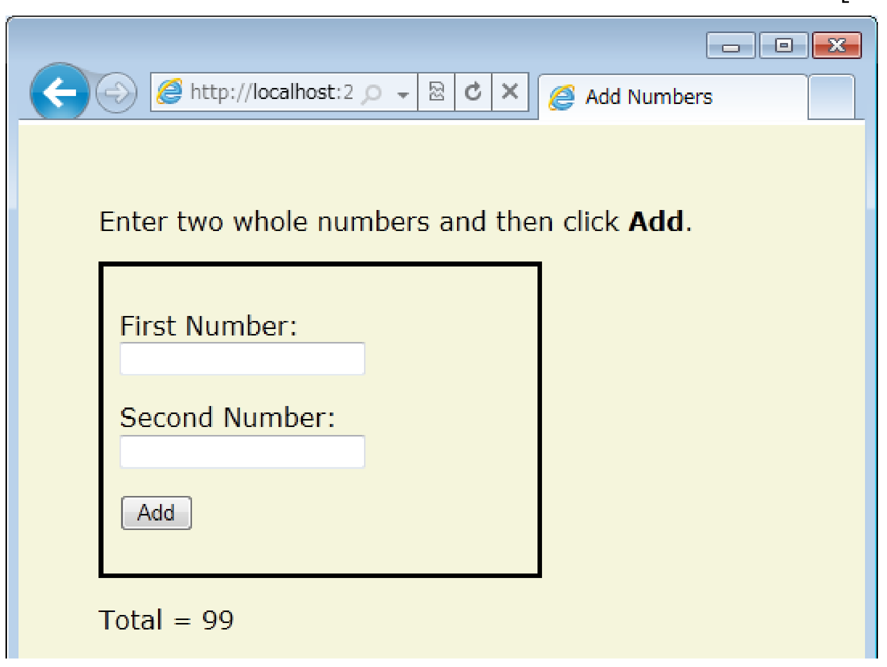
```
@{
    var total = 0;
    var totalMessage = "";
    if(IsPost) {
        // Retrieve the numbers that the user entered.
        var num1 = Request["text1"];
        var num2 = Request["text2"];

        // Convert the entered strings into integers numbers and add.
        total = num1.AsInt() + num2.AsInt();
        totalMessage = "Total = " + total;
    }
}

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Add Numbers</title>
    <meta charset="utf-8" />
    <style type="text/css">
      body {background-color: beige; font-family: Verdana, Arial;
            margin: 50px; }
      form {padding: 10px; border-style: solid; width: 250px;}
    </style>
  </head>
<body>
  <p>Enter two whole numbers and then click <strong>Add</strong>.</p>
  <form action="" method="post">
    <p><label for="text1">First Number:</label>
      <input type="text" name="text1" />
    </p>
    <p><label for="text2">Second Number:</label>
      <input type="text" name="text2" />
    </p>
    <p><input type="submit" value="Add" /></p>
  </form>
```

```
  <p>@totalMessage</p>

</body>
</html>
```

ここでは、以下の点に注意しておいてください。

- @文字は、このページの最初のコードブロックを開始し、ページの最後の方に埋め込まれている totalMessage 変数に先行しています。
- ページの先頭にあるブロックは、中カッコで囲まれています。
- 先頭のブロックで、すべての行はセミコロンで終わっています。
- 変数 total、num1、num2、totalMessage は、いくつかの数字とひとつの文字列を保持します。
- totalMessage 変数に代入されるリテラル文字列値は、ダブル クォーテーション記号で囲まれています。
- コードは大小文字を区別するため、ページの最後の方で totalMessage 変数が使われるとき、この変数の名前は先頭のものと完全に一致しなければなりません。
- 式「num.AsInt() + num2.AsInt()」は、オブジェクトとメソッドがどのように動作するかを示しています。それぞれの変数の AsInt メソッドはユーザーが入力した文字列を数（整数値）に変換するため、算術演算できるようになります。
- <form> タグは、「method="post"」という属性を含みます。これは、ユーザーが [Add] をクリックしたときに、このページが HTTP POST メソッドを使ってサーバーに送られることを指定するものです。ページが送信されるとき、「if(IsPost)」が真だと評価され、条件コードが実行されると、数値を加算した結果が表示されます。

3. ページを保存して、ブラウザーで実行します（実行する前に、「ファイル」ワークスペースでこのページが選択されていることを確認してください）。2 つの数字を入力して [Add] ボタンをクリックします。



## 基本的なプログラミングの概念

「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」や前述の例で示した通り、これまでプログラミングの経験がない人でも、WebMatrix ASP.NET Web ページと Razor 構文を使えば、洗練された機能を持つ動的な Web ページをすばやく作成でき、これらの処理のために多くのコードは必要ありません。

本章では、ASP.NET Web プログラミングの概要を提供しています。これは包括的な試験ではなく、しばしば使うであろうプログラミングの概念を通じたクイックツアーです。それでも、本書の残りの部分を通じて、必要とされるほとんどすべての内容がカバーされています。

ここでは、少しばかりテクニカルな背景を紹介します。

## Razor 構文、サーバーコード、そして ASP.NET

Razor 構文は、Web ページ中にサーバー ベースのコードを埋め込むためのシンプルなプログラミング構文です。Razor 構文を使う Web ページには、クライアント コンテンツとサーバー コードという 2 種類のコンテンツがあります。クライアント コンテンツとは、これまでの Web ページ中で使ってきたもの、HTML マークアップ（要素）、CSS のようなスタイル情報、JavaScript のようなクライアント スクリプト、プレーン テキストです。

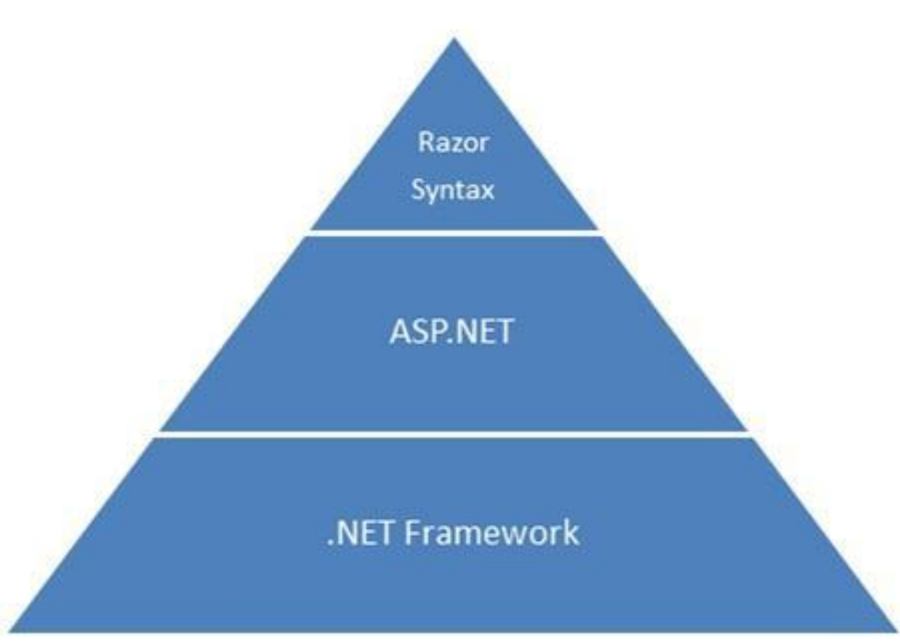
Razor 構文は、こうしたクライアント コンテンツにサーバー コードを追加できるようにします。ページ中にサーバー コードがあれば、サーバーはブラウザーにページを送る前にコードを実行します。コードをサーバー上で実行することで、クライアント コンテンツだけを使うよりも、サーバー ベースのデータベースをアクセスするといった、ずっと複雑なことを処理できるようになります。より重要なことは、サーバー コードは動的にクライアント コンテンツを生成できるということです。HTML マークアップや他のコンテンツをその場に応じて生成し、それを含む静的な HTML ページとともにブラウザーに送出できます。ブラウザーから見れば、サーバー コードで生成されたクライアント コンテンツは他のクライアント コンテンツと何の違いもありません。前述のとおり、必要なサーバー コードは極めてシンプルです。

Razor 構文を含む ASP.NET Web ページは、特別な拡張子（.cshtml または.vbhtml）を持ちます。サーバーは、これらの拡張子を認識して、Razor 構文を持つコードとして実行し、ページをブラウザーに送出します。

### ASP.NET はどこで取り込むのか

Razor 構文は、Microsoft .NET Framework に基づいた ASP.NET と呼ばれるマイクロソフトの技術を元にしています。.NET Framework は、Microsoft が提供する巨大かつ包括的なプログラミング フレームワークで、仮想的にはどんな種類のコンピューター アプリケーションも開発できます。ASP.NET は、.NET Framework において、特に Web アプリケーションを作成するために設計された部分です。世界中の開発者が、大規模かつ高負荷に耐える多くの Web サイトを作成するために ASP.NET を使ってきました。（サイトの URL として、ファイル名の拡張子に.aspx を見つけたら、ASP.NET を使って書かれていることがわかります。）

Razor 構文は、ASP.NET のすべてのパワーを提供しますが、単純化された構文を使うことは、初心者にとって学びやすく、エキスパートにとってより高い生産性をもたらします。この構文は、使い方はシンプルですが、ASP.NET や.NET Framework との強いつながりによって、Web サイトがより洗練されたものになり、大規模なフレームワークのパワーを活用できます。




**クラスとインスタンス**

ASP.NET サーバーコードはオブジェクトを使いますが、これはクラスの考えに基づいています。クラスは、オブジェクトにとっての定義やテンプレートです。たとえば、アプリケーションが Customer クラスを持つ場合、そのクラスは顧客オブジェクトが必要とするプロパティやメソッドを定義します。

アプリケーションが実際の顧客情報を必要とするときは、顧客オブジェクトのインスタンスを作成（インスタンス化）します。個々の顧客は、Customer クラスの独立したインスタンスです。どのインスタンスも同じプロパティとメソッドをサポートしていますが、どの顧客オブジェクトも独立しているため、個々のインスタンスのプロパティ値は通常異なっています。ある顧客オブジェクトでは LastName プロパティが"Smith" であり、別の顧客オブジェクトでは"Jones" かもしれません。

同じように、サイト上の個々の Web ページは Page クラスのインスタンスである Page オブジェクトです。ページ上のボタンは、Button クラスのインスタンスである Button オブジェクトです。どのインスタンスも、独自の特徴を持っていますが、それらはすべてオブジェクトのクラス定義の指定に基づいています。

## 言語と構文

前述の基本的な例で、ASP.NET Web ページを作成する方法や HTML マークアップにサーバー コードを追加する方法を示しました。ここでは、Razor 構文を使って ASP.NET サーバー コードを記述する基本、つまりプログラミング言語の規則について学びます。

プログラミングの経験があれば（とくに C、C++、C#、Visual Basic または JavaScript）、ここで読む内容のほとんどはおなじみのものでしょう。もしかすると、知るべきことは.cshtml ファイル中でサーバーコードにマークアップを追加する方法くらいかもしれません。

## コード ブロックにおけるテキスト、マークアップ、コードの連結

サーバー コード ブロックでは、しばしばページにテキストやマークアップ（またはその両方）を出力したいことがあります。サーバー コード ブロックが、コードではない、そのままレンダリングすべきテキストを含む場合、ASP.NET はコードとテキストを区別できる必要があります。このために、いくつかの方法があります。

- <p></p> や <em></em> のような HTML 要素中にテキストを囲む。

```
@if(IsPost) {
    // This line has all content between matched <p> tags.
    <p>Hello, the time is @DateTime.Now and this page is a postback!</p>
} else {
    // All content between matched tags, followed by server code.
    <p>Hello <em>stranger</em>, today is: <br /> </p> @DateTime.Now
}
```

HTML 要素はテキストや、追加の HTML 要素、サーバー コード式を含むことができます。ASP.NET は、開いた HTML タグを見つけると、その要素とそのコンテンツをすべてレンダリングし、ブラウザーに送ります。（かつ、サーバー コード式を解決します）

- @:演算子か、<text> 要素を使う。@: は、プレーン テキストや不一致の HTML タグを含む単一行のコンテンツを出力します。<text> 要素は、複数行をまとめて出力します。これらのオプションは、出力の一部を HTML 要素にレンダリングしたくない場合に便利です。

```
@if(IsPost) {
    // Plain text followed by an unmatched HTML tag and server code.
    @: The time is: <br /> @DateTime.Now
    // Server code and then plain text, matched tags, and more text.
    @DateTime.Now @:is the <em>current</em> time.
}
```

テキストや不一致の HTML タグを含む複数行を出力したい場合は、各行の先頭に@:を付けるか、<text>要素として行を囲むことができます。@:演算子と同じく、<text>タグは ASP.NET によってテキストコンテンツとして判別され、ページ出力にレンダリングされません。

```
@if(IsPost) {
    // Repeat the previous example, but use <text> tags.
    <text>
    The time is: <br /> @DateTime.Now
    @DateTime.Now is the <em>current</em> time.
    </text>
}
```

```
@{
    var minTemp = 75;
    <text>It is the month of @DateTime.Now.ToString("MMMM"), and
    it's a <em>great</em> day! <br /><p>You can go swimming if it's at
    least @minTemp degrees. </p></text>
}
```

最初の例は、前述の例と同じですが、レンダリングするテキストを囲むために<text>タグを使っています。2 番目の例は、含まれないテキスト、不一致の HTML タグ（<br />）、そしてサーバー コードと一致する HTML タグを持つ 3 行を<text>と</text>タグを使って囲んでいます。個々の行の先頭に@:を付けることもできます。どちらでも動作します。

**注意**　HTML 要素、@:演算子、<text>要素を使って、このセクションで示したテキストを出力する場合、ASP.NET は出力を HTML エンコードしません。（前述したとおり、ASP.NET は、@が先行するサーバー コード式やサーバー コード ブロックの出力を、このセクションで示した特別な場合を除いてエンコードします。）

## 空白

文中の余分な空白（文字列リテラルの外にあるもの）は、文には影響しません。

```
@{ var lastName =      "Smith"; }
```

文中の改行も文には影響しません。可読性のため文の途中で折り返せます。以下の文は等価です。

```
@{ var theName =
"Smith"; }

@{
    var
    personName
    =
    "Smith"
    ;
}
```

ただし、文字列リテラルの途中で行を折り返すことはできません。以下の例は、正しく動作しません。

```
@{ var test = "This is a long
    string"; } // Does not work!
```

## コード（およびマークアップ）のコメント

コメントを使うと、自分自身や他人のためのメモを残しておけます。また、当面の間、ページ中で実行させたくないコードやマークアップの部分を無効（コメントアウト）することもできます。

Razor コードと HTML マークアップのコメントには違いがあります。すべての Razor コードについて、Razor のコメントは、ページがブラウザーに送出される前にサーバー上で処理（削除）されます。このため、Razor のコメント構文ではコード中（さらにマークアップ中）にコメントを残すことができ、ファイルを編集するときに見ることができますが、ユーザーはページソースを確認しても見ることができません。

ASP.NET Razor のコメントでは、コメントは@*で始まり、*@で終わります。コメントは 1 行でも複数行でもかまいません。

```
@* A one-line code comment. *@

@*
    This is a multiline code comment.
    It can continue for any number of lines.
*@
```

以下は、コードブロック中のコメントです。

```
@{
    @* This is a comment. *@
    var theVar = 17;
}
```

以下は、同じコード ブロックで、コードをコメント化したため実行されません。

```
@{
    @* This is a comment. *@
    @* var theVar = 17; *@
}
```

コード ブロック中で、Razor コメント構文を使う代わりに、C#のようなプログラミング言語としてのコメント構文を使うこともできます。

```
@{
    // This is a comment.
    var myVar = 17;
    /* This is a multi-line comment
    that uses C# commenting syntax. */
}
```

C#では、//文字以降が単一行のコメントです。複数行のコメントは/*で始まり、*/で終わります。（Razor コメントと同じく、C#のコメントもブラウザーにはレンダリングされません。）

ご存知かもしれませんが、マークアップでは HTML のコメントを作成できます。

```
<!-- This is a comment. -->
```

HTML コメントは、<!-- 文字で始まり、-->で終わります。HTML コメントは、テキストだけでなく、ページ中で保持したままレンダリングさせたくないどんな HTML マークアップも囲むことができます。HTML コメントはタグやそこに含まれるコンテンツ全体を隠します。

```
<!-- <p>This is my paragraph.</p> -->
```

Razor コメントと違い、HTML コメントはページにレンダリングされ、ユーザーはページソースを確認することで、これを見ることができます。

## 変数

変数は、データを保存するために使う名前付きのオブジェクトです。変数にはどんな名前を付けることもできますが、名前は英字で始まり、スペースや予約記号を含むことはできません。

### 変数とデータ型

変数は、その変数に保存されたデータの種類で示される、特定のデータ型を持ちます。（"Hello world" のような）文字列値を保持する文字列変数や、（3 や 79 のような）整数値を保持する整数変数、（4/12/2010 や March 2009 のような）さまざまな形式の日付値を保持する日付変数が使えます。このほかにも使えるデータ型があります。

ただし、通常は変数の型を指定する必要はありません。ほとんどの場合、ASP.NET が、変数の中のデータの使われ方に基づいて、その型を決定できます。（ときには、本書の例にみられるとおり、型を指定しなければならないこともあります。）

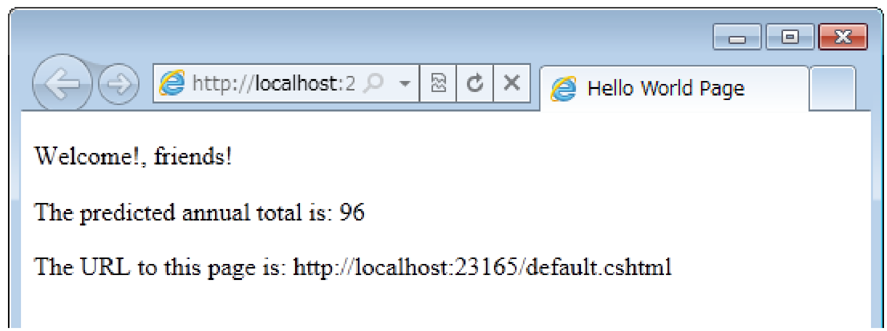
変数は、（型を指定したくない場合は）予約語 var を使うか、型名を使って宣言します。

```
@{
    // Assigning a string to a variable.
    var greeting = "Welcome!";

    // Assigning a number to a variable.
    var theCount = 3;

    // Assigning an expression to a variable.
    var monthlyTotal = theCount + 5;

    // Assigning a date value to a variable.
    var today = DateTime.Today;

    // Assigning the current page's URL to a variable.
    var myPath = this.Request.Url;

    // Declaring variables using explicit data types.
    string name = "Joe";
    int count = 5;
    DateTime tomorrow = DateTime.Now.AddDays(1);
}
```

以下の例は、Web ページにおける典型的な変数の使い方を示しています。

```
@{
    // Embedding the value of a variable into HTML markup.
    <p>@greeting, friends!</p>

    // Using variables as part of an inline expression.
    <p>The predicted annual total is: @( monthlyTotal * 12)</p>

    // Displaying the page URL with a variable.
    <p>The URL to this page is: @myPath</p>
}
```

結果は、次のようにブラウザーに表示されます。




## データ型の変換とテスト

通常、ASP.NET はデータ型を自動的に決定できますが、そうでない場合もあります。このような場合、明示的な変換を実行して、ASP.NET を手助けする必要があります。型を変換する必要がない場合でも、どのような型で処理されるかを確認するためにテストすることは役に立ちます。

もっともよくあるケースは、文字列から整数や日付のような他の型への変換しなければならないものです。以下の例は、文字列から数値へ変換しなければならない典型的な場合です。

```
@{
    var total = 0;

    if(IsPost) {
        // Retrieve the numbers that the user entered.
        var num1 = Request["text1"];
        var num2 = Request["text2"];
        // Convert the entered strings into integers numbers and add.
        total = num1.AsInt() + num2.AsInt();
    }
}
```

規則として、ユーザー入力は文字列として渡されます。ユーザーに数値を入力するよう促して、数字が入力された場合でも、ユーザー入力が送られてコード中で読み取る場合は、そのデータは文字列形式です。このため、文字列を数値に変換しなければなりません。この例で、変換せずに値を計算しようとすると、ASP.NET は 2 つの文字列を加算できないため、以下のようなエラーになります。

型 'string' を型 'int' に暗黙的に変換できません。

値を整数値に変換するには、AsInt メソッドを呼び出します。変換が成功すれば、値を加算できます。

以下の表は、変数に対する一般的な変換とテスト用メソッドの一覧です。

| メソッド | 説明 | 例 |
|---|---|---|
| AsInt(),<br>IsInt() | ("593" のような) 整数をあらわす文字列を整数値に変換する。 | `var myIntNumber = 0;`<br>`var myStringNum = "539";`<br>`if(myStringNum.IsInt()==true){`<br>`    myIntNumber = myStringNum.AsInt();`<br>`}` |
| AsBool(), Is-<br>Bool() | "true" や "false" のような文字列を、論理型に変換する。 | `var myStringBool = "True";`<br>`var myVar = myStringBool.AsBool();` |
| AsFloat(),<br>IsFloat() | "1.3" や "7.439" のような 10 進数をあらわす文字列を浮動小数値に変換する。 | `var myStringFloat = "41.432895";`<br>`var myFloatNum = myStringFloat.AsFloat();` |
| AsDecimal(),<br>IsDecimal() | "1.3" や "7.439" のような 10 進数をあらわす文字列を decimal 型に変換する。(ASP.NET では、decimal 型は浮動小数型よりも高精度。) | `var myStringDec = "10317.425";`<br>`var myDecNum = myStringDec.AsDecimal();` |
| AsDateTime(),<br>IsDateTime() | 日付と時間をあらわす文字列を ASP.NET の DateTime 型に変換する。 | `var myDateString = "12/27/2010";`<br>`var newDate = myDateString.AsDateTime();` |
| ToString() | 文字列以外の型を文字列に変換する。 | `int num1 = 17;`<br>`int num2 = 76;`<br>`// myString is set to 1776`<br>`string myString = num1.ToString() +`<br>`  num2.ToString();` |

## 演算子

演算子は、式の中でどんな種類のコマンドを実行するかを ASP.NET に伝えるための予約語や文字です。C#言語 (およびそれに基づく Razor 構文) は、多くの演算子をサポートしますが、最初に覚えておく必要があるのはわずかです。以下の表は、よく使われる演算子をまとめたものです。

| 演算子 | 説明 | 例 |
|---|---|---|
| +<br>-<br>*<br>/ | 数式における算術演算子。 | `@(5 + 13)`<br>`@{ var netWorth = 150000; }`<br>`@{ var newTotal = netWorth * 2; }`<br>`@(newTotal / 2)` |
| = | 代入。文における右辺の値を左辺のオブジェクトに代入する。 | `var age = 17;` |

<table>
<tr><td>==</td><td>等価。値が等しければ true を返す。（=演算子と==演算子の違いに注意すること。）</td><td>var myNum = 15;<br>if (myNum == 15) {<br>    // Do something.<br>}</td></tr>
<tr><td>!=</td><td>不等。値が等しくなければ true を返す。</td><td>var theNum = 13;<br>if (theNum != 15) {<br>    // Do something.<br>}</td></tr>
<tr><td><<br>><br><=<br>>=</td><td>より小さい、<br>より大きい、<br>より小さいか等しい、<br>より大きいか等しい。</td><td>if (2 < 3) {<br>    // Do something.<br>}<br>var currentCount = 12;<br>if(currentCount >= 12) {<br>    // Do something.<br>}</td></tr>
<tr><td>+</td><td>連結、文字列をつなぐ。ASP.NET は、この演算子と他の加算演算子を式のデータ型に基づいて区別する。</td><td>// The displayed result is "abcdef".<br>@("abc" + "def")</td></tr>
<tr><td>+=<br>-=</td><td>変数から値を加算または減算する。</td><td>int theCount = 0;<br>theCount += 1; // Adds 1 to count</td></tr>
<tr><td>.</td><td>ドット。オブジェクトとプロパティやメソッドを区別するために使う。</td><td>var myUrl = Request.Url;<br>var count = Request["Count"].AsInt();</td></tr>
<tr><td>()</td><td>カッコ。式をグループ化したり、メソッドにパラメーターを渡すために使う。</td><td>@(3 + 7)<br>@Request.MapPath(Request.FilePath);</td></tr>
<tr><td>[]</td><td>角カッコ。配列やコレクションの値をアクセスするために使う。</td><td>var income = Request["AnnualIncome"];</td></tr>
<tr><td>!</td><td>否定。true 値を false に（またはその逆）に逆転させる。通常、false（つまり、true でない）を判断する完結表現として使う。</td><td>bool taskCompleted = false;<br>// Processing.<br>if(!taskCompleted) {<br>    // Continue processing<br>}</td></tr>
<tr><td>&&<br>||</td><td>論理積と論理和。条件を結びつけるために使う。</td><td>bool myTaskCompleted = false;<br>int totalCount = 0;<br>// Processing.<br>if(!myTaskCompleted && totalCount < 12) {<br>    // Continue processing.<br>}</td></tr>
</table>

## コード中でファイルやフォルダー パスを扱う

しばしば、コード中でファイルやフォルダー パスを使うことがあります。以下は、開発コンピューターにおける Web サイトの物理フォルダーの構造の例です。

```
C:¥WebSites¥MyWebSite
    default.cshtml
    datafile.txt
    ¥images
        Logo.jpg
    ¥styles
        Styles.css
```

Web サーバー上で、Web サイトは、そのサイトの物理フォルダーに対応（マップ）する仮想フォルダー構造を持っています。（仮想パスについて思いつくひとつは、ドメイン名に続く URL の部分を構成するものです。） デフォルトでは、仮想フォルダー名は物理フォルダー名と同じです。仮想ルートは、コンピューターの C:ドライブのルートフォルダーをバックスラッシュ（＼）であらわすのと同じように、スラッシュ（/）であらわされます。（仮想フォルダーのパスは、常に/を使います。） 以下に、この構造における Styles.css ファイルの物理パスと仮想パスを示します。

- 物理パス: C:¥WebSites¥MyWebSiteFolder¥styles¥Styles.css
- （仮想ルートパスからの）仮想パス: /styles/Styles.css

コード中でファイルやフォルダーを扱う場合、処理するオブジェクトによって、あるときは物理パスを、あるときは仮想パスを参照する必要があり、ASP.NET はコード中でファイルやフォルダーを処理するための機能として、~演算子、Server.MapPath メソッド、Href メソッドを提供しています。

## ~演算子：仮想ルートを取得する

サーバー コードにおいて、フォルダーやファイルの仮想ルート パスを指定するために、~演算子を使います。これは、Web サイトを異なるフォルダーや場所に移動しても、コード中のパスを変更する必要がないため便利なものです。

```
@{
    var myImagesFolder = "~/images";
    var myStyleSheet = "~/styles/StyleSheet.css";
}
```

## Server.MapPath メソッド：仮想パスから物理パスへ変換する

Server.MapPath メソッドは、（/default.cshtml のような）仮想パスを（C:¥WebSites¥MyWebSiteFolder¥default.cshtml のような）絶対的な物理パスへ変換します。Web サーバー上でテキストファイルを読み書きする場合など、完全な物理パスを必要とする処理のために、このメソッドを使います。（通常、ホスティング サイトのサーバー上に置かれたサイトの絶対的な物理パスはわかりません。） ファイルやフォルダーの仮想パスを渡すと、物理パスを返します。

```
@{
    var dataFilePath = "~/dataFile.txt";
}
<!-- Displays a physical path C:¥Websites¥MyWebSite¥datafile.txt -->
<p>@Server.MapPath(dataFilePath)</p>
```

### Href メソッド: サイト リソースへのパスを作成する

WebPage オブジェクトの Href メソッドは、サーバー コード（~演算子を含む）で作成したパスを、ブラウザーが理解できるパスへ変換します。（~演算子は ASP.NET の演算子なので、ブラウザーは理解できません。） Href メソッドは、イメージ ファイルや他の Web ページ、CSS ファイルのようなリソースへのパスを作成するために使います。たとえば、<img>要素、<link>要素、<a>要素の属性を指定するために、HTML マークアップ中でこのメソッドを使います。

```
@{
    var myImagesFolder = "~/images";
    var myStyleSheet = "~/styles/StyleSheet.css";
}

<!-- This code creates the path "../images/Logo.jpg" in the src attribute. -->
<img src="@Href(myImagesFolder)/Logo.jpg" />

<!-- This produces the same result, using a path with ~ -->
<img src="@Href("~/images")/Logo.jpg" />

<!-- This creates a link to the CSS file. -->
<link rel="stylesheet" type="text/css" href="@Href(myStyleSheet)" />
```

## 条件文と繰り返し

ASP.NET のサーバーコードは、条件に基づく処理を実行したり、指定された数だけ文を繰り返したりするコード（ループを実行するコード）を書くことができます。

### 条件判定

単純な条件判定には if 文を使い、指定した判定が true か false を返します。

```
@{
    var showToday = true;
    if(showToday)
    {
        @DateTime.Today;
    }
}
```

予約語 if はブロックを開始します。実際の判定（条件）は、カッコの中にあり、true か false を返します。この判定が true の場合、中カッコに囲まれた文が実行されます。if 文は、else ブロックを含むことができ、条件が false の場合にそこに指定された文が実行されます。

```
@{
    var showToday = false;
    if(showToday)
    {
        @DateTime.Today;
```

```
    }
    else
    {
        <text>Sorry!</text>
    }
}
```

else if ブロックを使って、複数の条件を追加できます。

```
@{
    var theBalance = 4.99;
    if(theBalance == 0)
    {
        <p>You have a zero balance.</p>
    }
    else if (theBalance > 0 && theBalance <= 5)
    {
        <p>Your balance of $@theBalance is very low.</p>
    }
    else
    {
        <p>Your balance is: $@theBalance</p>
    }
}
```

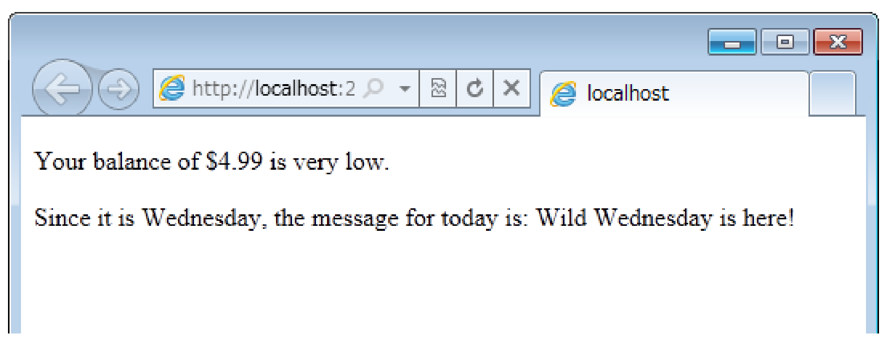
この例では、if ブロックの最初の条件が true でなければ、else if の条件が調べられます。その条件が真なら、else if ブロック中の文が実行されます。どの条件も合わなければ、else ブロックの文が実行されます。else if ブロックはいくつでも追加でき、さらに「そのほかのすべて」という条件として else ブロックを追加できます。

多数の条件を判定するためには、switch ブロックを使えます。

```
@{
    var weekday = "Wednesday";
    var greeting = "";

    switch(weekday)
    {
        case "Monday":
            greeting = "Ok, it's a marvelous Monday";
            break;
        case "Tuesday":
            greeting = "It's a tremendous Tuesday";
            break;
        case "Wednesday":
            greeting = "Wild Wednesday is here!";
            break;
        default:
            greeting = "It's some other day, oh well.";
            break;
    }

    <p>Since it is @weekday, the message for today is: @greeting</p>
}
```

判定する値は、カッコの中にあります（この例では、weekday 変数）。個々の判定値は、case 文に書き、コロン（:）で終わります。テスト値が case 文の値に一致する場合、その case ブロックのコードが実行されます。それぞれの case 文は、break 文で終わります。（C# では、各 case ブロックで break を付け忘れるとコンパイルエラーとなります。） switch ブロックは、しばしば「そのほかのすべて」を示す最後の case として default 文を持ち、どの case も真にならない場合に実行されます。

上記 2 つの条件ブロックは、ブラウザーで次のように表示されます。




## コードのループ

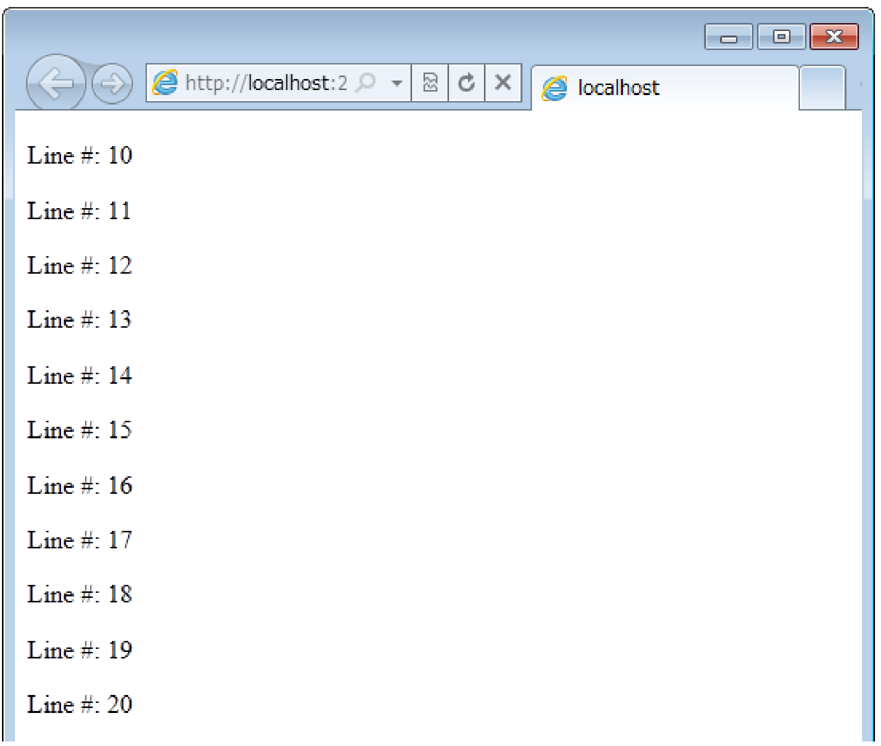
しばしば、同じ文を繰り返し実行する必要が生じます。これはループで実現します。たとえば、データの集まり（コレクション）のそれぞれの項目ごとに同じ文を実行することがあります。ループしたい回数が正確にわかっていれば、for ループを使えます。この種のループは、とくにカウントアップやカウントダウンに役立ちます。

```
@for(var i = 10; i < 21; i++)
{
    <p>Line #: @i</p>
}
```

このループは、予約語 for ではじまり、続いてカッコの中に 3 つの文があり、セミコロンで区切られています。

- カッコの中では、最初の文（var i = 10;）はカウンターを作成して 10 に初期化しています。カウンターの名前は i である必要はなく、変数名として正しいものなら何でも使えます。for ループを実行すると、このカウンターは（※訳注　3 番目の文により）自動的にインクリメントされます。
- 2 番目の文（i < 21;）は、最終的なカウントとどれくらい違うのかを示す条件です。ここでは、20 という最大値まで実行します（つまり、カウンターが 21 より小さい間繰り返し続けるということです）。
- 3 番目の文（i++）は、インクリメント演算子を使い、ループを繰り返すたびにカウンターに 1 を加算する指定を単純化しています。

中カッコの中には、ループのそれぞれの繰り返しで実行するコードがあります。このマークアップは、新しいパラグラフ（<p>要素）を作成し、毎回出力に、i（カウンター）の値を表示する行を追加しています。このページを実行すると、この例は各行が項目番号を示しているテキストを出力する 11 行が作成します。

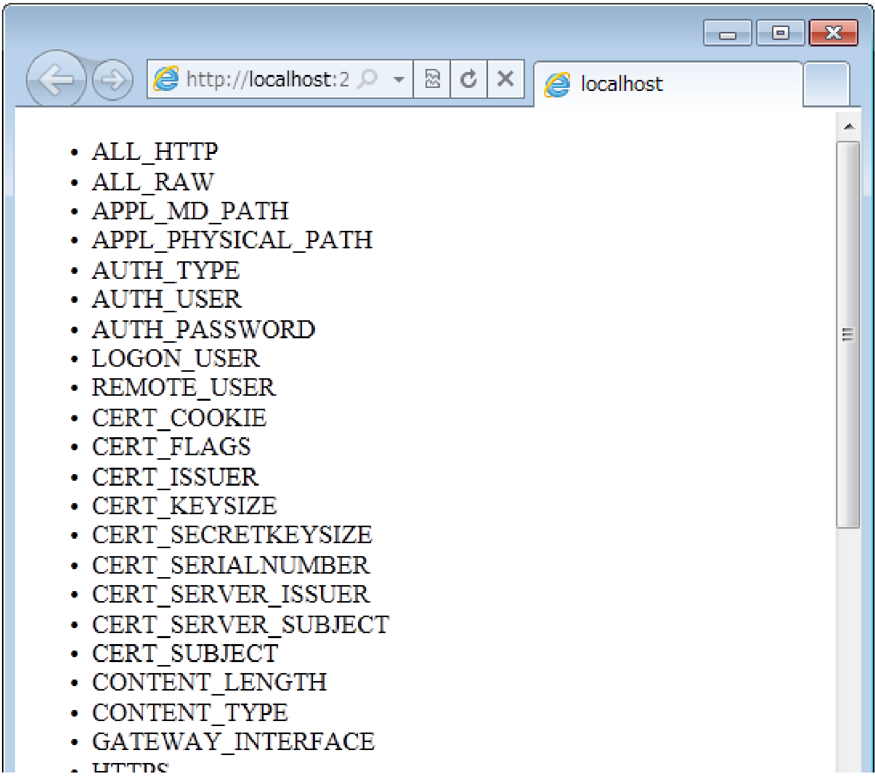



コレクションや配列を処理する場合は、たいてい foreach ループを使います。コレクションは、似たようなオブジェクトの集まりであり、foreach ループはそうしたコレクションの個々の要素に対して処理を適用できます。この種のループは、for ループと違ってカウンターをインクリメントしたり、上限を設定する必要がないので、コレクションを扱うのに便利です。その代り、foreach ループではコレクション全体を通じて最後まで順に処理します。

次の例では、Request.ServerVariables コレクション（Web サーバーに関する情報が含まれているオブジェクト）の項目を返します。ここでは foreach ループを使って、それぞれの要素の名前を、HTML の箇条書きリストの中に新しい<li>要素を作成して表示しています。

```
<ul>
@foreach (var myItem in Request.ServerVariables)
{
    <li>@myItem</li>
}
</ul>
```

予約語 foreach に続いてカッコがあり、コレクションの単一要素をあらわす変数宣言、予約語 in、このループで繰り返し処理したいコレクションが含まれています。foreach ループの本体には、ここで宣言した変数を使って現在の項目にアクセスできます。




より汎用的なループを作成するためには、while 文が使えます。

```
@{
    var countNum = 0;
    while (countNum < 50)
    {
        countNum += 1;
        <p>Line #@countNum: </p>
    }
}
```

while ループは、予約語 while ではじまり、続くカッコには、どれだけループを続けるかを示すかを指定し（ここでは、countNum が 50 未満の間）、その後ろのブロックが繰り返されます。ループは、通常、カウント用の変数やオブジェクトをインクリメント（加算）またはデクリメント（減算）します。この例では、+=演算子がループを実行するたびに countNum に 1 加算しています。（カウントダウン用にループ中で変数をデクリメントするときは、デクリメント演算子-=を使います。）

## オブジェクトとコレクション

ASP.NET Web サイトのほとんどすべてのものは、Web ページ自身を含めてオブジェクトです。このセクションでは、コード中でしばしば処理する重要なオブジェクトについて説明します。

### ページ オブジェクト

ASP.NET におけるもっとも基本的なオブジェクトは、ページです。ページ オブジェクトは、特別なオブジェクトの限定をすることなく、プロパティにアクセスできます。次のコードは、ページの Request オブジェクトを使って、そのページのファイル パスを取得しています。

```
@{
    var path = Request.FilePath;
}
```

現在のページのプロパティやメソッドを参照することを明らかにするために、コード中のページ オブジェクトをあらわす予約語 this を使うこともできます。前述のコード例は、次のようにページをあらわす this を追加できます。

```
@{
    var path = this.Request.FilePath;
}
```

Page オブジェクトのプロパティを使って、以下のような多くの情報を取得できます。

- Request。すでにご覧のとおり、これは現在のリクエストに関する情報のコレクションで、リクエストを発生したブラウザーの種類、ページの URL、ユーザー アイデンティティなどを含みます。
- Response。これは、サーバー コードの実行が完了したときにブラウザーに送出されるレスポンス（ページ）に関する情報のコレクションです。たとえば、このプロパティを使ってレスポンスへの情報を書き込むことができます。

```
@{
    // Access the page's Request object to retrieve the Url.
    var pageUrl = this.Request.Url;
}
<a href="@pageUrl">My page</a>
```

### コレクション オブジェクト（配列とディクショナリ）

コレクションは、同じ型のオブジェクトの集まりで、データベースの Customer オブジェクトのコレクションのようなものを言います。ASP.NET は、Request.Files コレクションのような多くのビルトイン コレクションを持っています。

しばしばコレクションのデータを扱います。2 つの汎用的なコレクションの種類に、配列とディクショナリがあります。配列は、同じ項目のコレクションを保持したい場合、かつそれぞれの項目のために異なる変数を作りたくない場合に役立ちます。

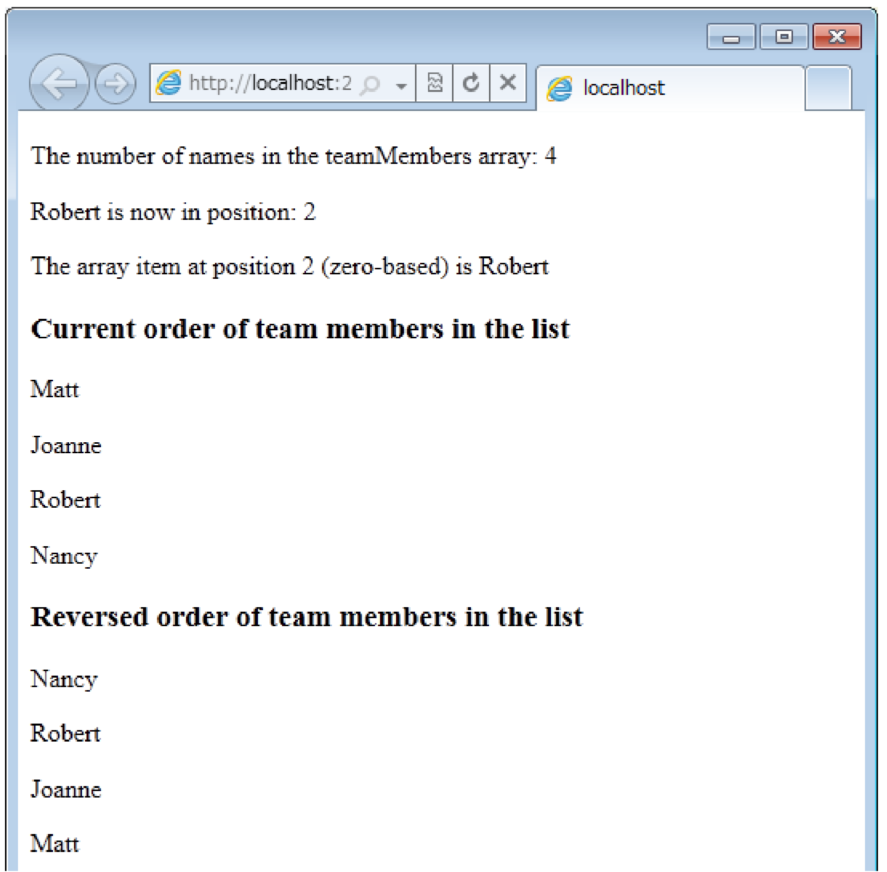
```
@* Array block 1: Declaring a new array using braces. *@
@{
    <h3>Team Members</h3>
    string[] teamMembers = {"Matt", "Joanne", "Robert", "Nancy"};
    foreach (var person in teamMembers)
    {
        <p>@person</p>
    }
}
```

配列を使って、string や int、DateTime のような指定した型を宣言できます。配列を含む変数を指定する場合、宣言には（string[] や int[] のように）角カッコを追加します。配列の要素にアクセスするときは、その位置（インデックス）を使うか、foreach 文を使います。配列のインデックスはゼロ ベース、つまり先頭の要素は 0 番目の位置、2 番目の要素は 1 番目の位置にあります。

```
@{
    string[] teamMembers = {"Matt", "Joanne", "Robert", "Nancy"};
    <p>The number of names in the teamMembers array: @teamMembers.Length </p>
    <p>Robert is now in position: @Array.IndexOf(teamMembers, "Robert")</p>
    <p>The array item at position 2 (zero-based) is @teamMembers[2]</p>
    <h3>Current order of team members in the list</h3>
    foreach (var name in teamMembers)
    {
        <p>@name</p>
    }
    <h3>Reversed order of team members in the list</h3>
    Array.Reverse(teamMembers);
    foreach (var reversedItem in teamMembers)
    {
        <p>@reversedItem</p>
    }
}
```

配列の要素の数は、その Length プロパティを取得することでわかります。配列中の指定した要素の場所を得るには（配列の検索）、Array.IndexOf メソッドを使います。また、配列の要素を逆順にして同じことをすることも（Array.Reverse メソッド）、コンテンツをソートすることも（Array.Sort メソッド）できます。

文字列配列のコードの出力は、ブラウザーに次のように表示されます。




ディクショナリは、キー/値の組み合わせのコレクションで、設定用のキー（または名前）を指定したり、対応する値を取り出したりできます。

```
@{
    var myScores = new Dictionary<string, int>();
    myScores.Add("test1", 71);
    myScores.Add("test2", 82);
    myScores.Add("test3", 100);
    myScores.Add("test4", 59);
}
<p>My score on test 3 is: @myScores["test3"]%</p>
@(myScores["test4"] = 79)
<p>My corrected score on test 4 is: @myScores["test4"]%</p>
```

ディクショナリを作成するためには、予約語 new を使って、新たに作成するディクショナリ オブジェクトを指定します。ディクショナリは、予約語 var を使って変数に代入できます。ディクショナリの項目のデータ型は、山カッコ（<>）を使って指定します。宣言の最後には、カッコのペアを追加しなければなりません。これが新たなディクショナリを作るためのメソッドです。

ディクショナリに新たな項目を追加するには、ディクショナリ変数（ここでは myScores）の Add メソッドを呼び出して、キーと値を指定します。あるいは、キーを指定する角カッコを使い、以下の例のような単純な代入文としても処理できます。

```
myScores["test4"] = 79;
```

ディクショナリから値を取り出すには、角カッコにキーを指定します。

```
var testScoreThree = myScores["test3"];
```

## パラメーター付きでメソッドを呼び出す

本章で前述したように、プログラムで使うオブジェクトはメソッドを持てます。たとえば、Database オブジェクトは Database.Connect メソッドを持ちます。メソッドの多くは、1 つ以上のパラメーターを持ちます。パラメーターは、メソッドが処理を完了できるように渡される値です。たとえば、Request.MapPath メソッドの宣言を見ると、次のように 3 つのパラメーターを受け取ります。

```
public string MapPath(string virtualPath, string baseVirtualDir, bool allowCrossAppMapping);
```

このメソッドは、指定された仮想パスに対応するサーバー上の物理パスを返します。このメソッドの 3 つのパラメーターは、VirtualPath、baseVirtualDir、allowCrossAppMapping です。（宣言においては、パラメーターは受け取るデータのデータ型付きで並べられていることに注意してください。） このメソッドを呼び出すときには、3 つすべてのパラメーターを渡さなければなりません。

Razor 構文は、メソッドにパラメーターを渡すために、位置パラメーターと名前パラメーターの 2 つの選択肢を提供しています。位置パラメーターを使ってメソッドを呼び出すには、メソッド宣言で指定された正確な順番をパラメーターに渡します。（この順序を知るには、通常メソッドのドキュメントを読みます。） 順序にしたがうためには、どのパラメーターもスキップできません。必要に応じて、値を持たない位置パラメーターには空文字列（""）か null を渡します。

以下の例は、Web サイトに scripts という名前のフォルダーがあることを想定しています。コードは、Request.MapPath メソッドを呼び出し、指定された順序で 3 つのパラメーターに値を渡しています。これで、結果としてマップされたパスを表示します。

```
// Pass parameters to a method using positional parameters.
var myPathPositional = Request.MapPath("/scripts", "/", true);
<p>@myPathPositional</p>
```

メソッドが多くのパラメーターを持つ場合は、名前パラメーターを使うことでコードの可読性を維持できます。名前パラメーターを使ってメソッドを呼び出すには、パラメーター名に続いてコロン（:）と値を指定します。名前パラメーターの利点は、好きな順序で値を渡せることです。（不利な点は、メソッド呼び出しが小さく収まらないことです。）

以下の例は、上記と同じメソッドを呼び出していますが、名前パラメーターをつかって値を渡しています。

```
//// Pass parameters to a method using named parameters.
var myPathNamed = Request.MapPath(baseVirtualDir: "/", allowCrossAppMapping: true,
virtualPath: "/scripts");
<p>@myPathNamed</p>
```

ご覧のとおり、パラメーターは異なる順序で渡されています。しかし、前述の例もこの例も、同じ値を返します。

## エラー処理

### Try-Catch 文

コードの中の文には、しばしば制御できない理由によって失敗する者が含まれます。たとえば、次のようなものです。

- コードがファイルを開いたり、作成したり、書き込む場合、さまざまな種類のエラーが発生するおそれがあります。ファイルが存在しない、ロックされている、コードが権限を持っていない、など。
- 同様に、コードがデータベースのレコードを更新しようとすると、権限の問題が起きるおそれがあります。データベースへの接続が落ちてしまったり、保存するデータが不正な場合、など。

プログラミング用語では、こうした状況を「例外」と呼びます。コードが例外に遭遇すると、エラーメッセージを生成（スロー）します。これは、よくても、ユーザーをいらだたせるものです。

コードがこのような状況に遭遇した場合、この種のエラーメッセージを避けるために、try/catch 文を使えます。try 文では、チェックしたいコードを実行します。1 つ以上の catch 文では、発生したものが指定したエラー（指定した例外の型）かどうかを調べます。ここでは、予想できるだけエラーに対応する catch 文を追加できます。

**注意**　try/catch 文で Response.Redirect メソッドを使うことは避けることをお勧めします。ページ中で例外を起こす恐れがあります。

以下の例は、最初のリクエストでテキストファイルを作成し、ユーザーにファイルを開くボタンを表示します。この例は、意図的に誤ったファイル名を使っているため、例外を発生します。起きうる 2 つの例外のためのコードは catch 文を含んでいます。ひとつはファイル名が間違っている場合の FileNotFoundException で、もうひとつは ASP.NET

がフォルダーを発見できない場合に発生する DirectoryNotFoundException です。（すべてが正常に動作するようすを確認するために、この例のコメントを外すこともできます。）

コードが例外を処理しないと、さきほどの画面ショットのようなエラー ページが表示されます。しかし、try/catch セクションは、この種のエラーが発生しても、そのような画面が表示されることを防ぎます。

```
@{
    var dataFilePath = "~/dataFile.txt";
    var fileContents = "";
    var physicalPath = Server.MapPath(dataFilePath);
    var userMessage = "Hello world, the time is " + DateTime.Now;
    var userErrMsg = "";
    var errMsg = "";

    if(IsPost)
    {
        // When the user clicks the "Open File" button and posts
        // the page, try to open the created file for reading.
        try {
            // This code fails because of faulty path to the file.
            fileContents = File.ReadAllText(@"c:¥batafile.txt");

            // This code works. To eliminate error on page,
            // comment the above line of code and uncomment this one.
            //fileContents = File.ReadAllText(physicalPath);
        }
        catch (FileNotFoundException ex) {
            // You can use the exception object for debugging, logging, etc.
            errMsg = ex.Message;
            // Create a friendly error message for users.
            userErrMsg = "A file could not be opened, please contact "
                + "your system administrator.";
        }
        catch (DirectoryNotFoundException ex) {
            // Similar to previous exception.
            errMsg = ex.Message;
            userErrMsg = "A directory was not found, please contact "
                + "your system administrator.";
        }
    }
    else
    {
        // The first time the page is requested, create the text file.
        File.WriteAllText(physicalPath, userMessage);
    }
}

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Try-Catch Statements</title>
    </head>
    <body>

    <form method="POST" action="" >
```

```
        <input type="Submit" name="Submit" value="Open File"/>
    </form>

    <p>@fileContents</p>
    <p>@userErrMsg</p>

    </body>
</html>
```

## その他のリソース

Visual Basic によるプログラミング

- 付録: ASP.NET Web ページ Visual Basic

参照ドキュメント

- ASP.NET
- C#言語

# 第３章　　一貫性のある外観の作成

サイトの Web ページを、より効率的に作成するために、Web サイトのコンテンツのブロック（ヘッダーやフッター）を再利用でき、すべてのページに一貫性のあるレイアウトを作成できます。

ここでは次のことを学びます。

- ヘッダーやフッターのような再利用性のあるコンテンツ ブロックを作成する方法
- レイアウト ページを使ったサイトで、すべてのページに一貫性のある外観を作成する方法
- レイアウト ページに実行時にデータを渡す方法
- シンプルなヘルパーの作成と使用方法

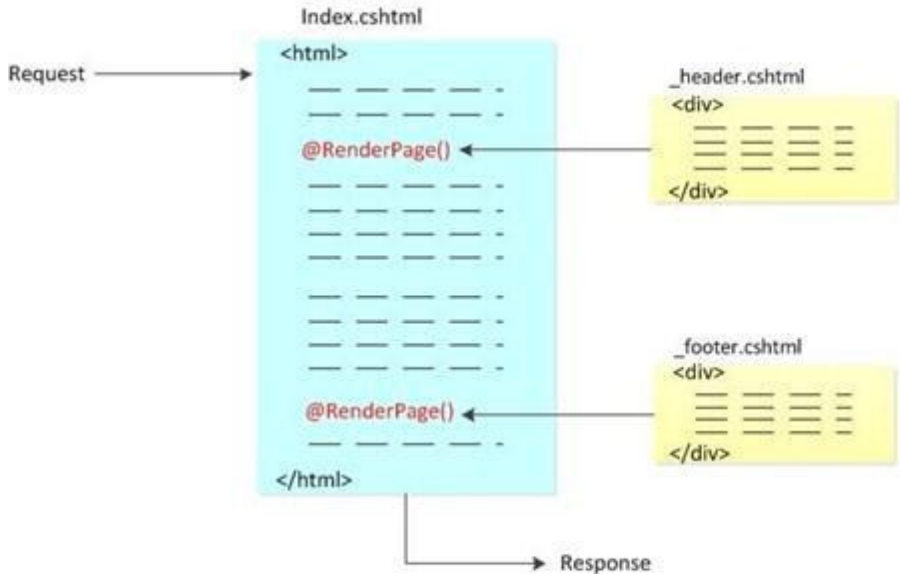
本章では、以下のような ASP.NET の機能が含まれます。

- コンテンツ ブロック。複数のページに挿入される HTML 形式のコンテンツを含むファイル。
- レイアウト ページ。Web サイト上のページで共有できる HTML 形式のコンテンツを含むページ。
- RenderPage、RenderBody、RenderSecion メソッド。ページ要素を挿入する場所を ASP.NET に伝える。
- PageData ディクショナリ。コンテンツ ブロックとレイアウト ページの間でデータを共有できる。

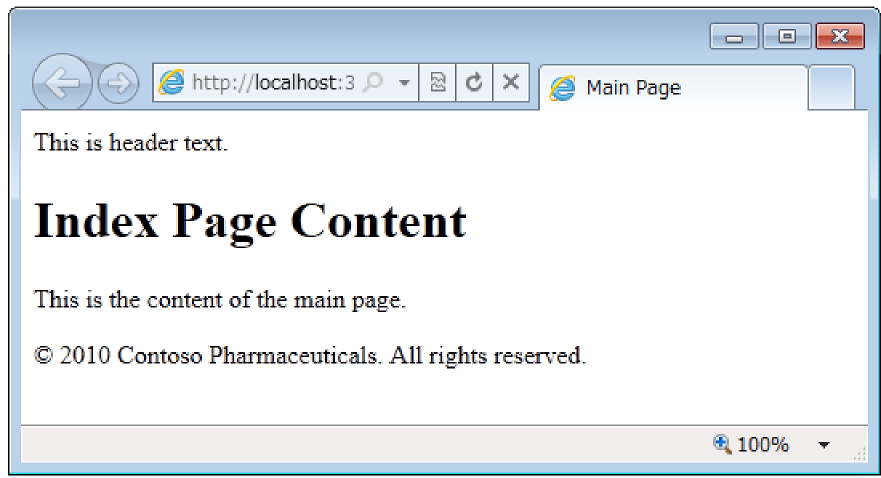
## 再利用できるコンテンツ ブロックの作成

多くの Web サイトは、ヘッダーやフッター、あるいはログイン中のユーザーを表示する領域のような、どのページにも表示されるコンテンツを持っています。ASP.NET では、あたかも標準的な Web ページのように、テキスト、マークアップ、コードを含むコンテンツ ブロックのために、独立したファイルを作成できます。そうすることで、そのサイトの中で情報を表示させたい別のページにコンテンツ ブロックを挿入できます。そうすれば、同じコンテンツをすべてのページにコピー アンド ペーストする必要はありません。このように汎用コンテンツを作成すれば、サイトを更新することも容易になります。コンテンツを変更する必要があれば、単一のファイルだけを更新すれば、その変更は、それを挿入した他のコンテンツに反映されるからです。

以下の図は、どのようにコンテンツ ブロックが動作するかを示しています。ブラウザーが Web サーバーにページを要求する場合、ASP.NET はコンテンツ ブロックをメインページ中で RenderPage メソッドが呼び出された場所に挿入します。その後、最終的な（マージされた）ページがブラウザーに送出されます。




この手順では、異なるファイルに配置された 2 つのコンテンツ ブロック（ヘッダーとフッター）を参照するページを作成しています。サイト上のどのページでも同じコンテンツ ブロックが使えます。完了すると、ページは次のように表示されます。




1. Web サイトのルートフォルダーに、Index.cshtml という名前のファイルを作成します。
   既存のマークアップを以下で置き換えます。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Main Page</title>
    </head>
    <body>

        <h1>Index Page Content</h1>
        <p>This is the content of the main page.</p>

    </body>
</html>
```

2. ルート フォルダーに、Shared という名前のフォルダーを作成します。

| **注意**　一般的な慣習として、Web ページ間で共有するファイルは、Shared という名前のフォルダーに保存します。 |
|---|

3. Shared フォルダーに、_Header.cshtml という名前のファイルを作成します。
4. 既存のマークアップを以下で置き換えます。

```
<div class="header">This is header text.</div>
```

_Header.cshtml というファイル名は、先頭にアンダースコア（_）が付いていることに注意してください。ASP.NET は、名前がアンダースコアからはじまるページをブラウザーに送出しません。これにより、（無意識かそうでないかに関わらず）そうしたページのリクエストを防止します。ユーザーにこうしたページをリクエストさせたいということはないので、コンテンツ ブロックを保持するページの名前にアンダースコアを使うのはよい考えです。これにより、これらは他のページに挿入されるためだけに存在することになります。

5. Shared フォルダーで、_Footer.cshtml という名前のファイルを作成し、既存のコンテンツを以下で置き換えます。

```
<div class="footer">&copy; 2010 Contoso Pharmaceuticals. All rights reserved.
</div>
```

6. Index.cshtml ページで、以下の強調部分のコードを追加します。これらは 2 つの RenderPage メソッドを呼び出しています。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Main Page</title>
    </head>
    <body>

        @RenderPage("/Shared/_Header.cshtml")

        <h1>Index Page Content</h1>
        <p>This is the content of the main page.</p>

        @RenderPage("/Shared/_Footer.cshtml")

    </body>
</html>
```

これは、Web ページ中にコンテンツ ブロックを挿入する方法を示しています。RenderPage メソッドを呼び出して、その場所に挿入したいコンテンツを持つファイル名を渡します。ここでは、_Header.cshtml と _Footer.cshtml ファイルのコンテンツを Index.cshtml ファイルに挿入しています。

7. ブラウザーで Index.cshtml ページを実行します。（「ファイル」ワークスペースで Index.cshtml が選択されていることを確認してください。）
8. ブラウザーで、ページ ソースを表示してください。（たとえば、Internet Explorer ではページを右クリックして、[ソースの表示] を選びます。）

すると、ブラウザーに送られた Web ページのマークアップが、インデックス ページにコンテンツ ブロックが組み合わされたものであることを確認できます。以下の例は、Index.cshtml にレンダリングされたページ ソースです。Index.cshtml に挿入した RenderPage の呼び出しが、ヘッダーとフッターの実際のコンテンツに置き換えられています。

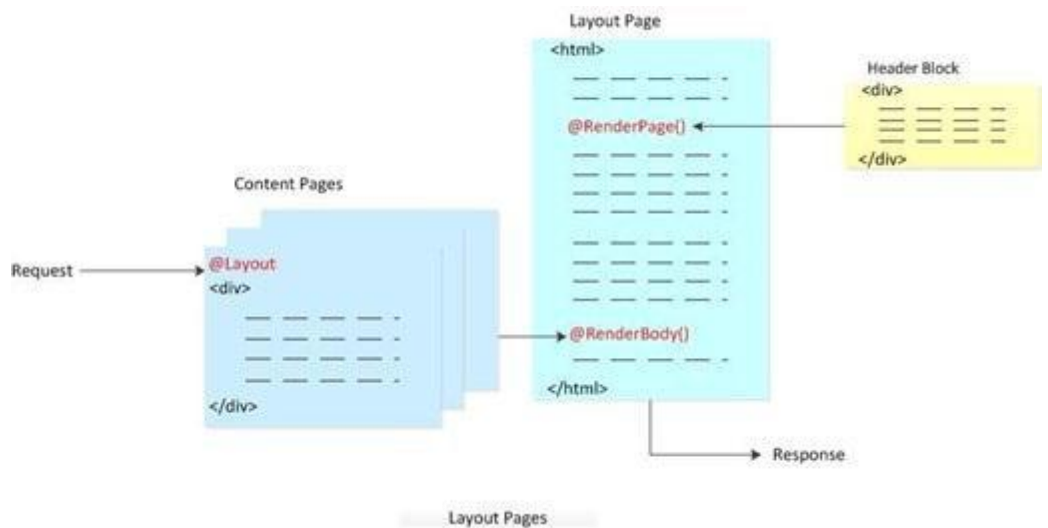
```
<!DOCTYPE html>
<html>
    <head>
        <title>Main Page</title>
    </head>
    <body>

        <div class="header">This is header text.</div>

        <h1>Index Page Content</h1>
        <p>This is the content of the main page.</p>

        <div class="footer">&copy; 2010 Contoso Pharmaceuticals. All rights reserved.
        </div>

    </body>
</html>
```

## レイアウト ページを使った一貫性のある外観の作成

ここまで、複数のページで容易に同じコンテンツを含めることを見てきました。一貫性のある外観のサイトを作るための、より構造化された方法が、レイアウト ページです。レイアウト ページは、Web ページの構造を定義しますが、実際には何のコンテンツも持ちません。レイアウト ページを作成すると、コンテンツを含む Web ページを作成して、それをレイアウト ページにリンクできます。こうしたページが表示されると、レイアウト ページに従った形式で表示されます。（この意味で、レイアウト ページは、他のページを定義するためのコンテンツのテンプレートとして機能します。）

レイアウト ページは、RenderBody メソッドへの呼び出しを含むことを除けば、HTML ページのようなものです。レイアウト ページにおける RenderBody メソッドの位置は、コンテンツ ページのどこから情報を取り込むかを決定します。

以下の図は、最終的な Web ページを生成するために、実行時にコンテンツ ページとレイアウト ページがどのように組み合わせられるかを示しています。ブラウザーは、コンテンツ ページをリクエストします。コンテンツ ページは、ページ構造のために使うレイアウト ページを指定するコードを持っています。コンテンツ ブロックは、前セクションで説明したとおり、RenderPage メソッドの呼び出しによってレイアウト ページの中に挿入されます。Web ページが完成すると、ブラウザーに送出されます。




以下の手順は、レイアウト ページの作成と、それに対するコンテンツ ページのリンクについて示しています。

1. Web サイトの Shared フォルダーに、_Layout1.cshtml という名前のファイルを作成します。
2. 既存のコンテンツを以下で置き換えます。

```
<!DOCTYPE html>
   <head>
     <title> Structured Content </title>
     <link href="@Href("/Styles/Site.css")" rel="stylesheet" type="text/css" />
   </head>
   <body>
       @RenderPage("/Shared/_Header2.cshtml")
       <div id="main">
           @RenderBody()
       </div>
       <div id="footer">
           &copy; 2010 Contoso Pharmaceuticals. All rights reserved.
       </div>
   </body>
</html>
```

レイアウト ページでは、コンテンツ ブロックを挿入するために RenderPage メソッドを使います。レイアウト ページには、1 回しか RenderBody メソッドを呼び出せません。

**注意**　Web サーバーは、この方法でハイパーリンクの参照（リンクの href 属性）を処理しません。このため、ASP.NET は@Href ヘルパーを提供し、パスを受け入れて、Web サーバーが期待する形式で Web サーバーへのパスを提供します。

3. Shared フォルダーの中に、_Header2.cshtml という名前のファイルを作成し、既存のコンテンツを以下で置き換えます。

```
<div id="header">Chapter 3: Creating a Consistent Look</div>
```

4. ルート フォルダーに新しいフォルダーを作成し、Styles という名前にします。

5. Styles フォルダーの中に、Site.css という名前のファイルを作成し、以下のスタイル定義を追加します。

```
h1 {
    border-bottom: 3px solid #cc9900;
    font: 2.75em/1.75em Georgia, serif;
    color: #996600;
}

ul {
    list-style-type: none;
}

body {
    margin: 0;
    padding: 1em;
    background-color: #ffffff;
    font: 75%/1.75em "Trebuchet MS", Verdana, sans-serif;
    color: #006600;
}

#list {
    margin: 1em 0 7em -3em;
    padding: 1em 0 0 0;
    background-color: #ffffff;
    color: #996600;
    width: 25%;
    float: left;
}

#header, #footer {
    margin: 0;
    padding: 0;
    color: #996600;
}
```

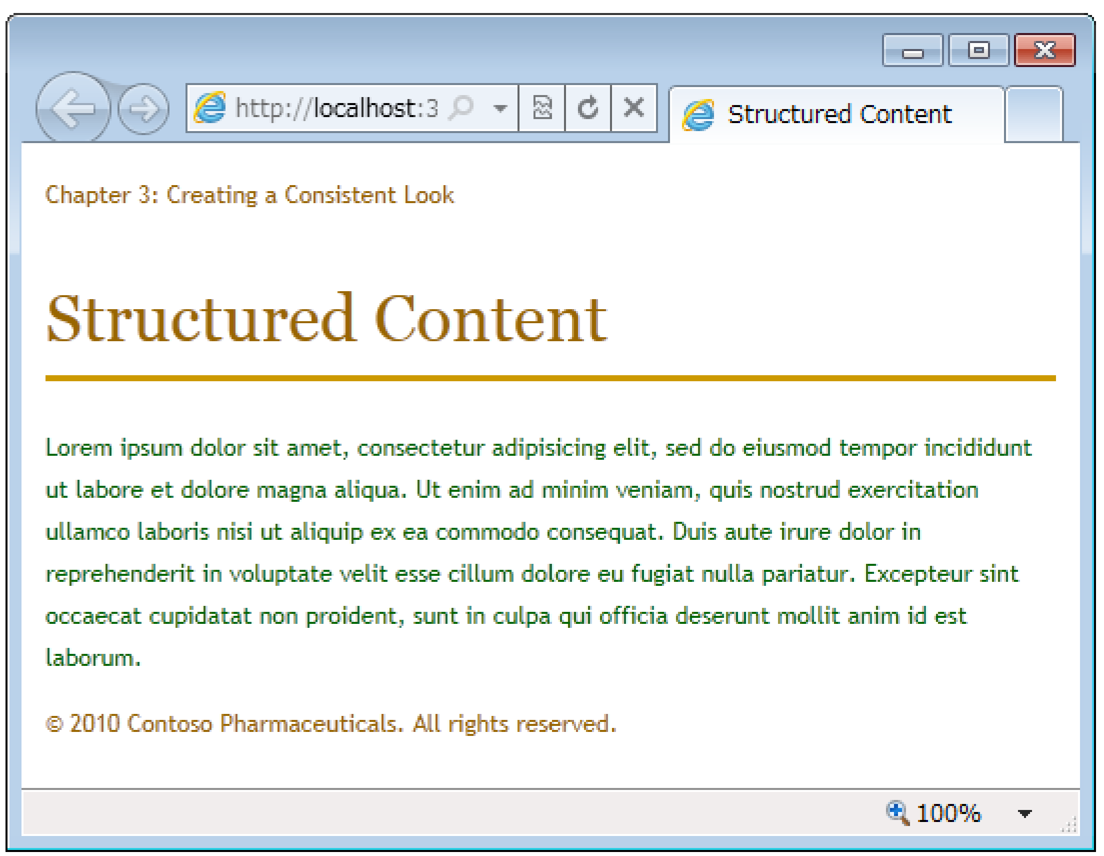
これらのスタイル定義は、レイアウト ページでどのようにスタイル シートを使えるかを示すためのものです。これらの要素に、好みのスタイルを定義してもかまいません。

6. ルート フォルダーで、Content1.cshtml という名前のファイルを作成し、既存のコンテンツを以下で置き換えます。

```
@{
    Layout = "/Shared/_Layout1.cshtml";
}

<h1> Structured Content </h1>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.</p>
```

これは、レイアウト ページを使うページです。このページの先頭のコード ブロックは、このコンテンツを書式化するために使うレイアウト ページを指定しています。

7. ブラウザーで Content1.cshtml を実行します。レンダリングされたページは、_Layout1.cshtml で定義された形式とスタイルシート、Content1.cshtml で定義されたテキスト（コンテンツ）を使います。




ステップ 6 を繰り返して、同じレイアウト ページを共有するコンテンツ ページを追加できます。

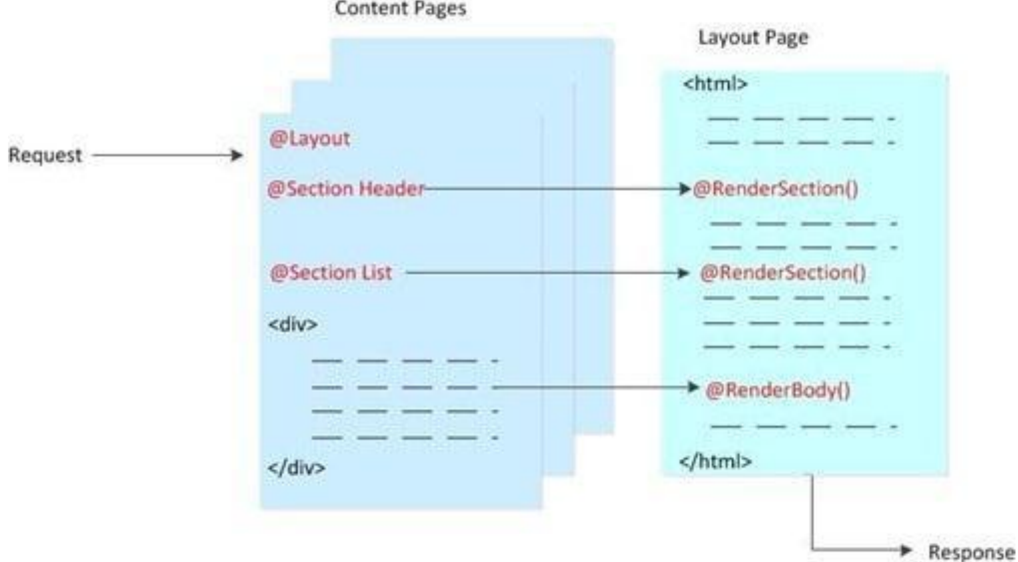
**注意**　フォルダー中のすべてのコンテンツ ページが同じレイアウト ページを自動的に使うよう、サイトを設定できます。詳細は、「第 18 章　サイト全体の動作をカスタマイズする」を参照してください。

## 複数のコンテンツ セクションを持つレイアウト ページのデザイン

コンテンツ ページは、複数のセクションを持つことができます。これは、置き換え用のコンテンツのために複数の領域を持つようなレイアウトを使いたいときに役立ちます。コンテンツ ページでは、それぞれのセクションに一意な名前を与えます。（デフォルト セクションは名前のないままです。） レイアウト ページでは、RenderBody メソッドを追加して、名前のない（デフォルトの）セクションが表示される場所を指定します。さらに、独立した RenderSection メソッドを追加して、それぞれの名前付きセクションをレンダリングします。

以下の図は、ASP.NET が複数のセクションに分割されたコンテンツをどのように扱うかを示しています。それぞれの名前付きセクションは、コンテンツ ページのセクション ブロックに保持されます。（この例では、Header と List という名前が付けられています。） フレームワークは、レイアウト ページの RenderSection メソッドが呼び出さ

れた場所にコンテンツ セクションを挿入します。前述のとおり、名前のない（デフォルトの）セクションは、RenderBody メソッドが呼び出された場所に挿入されます。




以下の手順は、複数のコンテンツ セクションを持つコンテンツ ページを作成し、複数のコンテンツ セクションをサポートするレイアウト ページを使ってレンダリングする方法を示しています。

1. Shared フォルダーの中に、_Layout2.cshtml という名前のファイルを作成します。
2. 既存のコンテンツを以下で置き換えます。

```
<!DOCTYPE html>
<html>
    <head>
     <title>Multisection Content</title>
     <link href="@Href("/Styles/Site.css")" rel="stylesheet" type="text/css" />
    </head>
    <body>
        <div id="header">
        @RenderSection("header")
    </div>
    <div id="list">
        @RenderSection("list")
    </div>
    <div id="main">
        @RenderBody()
    </div>
    <div id="footer">
        &copy; 2010 Contoso Pharmaceuticals. All rights reserved.
    </div>
    </body>
</html>
```

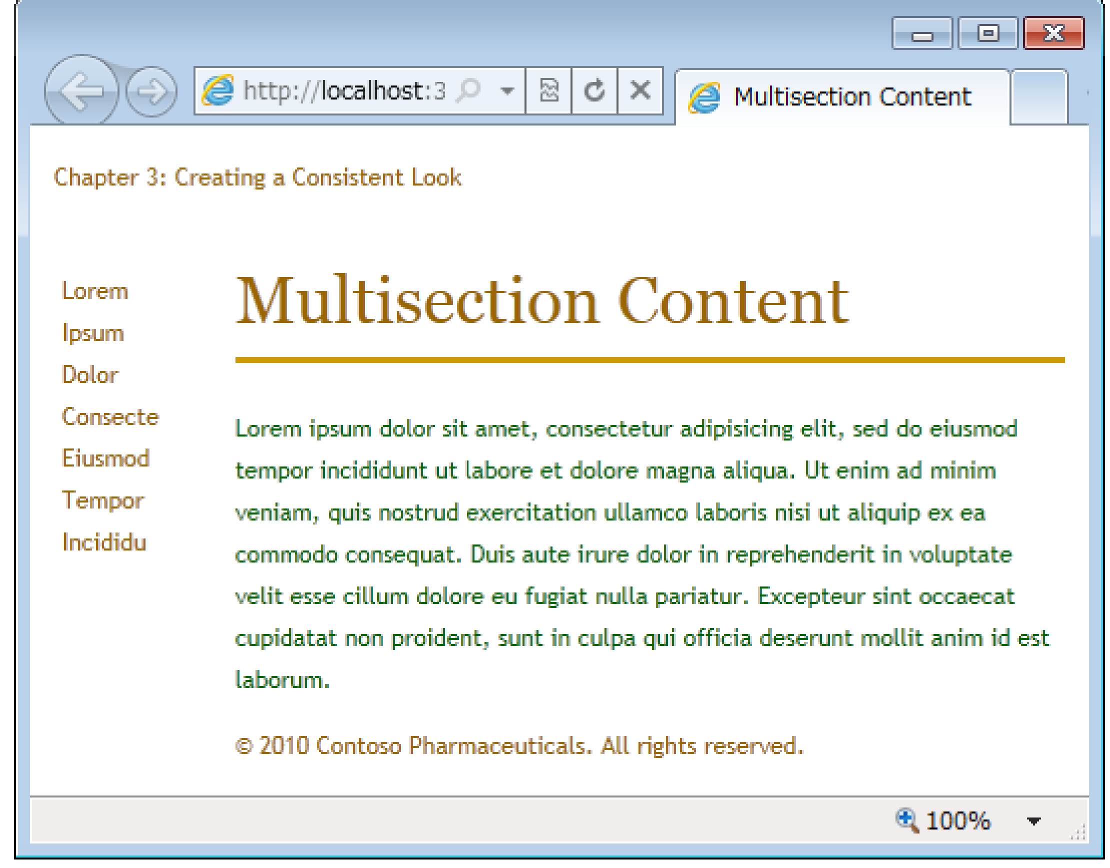
RenderSection メソッドを使って、header と list のセクションをレンダリングしています。

3. ルート フォルダーに、Content2.cshtml という名前のファイルを作成し、既存のコンテンツを以下で置き換えます。

```
@{
    Layout = "/Shared/_Layout2.cshtml";
}

@section header {
    <div id="header">
        Chapter 3: Creating a Consistent Look
    </div>
}

@section list {
    <ul>
        <li>Lorem</li>
        <li>Ipsum</li>
        <li>Dolor</li>
        <li>Consecte</li>
        <li>Eiusmod</li>
        <li>Tempor</li>
        <li>Incididu</li>
    </ul>
}

<h1>Multisection Content</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.</p>
```

コンテンツ ページは、ページの先頭にコード ブロックを含みます。それぞれの名前付きセクションは、セクション ブロックに含まれます。ページの残りは、デフォルトの（名前の無い）コンテンツ セクションを含みます。

4. ブラウザーで実行します。




## コンテンツ セクションをオプションにする

通常、コンテンツ ページの中に作るセクションは、レイアウト ページで定義されたセクションに一致する必要があります。以下の状況のいずれかに該当すると、エラーが発生します。

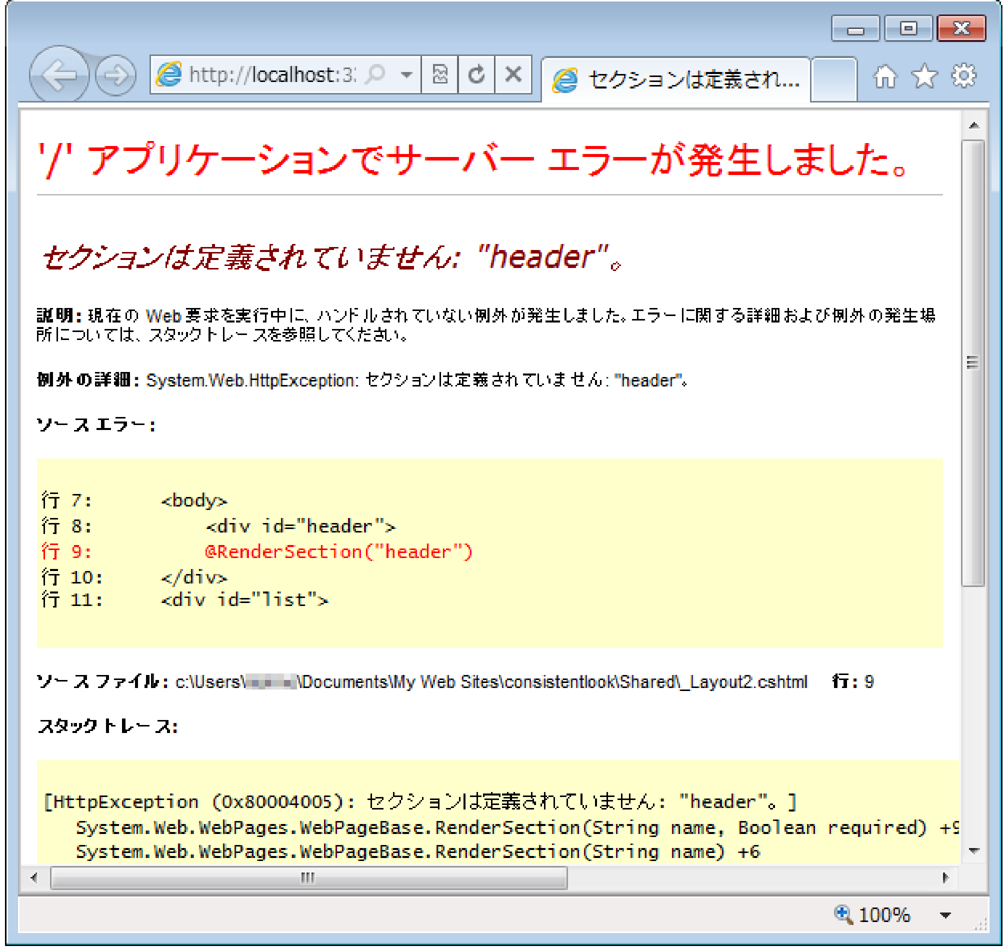
- コンテンツ ページが、レイアウト ページのセクションに対応しないセクションを含んでいる。
- レイアウト ページが、コンテンツを含まないセクションを含んでいる。
- レイアウト ページが、同じセクションを複数回レンダリングしようとするメソッド呼び出しを持っている。

ただし、レイアウト ページでセクションをオプションとして宣言することで、名前付きセクションの動作を変更できます。これにより、指定されたセクションのためのコンテンツを持つか持たないかに関わらず、レイアウト ページを共有できる複数のコンテンツ ページを定義できるようになります。

1. Content2.cshtml を開いて、以下のセクションを削除します。

```
@section header {
    <div id="header">
        Chapter 3: Creating a Consistent Look
    </div>
}
```

2. ページを保存して、ブラウザーで実行します。このコンテンツ ページには、レイアウト ページで定義された Header という名前のセクションのためのコンテンツが提供されないため、エラーが表示されます。




3. Shared フォルダーで、_Layout2.cshtml ページを開き、次の行を、

```
@RenderSection("header")
```

次の行で置き換えます。

```
@RenderSection("header", required: false)
```

あるいは、上記のコード行を、以下のコード ブロックで置き換えても同じ結果になります。

```
@if (IsSectionDefined("header")) {
    @RenderSection("header")
}
```

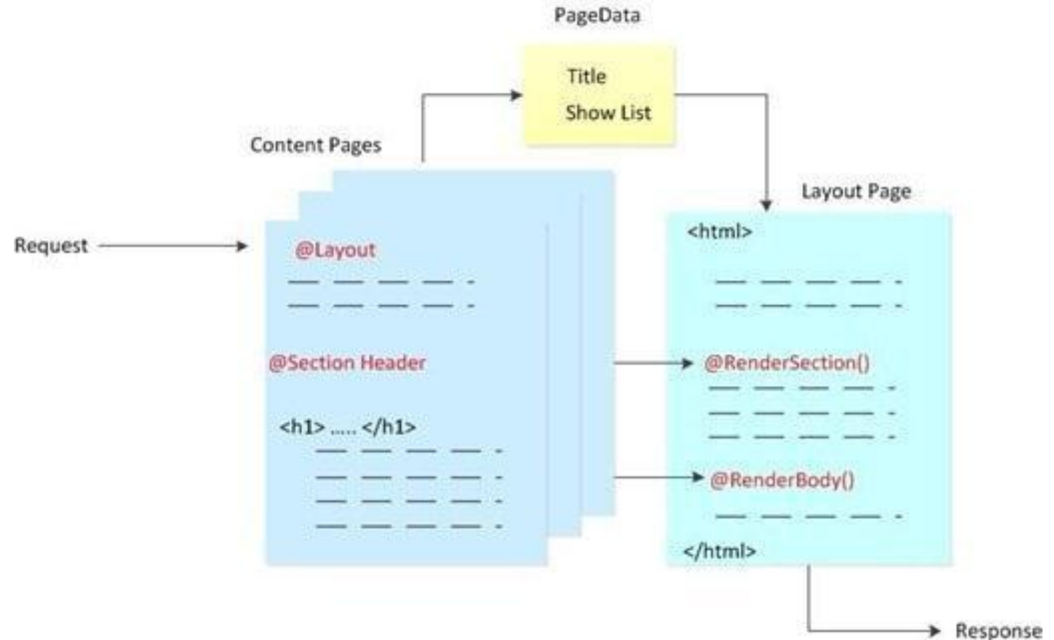
4. もう一度、Content2.cshtml ページをブラウザーで実行します。（このページをブラウザーで開いたままにしていた場合は、読み込みなおしてください。） 今回は、このページは、ヘッダーはないものの、エラーなしで表示されます。

## レイアウト ページにデータを渡す

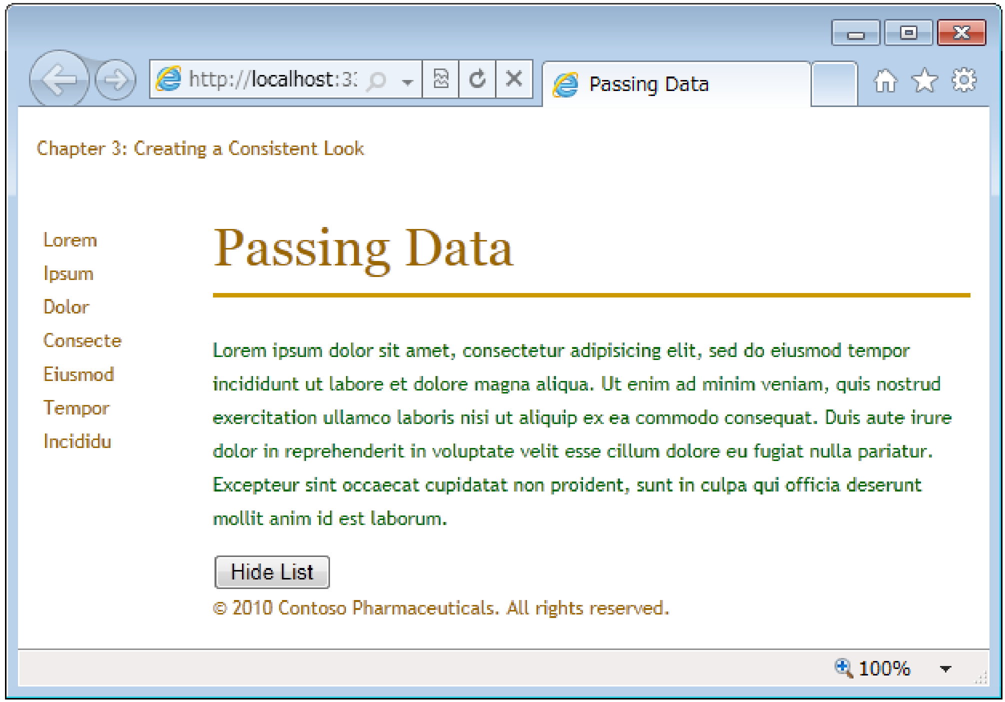
コンテンツ ページ中で定義されたデータを、レイアウト ページで参照する必要があるかもしれません。その場合、コンテンツ ページからレイアウト ページにデータを渡す必要があります。たとえば、ユーザーのログイン状態を表示したい場合、あるいは、ユーザー入力に基づいてコンテンツを表示したり隠したりしたいかもしれません。

コンテンツ ページからレイアウト ページにデータを渡すためには、コンテンツ ページの PageData プロパティに値を置きます。PageData プロパティは、ページ間で受け渡ししたいデータを保持する、名前と値のペアからなるコレクションです。こうすることで、レイアウト ページで、PageData プロパティから値を読みだせます。

もう一つの図を示します。これは、コンテンツ ページからレイアウト ページに値を渡す PageData プロパティを、ASP.NET がどのように使っているかを示すものです。ASP.NET は、Web ページを構築しはじめるときに、PageData コレクションを作成します。コンテンツ ページでは、PageData コレクションにデータを置くためのコードを書きます。PageData コレクションの値は、コンテンツ ページの他のセクションや、他のコンテンツ ブロックからアクセスできます。




以下の手順は、コンテンツ ページからレイアウト ページにデータを渡す方法を示しています。ページを実行すると、レイアウト ページで定義されたリストを隠したり、表示するためのボタンを表示します。ユーザーがボタンをクリックすると、PageData プロパティの値が true/false（論理値）に設定されます。レイアウト ページは値を読み込み、false ならばリストを隠します。この値は、コンテンツ ページで［Hide List］ボタンか［Show List］ボタンのどちらを表示するかを決定するためにも使われます。




1. ルート フォルダーで、Content3.cshtml という名前のファイルを作成し、既存のコンテンツを以下の内容で置き換えます。

```
@{
    Layout = "/Shared/_Layout3.cshtml";

    PageData["Title"] = "Passing Data";
    PageData["ShowList"] = true;

    if (IsPost) {
        if (Request["list"] == "off") {
            PageData["ShowList"] = false;
        }
    }
}

@section header {
    <div id="header">
        Chapter 3: Creating a Consistent Look
    </div>
}

<h1>@PageData["Title"]</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.</p>

@if (PageData["ShowList"] == true) {
    <form method="post" action="">
        <input type="hidden" name="list" value="off" />
```

```
        <input type="submit" value="Hide List" />
    </form>
}
else {
    <form method="post" action="">
        <input type="hidden" name="list" value="on" />
        <input type="submit" value="Show List" />
    </form>
}
```

このコードは、PageData プロパティに 2 つのデータの断片 - Web ページのタイトルとリストを表示するかどうかを指定する true または false - を保存します。

ASP.NET が、コード ブロックを使って条件的にページのマークアップを配置できることに注意してください。たとえば、このページの本体にある if/else ブロックは、PageData["ShowList"] が true に設定されているかどうかによって、どちらのフォームを表示するかを決定します。

2. Shared フォルダーで、_Layout3.cshtml という名前のファイルを作成し、既存のコンテンツを以下で置き換えます。

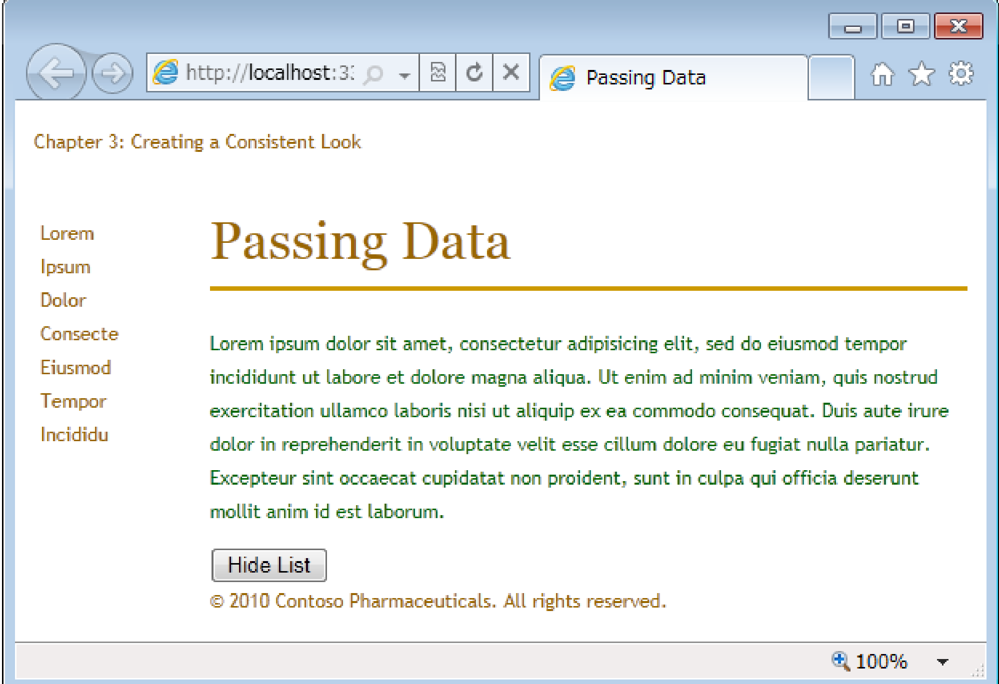
```
<!DOCTYPE html>
<html>
    <head>
     <title>@PageData["Title"]</title>
     <link href="@Href("/Styles/Site.css")" rel="stylesheet" type="text/css" />
    </head>
    <body>
        <div id="header">
            @RenderSection("header")
        </div>
        @if (PageData["ShowList"] == true) {
            <div id="list">
                @RenderPage("/Shared/_List.cshtml")
            </div>
        }
        <div id="main">
            @RenderBody()
        </div>
        <div id="footer">
            &copy; 2010 Contoso Pharmaceuticals. All rights reserved.
        </div>
    </body>
</html>
```

レイアウト ページに含まれる <title> 要素の式は、PageData プロパティから title の値を取り出します。また、PageData プロパティの ShowList 値を使って、list コンテンツ ブロックを表示するかどうかを決定します。

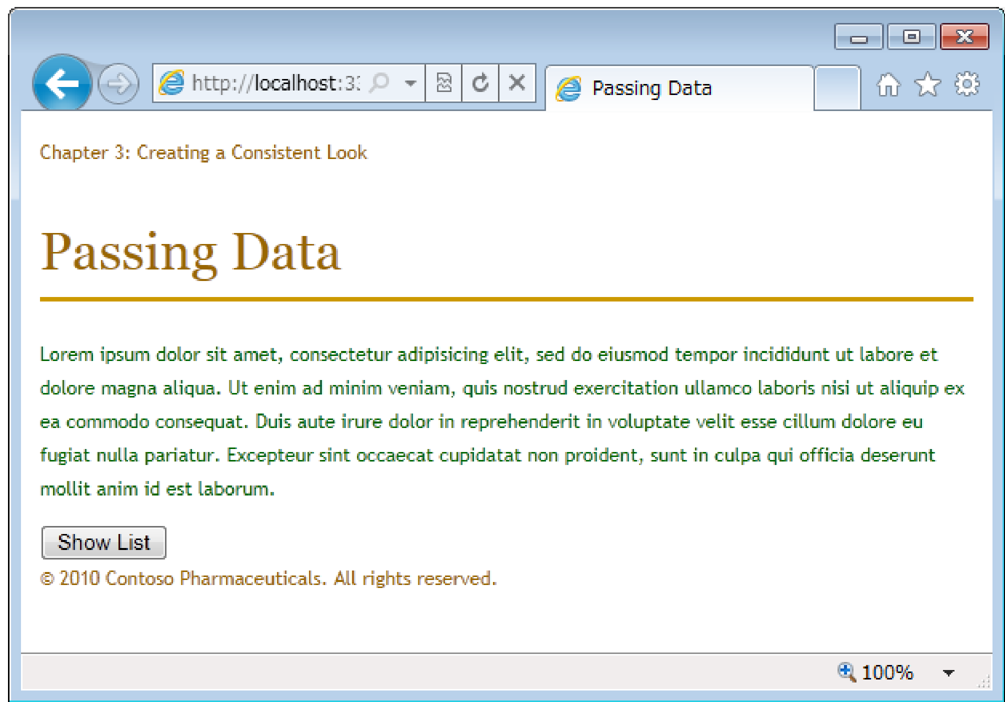
3. Shared フォルダーに、_List.cshtml という名前のファイルを作成し、既存のコンテンツを以下で置き換えます。

```
<ul>
    <li>Lorem</li>
    <li>Ipsum</li>
    <li>Dolor</li>
    <li>Consecte</li>
    <li>Eiusmod</li>
    <li>Tempor</li>
    <li>Incididu</li>
</ul>
```

4. ブラウザーで、Content3.cshtml ページを実行します。このページには、左側に表示中のリストと、下部に [Hide List] ボタンがあります。




5. [Hide List] をクリックします。リストは消えて、ボタンは [Show List] に変わります。



6. [Show List] ボタンをクリックすると、ふたたびリストが表示されます。

## 基本的なヘルパーの作成と使用

Webサイトで一貫性のある外観を作成するための、もうひとつの選択肢がカスタム ヘルパーを作成することです。「第 1 章 - はじめての WebMatrix と ASP.NET Web ページ」で学んだとおり、ヘルパーは単一行のコードを使って処理を実現するための部品です。ASP.NET は、多くのヘルパーを提供しており、以降の章でこれらの多くを使います。ヘルパーの完全なリストは、「ASP.NET API クイックリファレンス」に記載されています。

ヘルパーは、複数のページをまたがる一般的なコードのブックを使うことで、Web サイトに一貫性のある外観を作ることを手助けします。ページ中に、通常のパラグラフとは別に注釈項目を繰り返し作成するため、四角い枠をスタイルとするため<div>要素を使うことを考えてみてください。どのページにも同じマークアップを追加する代わりに、ヘルパーとしてパッケージングし、必要な場所に単一行で注釈を挿入することができます。これにより、ページ中のコードは単純化され、読みやすくなります。注釈の外観を変更する場合も、一カ所のマークアップを変更するだけで済むため、サイトの保守も容易になります。

以下の手順は、ここで述べた注釈を作成するヘルパーを作成する方法について示しています。これは単純な例ですが、カスタム ヘルパーは必要に応じてどんなマークアップや ASP.NET コードも含むことができます。

1. Web サイトのルート フォルダーに、App_Code という名前のフォルダーを作成します。
2. App_Code フォルダーの中に、新しい.cshtml ファイルを作成し、MyHelpers.cshtml と名付けます。
3. 既存のコンテンツを以下と置き換えます。

```
@helper MakeNote(string content) {
    <div class="note" style="border: 1px solid black; width: 90%; padding: 5px;
margin-left: 15px;">
        <p>
            <strong>Note</strong>   @content
        </p>
    </div>
}
```

このコードは、MakeNote という名前の新しいヘルパーを宣言する@helper 構文を使っています。この特別なヘルパーは、テキストとマークアップの組み合わせを含む content という名前のパラメーターを渡します。ヘルパーは、@content 変数を使って注釈の本体に文字列を挿入します。

4. 保存してファイルを閉じます。

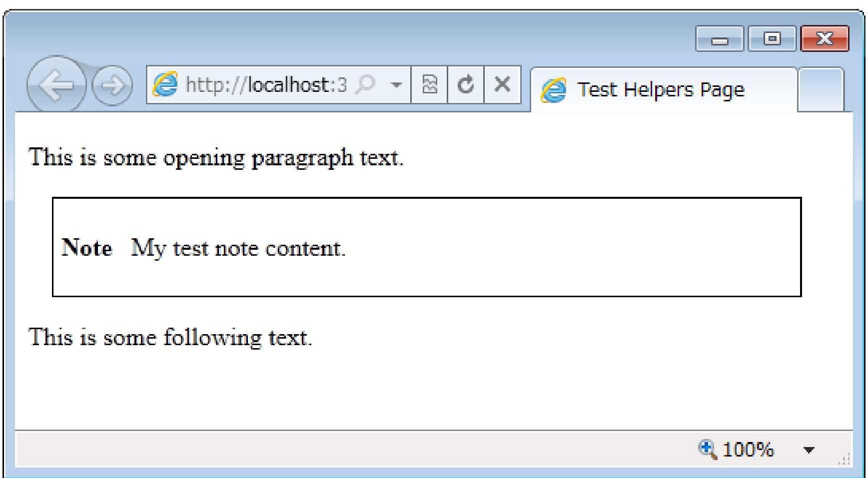
次の手順は、Web ページ中に新しい注釈項目を挿入するためのヘルパーの使い方について示しています。

1. ルートフォルダーで、TestHelper.cshtml という空のファイルを作成します。
2. このファイルに、以下のコードを追加します。

```
<!DOCTYPE html>
    <head>
        <title>Test Helpers Page</title>
    </head>
    <body>
        <p>This is some opening paragraph text.</p>

        <!-- Insert the call to your note helper here. -->
        @MyHelpers.MakeNote("My test note content.")

        <p>This is some following text.</p>
    </body>
</html>
```

作成したヘルパーを呼び出すために、@に続けて、ヘルパーのある場所、ドット、ヘルパー名を指定します。（App_Code フォルダー中に複数のフォルダーがある場合は、「@フォルダー名.ファイル名.ヘルパー名」という構文を使って、どんなネストされたフォルダーにあるヘルパーも呼び出せます。） カッコの中のクォーテーション記号に追加されたテキストは、ヘルパーが Web ページに注釈の一部として表示するテキストです。

3. ページを保存して、ブラウザーで実行します。ヘルパーは、ヘルパーを呼び出した場所、つまり 2 つのパラグラフの間に注釈項目を生成します。




## その他のリソース

- 「第 18 章　サイトー全体の動作をカスタマイズする」

# 第４章　　フォームを扱う

フォームとは、テキストボックスやチェックボックス、ラジオボタン、プルダウン リストのようなユーザー入力コントロールを配置した HTML ドキュメントのセクションです。ユーザー入力を集めて処理したいときにフォームを使います。

ここでは次のことを学びます。

- HTML フォームを作成する方法
- フォームからユーザー入力を読み取る方法
- ユーザー入力を検証する方法
- ページを送出した後で値を復元する方法

本章には、以下のような ASP.NET のプログラミング概念が使われています。

- Request オブジェクト
- 入力の検証
- HTML エンコーディング

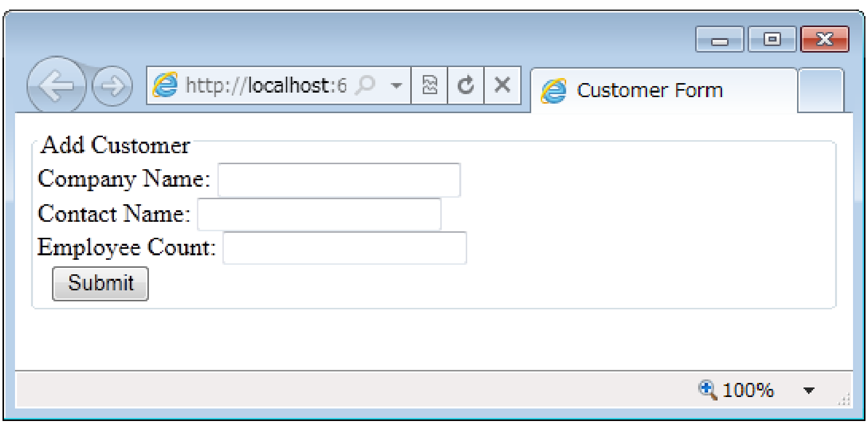
## シンプルな HTML フォームを作成する

1. 新しい Web サイトを作成します。
2. ルート フォルダーで、Form.cshtml と名付けられた Web ページを作成し、以下のマークアップを入力します。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Customer Form</title>
    </head>
    <body>
        <form method="post" action="">
            <fieldset>
                <legend>Add Customer</legend>
                <div>
                    <label for="CompanyName">Company Name:</label>
                    <input type="text" name="CompanyName" value="" />
                </div>
                <div>
                    <label for="ContactName">Contact Name:</label>
                    <input type="text" name="ContactName" value="" />
                </div>
                <div>
                    <label for="Employees">Employee Count:</label>
                    <input type="text" name="Employees" value="" />
                </div>
                <div>
                    <label> </label>
                    <input type="submit" value="Submit" class="submit" />
                </div>
```

```
            </fieldset>
        </form>
    </body>
</html>
```

3. ブラウザーでページを呼び出します。（実行する前に「ファイル」ワークスペースで、このページが選択されていることを確認してください。） 3 つの入力項目と [Submit] ボタンを持つシンプルなフォームが表示されます。




## フォームからユーザー入力を読み取る

フォームを処理するために、項目に入力された値を読み取り、それらに対して何かしら処理するコードを追加します。以下の手順は、項目を読み込み、ページ上にユーザー入力を表示する方法を示しています。（実用的なアプリケーションでは、通常、ユーザー入力に対してより意味のあることをします。それについては、データベースに関する章で取り上げます。）

1. Form.cshtml ファイルの先頭に、次のコードを入力します。

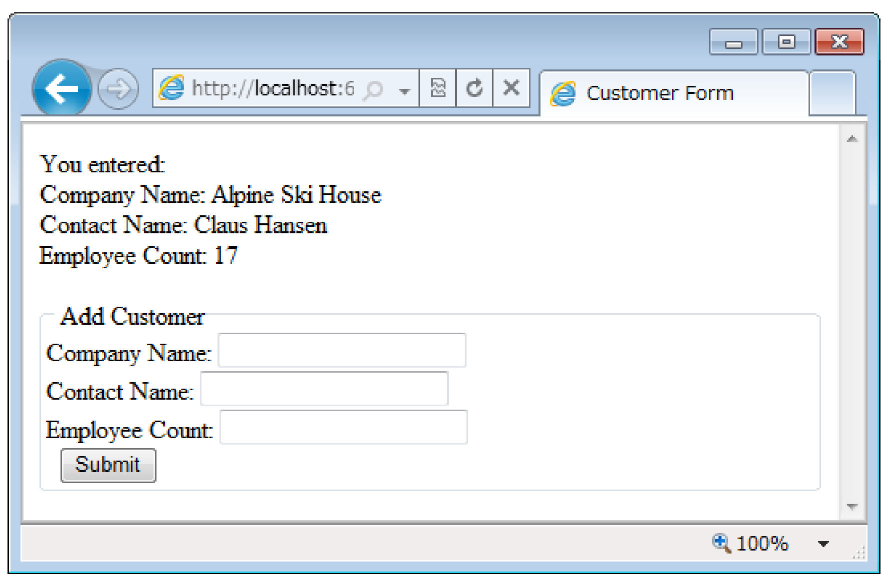
```
@{
    if (IsPost) {
        string companyname = Request["companyname"];
        string contactname = Request["contactname"];
        int employeecount = Request["employees"].AsInt();

        <text>
            You entered: <br />
            Company Name: @companyname <br />
            Contact Name: @contactname <br />
            Employee Count: @employeecount <br />
        </text>
    }
}
```

このページは、ユーザーが最初にページをリクエストしたときには、空のフォームが表示されます。ユーザー（ここではあなた）はフォームに入力して、[Submit] をクリックします。これにより、ユーザー入力

がサーバーに送信（POST）されます。このフォームは上記の手順と同じように作成し、form 要素の action 属性を空にしたままなので、（Form.cshtml という名前の）同じページへのリクエストになります。

```
<form method="post" action="">
```

今回ページを送信すると、入力した値がフォームの真上に表示されます。

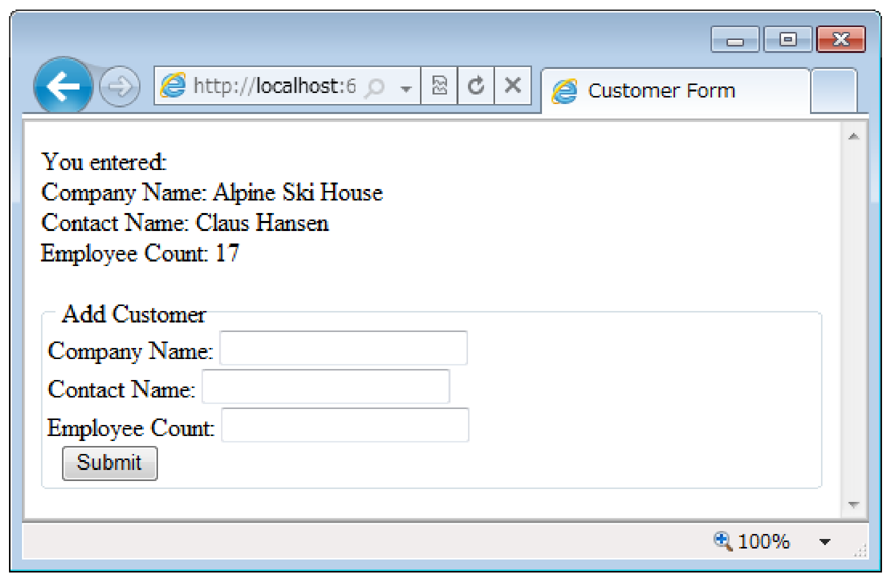



ページのコードを見てください。最初に IsPost メソッドを使ってページが POST されたものかどうか、つまりユーザーが [Submit] ボタンを押したのかどうかを判断しています。POST であれば、IsPost は true を返します。これは、ASP.NET Web ページが初期リクエスト（GET リクエスト）とポストバック（POST リクエスト）を判別するための標準的な方法です。（GET と POST のより詳細な情報については、「第 2 章 Razor 構文を使った ASP.NET Web プログラミングの紹介」の「HTTP GET と POST と IsPost プロパティ」を参照してください。）

次に、Request オブジェクトからユーザーが入力した値を取得し、後でそれらを変数に設定します。Request オブジェクトは、ページとともに送信されたすべての値を保持しており、それぞれはキーによって区別されます。キーは、読み取りたいフォーム項目の name 属性と同じです。たとえば、companyname 項目（テキストボックス）を読み取るためには、「Request["companyname"]」を使います。

フォームの値は Request オブジェクトに文字列として保存されています。このため、数値や日付やその他の型として値を扱う場合は、その値を文字列から適切な型に変換しなければなりません。たとえば、（従業員の数を含む）employees 項目を整数値に変換するために、Request の AsInt メソッドを使います。

2. ブラウザーでページを呼び出し、フォームの項目に入力し、[Submit] をクリックします。ページは入力した値を表示します。




**外観とセキュリティのための HTML エンコーディング**

HTML は、<、>、&のような文字を特殊な目的で使います。これらの特殊文字が予期せぬ場所にあらわれると、Web ページの外観や機能を損なうおそれがあります。たとえば、ブラウザーは、（後ろにスペースがない限り）<文字を読み取ると、<b> や <input …> のような HTML 要素の先頭であると解釈します。ブラウザーが要素を理解しないと、< ではじまった文字列は何か理解できるものに出会うまで捨てられます。この結果、明らかにページのレンダリングがおかしくなります。

HTML エンコーディングは、こうした予約文字をブラウザーが解釈できる正しいシンボルのコードに置き換えます。たとえば、<文字は<で、>文字は>で置き換えられます。ブラウザーは、これらの置き換えた文字を表示したい文字としてレンダリングします。

ユーザーから受け取った文字列（入力）を表示するときはいつでも HTML エンコーディングを使うようにするとよいでしょう。そうしないと、ユーザーが Web ページ上で悪意のあるスクリプトを実行したり、サイトのセキュリティを危険にさらしたり、意図しない動作を引き起こすおそれがあります。（これは、ユーザーの入力を受け付け、どこかに保存し、後で表示するような場合、たとえばブログのコメントやユーザー レビューやそれに類する処理を行う場合には特に重要です。）

## ユーザー入力の検証

ユーザーは間違えます。項目に入力するように促しても、それを忘れたり、従業員数を入力するように尋ねても、代わりに名前を入力したりすることがあります。フォームの入力を処理する前に、正しく入力されたかどうかを確認するには、ユーザー入力を検証します。

以下の手順は、ユーザーが 3 つのフォーム項目を空にしたままにしていないかどうかを検証する方法を示しています。また、従業員数の値が数値であるかどうかも確認しています。エラーがあれば、ユーザーにどの値が検証を通らなかったかを示すエラーメッセージを表示します。

1. Form.cshtml ファイルで、最初のコードブロックを以下のコードに置き換えます。

```
@{
    if (IsPost) {
        var errors = false;
        var companyname = Request["companyname"];
        if (companyname.IsEmpty()) {
            errors = true;
            @:Company name is required.<br />
        }
        var contactname = Request["contactname"];
        if (contactname.IsEmpty()) {
            errors = true;
            @:Contact name is required.<br />
        }
        var employeecount = 0;
        if (Request["employees"].IsInt()) {
            employeecount = Request["employees"].AsInt();
        } else {
            errors = true;
            @:Employee count must be a number.<br />
        }
        if (errors == false) {
            <text>
            You entered: <br />
            Company Name: @companyname <br />
            Contact Name: @contactname <br />
            Employee Count: @employeecount <br />
            </text>
        }
    }
}
```

このコードは、置き換え前のコードに似ていますが、いくつかの違いがあります。最初の違いは、errors という名前の変数を false に初期化していることです。いずれかの検証テストが失敗した場合、この変数には true を設定します。

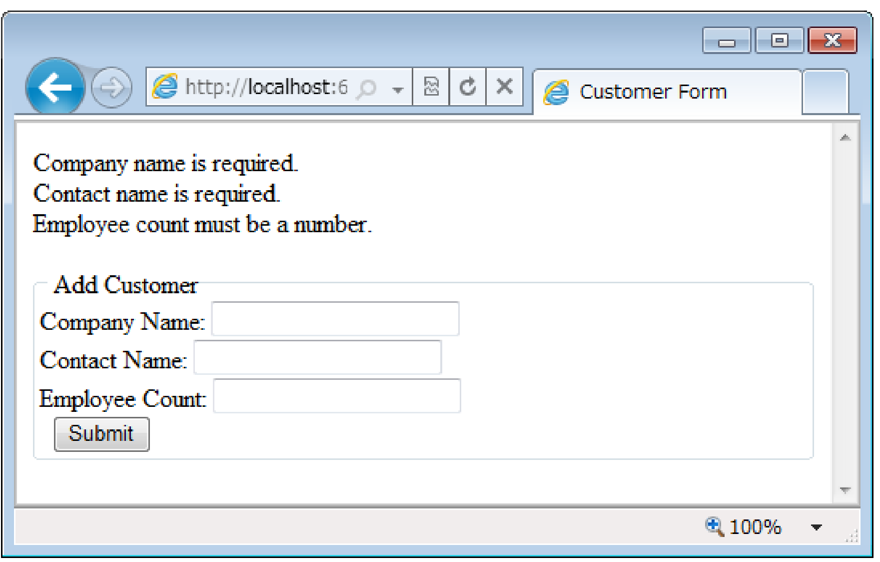
フォーム項目の値を読み取るコードは、毎回検証テストを行います。companyname と contact 項目は、IsEmpty 関数を呼び出して検証します。テストが失敗すると（つまり、IsEmptry が true を返すと）、コードは errors 変数に true を設定して、適切なエラーメッセージを表示します。

次のステップは、ユーザーが従業員数として数値（整数）を入力したかどうかを確認することです。これを行うため、IsInt 関数を呼び出します。この関数は、テストしたい値が文字列から整数に変換できる場合に true を返します。（もちろん、数値が変換できない場合は false を返します。） Request オブジェクトに含まれるすべての値は文字列であることを思い出してください。この例では、あまり重要ではありませんが、値に対して演算操作したい場合には、値は数値に変換できるものでなければなりません。

IsInt で値が整数であることがわかれば、employeecount 変数にその値を設定します。ただし、employeecount を初期化するときには int を使って型指定されているので、これを行う前に実際に整数に変換しておかなければなりません。IsInt 関数は整数かどうかを知らせ、次の行の AsInt 関数が実際の変換を実

行する、というパターンに注意してください。IsInt が true を返さなければ、else ブロックの文で errors 変数を true に設定します。

最後に、すべてのテストが終了した後、コードは errors 変数がまだ false であるかどうかを判断します。もしそうなら、コードはユーザーが入力した値を含むテキストブロックを表示します。このページをブラウザーで呼び出し、項目を空にしたまま、[Submit] をクリックしてください。エラーが表示されます。




2. フォーム項目に値を入力して、[Submit] をクリックしてください。さきほどと同じように入力した値が表示されることがわかります。

## ポストバック後のフォームの値を復元する

前セクションのページをテストするとき、検証エラーがあると、入力した内容はすべて（不正なデータでなくても）破棄されてしまい、もう一度すべての項目を再入力しなければならないことがわかります。これは重要なことを示しています。ページを送出し、処理し、ふたたびページをレンダリングするときには、ページは最初から再作成されるということです。ご覧のとおり、これはページに入力した値が送信されるときに失われるということを意味しています。

しかし、これは簡単に修正できます。（Request オブジェクトで）入力された値にアクセスできるため、ページをレンダリングするときに、これらの値をフォーム項目に戻しておくことができます。

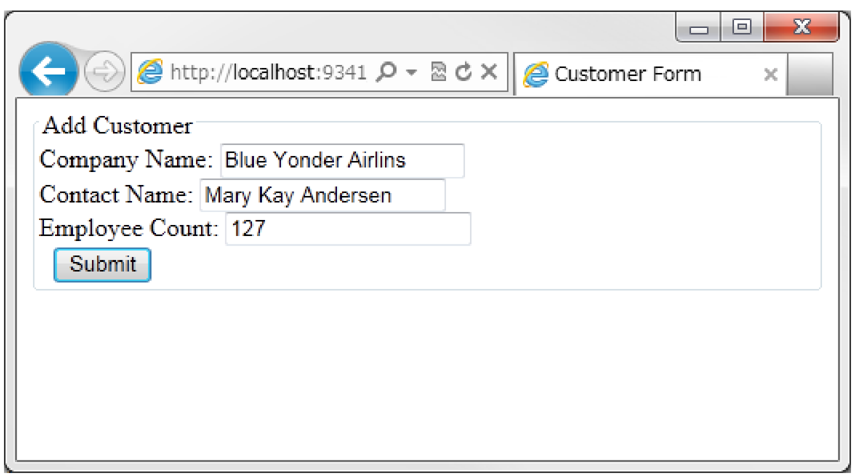
1. Form.cshtml ファイルで、デフォルト ページを以下のマークアップで置き換えます。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Customer Form</title>
    </head>
    <body>
        <form method="post" action="">
            <fieldset>
                <legend>Add Customer</legend>
                <div>
```

```
                <label for="CompanyName">Company Name:</label>
                <input type="text" name="CompanyName"
                    value="@Request["companyname"]" />
            </div>
            <div>
                <label for="ContactName">Contact Name:</label>
                <input type="text" name="ContactName"
                    value="@Request["contactname"]" />
            </div>
            <div>
                <label for="Employees">Employee Count:</label>
                <input type="text" name="Employees" val-
ue="@Request["employees"]" />
            </div>
            <div>
                <label> </label>
                <input type="submit" value="Submit" class="submit" />
            </div>
        </fieldset>
    </form>
  </body>
</html>
```

<input>要素の value 属性に、Request オブジェクトから読みだした項目値を動的に設定します。ページが最初にリクエストされたときは、Request オブジェクトはすべて空です。フォームは空欄になるため、これで問題ありません。

2. このページをブラウザーで呼び出し、フォームの項目を入力したり、空欄のままにして、[Submit] をクリックします。入力した値が表示されます。



## その他のリソース

- Web ユーザーから入力を受け取る 1,001 通りの方法
- Using Forms and Processing User Input
- Using AutoComplete in HTML Forms
- Gathering Information With HTML Forms
- AJAX を使用した HTML フォームの拡張

# 第５章　　データを扱う

本章では、ASP.NET Web ページを使って、データベースのデータにアクセスし、表示する方法を解説します。

ここでは次のことを学びます。

- データベースを作成する方法
- データベースへ接続する方法
- Web ページにデータを表示する方法
- データベースレコードを挿入、更新、削除する方法

本章では、以下の機能を紹介します。

- Microsoft SQL Server Compact Edition データベースの処理
- SQL クエリの処理
- Database クラス

## データベースの紹介

一般的なアドレス帳を考えてみてください。アドレス帳のどの項目も（つまり、どの人も）、姓、名、住所、電子メールアドレス、電話番号のような情報の断片を持っています。

データをイメージしやすいよう、行と列からなる表とします。データベース用語では、それぞれの行をレコードと呼びます。どの列（項目と呼ばれることもあります）も、それぞれのデータ（姓、名など）の型に対応する値を含みます。

| ID | FirstName | LastName | Address | Email | Phone |
|---|---|---|---|---|---|
| 1 | Jim | Abrus | 210 100th St SE Orcas WA 98031 | jim@contoso.com | 555 0100 |
| 2 | Terry | Adams | 1234 Main St. Seattle WA 99011 | terry@cohowinery.com | 555 0101 |

ほとんどのデータベース テーブルでは、テーブルは顧客番号やアカウント番号のような一意の識別子を含む列を持っているものです。これはテーブルのプライマリー キーとして知られており、テーブルの個々の行を識別するために使われます。たとえば、このアドレス帳では ID 列がプライマリー キーです。

このデータベースの基本的な理解があれば、単純なデータベースを作成し、データの追加、修正、削除のような操作を実行する方法について学ぶことができます。

**リレーショナル データベース**

データは、テキストファイルやスプレッドシートなど、さまざまな形式で保存できます。しかし、ほとんどの業務用途においては、データはリレーショナル データベースに保存されます。

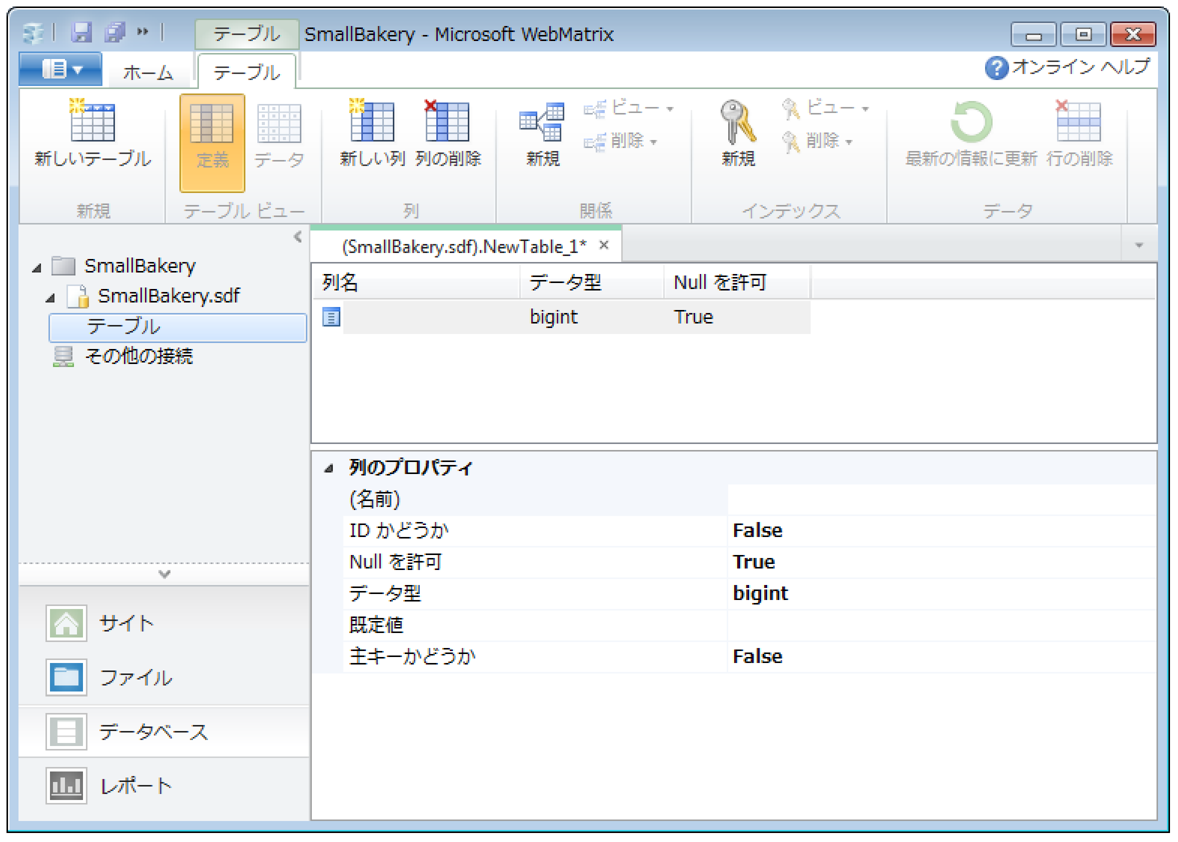
本章では、データベースの詳細には立ち入りません。しかし、これらをよく理解することはとても役立つことがわかるでしょう。リレーショナル データベースにおいては、情報は分離されたテーブルへ論理的に分割されます。たとえば、学校のためのデータベースは、生徒のため、あるいはクラスのために提供される異なるテーブルを含みます。

（SQL Server のような）データベース ソフトウェアは、テーブル同士の関係を動的に確立するための強力なコマンドをサポートします。たとえば、予定を作成するために生徒とクラスの論理的な関係を確立するためにリレーショナル データベースを使うことができます。異なるテーブルにデータを保存することは、テーブル構造の複雑性を緩和し、テーブル中に冗長なデータを保持する必要性を減らします。

## データベースの作成

以下の手順は、WebMatrix に含まれる SQL Server Compact Database デザイン ツールを使って SmallBakery と名付けられたデータベースを作成する方法を示しています。コードを使ってデータベースを作成することもできますが、より一般的には WebMatrix のようなデザインツールを使ってデータベースやデータベース テーブルを作成します。

1. WebMatrix を起動し、「クイック スタート」ページで「テンプレートからサイトを作成する」をクリックします。
2. 「空のサイト」を選び、「サイト名」ボックスに "SmallBakery" と入力し、[OK] をクリックします。WebMatrix にサイトが作成され、表示されます。
3. 左ペインで、「データベース」ワークスペースをクリックします。
4. リボンで、「新しいデータベース」をクリックします。サイトと同じ名前を持つ、空のデータベースが作成されます。
5. 左ペインで、SmallBakery.sdf ノードを展開し、「テーブル」をダブルクリックします。
6. リボンで、「新しいテーブル」をクリックします。WebMatrix はテーブル デザイナーを開きます。




7. 列のプロパティで、「(名前)」欄に"Id"と入力します。

8. 新しい Id 列で、「ID かどうか」「主キーかどうか」を True に設定します。

  項目名が示す通り、「主キーかどうか」は、データベースに対して、その項目がテーブルのプライマリー キーになることを伝えます。「ID かどうか」は、データベースに対して、すべての新しいレコードに対する ID 番号を作成して次の値（最初は 1）を割り当てることを伝えます。

9. リボンで、「新しい列」をクリックします。
10. 列のプロパティで、「(名前)」欄に"Name"と入力します。
11. 「Null を許可」を false に設定します。これにより、Name 列は空のままにしておけなくなります。
12. 「データ型」を"nvarchar"に設定します。Nvarchar の var は、この列のデータが文字列であり、そのサイズがレコードごとに変わる（vary）ことをデータベースに伝えます。（プレフィックスの n は national を意味し、項目が保持する文字データがどんな文字セットでもあらわせる、つまり項目は Unicode データを保持するという指定です。）
13. 同じように、Description という名前の列を作成します。「Null を許可」は false に、「データ型」は"nvarchar"に設定します。
14. Price という名前の列を作成します。「Null を許可」は false に、「データ型」は"money"に設定します。
15. [CTRL]+[S] を押してテーブルを保存し、"Product"という名前を付けます。

  完了すると、定義は次のように表示されます。

テーブル - (SmallBakery.sdf).Product ×

| 列名 | データ型 | Null を許可 |
|---|---|---|
| Id | bigint | False |
| Name | nvarchar | True |
| Description | nvarchar | True |
| Price | money | True |

**注意** テーブルの列の定義を編集するときは、プロパティ名から値に移動するために [Tab] キーを使うことはできません。代わりに、編集するプロパティを選び、それがテキスト項目ならばプロパティ値を変更するようタイプしてください。または、ドロップダウン項目ならば [F4] を押してプロパティ値を変更してください。

## データベースへのデータの追加

これで、データベースに本章の後半で使うためのサンプルデータを追加できるようになります。

1. 左ペインで、SmallBakery.sdf ノードを展開して「テーブル」をクリックします。
2. Product テーブルを右クリックして、「データ」をクリックします。
3. 編集ペインで、以下のレコードを入力します。

| Name | Description | Price |
|---|---|---|
| Bread | Baked fresh every day. | 2.99 |
| Strawberry Short-cake | Made with organic strawberries from our garden. | 9.99 |
| Apple Pie | Second only to your mom's pie. | 12.99 |
| Pecan Pie | If you like pecans, this is for you. | 10.99 |
| Lemon Pie | Made with the best lemons in the world. | 11.99 |
| Cupcakes | Your kids and the kid in you will love these. | 7.99 |

4. Id 列には何も入力する必要がないことを覚えておいてください。Id 列を作成するとき、「ID かどうか」プロパティを true にしたため、この項目は自動的に入力されます。
5. データを入力すると、テーブル デザイナーは次のように表示されます。

テーブル - (SmallBakery.sdf).Product ×

| | Id | Name | Description | Price |
|---|---|---|---|---|
| | 1 | Bread | Baked fresh every day. | 2.99 |
| | 2 | Strawberry Shortcake | Made with organic strawberries from our garden. | 9.99 |
| | 3 | Apple Pie | Second only to your mom's pie. | 12.99 |
| | 4 | Pecan Pie | If you like pecans, this is for you. | 10.99 |
| | 5 | Lemon Pie | Made with the best lemons in the world. | 11.99 |
| | 6 | Cupcakes | Your kids and the kid in you will love these. | 7.99 |
| ▸* | NULL | NULL | NULL | NULL |

6. データベース データを含むタブを閉じます。

## データベースのデータの表示

データベースにデータを入力したら、ASP.NET Web ページでデータを表示できます。表示するテーブルの行を選択するには、SQL 文を使います。これはデータベースに渡すコマンドです。

1. 左ペインで、「ファイル」ワークスペースをクリックします。
2. Web サイトのルートで、新たに ListProducts.cshtml という名前の CSHTML ページを作成します。
3. 既存のマークアップを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product ORDER BY Name";
}
<!DOCTYPE html>
<html>
    <head>
        <title>Small Bakery Products</title>
        <style>
            table, th, td {
                border: solid 1px #bbbbbb;
                border-collapse: collapse;
                padding: 2px;
            }
        </style>
    </head>
    <body>
        <h1>Small Bakery Products</h1>
        <table>
            <thead>
                <tr>
                    <th>Id</th>
                    <th>Product</th>
                    <th>Description</th>
                    <th>Price</th>
                </tr>
            </thead>
```

```
            <tbody>
                @foreach(var row in db.Query(selectQueryString)){
                <tr>
                    <td>@row.Id</td>
                    <td>@row.Name</td>
                    <td>@row.Description</td>
                    <td>@row.Price</td>
                </tr>
                }
            </tbody>
        </table>
    </body>
</html>
```

最初のコードブロックで、さきほど作成した SmallBakery.sdf ファイル（データベース）を開きます。Database.Open メソッドは、.sdf ファイルが Web サイトの App_Data フォルダーにあることを想定します。（拡張子.sdf を指定する必要がないことに注意してください。実際、指定すると、Open メソッドは正しく動作しません。）

**注意**　App_Data フォルダーは、ASP.NET でデータファイルを保持するために使われる特別なフォルダーです。詳細については、本章の「データベースへの接続」を参照してください。

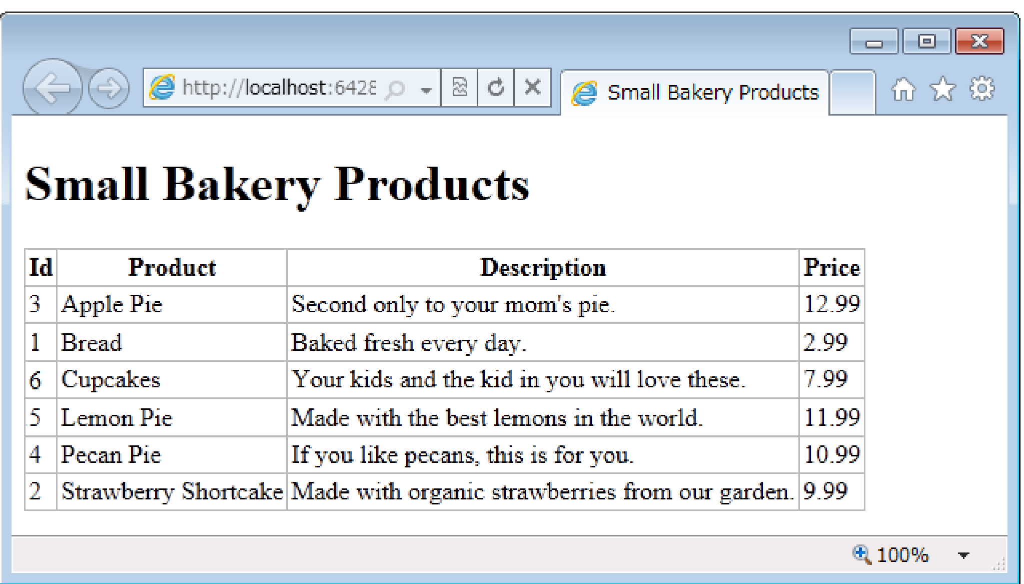
これで、次のような SQL の Select 文を使って、データベースへの問い合わせをリクエストします。

```
SELECT * FROM Product ORDER BY Name
```

この文で、Product は問い合わせるテーブルを識別します。*文字は、この問い合わせが、テーブルからすべての列を返すように指定するものです。（もし、いくつかの列だけを見たいのであれば、それぞれの列をカンマで区切って個別に列挙することもできます。）　Order By 節は、データをどのようにソートするか － ここでは Name 列を使うことを指示します。これはデータを各行の Name 列の値に基づきアルファベット順にソートするという意味です。

ページの本体では、マークアップはデータを表示するために使われる HTML テーブルを作成しています。<tbody>要素の中で、foreach ループを使い、問い合わせが返した各データ行を個別に取り出します。それぞれのデータ行のために、HTML テーブルの行（<tr>要素）を作成します。さらに、各列のために HTML テーブルのセル（<td>要素）を作成します。ループを実行するたびに、データベースで次に有効な行が row 変数に格納されます。（これは foreach 文で設定されます。）　行の個々の列を取り出すためには、row.Name や row.Description のように、使いたい列名を指定します。

4. このページをブラウザーで実行します。（実行する前に「ファイル」ワークスペースで、このページが選択されていることを確認してください。）　このページは、以下のようにリストを表示します。




Small Bakery Products

| Id | Product | Description | Price |
|---|---|---|---|
| 3 | Apple Pie | Second only to your mom's pie. | 12.99 |
| 1 | Bread | Baked fresh every day. | 2.99 |
| 6 | Cupcakes | Your kids and the kid in you will love these. | 7.99 |
| 5 | Lemon Pie | Made with the best lemons in the world. | 11.99 |
| 4 | Pecan Pie | If you like pecans, this is for you. | 10.99 |
| 2 | Strawberry Shortcake | Made with organic strawberries from our garden. | 9.99 |

**構造化問い合わせ言語（SQL）**

SQL は、ほとんどのリレーショナル データベースで、データベースのデータを管理するために使われている言語です。データを取り出したり更新したり、あるいはデータベース テーブルを作成、変更、管理できるようにするためのコマンドを含みます。SQL は、（WebMatrix で使われているような）プログラミング言語と違って、どうしたいかという考えをデータベースに伝えるもので、データベース処理は、それを元にどのようにデータを取得したり、タスクを実行するかを特定します。SQL コマンドの例と、その動作を紹介します。

SELECT Id, Name, Price FROM Product WHERE Price > 10.00 ORDER BY Name

これは、Product テーブルのレコードから、Price が 10 を超えている場合に、Id、Name、Price 列を取り出し、Name 列の値に基づいたアルファベット順で結果を返します。このコマンドは、条件に一致したレコードを含むか、条件に一致したものがなければ空の結果セットを返します。

INSERT INTO Product (Name, Description, Price) VALUES ("Croissant", "A flaky delight", 1.99)

これは、Name 列を"Croissant"、Description 列を"A flaky delight"、Price 列を 1.99 にした新しいレコードを Product テーブルに挿入します。

DELETE FROM Product WHERE ExpirationDate < "01/01/2008"

このコマンドは、ExpirationDate 列が 2008 年 1 月 1 日よりも前のレコードを Product テーブルから削除します。（もちろん、これは Product テーブルがそのような列を持っていると想定しています。） ここでは、日付は MM/DD/YYYY 形式で入力されていますが、ここはお使いのロケールで使われている形式で指定しなければなりません。

Insert Into や Delete コマンドは、結果セットを返しません。代わりに、そうしたコマンドによって影響を受けたレコードの数を返します。

（レコードの挿入や削除のような）いくつかの操作において、プロセスは、データベースにおける適切な権限を持っている必要があります。これは、実際のデータベースを使うためには、データベースへ接続するために、通常ユーザー名やパスワードを提供しなければなりません。

いくつもの SQL コマンドがありますが、それらはすべてこうしたパターンに沿っています。データベース テーブルを作成したり、テーブル中のレコードの数を数えたり、価格を計算したり、その他の多くの操作を実行できます。

## データベースのデータの挿入

このセクションでは、ユーザーが Product データベース テーブルに新しい製品（Product）を追加できるページを作成する方法を示します。新しい製品レコードが挿入された後、ページは、前セクションで作成した ListProducts.cshtml ページを使って更新されたテーブルを表示します。

このページは、ユーザーが入力したデータがデータベースにとって正しいかどうか確認する検証を含みます。たとえば、ページ中のコードは、すべての要求された列に入力された値を確認します。

**注意**　（レコードの挿入や削除のような）いくつかの操作において、プロセスは、データベースにおける適切な権限を持っている必要があります。（WebMatrix 中で扱っているテスト用のデータベースと違って）実際のデータベースでは、データベースへ接続するために、通常ユーザー名やパスワードを指定しなければなりません。

1. Web サイトで、新たに InsertProducts.cshtml という名前の CSHTML ファイルを作成します。
2. 既存のマークアップを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery");
    var Name = Request["Name"];
    var Description = Request["Description"];
    var Price = Request["Price"];

    if (IsPost) {

        // Read product name.
        Name = Request["Name"];
        if (Name.IsEmpty()) {
            ModelState.AddError("Name", "Product name is required.");
        }

        // Read product description.
        Description = Request["Description"];
        if (Description.IsEmpty()) {
            ModelState.AddError("Description",
                "Product description is required.");
        }

        // Read product price
        Price = Request["Price"];
        if (Price.IsEmpty()) {
            ModelState.AddError("Price", "Product price is required.");
```

```
        }

        // Define the insert query. The values to assign to the
        // columns in the Product table are defined as parameters
        // with the VALUES keyword.
        if(ModelState.IsValid) {
            var insertQuery = "INSERT INTO Product (Name, Description, Price) " +
                "VALUES (@0, @1, @2)";
            db.Execute(insertQuery, Name, Description, Price);
            // Display the page that lists products.
            Response.Redirect(@Href("~/ListProducts"));
        }
    }
}

<!DOCTYPE html>
<html>
<head>
    <title>Add Products</title>
    <style type="text/css">
        label {float:left; width: 8em; text-align: right;
            margin-right: 0.5em;}
        fieldset {padding: 1em; border: 1px solid; width: 35em;}
        legend {padding: 2px 4px; border: 1px solid; font-weight:bold;}
        .validation-summary-errors {font-weight:bold; color:red; font-size: 11pt;}
    </style>
</head>
<body>
    <h1>Add New Product</h1>

    @Html.ValidationSummary("Errors with your submission:")

    <form method="post" action="">
        <fieldset>
            <legend>Add Product</legend>
            <div>
                <label>Name:</label>
                <input name="Name" type="text" size="50" value="@Name" />
            </div>
            <div>
                <label>Description:</label>
                <input name="Description" type="text" size="50"
                    value="@Description" />
            </div>
            <div>
                <label>Price:</label>
                <input name="Price" type="text" size="50" value="@Price" />
            </div>
            <div>
                <label> </label>
                <input type="submit" value="Insert" class="submit" />
            </div>
        </fieldset>
    </form>
</body>
</html>
```

ページの本体は、名前（Name）、説明（Description）、価格（Price）を入力できる、3 つのテキストボックスを持つ HTML フォームを含みます。ユーザーが [Insert] ボタンをクリックすると、ページの先頭にあるコードが SmallBakery.sdf データベースへの接続を開きます。そして、Request オブジェクトを使ってユーザーが入力した値を取得し、それらの値をローカル変数に代入します。

それぞれの必要な列に対してユーザーが入力した値を検証するためには、次のようにします。

```
Name = Request["Name"];
if (Name.IsEmpty()) {
    ModelState.AddError("Name",
        "Product name is required.");
}
```

Name 列の値が空ならば、ModelState.AddError メソッドを使ってエラーメッセージを渡します。これを検査したいそれぞれの列で繰り返します。すべての列を検査したら、次のテストを実行します。

```
if(ModelState.IsValid) { // ... }
```

すべての列で問題なければ（空のものがない）、以下に示すように、データを挿入する SQL 文を作成し、実行します。

```
var insertQuery =
    "INSERT INTO Product (Name, Description, Price) VALUES (@0, @1, @2)";
```

挿入する値のために、パラメーター プレースホルダーを使っています（@0、@1、@2）。

**注意**　セキュリティ対策として、上記の例に示したとおり、SQL 文へ値を渡す際にはパラメーターを使います。これはユーザーデータを検証することに加えて、データベースに対して悪意のあるコマンドを送られることを防止します。（SQL インジェクション攻撃と呼ばれます。）

問い合わせを実行するためには、次の文を使って、プレースホルダーを置き換えるための値を含む変数を渡します。

```
db.Execute(insertQuery, Name, Description, Price);
```

Insert Into 文を実行した後、以下の行を使って製品を一覧するページをユーザーに創出します。

```
Response.Redirect("~/ListProducts");
```

検証が成功しなかった場合は、挿入をスキップします。代わりに、たまっているエラーメッセージ（もしあれば）を表示するヘルパーを使います。

```
@Html.ValidationSummary("Errors with your submission:")
```

　このマークアップのスタイルブロックは、.validation-summary-errors という名前の CSS クラス定義を含んでいることに注意してください。これは、すべての検証エラーを含む<div>要素においてデフォルトで使われる CSS クラスの名前です。この場合では、CSS クラスは検証のまとめを赤字かつ太字で表示するよう指示しますが、好みの形式で表示できるよう.validation-summary-errors クラスを定義することもできます。

3. ページをブラウザーで表示します。このページは、以下のようなフォームを表示します。

4. Price 列を空白にしておき、その他の、すべての列に値を入力します。
5. [Insert] をクリックします。ページは以下に示すようなエラーメッセージを表示します。（新しいレコードは作成されません。）

**Errors with your submission:**

- **Product price is required.**

**Add Product**

Name: Cherry Pie
Description: Made with organic cherries from our garden
Price:
Insert

## データベースのデータの更新

　テーブルにデータを入力した後、更新する必要があるかもしれません。以下の手順は、先に作成したデータを挿入するためのページに似た、2 つのページの作成方法を示しています。最初のページは、製品を表示し、ユーザーは変更するものを選択できます。2 つめのページは、実際にユーザーが内容を編集して、保存するためのものです。

**重要**　実用目的の Web サイトでは、通常、誰にデータの変更を許可するかを厳密に制限します。メンバーシップの設定やサイトに関する作業をユーザーに許可する方法についての詳細は、「第 16 章　セキュリティとメンバーシップの追加」を参照してください。

1. Web サイトで、新たに EditProducts.cshtml という名前の CSHTML ファイルを作成します。
2. ファイル中の既存のマークアップを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product ORDER BY Name";
}
<!DOCTYPE html>
<html>
<head>
    <title>Edit Products</title>
    <style type="text/css">
        table, th, td {
            border: solid 1px #bbbbbb;
            border-collapse: collapse;
            padding: 2px;
        }
    </style>
</head>
<body>
    <h1>Edit Small Bakery Products</h1>
    <table>
        <thead>
            <tr>
                <th> </th>
                <th>Name</th>
                <th>Description</th>
                <th>Price</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var row in db.Query(selectQueryString)) {
                <tr>
                    <td><a href="@Href("~/UpdateProducts", row.Id)">Edit</a></td>
                    <td>@row.Name</td>
                    <td>@row.Description</td>
                    <td>@row.Price</td>
                </tr>
            }
        </tbody>
    </table>
</body>
</html>
```

このページと先の ListProducts.cshtml ページの唯一の違いは、このページの HTML ページが [Edit] リンクとして表示される余分の列を含んでいることです。このリンクをクリックすると、（次に作成する）UpdateProducts.cshtml ページが表示され、選択したレコードを編集できます。[Edit] リンクを作成するコードを見てください。

```
<a href="@Href("~/UpdateProducts", row.Id)">Edit</a></td>
```

これは、HTML アンカー（<a>要素）と呼ばれるものを作成し、href 属性を動的に設定します。href 属性は、ユーザーがリンクをクリックしたときに表示するページを指定します。また、リンクへの現在の行の Id 値を渡します。このページを実行すると、ページソースは以下のようなリンクを含んでいます。

```
<a href="UpdateProducts/1">Edit</a></td>
<a href="UpdateProducts/2">Edit</a></td>
<a href="UpdateProducts/3">Edit</a></td>
```

href 属性が、「UpdateProducts/*n*」（n は製品番号）に設定されていることに注意してください。ユーザーがこれらのリンクをクリックすると、その結果の URL は次のようになります。

```
http://localhost:18816/UpdateProducts/6
```

言い換えると、編集する製品番号が URL として渡されるのです。

3. ブラウザーでこのページを表示します。ページは、以下のような形式でデータを表示します。

| | Name | Description | Price |
|---|---|---|---|
| Edit | Apple Pie | Second only to your mom's pie. | 12.99 |
| Edit | Bread | Baked fresh every day. | 2.99 |
| Edit | Cherry Pie | Made with organic cherries from our garden | 8.99 |
| Edit | Cupcakes | Your kids and the kid in you will love these. | 7.99 |
| Edit | Lemon Pie | Made with the best lemons in the world. | 11.99 |
| Edit | Pecan Pie | If you like pecans, this is for you. | 10.99 |
| Edit | Strawberry Shortcake | Made with organic strawberries from our garden. | 9.99 |

次に、実際にユーザーがデータを変更できるページを作成します。更新ページは、ユーザーが入力したデータを確認する検証を含みます。たとえば、このページのコードは、すべての必須項目に値が入力されているかどうかを確認します。

4. Web サイトで、新たに UpdateProducts.cshtml という名前の CSHTML ファイルを作成します。
5. このファイル中の既存のマークアップを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product WHERE Id=@0";

    var ProductId = UrlData[0];

    if (ProductId.IsEmpty()) {
        Response.Redirect(@Href("~/EditProducts"));
    }

    var row = db.QuerySingle(selectQueryString, ProductId);

    var Name = row.Name;
    var Description = row.Description;
    var Price = row.Price;
```

```
    if (IsPost) {
        Name = Request["Name"];
        if (String.IsNullOrEmpty(Name)) {
            ModelState.AddError("Name", "Product name is required.");
        }

        Description = Request["Description"];
        if (String.IsNullOrEmpty(Description)) {
            ModelState.AddError("Description",
                "Product description is required.");
        }

        Price = Request["Price"];
        if (String.IsNullOrEmpty(Price)) {
            ModelState.AddError("Price", "Product price is required.");
        }

        if(ModelState.IsValid) {
            var updateQueryString =
                "UPDATE Product SET Name=@0, Description=@1, Price=@2 WHERE Id=@3" ;
            db.Execute(updateQueryString, Name, Description, Price, ProductId);
            Response.Redirect(@Href("~/EditProducts"));
        }
    }
}

<!DOCTYPE html>
<html>
<head>
    <title>Add Products</title>
    <style type="text/css">
        label { float: left; width: 8em; text-align: right;
            margin-right: 0.5em;}
        fieldset { padding: 1em; border: 1px solid; width: 35em;}
        legend { padding: 2px 4px; border: 1px solid; font-weight: bold;}
        .validation-summary-errors {font-weight:bold; color:red; font-size:11pt;}
    </style>
</head>
<body>
    <h1>Update Product</h1>

    @Html.ValidationSummary("Errors with your submission:")

    <form method="post" action="">
        <fieldset>
            <legend>Update Product</legend>
            <div>
                <label>Name:</label>
                <input name="Name" type="text" size="50" value="@Name" />
            </div>
            <div>
                <label>Description:</label>
                <input name="Description" type="text" size="50"
                    value="@Description" />
            </div>
            <div>
                <label>Price:</label>
```

```
                <input name="Price" type="text" size="50" value="@Price" />
            </div>
            <div>
                <label> </label>
                <input type="submit" value="Update" class="submit" />
            </div>
        </fieldset>
    </form>
</body>
</html>
```

ページの本体は、製品が表示される場所とユーザーが編集する場所を含んでいます。表示する製品を取得するために、次の SQL 文を使います。

```
SELECT * FROM Product WHERE Id=@0
```

これは、ID が@0 パラメーターに渡された値に一致する製品を選択します。（Id はプライマリーキーであり、一意なので、これによって選ばれる製品レコードはひとつだけです。） この Select 文に渡される ID 値を取得するために、以下の構文を使って URL の一部としてページに渡される値を読み取ります。

```
var ProductId = UrlData[0];
```

製品レコードを実際に読み取るには、単一レコードだけを返す QuerySingle メソッドを使います。

```
var row = db.QuerySingle(selectQueryString, ProductId);
```

単一行が row 変数に返されます。以下のようにして、各列のデータを取り出し、ローカル変数に代入できます。

```
var Name = row.Name;
var Description = row.Description;
var Price = row.Price;
```

フォームのマークアップでは、以下のような埋め込みコードを使うことで、これらの値が自動的にそれぞれのテキストボックスに表示されます。

```
<input name="Name" type="text" size="50" value="@Name" />
```

コードのこの部分は、更新する製品レコードを表示します。レコードが表示されると、ユーザーはそれぞれの項目を編集できます。

ユーザーが ［Update］ ボタンをクリックしてフォームを送信すると、「if(IsPost)」ブロックのコードが実行されます。これは、Request オブジェクトからユーザーの入力を取り出し、変数に値を保存し、それぞれの列が入力されているかを確認します。検証がパスすれば、コードは以下の SQL の Update 文を作成します。

```
UPDATE Product SET Name=@0, Description=@1, Price=@2, WHERE ID=@3
```

SQL の Update 文で、更新する列を指定し、値を設定します。このコードでは、値はパラメーター プレースホルダー—@0、@1、@2 などを使って指定します。（前述したとおり、セキュリティのためには常にパラメーターを使って SQL 文に値を渡します。）

db.Execute メソッドを呼び出すと、値は対応する順序で SQL 文中のパラメーターに渡されます。

```
db.Execute(updateQueryString, Name, Description, Price, ProductId);
```

Update 文が実行された後、ユーザーを編集ページにリダイレクトして戻すために、以下のメソッドを呼び出します。

```
Response.Redirect(@Href("~/EditProducts"));
```

これにより、ユーザーはデータベースの更新されたデータを見ることができ、他の製品を編集できるようになります。

6. ページを保存します。
7. EditProducts.cshtml ページを実行し（更新ページではありません）、[Edit] をクリックして編集する製品を選びます。UpdateProducts.cshtml ページが表示され、選択レコードが表示されます。

**Update Product**

Name: Cherry Pie
Description: Made with organic cherries from our garden.
Price: 8.99
Update

8. 変更したら、[Update] をクリックします。更新されたデータを含む製品一覧が、再び表示されます。

## データベースのデータの削除

このセクションでは、Product データベース テーブルからユーザーに製品を削除させる方法を示します。この例は、2 つのページを含みます。最初のページでは、ユーザーは削除するレコードを選択します。削除するレコードは 2 番目のページに表示され、ここでユーザーがレコードを削除したいかどうかを確認します。

**重要** 実用目的の Web サイトでは、通常、誰にデータの変更を許可するかを厳密に制限します。メンバーシップの設定やサイトに関する作業をユーザーに許可する方法についての詳細は、「第 16 章 セキュリティとメンバーシップの追加」を参照してください。

1. Web サイトで、新たに ListProductsForDelete.cshtml という名前の CSHTML ファイルを作成します。
2. 既存のマークアップを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product ORDER BY Name";
}
<!DOCTYPE html>
<html>
<head>
    <title>Delete a Product</title>
    <style>
        table, th, td {
            border: solid 1px #bbbbbb;
            border-collapse: collapse;
            padding: 2px;
        }
    </style>
</head>
<body>
    <h1>Delete a Product</h1>
    <form method="post" action="" name="form">
        <table border="1">
            <thead>
                <tr>
                    <th> </th>
                    <th>Name</th>
                    <th>Description</th>
                    <th>Price</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var row in db.Query(selectQueryString)) {
                    <tr>
                        <td><a href="@Href("~/DeleteProduct", row.Id)">Delete</a></td>
                        <td>@row.Name</td>
                        <td>@row.Description</td>
                        <td>@row.Price</td>
                    </tr>
                }
            </tbody>
        </table>
    </form>
</body>
</html>
```

このページは、先の EditProducts.cshtml ページに似ています。ただし、それぞれの製品に対する [Edit] リンクを表示する代わりに、[Delete] リンクを表示します。[Delete] リンクは、マークアップ中に以下の埋め込みコードを使って作成されます。

```
<a href="@Href("~/DeleteProduct", row.Id)">Delete</a>
```

これは、ユーザーがリンクをクリックするときに以下のように見える URL を作成します。

```
http://<server>/DeleteProduct/4
```

この URL は、DeleteProduct.cshtml という名前のページ（次に作成します）を呼び出し、削除する製品の ID を渡します。

3. ファイルを保存します。このファイルは開いたままにします。
4. もうひとつ、DeleteProduct.cshtml という名前の CSHTML ファイルを作成し、既存のコンテンツを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery");
    var ProductId = UrlData[0];
    if (ProductId.IsEmpty()) {
        Response.Redirect(@Href("~/ListProductsForDelete"));
    }
    var prod = db.QuerySingle("SELECT * FROM PRODUCT WHERE ID = @0", ProductId);
    if( IsPost && !ProductId.IsEmpty()) {
        var deleteQueryString = "DELETE FROM Product WHERE Id=@0";
        db.Execute(deleteQueryString, ProductId);
        Response.Redirect("~/ListProductsForDelete");
    }

}
<!DOCTYPE html>
<html
<head>
    <title>Delete Product</title>
</head>
<body>
    <h1>Delete Product - Confirmation</h1>
    <form method="post" action="" name="form">
        <p>Are you sure you want to delete the following product?</p>

    <p>Name: @prod.Name <br />
        Description: @prod.Description <br />
        Price: @prod.Price</p>
    <p><input type="submit" value="Delete" /></p>
    </form>
</body>
</html>
```

このページは、ListProductsForDelete.cshtml から呼び出され、ユーザーが製品を削除したいかどうかを確認します。削除する製品を表示するため、以下のコードを使って URL から削除する製品の ID を取得します。

```
var ProductId = UrlData[0];
```

その後、このページは、実際にレコードを削除するためのボタンをクリックすることでユーザーに確認します。これは、セキュリティ面で重要なことです。データの更新や削除のような機微な操作を実行するとき、そのような操作は、GET 操作ではなく常に POST 操作を使って処理するべきです。もし、GET を使って削除操作を実行するように設定すると、誰もが「http://<server>/DeleteProduct/4」という URL を使って、データベースからあらゆるものを削除できてしまいます。確認を追加し、ページのコードを POST だけを使って実行することで、サイトにある程度のセキュリティを追加できます。

実際の削除操作は、以下のコードを使って処理されます。最初は、これが POST 操作で行われていること、続いて ID が空でないことを確認します。

```
if( IsPost && !ProductId.IsEmpty()) {
    var deleteQueryString = "DELETE FROM Product WHERE Id=@0";
    db.Execute(deleteQueryString, ProductId);
    Response.Redirect("~/ListProductsForDelete");
}
```

このコードは、指定したレコードを削除する SQL 文を実行し、ユーザーを一覧ページにリダイレクトします。

5. ブラウザーで ListProductsForDelete.cshtml を実行します。

| | Name | Description | Price |
|---|---|---|---|
| Delete | Apple Pie | Second only to your mom's pie. | 12.99 |
| Delete | Bread | Baked fresh every day. | 2.99 |
| Delete | Cherry Pie | Made with organic cherries from our garden | 8.99 |
| Delete | Cupcakes | Your kids and the kid in you will love these. | 7.99 |
| Delete | Lemon Pie | Made with the best lemons in the world. | 11.99 |
| Delete | Pecan Pie | If you like pecans, this is for you. | 10.99 |
| Delete | Strawberry Shortcake | Made with organic strawberries from our garden. | 9.99 |

6. 製品のひとつの [Delete] をクリックします。DeleteProduct.cshtml ページが表示されて、レコードを削除したいか確認されます。
7. [Delete] ボタンをクリックします。製品は削除され、ページは更新された製品一覧で再表示されます。

**データベースへの接続**

データベースに接続するには、2 つの方法があります。ひとつは、Database.Open メソッドを使い、データベースファイル名（拡張子.sdf を除いたもの）を指定する方法です。

var db = Database.Open("SmallBakery");

Open メソッドは、.sdf ファイルが Web サイトの「App_Data」フォルダーに置かれていることを想定します。このフォルダーは、特別にデータを保持するために用意されたものです。たとえば、Web サイトがデータを読み書きするための適切な許可を提供し、WebMatrix はこのフォルダーのファイルへのアクセスを許可しません。

2 つめは、接続文字列を使う方法です。接続文字列は、データベースに接続するための情報を含んでいます。これは、ファイルパスや、ローカルやリモートサーバー上にある SQL Server データベースの名前を、サーバーに接続するためのユーザー名やパスワードとともに含みます。（データが、ホスティング プロバイダー サイトのような、中央管理された SQL Server に置かれている場合は、常にデータ接続情報を指定した接続文字列を使います。）

WebMatrix では、通常、接続文字列は Web.config という名前の XML ファイルに保存されます。その名が示す通り、Web サイトのルートにある Web.config を使って、サイトの設定情報を保存できます。これにはサイトが必要とする接続文字列も含まれます。Web.config 中の接続文字列は、たとえば以下のようなものです。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add
      name="SQLServerConnectionString"
      connectionString="server=myServer;database=myDatabase;uid=username;pwd=password"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

この例では、接続文字列は（ローカルの.sdf ファイルと違って）どこかにあるサーバーで実行されている SQL Server のインスタンス中にあるデータベースを指しています。myServer や myDatabase は適切な名前で置き換え、username や password には SQL Server のログイン値を指定する必要があります。（username と password の値は、Windows 認証と同じ場合や、ホスティング プロバイダーがサーバーに対してログインを提供している場合には必要ありません。必要な値は管理者に確認してください。）

Database.Open メソッドは、データベース.sdf ファイルの名前も、Web.config 中に保存された接続文字列の名前も渡すことができるため、柔軟性があります。以下の例は、上記の例に示された接続文字列を使って、データベースに接続する方法を示しています。

```
@{
    var db = Database.Open("SQLServerConnectionString");
}
```

前述したとおり、Database.Open メソッドは、データベース名も接続文字列も渡すことができ、どちらを使うかを決定します。これは、Web サイトを配布（発行）する際に非常に有用です。サイトを開発したり、テストしたりするときには、「App_Data」フォルダー中の.sdf ファイルを使うことができます。その後、実際に運用するサーバーに移動するときは、.sdf ファイルと同じ名前を持ち、ホスティング プロバイダーのデータベースを指すようにした Web.config の接続文字列を使うことができ、コードを変更する必要はまったくありません。

最後に、接続文字列を直接処理したい場合は、Database.OpenConnectionString メソッドを使い、Web.config ファイル中の名前を選ぶ代わりに、実際の接続文字列を渡します。これは、ページを実行する前のような、何らかの理

由で接続文字列（.sdf ファイル名のような値）にアクセスできないような場合に役立ちます。しかし、ほとんどの場合、本章で説明した Database.Open を使うことができます。

## その他のリソース

- SQL Server Compact
- Connecting to a SQL Server or MySQL Database in WebMatrix

# 第６章　　データをグリッドで表示する

本章では、データを HTML の表形式（グリッド形式）で表示するヘルパーの使い方について解説します。

ここでは次のことを学びます。

- WebGrid ヘルパーを使って Web ページにデータを表示する方法
- グリッド中に表示されるデータを書式化する方法
- グリッドにページングを追加する方法

本章で紹介される ASP.NET プログラミングの機能は以下のとおりです。

- WebGrid ヘルパー

## WebGrid ヘルパー

前章では、ページ中にデータを表示するための作業をすべて自分自身で行っていました。しかし、データを表示するより簡単な方法があります。WebGrid ヘルパーを使うことです。このヘルパーは、表示したいデータを HTML の表形式でレンダリングできます。ヘルパーは、書式化のオプションもサポートしており、データをページングする方法や、ユーザーが列ヘッダーをクリックすることでソートさせることもできます。

## WebGrid ヘルパーを使ってデータを表示する

以下の手順は、もっとも簡単な設定を使って WebGrid ヘルパーでデータを表示する方法を示しています。

1. 「第 5 章　データを扱う」で作成した Web サイトを開きます。

   前章の手順を実行していなければ、今、ここで実行する必要はありません。しかし、第 5 章の冒頭で作成した SmallBakery.sdf データベースファイルは必要です。このファイルは、これから作業する Web サイトの「App_Data」フォルダーに配置しなければなりません。

2. Web サイトで、新たに WebGridBasic.cshtml という名前の CSHTML ファイルを作成します。
3. 既存のマークアップを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery") ;
    var selectQueryString = "SELECT * FROM Product ORDER BY Id";
    var data = db.Query(selectQueryString);
    var grid = new WebGrid(data);
}
<!DOCTYPE html>
<html>
    <head>
        <title>Displaying Data Using the WebGrid Helper</title>
    </head>
    <body>
        <h1>Small Bakery Products</h1>
```

```
        <div id="grid">
            @grid.GetHtml()
        </div>
    </body>
</html>
```

このコードは、まず SmallBakery.sdf データベース ファイルを開き、SQL の Select 文を作成します。

```
SELECT * FROM Product ORDER BY Id
```

Data という名前の変数は、SQL の Select 文で返されたデータをあらわします。そして、WebGrid ヘルパーは、データから新しいグリッドを作成するために使います。

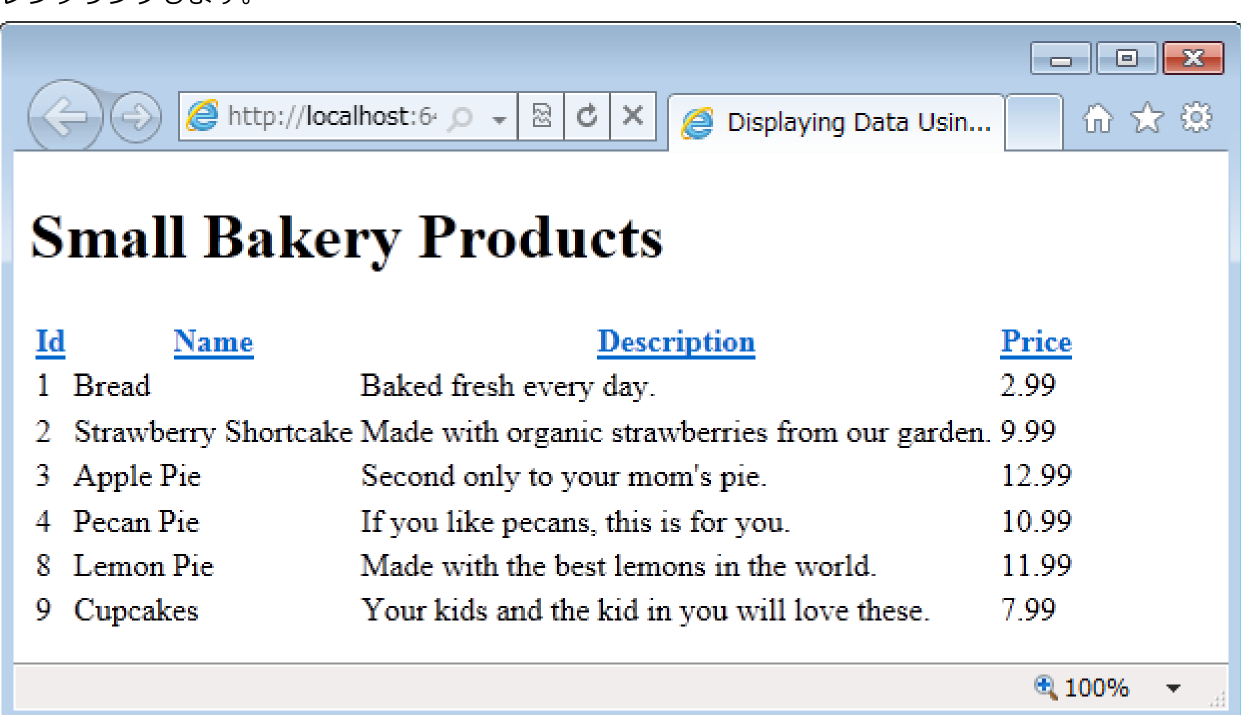
```
var data = db.Query(selectQueryString);
var grid = new WebGrid(data);
```

このコードは、新しい WebGrid オブジェクトを作成して、grid 変数に代入します。ページの本体では、WebGrid ヘルパーに対して次のようなコードを使ってデータをレンダリングします。

```
@grid.GetHtml()
```

grid 変数は、WebGrid オブジェクトを作成したときに、作成した値です。

4. ページを実行します。（実行する前に、「ファイル」ワークスペースで、このページが選択されていることを確認してください。） WebGrid ヘルパーは、SQL の Select 文に基づいて選択されたデータを含む HTML の表をレンダリングします。




| Id | Name | Description | Price |
|---|---|---|---|
| 1 | Bread | Baked fresh every day. | 2.99 |
| 2 | Strawberry Shortcake | Made with organic strawberries from our garden. | 9.99 |
| 3 | Apple Pie | Second only to your mom's pie. | 12.99 |
| 4 | Pecan Pie | If you like pecans, this is for you. | 10.99 |
| 8 | Lemon Pie | Made with the best lemons in the world. | 11.99 |
| 9 | Cupcakes | Your kids and the kid in you will love these. | 7.99 |

列名をクリックすると、その列のデータによって表がソートされることに注意してください。

ご覧のとおり、WebGrid ヘルパーにもっとも単純なコードを使うだけでも、データを表示（およびソート）するときに必要な膨大な作業を肩代わりしてくれます。それだけではありません。本章の残りの部分では、WebGrid ヘルパーをカスタマイズして以下のような処理を行う方法について説明しています。

- 表示するデータ列を指定し、それらの列の表示を書式化する方法
- グリッド全体のスタイル指定
- データのページング

## 表示する列の指定と書式化

デフォルトでは、WebGrid ヘルパーは SQL の問い合わせで返されるすべてのデータ列を表示します。このデータの表示は、以下の方法でカスタマイズできます。

- ヘルパーが表示する列、およびその順序を指定します。これは、SQL の問い合わせで返されるデータ列の一部だけを表示したい場合に使います。
- データをどのように表示するかの書式を指定します。たとえば、金額をあらわすためにデータに（$のような）通貨記号を追加します。

以下の手順は、個々の列を書式化する WebGrid ヘルパーのオプションを使います。

1. Web サイトで、新たに WebGridColumnFormat.cshtml という名前のページを作成します。
2. 既存のマークアップを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery") ;
    var selectQueryString = "SELECT * FROM Product ORDER BY Id";
    var data = db.Query(selectQueryString);
var grid = new WebGrid(data);
}
<!DOCTYPE html>
<html>
    <head>
        <title>Displaying Data Using the WebGrid Helper (Custom Formatting)</title>
        <style type="text/css">
            .product { width: 200px; font-weight:bold;}
        </style>
    </head>
    <body>
        <h1>Small Bakery Products</h1>
        <div id="grid">
            @grid.GetHtml(
                columns: grid.Columns(
                    grid.Column("Name", "Product", style: "product"),
```

```
                    grid.Column("Description", format:@<i>@item.Description</i>),
                    grid.Column("Price", format:@<text>$@item.Price</text>)
                )
            )
        </div>
    </body>
</html>
```

この例は、前述のものに似ていますが、ページの本体では grid.GetHtml を呼び出してグリッドをレンダリングするときに、表示する列と、それを表示する方法を指定しています。以下のコードは、どの項目を、どの順序で表示するかを指定する方法を示しています。

```
@grid.GetHtml(
    columns: grid.Columns(
        grid.Column("Name", "Product", style: "product"),
        grid.Column("Description", format:@<i>@item.Description</i>),
        grid.Column("Price", format:@<text>$@item.Price</text>)
    )
)
```

ヘルパーに表示する列を伝えるには、WebGrid ヘルパーの GetHtml メソッドに columns パラメーターを含み、列のコレクションを渡さなければなりません。このコレクションには、表示したいそれぞれの列を指定します。grid.Column オブジェクトを含むことで表示する個々の列を指定できます。この例では、WebGrid オブジェクトは 3 つの列、つまり Name、Description、Price だけを表示します。（これらの列は、SQL の問い合わせ結果に含まれていなければなりません。ヘルパーは問い合わせが返さなかった列を表示することはできません。）

ここで、グリッドに列名を渡すだけでなく、書式化についての指示も渡せることに注意してください。この例では、以下のコードを使って Name 列を表示します。

```
grid.Column("Name", "Product", style: "product")
```

これは WebGrid ヘルパーに次のことを伝えています。

- Name データ列の値を表示する
- 列のヘッダーとして、ヘッダーのデフォルト名（ここでは"Name"）ではなく、"Product"を使う
- "product"という名前の CSS スタイルクラスを適用する。このページの例では、CSS クラスは列幅（200 ピクセル）とフォント形式（太字）を設定する。

この例では、Description 列は次のコードを使います。

```
grid.Column("Description", format:@<i>@item.Description</i>)
```

これは、ヘルパーに Description 列を表示することを伝えます。次のような HTML マークアップで、データ列で値を囲む表記で書式を指定します。

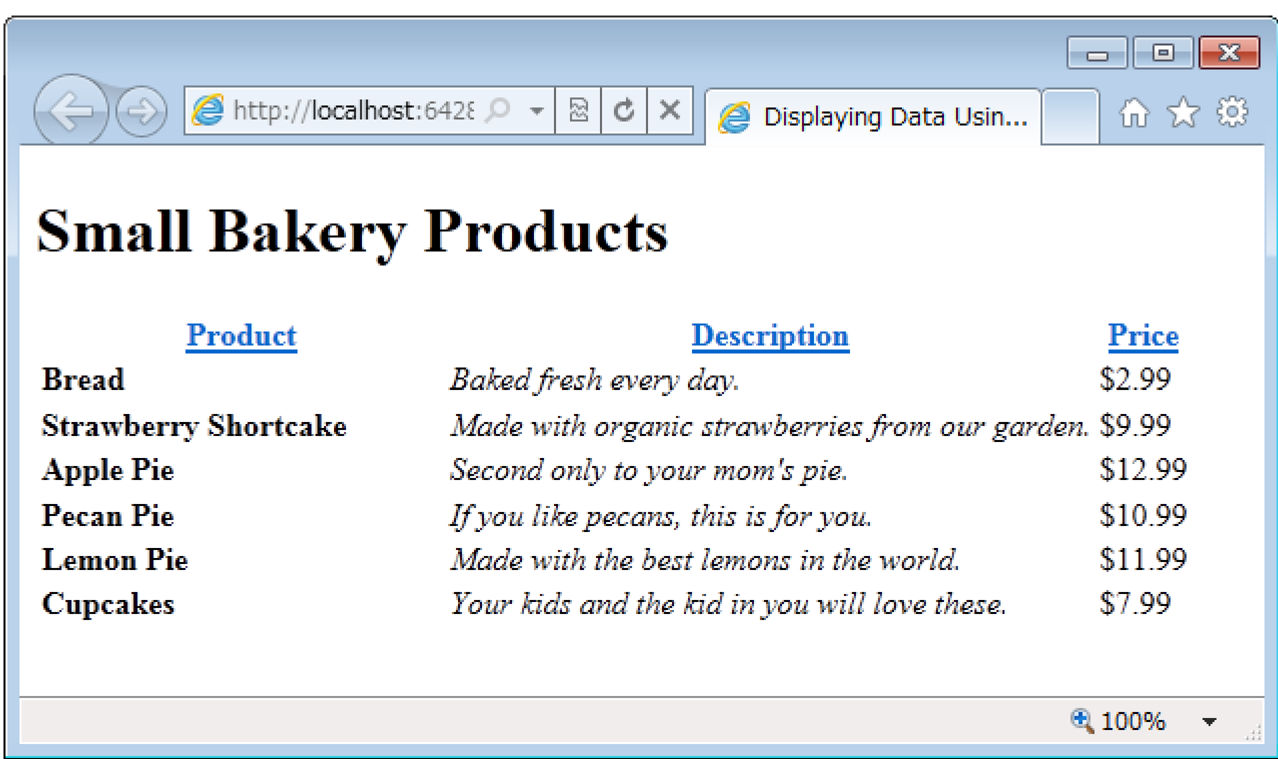
```
@<i>@item.Description</i>
```

この例では、Price 列にもう一つの方法として format プロパティを指定する方法を示しています。

```
grid.Column("Price", format:@<text>$@item.Price</text>)
```

これはレンダリングする HTML マークアップを指定し、列の値の前にドル記号（$）を追加します。

3. このページをブラウザーで表示します。




今回は 3 つの列しか表示されていません。Name 列は、列のヘッダーと、サイズ、フォントがカスタマイズしています。Description 列は、イタリック体（斜体）で、Price 列はドル記号付きで表示されます。

## グリッド全体をスタイル化する

表示する個々の列をどのように表示するかを指定するだけでなく、グリッド全体を書式化できます。このためには、CSS クラスを定義して、HTML の表をどのようにレンダリングするかを指定します。

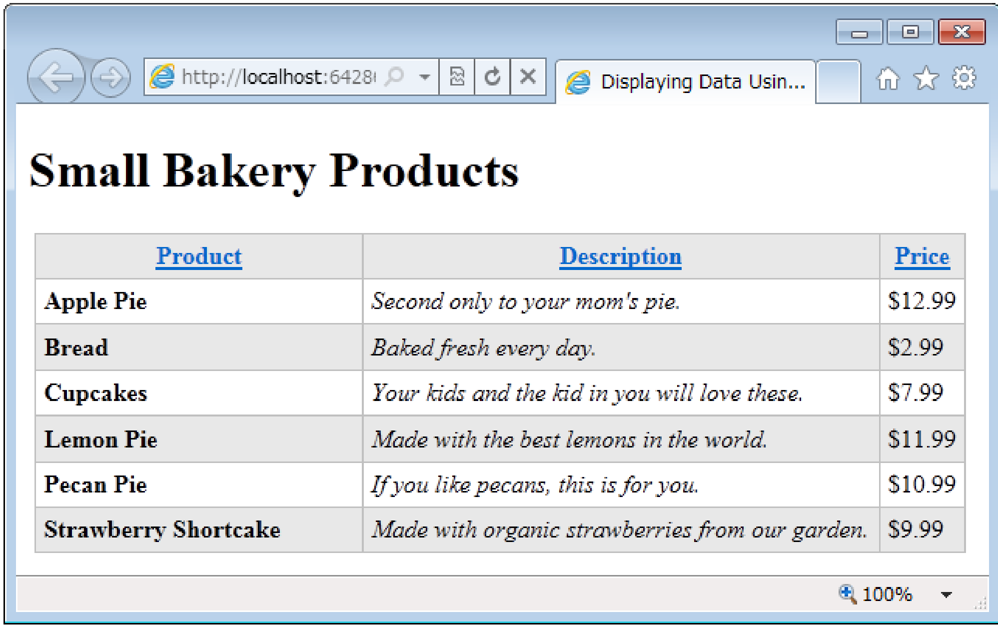
1. Web サイトで、新たに WebGridTableFormat.cshtml という名前のページを作成します。
2. 既存のマークアップを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product ORDER BY Id";
    var data = db.Query(selectQueryString);
    var grid = new WebGrid(source: data, defaultSort: "Name");
}
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>Displaying Data Using the WebGrid Helper (Custom Table Format-
ting)</title>
        <style type="text/css">
            .grid { margin: 4px; border-collapse: collapse; width: 600px; }
            .head { background-color: #E8E8E8; font-weight: bold; color: #FFF; }
            .grid th, .grid td { border: 1px solid #C0C0C0; padding: 5px; }
            .alt { background-color: #E8E8E8; color: #000; }
            .product { width: 200px; font-weight:bold;}
        </style>
    </head>
    <body>
        <h1>Small Bakery Products</h1>
        <div id="grid">
            @grid.GetHtml(
                tableStyle: "grid",
                headerStyle: "head",
                alternatingRowStyle: "alt",
                columns: grid.Columns(
                    grid.Column("Name", "Product", style: "product"),
                    grid.Column("Description", format:@<i>@item.Description</i>),
                    grid.Column("Price", format:@<text>$@item.Price</text>)
                )
            )
        </div>
    </body>
</html>
```

このコードは、前述の例において、新しいスタイルクラス（grid、head、grid th、grid td、および alt）を作成する方法を示しています。続いて、grid.GetHtml メソッドでこれらのスタイルを tableStyle、headerStyle および alternatingRowStyle パラメーターを使ってグリッドの様々な要素に割り当てます。

3. このページをブラウザーで表示します。今回は、表全体に適用されるさまざまなスタイルを使ってグリッドが表示されます。たとえば、1 行おきに色が指定されています。



## データをページングする

すべてのデータをグリッドとして一度に表示する代わりに、ユーザーにデータをページングさせることもできます。ここで作業するデータは少ないので、ページングはさほど重要ではありません。しかし、何百、何千というデータ行を表示する場合は、ページングは非常に役立ちます。

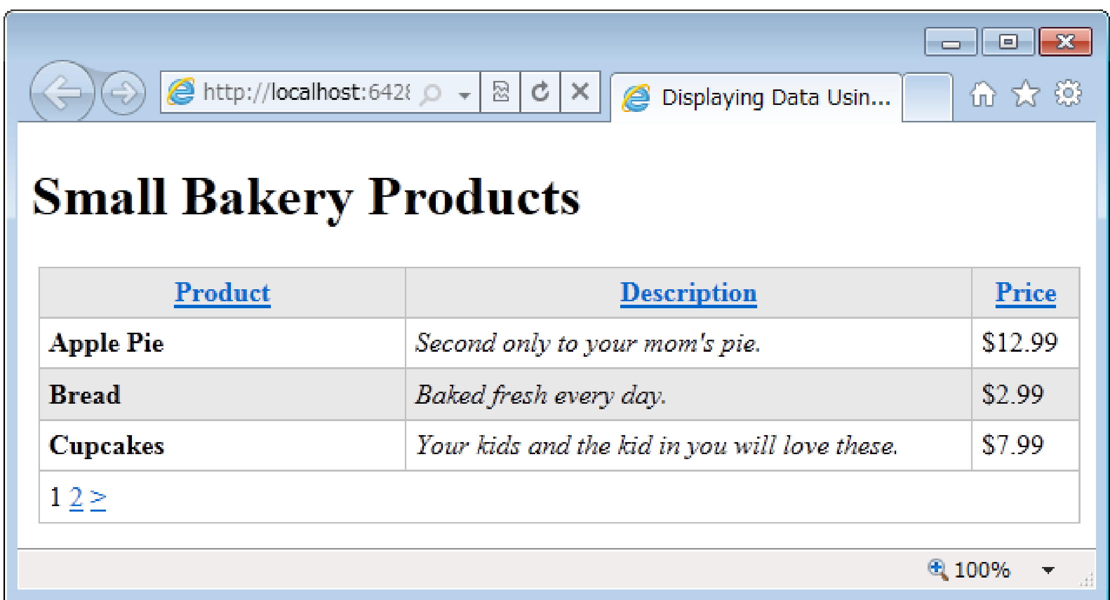
レンダリングするグリッドにページングを追加するには、WebGrid ヘルパーに追加のパラメーターを指定します。

1. Web サイトに、新たに WebGridPaging.cshtml という名前のページを作成します。
2. 既存のマークアップを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product ORDER BY Id";
    var data = db.Query(selectQueryString);
    var grid = new WebGrid(source: data,
                           defaultSort: "Name",
                           rowsPerPage: 3);
}
<!DOCTYPE html>
<html>
    <head>
        <title>Displaying Data Using the WebGrid Helper (with Paging)</title>
        <style type="text/css">
            .grid { margin: 4px; border-collapse: collapse; width: 600px; }
            .head { background-color: #E8E8E8; font-weight: bold; color: #FFF; }
            .grid th, .grid td { border: 1px solid #C0C0C0; padding: 5px; }
            .alt { background-color: #E8E8E8; color: #000; }
            .product { width: 200px; font-weight:bold;}
        </style>
    </head>
    <body>
        <h1>Small Bakery Products</h1>
        <div id="grid">
            @grid.GetHtml(
                tableStyle: "grid",
                headerStyle: "head",
                alternatingRowStyle: "alt",
                columns: grid.Columns(
                    grid.Column("Name", "Product", style: "product"),
                    grid.Column("Description", format:@<i>@item.Description</i>),
                    grid.Column("Price", format:@<text>$@item.Price</text>)
                )
            )
        </div>
    </body>
</html>
```

このコードは、前述のサンプルで WebGrid オブジェクトを作成するときに rowsPerPage パラメーターを追加して拡張したものです。このパラメーターは、表示する行の数を設定します。このパラメーターを追加すると、自動的にページングが有効になります。

3. このページをブラウザーで表示します。3 つの行しか表示されないことに注意してください。グリッドの下部には、残りのデータ行をページングできるようなコントロールがあります。




## その他のリソース

- 第 5 章　データを扱う
- 第 7 章　データをグラフで表示する
- ASP.NET Web Pages with Razor Syntax Reference

## 第７章　　　データをグラフで表示する

本章では、データをグラフで表示する方法を解説します。

前章では、データを手作業でグリッドに表示する方法を学びました。本章では、Chart ヘルパーを使ってデータを表示する方法を解説します。

ここでは次のことを学びます。

- データをグラフで表示する方法
- 組み込みテーマを使ってグラフをスタイル化する方法
- グラフを保存する方法、およびパフォーマンス改善のためにキャッシュする方法

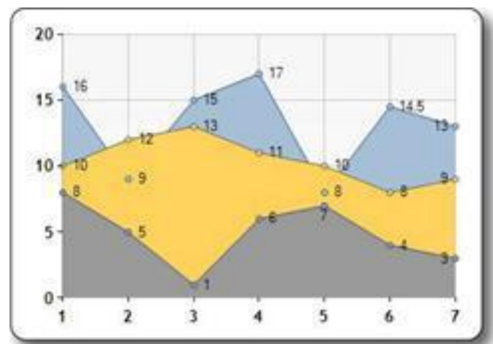
本章で紹介する ASP.NET プログラミングの機能は、以下の通りです。

- Chart ヘルパー

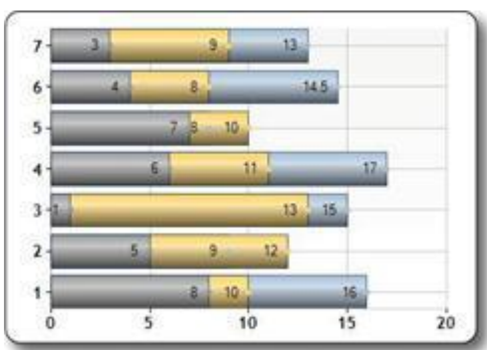
### Chart ヘルパー

データをグラフィカルな形式で表示したい場合、Chart ヘルパーが使えます。Chart ヘルパーは、さまざまな形式のグラフでデータを表示するイメージをレンダリングできます。また、書式化やラベルのために多くのオプションをサポートしています。Chart ヘルパーは、Microsoft Excel や他のツールなどで知られている、30 種類以上のグラフをレンダリングできます。これには、面グラフ（area chart）、横棒グラフ（bar chart）、縦棒グラフ（column chart）、線グラフ（line chart）、円グラフ（pie chart）、あるいは株価グラフ（stock chart）のような特殊なグラフを含みます。

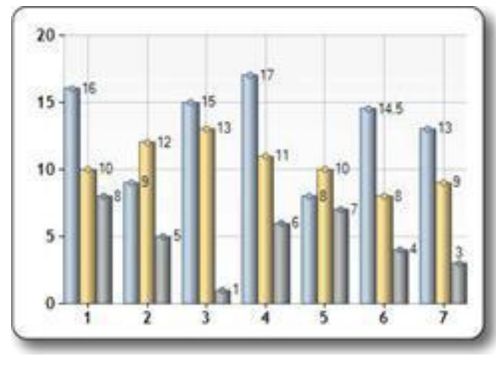
面グラフ（area chart）



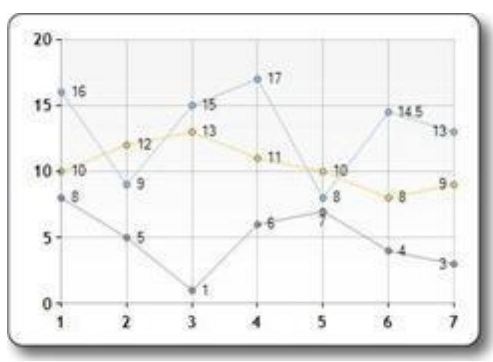

横棒グラフ（bar chart）



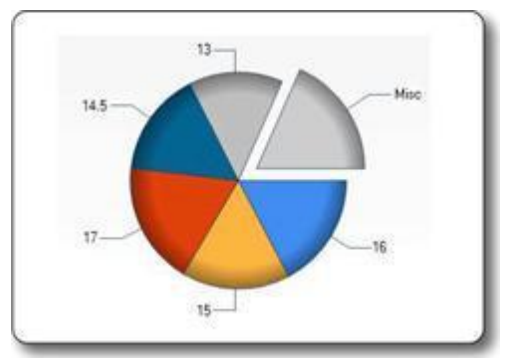

縦棒グラフ（column chart）



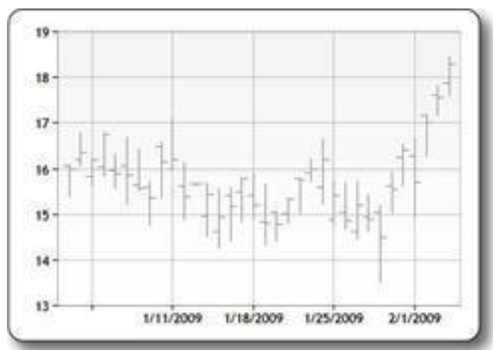

線グラフ（line chart）



円グラフ（pie chart）

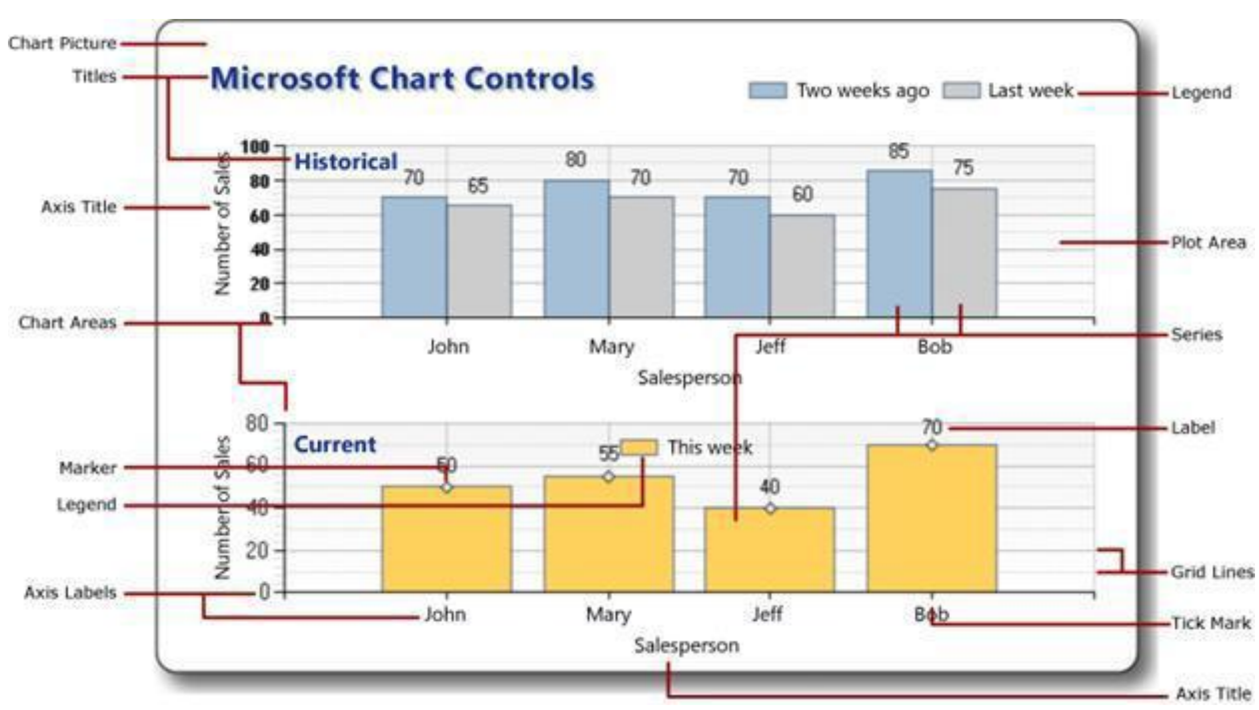



株価グラフ（stock chart）




## グラフの要素

グラフは、データに加えて凡例や軸、系列など追加の要素を表示します。以下の図は、グラフの多くの要素を表示しています。これらは、Chart ヘルパーを使うときにカスタマイズできます。本章では、これらの要素のいくつか（全部ではありません）を設定する方法を示します。




## データからグラフを作成する

グラフに表示するデータとしては、配列にあるもの、データベースから返された結果、あるいは XML ファイル中のデータが使えます。

### 配列を使う

「第 2 章　Razor 構文を使った ASP.NET Web プログラミングの紹介」で解説したとおり、配列とは、単一の変数に同じような項目の集まりを保存できるものです。グラフに含めたいデータを含む配列が使えます。

以下の手順は、デフォルトのグラフを使って、配列中のデータからグラフを作成する方法を示しています。また、このグラフをページ中に表示する方法も示します。

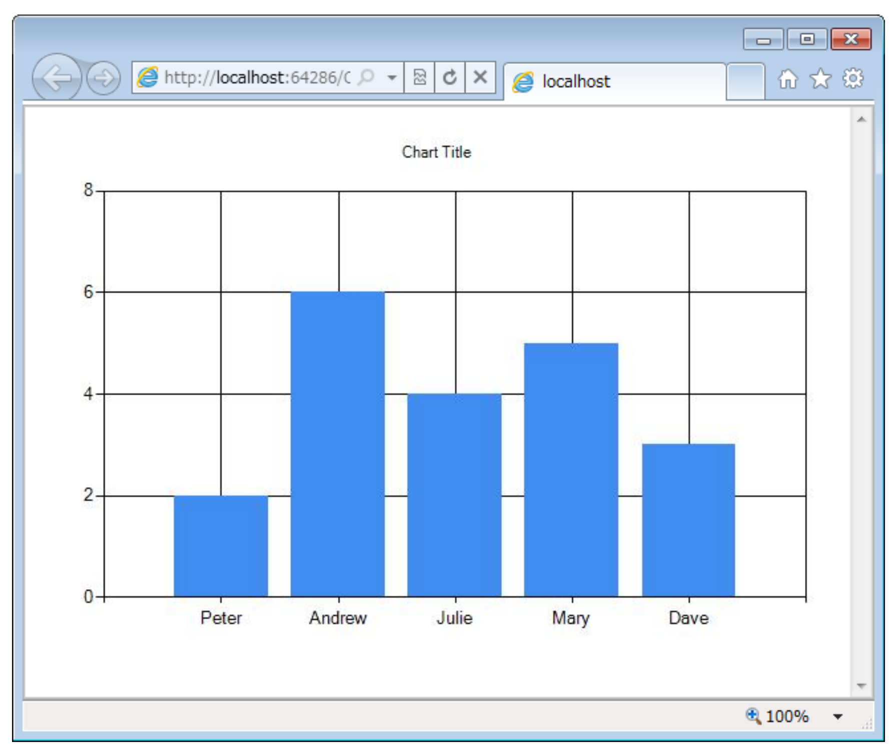
1. 新たに ChartArrayBasic.cshtml という名前のファイルを作成します。
2. 既存のコードを以下で置き換えます。

```
@{
    var myChart = new Chart(width: 600, height: 400)
        .AddTitle("Chart Title")
        .AddSeries(
            name: "Employee",
            xValue: new[] { "Peter", "Andrew", "Julie", "Mary", "Dave" },
            yValues: new[] { "2", "6", "4", "5", "3" })
        .Write();
}
```

　このコードは、まず新しいグラフを作成し、その幅と高さを設定します。AddTitle メソッドを使ってグラフのタイトルを指定します。データを追加するには、AddSeries メソッドを使います。この例では、AddSeries メソッドの name、xValue、yValues パラメーターを使います。name パラメーターは、グラフの凡例に表示されます。xValue パラメーターは、グラフの水平軸に沿って表示されるデータの配列を含みます。yValues パラメーターは、グラフの垂直座標を描画するために使うデータの配列を含みます。

　Write メソッドは、実際にグラフをレンダリングします。ここでは、グラフの種類を指定していないため、Chart ヘルパーはデフォルトのグラフ、つまり縦棒グラフとしてレンダリングします。

3. このページをブラウザーで実行します。（実行する前に、このページが「ファイル」ワークスペースで選択されていることを確認してください。）　ブラウザーはグラフを表示します。



## グラフのデータとしてデータベースの問い合わせを使う

グラフ化したい情報がデータベースにある場合、データベースの問い合わせを実行した後、その結果得られたデータを使ってグラフを作成します。以下の手順は、前述した例で作成したデータを読み取り、データを表示する方法を示しています。

1. Web サイトのルートに「App_Data」フォルダーがなければ、このフォルダーを追加します。
2. 「App_Data」フォルダーで、「第 5 章　データを扱う」で作成した SmallBakery.sdf という名前のデータベースファイルを追加します。
3. 新たに ChartDataQuery.cshtml という名前のファイルを作成します。
4. 既存のコードを以下で置き換えます。

```
@{
    var db = Database.Open("SmallBakery");
    var data = db.Query("SELECT Name, Price FROM Product");
    var myChart = new Chart(width: 600, height: 400)
        .AddTitle("Product Sales")
        .DataBindTable(dataSource: data, xField: "Name")
        .Write();
}
```

このコードは、まず SmallBakery データベースを開き、db という名前の変数に代入します。この変数は、このデータベースを読み書きするために使われる Database オブジェクトをあらわしています。次に、このコードは SQL 問い合わせを実行し、各製品の名前と価格を取得します。コードは、新しいグラフを作成し、グラフの DataBindTable メソッドを呼び出すことで、データベースの問い合わせを渡します。このメソッドは 2 つのパラメーターを取ります。ひとつは dataSource パラメーターで問い合わせが返すデータをあらわし、もうひとつの xField パラメーターはどのデータ列をグラフの X 軸として使うかを指定するものです。

DataBindTable メソッドを使う代わりに、Chart ヘルパーの AddSeries メソッドを使うこともできます。AddSeries メソッドでは、xValue と yValues パラメーターを設定できます。たとえば、次のように DataBindTable メソッドを使う代わりに、

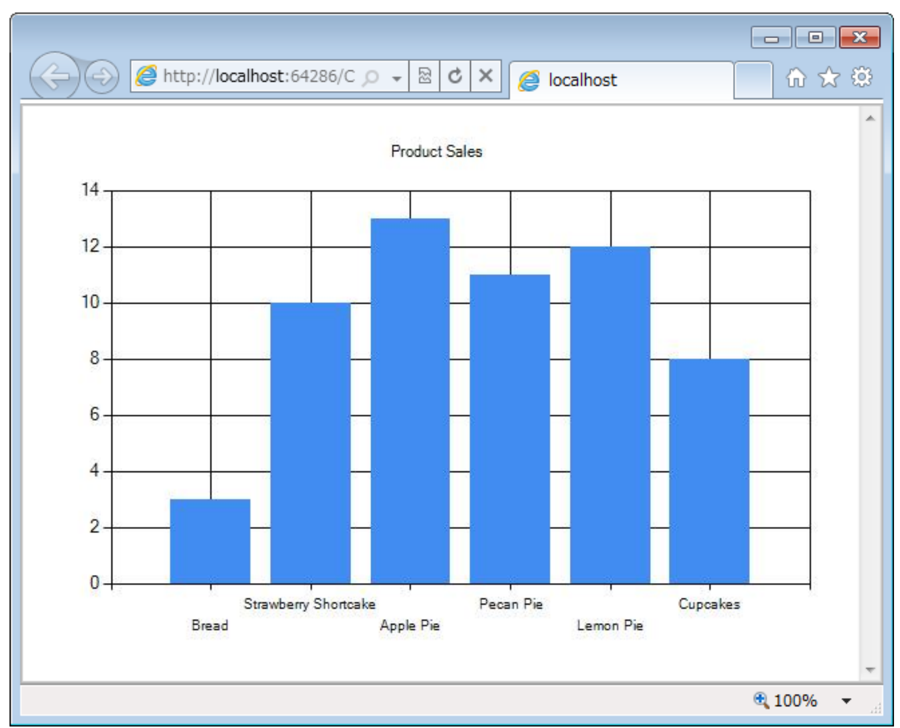
```
.DataBindTable(data, "Name")
```

次のように AddSeries メソッドを使うことができます。

```
.AddSeries("Default",
    xValue: data, xField: "Name",
    yValues: data, yFields: "Price")
```

どちらのレンダリング結果も同じです。グラフの種類やデータを明示的に指定できるため、AddSeries メソッドはより柔軟性があります。ただし、そうした拡張的な柔軟性が必要なければ DataBindTable の方が使いやすいでしょう。

5.　このページをブラウザーで実行します。




## XML データを使う

グラフを使うための 3 つめの選択肢は、グラフのためのデータとして XML ファイルを使うことです。このために、XML ファイルは XML の構造を記述したスキーマファイル（.xsd ファイル）も必要とします。以下の手順は、XML ファイルからデータを読み取る方法を示しています。

1.　「App_Data」フォルダーで、新たに data.xml という名前のファイルを作成します。
2.　既存の XML を以下で置き換えます。これは架空の会社の従業員に関する XML データです。

```
<?xml version="1.0" standalone="yes" ?>
<NewDataSet xmlns="http://tempuri.org/data.xsd">
    <Employee>
        <Name>Erin</Name>
        <Sales>10440</Sales>
    </Employee>
    <Employee>
        <Name>Kim</Name>
        <Sales>17772</Sales>
    </Employee>
    <Employee>
        <Name>Dean</Name>
        <Sales>23880</Sales>
    </Employee>
    <Employee>
        <Name>David</Name>
        <Sales>7663</Sales>
    </Employee>
    <Employee>
```

```
        <Name>Sanjay</Name>
        <Sales>21773</Sales>
    </Employee>
    <Employee>
        <Name>Michelle</Name>
        <Sales>32294</Sales>
    </Employee>
</NewDataSet>
```

3. 「App_Data」フォルダーで、新たに data.xsd という名前のファイルを作成します。（今回の拡張子が.xsd であることに注意してください。）
4. 既存の XML を以下で置き換えます。

```
<?xml version="1.0" ?>
<xs:schema
    id="NewDataSet"
    targetNamespace="http://tempuri.org/data.xsd"
    xmlns:mstns="http://tempuri.org/data.xsd"
    xmlns="http://tempuri.org/data.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    attributeFormDefault="qualified"
    elementFormDefault="qualified">
    <xs:element name="NewDataSet"
        msdata:IsDataSet="true"
        msdata:EnforceConstraints="False">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded">
                <xs:element name="Employee">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element
                                name="Name"
                                type="xs:string"
                                minOccurs="0" />
                            <xs:element
                                name="Sales"
                                type="xs:double"
                                minOccurs="0" />
                            </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

5. Web サイトのルートで、新たに ChartDataXML.cshtml という名前のファイルを作成します。
6. 既存のコードを以下で置き換えます。

```
@using System.Data;
@{
    var dataSet = new DataSet();
```

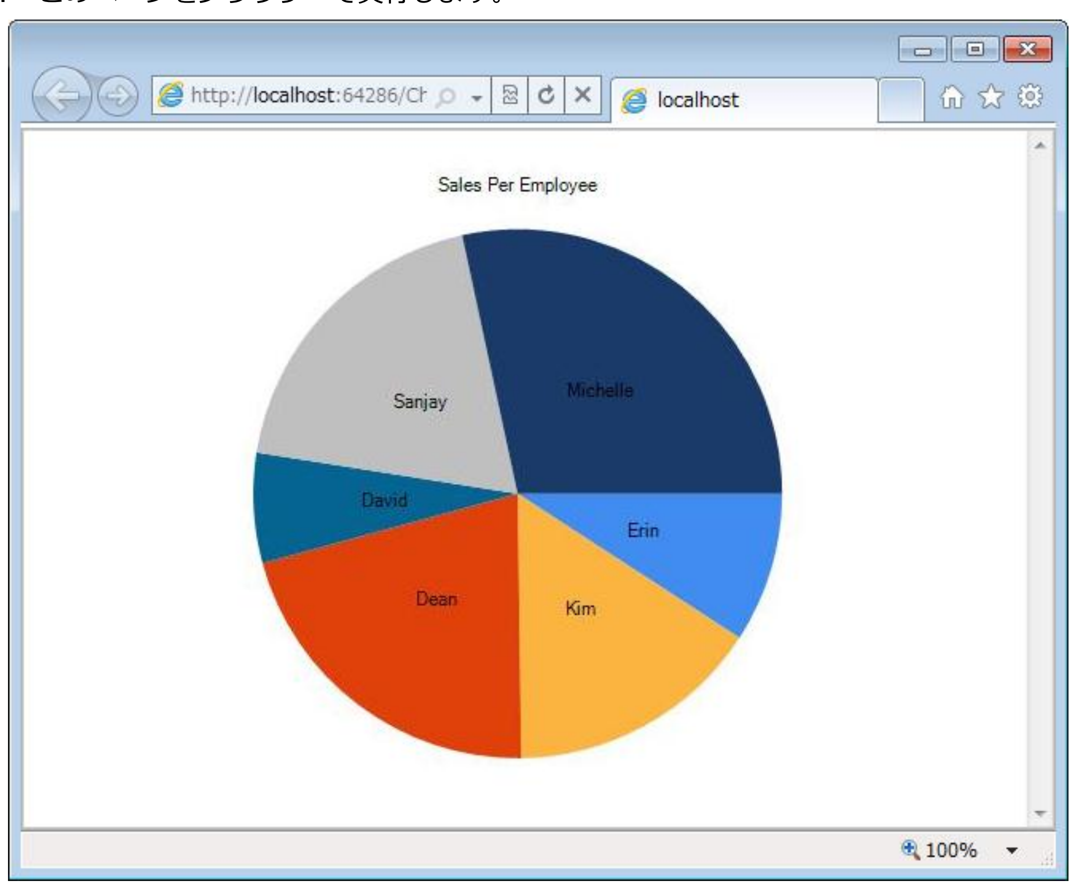
```
    dataSet.ReadXmlSchema(Server.MapPath("~/App_Data/data.xsd"));
    dataSet.ReadXml(Server.MapPath("~/App_Data/data.xml"));
    var dataView = new DataView(dataSet.Tables[0]);

    var myChart = new Chart(width: 600, height: 400)
        .AddTitle("Sales Per Employee")
        .AddSeries("Default", chartType: "Pie",
            xValue: dataView, xField: "Name",
            yValues: dataView, yFields: "Sales")
        .Write();
}
```

このコードは、まず DataSet オブジェクトを作成します。このオブジェクトは、XML ファイルから読み込みんだデータを管理するために使われ、スキーマファイルの情報に基づいて体系化します。(コードの先頭には、using System.Data 文が含まれていることに注意してください。これは、DataSet オブジェクトを扱うために必要なものです。詳細については、「"Using"文と完全な限定名」を参照してください。

次に、このコードはデータセットに基づく DataView オブジェクトを作成します。データビューは、グラフがバインド可能な、つまり、読み込みや描画ができるオブジェクトを提供します。グラフは、前述した配列のグラフ化と同じように AddSeries メソッドを使ってデータにバインドします。ただし、今回は xValue や yValues パラメーターは DataView オブジェクトに設定します。

この例は、グラフの種類を指定する方法も示しています。データが AddSeries メソッドに追加されるとき、chartType パラメーターも円グラフを表示するように設定されます。

7. このページをブラウザーで実行します。



**"Using"文と完全な限定名**

Razor 構文を使う ASP.NET Web ページのもとになる.NET Framework は、数千ものコンポーネント（クラス）から構成されています。こうしたすべてのクラスとともに使えるよう、これらは名前空間（namespace）の中で体系化されています。これは図書館（ライブラリ）のようなものです。たとえば、System.Web 名前空間は、ブラウザー/サーバー通信をサポートするクラスを含み、System.Xml 名前空間は XML ファイルの作成や読み込みのために使われるクラスを含み、System.Data 名前空間はデータを扱うクラスを含みます。

.NET Framework で定義されているクラスにアクセスするには、コードはクラス名だけでなく、そのクラスが含まれている名前空間も知っておく必要があります。たとえば、Chart ヘルパーを使うためには、コードは System.Web.Helpers.Chart クラスを見つける必要があります。これは、名前空間（System.Web.Helpers）とクラス名（Chart）を組み合わせたものです。これは、クラスの「完全な限定名」として知られており、広大な.NET Framework の中で、完全で、あいまいさのない場所をあらわします。

コードでは、これは以下のようになります。

```
var myChart = new System.Web.Helpers.Chart(width: 600, height: 400) // etc.
```

しかし、クラスやヘルパーを参照しようとするたびに、毎回完全な限定名を使おうとすることは厄介（かつ間違えやすい）です。そこで、クラス名を使いやすくするために、関心のある名前空間を「インポート」でき、.NET Framework の多くの名前空間から見つけやすくすることができます。名前空間をインポートすると、完全な限定名（System.Web.Helpers.Chart）ではなく、ただクラス名（Chart）を使うことができます。コードがクラス名に遭遇すると、インポートされた名前空間の中からクラスを見つけようとします。

Razor 構文を使う ASP.NET Web ページを使って Web ページを作成するときは、通常、WebPage クラス、さまざまなヘルパーなどのような同じクラスを毎回使います。Web サイトを作成するたびに、毎回関連する名前空間をインポートする作業を節約するため、どの Web サイトでも使われるようなコアの名前空間を自動的にインポートするよう、ASP.NET は設定されています。ここまで名前空間を扱ったり、インポートしなくてもよかったのは、そのためです。扱っていたすべてのクラスは、すでにインポートされていた名前空間にあるものです。

しかし、自動的にインポートされていない名前空間にないクラスを使いたいこともあります。この場合、クラスの完全な限定名を使うか、そのクラスを含む名前空間を自分でインポートします。名前空間をインポートするには、すでに本章で述べたとおり using 文（Visual Basic では import 文）を使います。

たとえば、DataSet クラスは System.Data 名前空間にあります。System.Data 名前空間は、ASP.NET Razor ページでは自動的に有効になりません。このため、完全な限定名を使って DataSet クラスを扱うには、次のようなコードを使います。

```
var dataSet = new System.Data.DataSet();
```

DataSet クラスを繰り返し使わなければならないときは、次のように名前空間をインポートして、コードではクラス名だけを使うようにします。

```
@using System.Data;
```

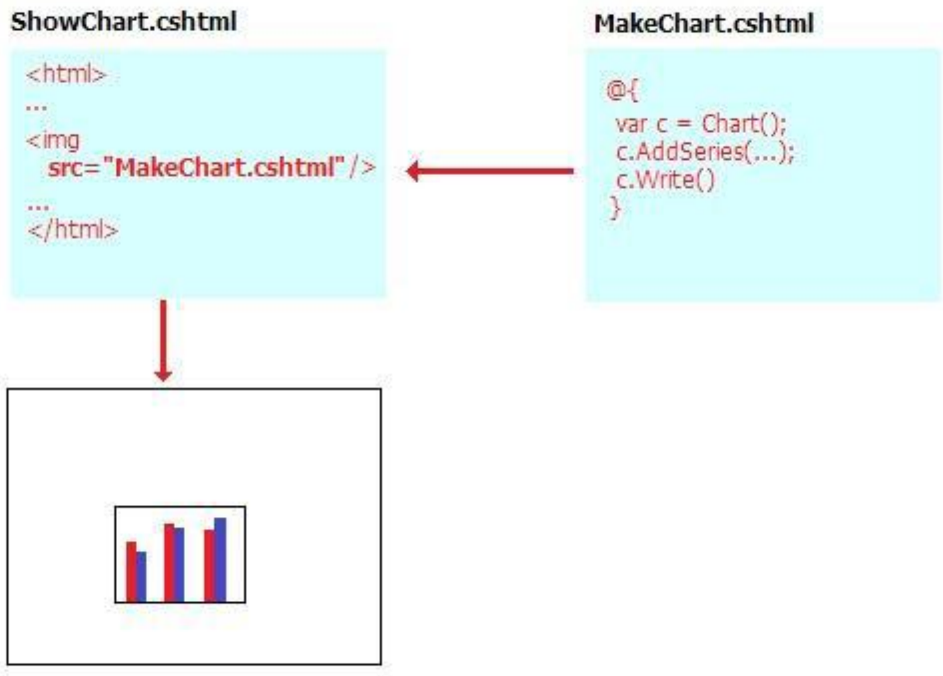
```
@{
    var dataSet = new DataSet();
    // etc.
}
```

他の.NET Framework の名前空間を参照したいときは、using 文を追加できます。ただし、前述のとおり、そうしなければならないことはあまりありません。.cshtml や.vbhtml ページで使うクラスのほとんどは、ASP.NET によって自動的にインポートされる名前空間に含まれているからです。

## Web ページ中にグラフを表示する

ここまでの例では、グラフを作成し、そのグラフを直接グラフィックとしてブラウザーにレンダリングしていました。しかし、たいていの場合、グラフはブラウザーに直接表示するのではなく、ページの一部として表示したいものです。これには、2 ステップの手順が必要です。最初のステップは、すでに説明したとおり、グラフを生成するページを作成することです。

2 つめのステップは、結果として得られたイメージを他のページに表示することです。イメージを表示するには、他のイメージを表示する方法と同じように HTML の<img>要素を使います。ただし、.jpg や.png ファイルを参照する代わりに、<img>要素はグラフを作成した Chart ヘルパーを含む.cshtml ファイルを参照します。表示用のページを実行するとき、<img> 要素は Chart ヘルパーの出力を受け取り、グラフをレンダリングします。




1. ShowChart.cshtml という名前のファイルを作成します。
2. 既存のコードを以下で置き換えます。

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <title>Chart Example</title>
    </head>
    <body>
        <h1>Chart Example</h1>
        <p>The following chart is generated by the <em>ChartArrayBasic.cshtml</em>
            file, but is shown in this page.</p>
        <p><img src="ChartArrayBasic.cshtml" /> </p>
    </body>
</html>
```

このコードは､前述の ChartArrayBasic.cshtml ファイルで作成したグラフを表示する<img>要素を使います。

3. このページをブラウザーで実行します。ShowChart.cshtml ファイルは、ChartArrayBasic.cshtml ファイルに含まれるコードに基づき、グラフのイメージを表示します。

## グラフのスタイル化

Chart ヘルパーは、グラフの外観をカスタマイズするための数多くのオプションをサポートしています。色やフォント、枠などを設定できます。グラフの外観を容易にカスタマイズするには「テーマ」が使えます。テーマは、フォントや色、ラベル、パレット、枠、そして効果を使ってグラフをレンダリングする方法を指定するための情報の集まりです。（グラフのスタイルは、グラフの種類を意味していないことに注意してください。）

以下の表は、組み込みのテーマの一覧です。

| テーマ | 説明 |
|---|---|
| Vanilla | 白い背景に赤い列を表示します。 |
| Blue | 青いグラデーションの背景に青い列を表示します。 |
| Green | 緑のグラデーションの背景に青い列を表示します。 |
| Yellow | 黄色いグラデーションの背景にオレンジ色の列を表示します。 |
| Vanilla3D | 白い背景に 3D の赤い列を表示します。 |

新しいグラフを作成するときに、使いたいテーマを指定できます。

1. 新たに ChartStyleGreen.cshtml という名前のファイルを作成します。
2. このページのデフォルトのマークアップとコードを以下で置き換えます。

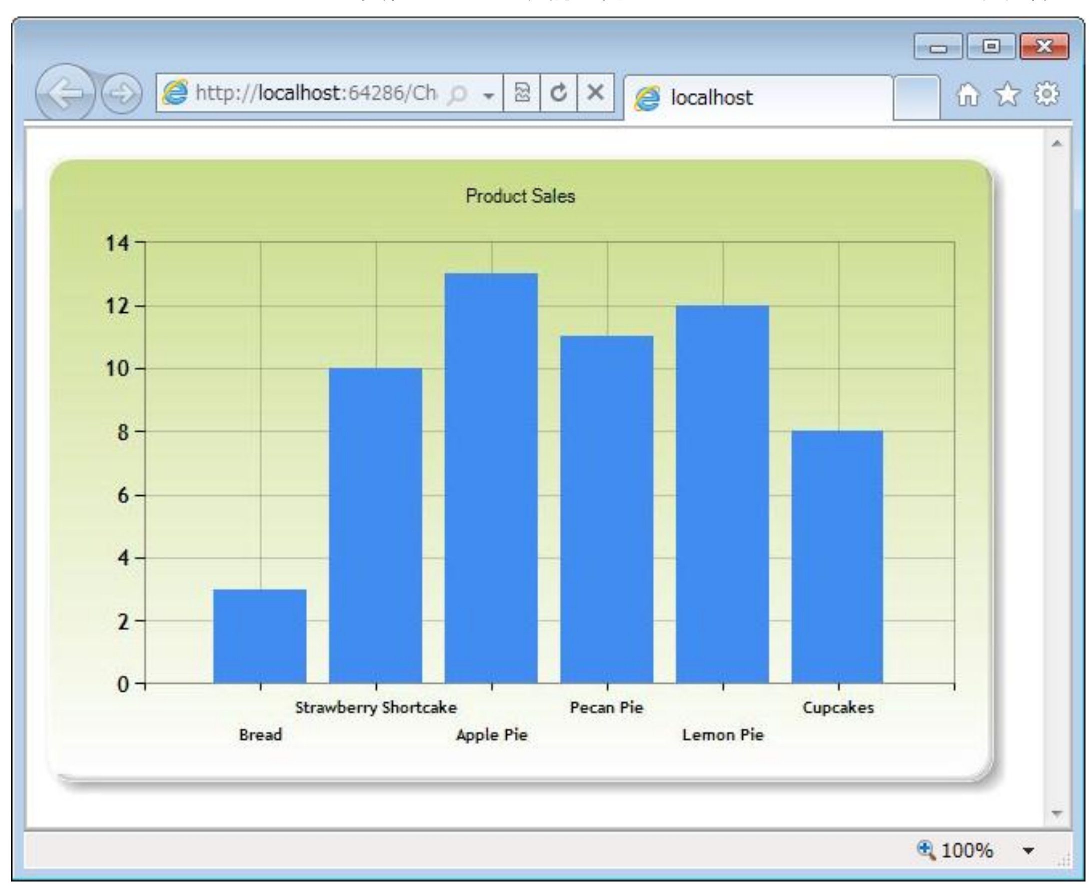
```
@{
    var db = Database.Open("SmallBakery");
    var data = db.Query("SELECT Name, Price FROM Product");
    var myChart = new Chart(width: 600,
                          height: 400,
                          theme: ChartTheme.Green)
        .AddTitle("Product Sales")
```

```
            .DataBindTable(data, "Name")
            .Write();
    }
```

　このコードは、データベースをデータとして使う前述の例と同じですが、Chart オブジェクトを作成するときに theme パラメーターを追加しています。以下が変更されたコードです。

```
var myChart = new Chart(width: 600,
                        height: 400,
                        theme: ChartTheme.Green)
```

3. このページをブラウザーで実行します。以前と同じデータですが、グラフはより洗練されています。




## グラフの保存

　これまで本章で見てきたように Chart ヘルパーを使うときには、ヘルパーは呼び出されるたびにスクラッチからグラフを再作成します。必要に応じて、グラフのためのコードはデータベースを再度問い合わせしたり、データを読み取るために XML ファイルを再読み込みしたりします。問い合わせるデータベースが大規模な場合や、XML ファイルが膨大なデータを含む場合などでは、ときには、これは複雑な操作になるかもしれません。グラフは大量のデータを必要としないにも関わらず、動的にイメージを作成するための処理は、サーバーリソースを消費し、多くの人がそのページや、あるいはグラフを表示するページをリクエストする場合、Web サイトのパフォーマンスに影響しかねません。

グラフを作成するための潜在的なパフォーマンスの影響を軽減するために、最初に必要なグラフを作成し、それを保存しておくことができます。再びグラフが必要なときには、再生成する代わりに、保存したバージョンを読み込んでレンダリングできます。

グラフは、以下の方法で保存できます。

- （サーバー上の）コンピューターメモリーでグラフをキャッシュします
- イメージファイルとしてグラフを保存します
- XML ファイルとしてグラフを保存します。この場合、保存する前にグラフを加工できます

## グラフのキャッシュ

グラフを作成した後で、キャッシュできます。グラフのキャッシュすることで、そのグラフを再度表示する必要があるときに再作成しなくてもよくなります。グラフをキャッシュに保存するときは、グラフに一意となるキーを与えます。

キャッシュに保存されたグラフは、サーバーのメモリが少なくなったときに削除されます。また、何らかの理由でアプリケーションが再起動されると、キャッシュはクリアされます。このため、キャッシュされたグラフを使うときには、まずキャッシュ中のグラフが使えるかどうかを確認し、使えない場合は、作成（または再作成）するのが標準的な手法です。

1. Web サイトのルートに、ShowCachedChart.cshtml という名前のファイルを作成します。
2. 既存のコードを以下で置き換えます。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Chart Example</title>
    </head>
    <body>
        <h1>Chart Example</h1>
        <img src="ChartSaveToCache.cshtml?key=myChartKey" />
    </body>
</html>
```

<img>タグは、ChartSaveToCache.cshtml ファイルを指し、問合せ文字列としてページのキーを渡す src 属性を含んでいます。キーは、myChartKey という値を含みます。ChartSaveToCache.cshtml ファイルは、グラフを作成する Chart ヘルパーを含みます。このページは次で作成します。

3. Web サイトのルートに、新たに ChartSaveToCache.cshtml という名前のファイルを作成します。
4. 既存のコードを以下で置き換えます。

```
@{
    var chartKey = Request["key"];
```

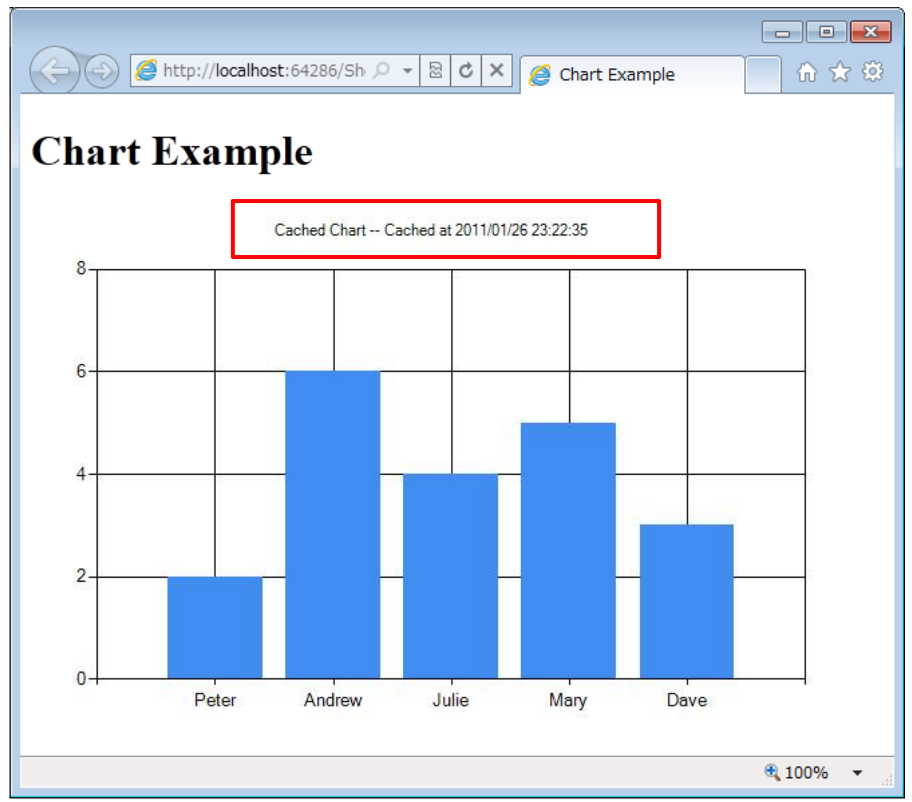
```
    if (chartKey != null) {
        var cachedChart = Chart.GetFromCache(key: chartKey);
        if (cachedChart == null) {
            cachedChart = new Chart(600, 400);
            cachedChart.AddTitle("Cached Chart -- Cached at " + DateTime.Now);
            cachedChart.AddSeries(
                name: "Employee",
                axisLabel: "Name",
                xValue: new[] { "Peter", "Andrew", "Julie", "Mary", "Dave" },
                yValues: new[] { "2", "6", "4", "5", "3" });
            cachedChart.SaveToCache(key: chartKey,
                minutesToCache: 2,
                slidingExpiration: false);
        }
        Chart.WriteFromCache(chartKey);
    }
}
```

　このコードは、まず問合せ文字列にキーの値として渡されたものがあるかを確認します。もしあれば、コードは GetFromCache メソッドにキーを渡して呼び出し、キャッシュからグラフを読みだそうとします。そのキーでキャッシュが見つからなければ（グラフが最初にリクエストされたときにはそうなります）、コードは通常どおりグラフを作成します。グラフが完成したら、コードは SaveToCache を呼び出してキャッシュに保存します。このメソッドはキーと（これで後からグラフをリクエストできます）、キャッシュに保存しておく時間を必要とします。（グラフをキャッシュする正確な時間は、そのデータがどれくらいの頻度で変更されるかに依存します。）SaveToCache メソッドは、slidingExpiration パラメーターも必要とします。これが true に設定されていると、グラフがアクセスされるたびにタイムアウト カウンターがリセットされます。ここでは、グラフのキャッシュ エントリは、誰かがグラフを最後にアクセスした時間から 2 分経過すると失効するようになっています。（slidingExpiration のもう一つの設定では、絶対的な失効時間をあらわし、キャッシュに保存されたときからちょうど 2 分間が経過したときに、そのキャッシュ エントリが失効します。どれだけアクセスされたかは影響しません。）

　最後に、このコードは WriteFromCache メソッドを使ってキャッシュからグラフを読み出し、レンダリングします。このメソッドは、キャッシュを確認する if ブロックの外側にあることに注意してください。グラフが、ここではじめて作成されたか、以前に作成されてキャッシュに保存されたものかに関わらず、キャッシュからグラフを取り出すためです。

　この例では、AddTitle メソッドがタイムスタンプを含んでいることにも注意してください。（現在の日付と時刻、DateTime.Now をタイトルに追加します。）

5. ShowCachedChart.cshtml ページをブラウザーで実行します。このページは、ChartSaveToCache.cshtml ファイルに含まれるコードに基づいたグラフのイメージを表示します。グラフのタイトルがどんなタイムスタンプになっているかを確認してください。




6. ブラウザーを閉じます。
7. もう一度、ShowCachedChart.cshtml を実行します。タイムスタンプが前回と変わっていないことに注意してください。これは、グラフが再生成されておらず、キャッシュから読み込まれているということです。
8. WebMatrix の「ホーム」タブの「サイト」グループにある、「再起動」をクリックします。これは、IIS Express を停止させ、再起動します。これにより、Web サイトアプリケーションを再起動する効果がわかります。

あるいは、キャッシュ エントリが失効するまで 2 分間待ってください。

9. もう一度、ShowCachedChart.cshtml を実行します。アプリケーションを再起動したことでキャッシュがクリアされたため、今回はタイムスタンプが変わっています。このため、コードはグラフを再生成し、キャッシュに書き戻します。

## グラフをイメージファイルとして保存する

グラフをサーバー上のイメージファイル（たとえば、.jpg ファイル）として保存することもできます。こうすると、どんなイメージも使えます。一時的なキャッシュとして保存しておくよりもファイルとして保存するのは利点があります。異なる時間（たとえば、一時間ごと）に新たなグラフを作成し、時間が過ぎるたびに永続的な記録を保持する

ことができます。Web アプリケーションがイメージファイルを保持しておきたいサーバー上のフォルダーにファイルを書き込む権限があることを確認しておかなければなりません。

1. Web サイトのルートに、_ChartFiles という名前のフォルダーを作成します（まだ作成していない場合）
2. Web サイトのルートに、新たに ChartSave.cshtml という名前のファイルを作成します。
3. 既存のコードを以下で置き換えます。

```
@{
    var filePathName = "_ChartFiles/chart01.jpg";
    if (!File.Exists(Server.MapPath(filePathName))) {
        var chartImage = new Chart(600, 400);
        chartImage.AddTitle("Chart Title");
        chartImage.AddSeries(
            name: "Employee",
            axisLabel: "Name",
            xValue: new[] { "Peter", "Andrew", "Julie", "Mary", "Dave" },
            yValues: new[] { "2", "6", "4", "5", "3" });
        chartImage.Save(path: filePathName);
    }
}
<!DOCTYPE html>
<html>
    <head>
        <title>Chart Example</title>
    </head>
    <body>
        <img src="@filePathName" />
    </body>
</html>
```

　このコードは、まず File.Exists メソッドを呼び出して.jpg ファイルが存在するかどうかを確認します。ファイルが存在しなければ、コードは配列から新しいグラフを作成します。今回、コードは Save メソッドを呼び出して、グラフを保存する場所を指定するファイル パスとファイル名を path パラメーターに渡します。ページの本体で、<img> 要素は表示する .jpg ファイルへのパスを使っています。

4. ChartSave.cshtml ファイルを実行します。

## グラフを XML ファイルとして保存する

　最後に、グラフはサーバー上の XML ファイルとして保存できます。グラフをキャッシュしたり、ファイルに保存したりする代わりに、この方法を使う利点は、必要に応じてグラフを表示する前に XML を加工できることです。アプリケーションは、ファイルを保存したい場所で、サーバー上のフォルダーに読み書きの権限を持っている必要があります。

1. Web サイトのルートで、新たに ChartSaveXml.cshtml という名前のファイルを作成します。

2. 既存のコードを以下で置き換えます。

```
@{
    Chart chartXml;
    var filePathName = "_ChartFiles/XmlChart.xml";
    if (File.Exists(Server.MapPath(filePathName))) {
        chartXml = new Chart(width: 600,
                             height: 400,
                             themePath: filePathName);
    }
    else {
        chartXml = new Chart(width: 600,
                             height: 400);
        chartXml.AddTitle("Chart Title -- Saved at " + DateTime.Now);
        chartXml.AddSeries(
            name: "Employee",
            axisLabel: "Name",
            xValue: new[] { "Peter", "Andrew", "Julie", "Mary", "Dave" },
            yValues: new[] { "2", "6", "4", "5", "3" });
        chartXml.SaveXml(path: filePathName);
    }
    chartXml.Write();
}
```

このコードは、先のキャッシュにグラフを保存するコードに似ていますが、XML ファイルを使っています。このコードは、まず File.Exists メソッドを呼び出して、XML ファイルが存在するかどうかを確認します。ファイルが存在すれば、このコードは themePath パラメーターとしてファイル名を渡して、新たに Chart オブジェクトを作成します。これは、XML ファイルにあるものに基づいてグラフを作成します。XML ファイルが存在しなければ、このコードは通常通りグラフを作成して、これを SaveXml を呼び出して保存します。グラフは、先に見たとおり Write メソッドを使ってレンダリングされます。

キャッシュを示すページと同じように、このコードはグラフのタイトルにタイムスタンプを含みます。

3. 新たに ChartDisplayXMLChart.cshtml という名前のページを作成し、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Display chart from XML</title>
    </head>
    <body>
        <img src="ChartSaveXML.cshtml" />
    </body>
</html>
```

4. ChartDisplayXMLChart.cshtml ページを実行します。グラフが表示されます。グラフのタイトルにあるタイムスタンプを確認してください。
5. ブラウザーを閉じます。

6. WebMatrix で、_ChartFiles フォルダーを右クリックし、[最新の情報に更新] をクリックして、フォルダーを開きます。このフォルダーの XMLChart.xml ファイルは、Chart ヘルパーによって作成されたものです。
7. もう一度、ChartDisplayXMLChart.cshtml ページを実行します。グラフは、最初にページを実行したときと同じタイムスタンプを表示します。これは、グラフが先に保存した XML から生成されたためです。
8. WebMatrix で、_ChartFiles フォルダーを開き、XMLChart.xml ファイルを削除します。
9. もう一度、ChartDisplayXMLChart.cshtml ページを実行します。今回は、Chart ヘルパーが XML ファイルを再生成したため、タイムスタンプが更新されます。_ChartFiles フォルダーを確認すると、XML ファイルが戻っていることがわかります。

## その他のリソース

- 第 5 章　データを扱う
- 第 6 章　データをグリッドで表示する
- 第 15 章　Web サイトのパフォーマンスを改良するためのキャッシング
- グラフ コントロール
- ASP.NET Web Pages with Razor Syntax

# 第８章　　ファイルを扱う

本章は、ファイルの読み込み、書き込み、追加、削除、アップロードについて解説します。

前章では、データベースにデータを保存する方法について学びました。しかし、Web サイトではテキストファイルを扱うことがあるかもしれません。たとえば、サイトのデータを保存するシンプルな方法としてテキストファイルを使うかもしれません。（データを保存するために使われるテキストファイルのことを、フラットファイルと呼ぶことがあります。）　テキストファイルは、.txt、.xml、csv（カンマ区切りの値）のように、異なる形式を持つことができます。

ここでは次のことを学びます。

- テキストファイルを作成し、そこにデータを書き込む方法
- 既存のファイルにデータを追加する方法
- ファイルを読み込み、表示する方法
- Web サイトからファイルを削除する方法
- ユーザーに、一つのファイル、または複数のファイルをアップロードさせる方法

本章で紹介する ASP.NET プログラミングの機能は、以下の通りです。

- ファイルを管理する方法を提供する、File オブジェクト
- FileUpload ヘルパー
- パスやファイル名を操作するメソッドを持つ Path オブジェクト

**注意**　イメージをアップロードしたり、操作したりする場合は（たとえば、反転やリサイズ）、「第 9 章　イメージを扱う」を参照してください。

## テキストファイルの作成とデータの書き込み

データをテキストファイルに保存したい場合、File.WriteAllText メソッドを使って、作成するファイルや、そこに書き込むデータを指定できます。以下の手順では、3 つの input 要素（名前、苗字、email アドレス）と［Submit］ボタンを持つシンプルなフォームを含むページを作成します。ユーザーがフォームを送出すると、ユーザーの入力をテキストファイルに保存します。

1. App_Data という名前のフォルダーがなければ、新たに作成します。
2. Web サイトのルートに、新たに UserData.cshtml という名前のファイルを作成します。
3. デフォルトのマークアップとコードを以下で置き換えます。

```
@{
    var result = "";
    if (IsPost)
    {
        var firstName = Request["FirstName"];
        var lastName = Request["LastName"];
```

```
        var email = Request["Email"];
        var userData = firstName + "," + lastName +
            "," + email + Environment.NewLine;
        var dataFile = Server.MapPath("~/App_Data/data.txt");
        File.WriteAllText(@dataFile, userData);
        result = "Information saved.";
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Write Data to a File</title>
</head>
<body>
    <form id="form1" method="post">
    <div>
        <table>
            <tr>
                <td>First Name:</td>
                <td><input id="FirstName" name="FirstName" type="text" /></td>
            </tr>
            <tr>
                <td>Last Name:</td>
                <td><input id="LastName" name="LastName" type="text" /></td>
            </tr>
            <tr>
                <td>Email:</td>
                <td><input id="Email" name="Email" type="text" /></td>
            </tr>
            <tr>
                <td></td>
                <td><input type="submit" value="Submit"/></td>
            </tr>
        </table>
    </div>
    <div>
    @if(result != ""){
    <p>Result: @result</p>
    }
    </div>
    </form>
</body>
</html>
```

HTML マークアップは、3 つのテキストボックスを持つフォームを作成します。コードでは、IsPost プロパティを使って、処理を始める前にページが送出されたものかどうかを判別しています。

最初の処理は、ユーザー入力を取得して、変数に代入することです。このコードは、別々の変数を連結して、ひとつのカンマ区切りの文字列にし、別の変数に保存します。作成する大きな文字列にカンマを文字として埋め込むため、カンマ区切りはクォーテーションマークで囲まれています(",")。連結したデータの末尾には、Environment.NewLine を追加します。これは改行（改行文字）を追加します。すべてを連結したものは、次のような文字列になります。

```
David,Jones,davidj@contoso.com
```

（末尾には目に見えない改行が含まれています。）

次に、このデータを保存するファイルの場所と名前を含む変数（dataFile）を作成します。場所を設定するには、いくらか特殊な処理が必要です。Web サイトでは、Web サーバー上のファイルとして C:¥Folder¥File.txt のような絶対パスを使うことは、よくない習慣です。Web サイトを移動すると、絶対パスは間違った場所を指してしまいます。さらに、（自分自身のコンピューターではなく）ホストされたサイトでは、コードを書くときに正しいパスを知ることさえできないことがあります。

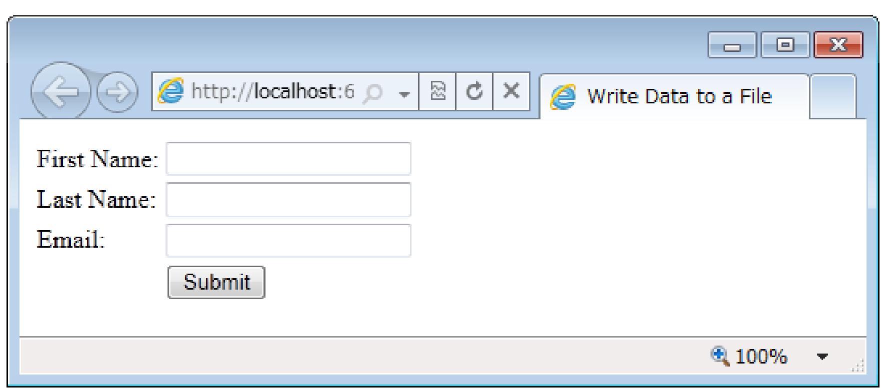
しかし、（いまここで、ファイルを書き込むように）完全なパスが必要なことがあります。その解決策となるのが、Server オブジェクトの MapPath メソッドを使うことです。これは Web サイトでの完全なパスを返します。Web サイトのルートのパスを取得するには、MapPath に"~"を渡します。（サブフォルダーのパスを取得したい場合は、~/AppData/のようなサブフォルダー名を渡すこともできます。） その後、メソッドが完全なパスを作成するために返した値を、追加の情報と連結できます。（この他のファイルとフォルダーパスの扱い方については、「第 2 章　Razor 構文を使った ASP.NET Web プログラミングの紹介」を参照してください。）

ファイルは、App_Data フォルダーに保存されます。「第 5 章　データを扱う」で述べたとおり、このフォルダーは、ASP.NET がデータファイルを保存するために使う特別なフォルダーです。

File オブジェクトの WriteAllText メソッドは、データをファイルへ書き出します。このメソッドは、書き込むファイルの name（パス付き）と、実際に書き込むデータの、2 つのパラメーターを持っています。最初のパラメーターの名前が、プレフィックスとして@文字を持っていることに注意してください。これは、ASP.NET に逐語的な文字列リテラルを渡していることを伝えます。"/"のような文字は、特別な形式に解釈されません。（詳細については、第 2 章を参照してください。）

**注意**　コードが App_Data フォルダーにファイルを保存するため、アプリケーションはこのフォルダーに対する読み書きの権限が必要です。開発用のコンピューターでは、通常問題になることはありません。しかし、Web サイトをホスティング プロバイダーの Web サーバーに発行するときには、この権限を明示的に設定する必要があるかもしれません。ホスティング プロバイダーのサーバーでこのコードを実行して、エラーが発生したら、ホスティング プロバイダーに権限を設定する方法について確認してください。

4. このページをブラウザーで実行します。（実行する前に、「ファイル」ワークスペースでこのページが選択されていることを確認してください。）



5. 項目に値を入力して、[Submit] をクリックします。
6. ブラウザーを閉じます。
7. プロジェクトに戻って、表示を最新の情報に更新します。
8. data.txt を開きます。フォームで登録したデータが、ファイルに保存されています。

9. data.txt を閉じます。

## 既存ファイルへのデータの追加

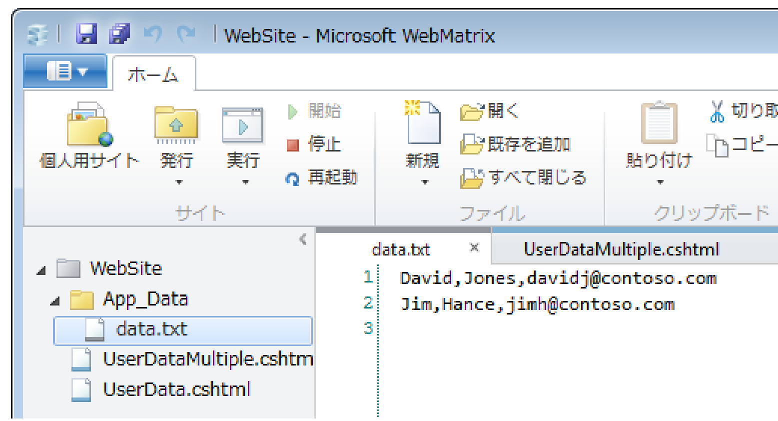
前のサンプルでは、データの一片を持つテキストファイルを作成するために WriteAllText を使いました。もう一度このメソッドを呼び出し、同じファイル名を渡すと、既存のファイルは完全に上書きされます。しかし、ファイルを作成した後には、しばしばファイルの末尾に新しいデータを追加したいことがあります。これは、File オブジェクトの AppendAllText メソッドを使って実現できます。

1. Web サイトで、UserData.cshtml ファイルのコピーを作成し、このコピーに UserDataMultiple.cshtml という名前を付けます。
2. 開始タグ<!DOCTYPE html>の前にあるコードブロックを、以下のコードブロックで置き換えます。

```
@{
    var result = "";
    if (IsPost)
    {
        var firstName = Request["FirstName"];
        var lastName = Request["LastName"];
        var email = Request["Email"];

        var userData = firstName + "," + lastName +
            "," + email + Environment.NewLine;

        var dataFile = Server.MapPath("~/App_Data/data.txt");
        File.AppendAllText (@dataFile, userData);
        result = "Information saved.";
    }
}
```

このコードは、前のサンプルから一箇所変更しています。WriteAllText を使う代わりに、AppendAllText メソッドを使っています。メソッドは似ていますが、AppendAllText はファイルの末尾にデータを追加します。WriteAllText と同じく、AppendAllText はファイルが存在しなければ作成します。

3. このページをブラウザーで実行します。
4. 項目に値を入力し、[Submit] をクリックします。
5. さらにデータを入力し、フォームを送信します。
6. プロジェクトに戻り、プロジェクト フォルダーを右クリックして、[最新の情報に更新] をクリックします。
7. data.text ファイルを開きます。入力した新しいデータが含まれています。




## ファイルからのデータの読み込みと表示

テキストファイルへデータを書き込む必要がない場合でも、ときにはデータを読み込む必要があるかもしれません。このために、ふたたび File オブジェクトを使います。File オブジェクトを使って、（改行で区切られた）1 行ごとに読み込んだり、区切られ方に関わらず個々の項目を読み込んだりできます。

以下の手順は、前のサンプルで作成したデータを読み込んで、表示する方法を示しています。

1. Web サイトのルートで、新たに DisplayData.cshtml という名前のファイルを作成します。
2. 既存のコードを以下で置き換えます。

```
@{
    var result = "";
    Array userData = null;
    char[] delimiterChar = {','};

    var dataFile = Server.MapPath("~/App_Data/data.txt");
    if (File.Exists(dataFile)) {
        userData = File.ReadAllLines(dataFile);
        if (userData == null) {
            // Empty file.
            result = "The file is empty.";
        }
    }
    else {
        // File does not exist.
```

```
        result = "The file does not exist.";
    }
}
<!DOCTYPE html>

<html>
<head>
    <title>Reading Data from a File</title>
</head>
<body>
    <div>
        <h1>Reading Data from a File</h1>
        @result
        @if (result == "") {
            <ol>
            @foreach (string dataLine in userData) {
            <li>
                User
                <ul>
                @foreach (string dataItem in dataLine.Split(delimiterChar)) {
                    <li>@dataItem</li >
                }
            </ul>
            </li>
            }
            </ol>
        }
    </div>
</body>
</html>
```

このコードは、まず、前のサンプルで作成したファイルを次のメソッド呼び出しで userData という名前の変数に読み込みます。

```
File.ReadAllLines(dataFile)
```

これを行うコードは if 文の中にあります。ファイルを読み込みたいとき、File.Exists メソッドを使ってそのファイルが存在するかどうかを確認するのはよい考えです。このコードは、ファイルが空であるかどうかも確認します。

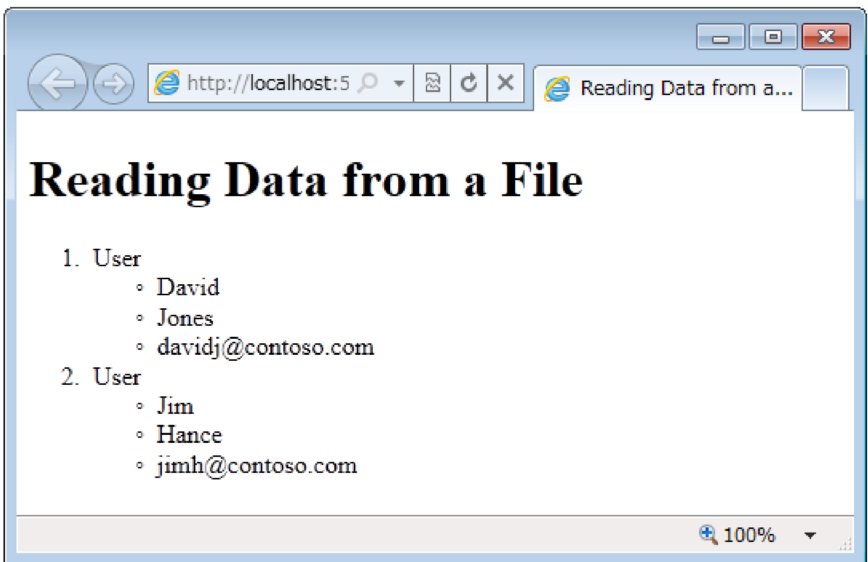
ページの本体は、ネストされた 2 つの foreach ループを含んでいます。外側の foreach ループは、データファイルから一度に 1 行ずつ取得します。ここでは、行はファイル中の改行によって定義されています。つまり、それぞれの各データ項目は、それぞれの行にあります。外側のループは、順序リスト（<ol>要素）の中に新しい項目（<li>要素）を作成します。

内側のループは、それぞれのデータ行を、カンマを区切りとして項目に分割します。（前の例に基づき、ここでは各行は、名前、苗字、email アドレスの 3 つの項目を含みます。） 内側のループは、<ul>リストを作成し、データ行の各項目をそれぞれ一つのリスト項目として表示します。

以下のコードは、配列と char データ型の、2 つのデータ型を使う方法を示しています。配列が必要なのは、File.ReadAllLines メソッドはデータを配列として返すためです。char データ型が必要なのは、Split メソッドが

各要素を char 型の配列に返すためです。（配列の詳細については、「第 2 章　Razor 構文を使った ASP.NET Web プログラミングの紹介」を参照してください。）

3.　このページをブラウザーで実行します。前の例で入力したデータが表示されます。



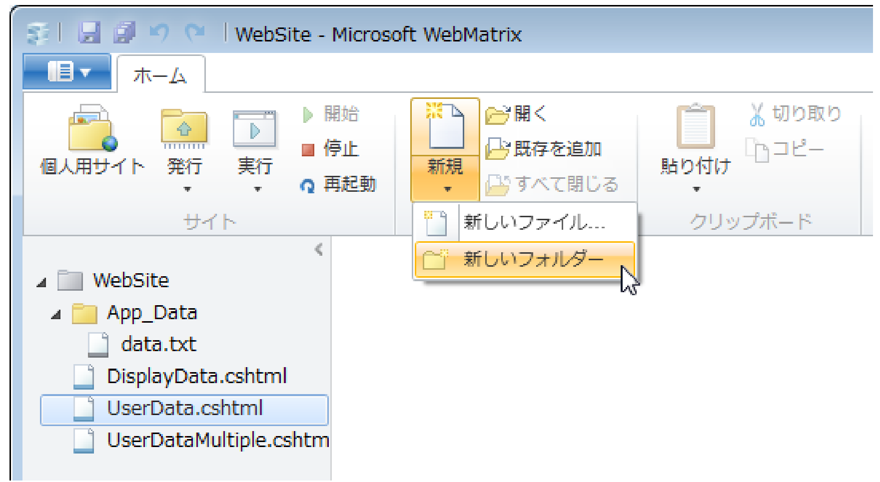

**Microsoft Excel のカンマ区切りファイルからデータを表示する**
　スプレッドシートに含まれるデータをカンマ区切りのファイル（.csv ファイル）として保存するために、Microsoft Excel が使えます。このとき、ファイルは Excel 形式ではなく、プレーン テキストとして保存されます。スプレッドシートの各行は、テキスト ファイルでは改行で区切られ、各データ項目はカンマで区切られます。前の例で示されたコードを使って、コード中のデータファイルの名前を変更するだけで、Excel のカンマ区切りファイルを読み込むことができます。

## ファイルの削除

　Web サイトのファイルを削除するために、File.Delete メソッドが使えます。以下の手順は、ユーザーがファイル名を分かっている場合に、images フォルダーにあるイメージ（.jpg）ファイルを削除させる方法を示しています。

**重要**　実運用する Web サイトでは、通常、データを変更できる人を制限します。メンバーシップの設定やサイト上の処理を実行できる権限をユーザーに与える方法については、「第 16 章　セキュリティとメンバーシップの追加」を参照してください。

1.　Web サイトで、images という名前のサブフォルダーを作成します。




2. images フォルダーに、1 つか複数の.jpg ファイルをコピーします。
3. Web サイトのルートで、新たに FileDelete.cshtml という名前のファイルを作成します。
4. デフォルトのマークアップとコードを以下で置き換えます。

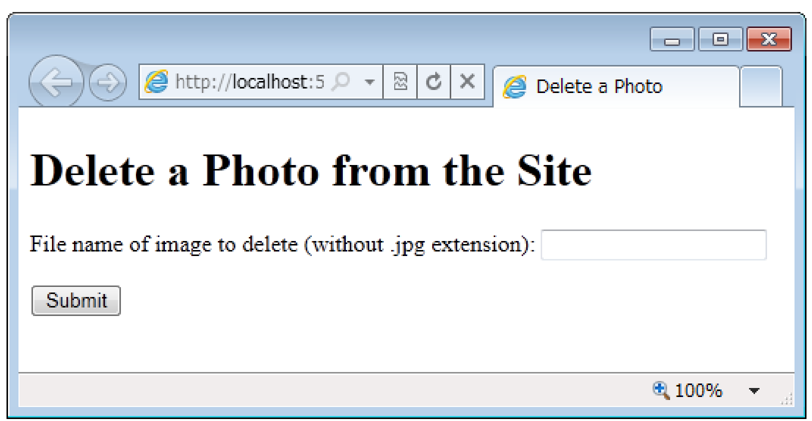
```
@{
    bool deleteSuccess = false;
    var photoName = "";
    if (IsPost) {
        photoName = Request["photoFileName"] + ".jpg";
        var fullPath = Server.MapPath("~/images/" + photoName);

        if (File.Exists(fullPath))
        {
            File.Delete(fullPath);
            deleteSuccess = true;
        }
    }
}
<!DOCTYPE html>
<html>
    <head>
        <title>Delete a Photo</title>
    </head>
    <body>
        <h1>Delete a Photo from the Site</h1>
        <form name="deletePhoto" action="" method="post">
            <p>File name of image to delete (without .jpg extension):
            <input name="photoFileName" type="text" value="" />
            </p>
            <p><input type="submit" value="Submit" </p>
        </form>

        @if(deleteSuccess) {
            <p>
            @photoName deleted!
            </p>
        }
    </body>
</html>
```

　このページは、ユーザーがイメージファイルの名前を入力できるフォームを含みます。ファイル名の拡張子.jpg は入力しません。このようにファイル名を制限することで、サイト上のファイルを自由に削除されてしまうことを防ぎます。

　このコードは、ユーザーが入力したファイル名を読み取って、完全なパスを構成します。パスを作成するために、コードは（Server.MapPath メソッドで返される）現在の Web サイトのパス、images フォルダー名、ユーザーが提供した名前、リテラル文字列としての".jpg"を使います。

　ファイルを削除するために、コードは、構成した完全パスを渡して File.Delete メソッドを呼び出します。マークアップの最後で、コードはファイルが削除されたことを確認するメッセージを表示します。

5. このページをブラウザーで実行します。




6. 削除するファイル名を入力し、[Submit] をクリックします。ファイルが削除されると、ファイル名がページの下部に表示されます。

## ユーザーにファイルをアップロードさせる

　FileUpload ヘルパーは、ユーザーにファイルを Web サイトへアップロードさせます。以下の手順は、ユーザーに単一のファイルをアップロードさせる方法を示しています。

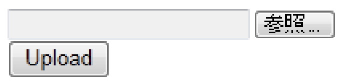
1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。
2. App_Data フォルダーで、新たにフォルダーを作成し、UploadedFiles という名前を付けます。
3. ルートで、新たに FileUpload.cshtml という名前のファイルを作成します。
4. ページ中のデフォルトのマークアップとコードを以下で置き換えます。

```
@{
    var fileName = "";
    if (IsPost) {
        var fileSavePath = "";
        var uploadedFile = Request.Files[0];
        fileName = Path.GetFileName(uploadedFile.FileName);
        fileSavePath = Server.MapPath("~/App_Data/UploadedFiles/" +
            fileName);
```

```
        uploadedFile.SaveAs(fileSavePath);
    }
}
<!DOCTYPE html>
<html>
    <head>
        <title>FileUpload - Single-File Example</title>
    </head>
    <body>
        <h1>FileUpload - Single-File Example</h1>
        @FileUpload.GetHtml(
            initialNumberOfFiles:1,
            allowMoreFilesToBeAdded:false,
            includeFormTag:true,
            uploadText:"Upload")
    @if (IsPost) {
        <span>File uploaded!</span><br/>
    }
    </body>
</html>
```

このページの本体部分は、FileUpload ヘルパーを使って、よく見かけるアップロードボックスとボタンを作成します。

FileUpload




FileUpload ヘルパーのプロパティを設定して、アップロードするファイルのための単一のボックスが必要であること、読み込みのための送信ボタンとして［Upload］が必要であることを指定します。（本章の後で、複数のボックスを追加します。）

ユーザーが［Upload］をクリックすると、ページの先頭にあるコードがファイルを取得して保存します。通常、フォームの項目から値を取り出すために使う Request オブジェクトはアップロードされたひとつのファイル（または複数のファイル）を含む Files 配列を持っています。配列の指定した位置から個々のファイルを取り出すことができます。たとえば、最初にアップロードされたファイルを取り出すには、Request.Files[0]を使い、2 番目のファイルを取り出すには、Request.Files[1]を使います。（プログラミングでは、一般にゼロから数えることを思い出してください。）

アップロードされたファイルを取り込むとき、これを変数に置くことで（ここでは、uploadedFile）、これを操作できます。アップロードされたファイルの名前は、FileName プロパティで取り出せます。しかし、ユーザーがファイルをアップロードするとき、FileName はユーザーの元の名前を含むため、完全なパスが含まれています。これは次のようなものです。

```
C:¥Users¥Public¥Sample.txt
```

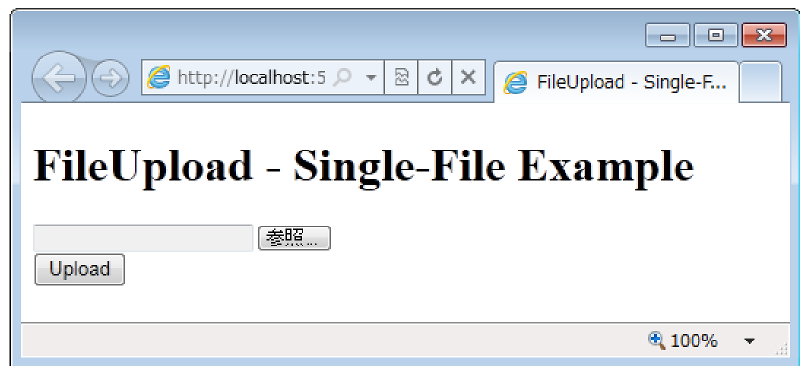
すべてのパスの情報は必要ないでしょう。これはユーザー コンピューター上のパスであり、サーバー上のものではないためです。必要なのは、実際のファイル名（Sample.txt）だけです。Path.GetFileName メソッドを次のように使うことで、パスからファイル名だけを取り出すことができます。

```
Path.GetFileName(uploadedFile.FileName)
```

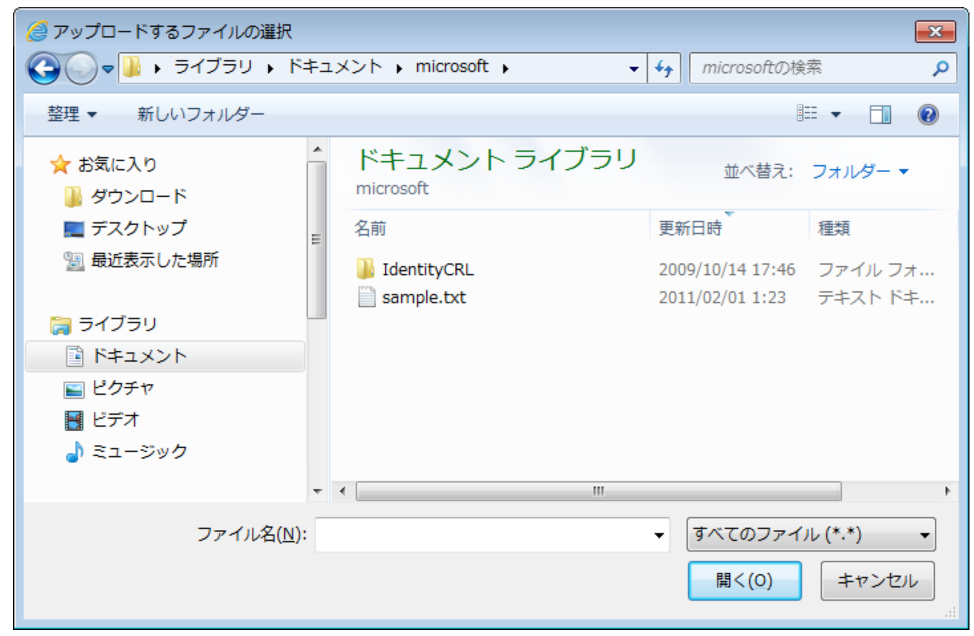
Path オブジェクトは、このようにパスを分割したり、組み合わせたりするようなメソッドをたくさん持っているユーティリティです。

アップロードされたファイルの名前を取得したら、Web サイト上でアップロードファイルを保存したい場所のための新しいパスを作成できます。ここでは、Server.MapPath、フォルダー名（App_Data/UploadedFiles）、新しいパスを作成するために取り出された新たなファイル名を組み合わせます。これで、実際にファイルを保存するためアップロードされたファイルの SaveAs メソッドを呼び出せます。

5. このページをブラウザーで実行します。

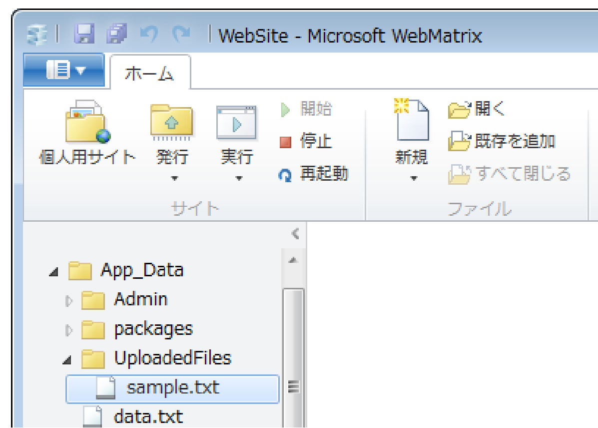



6. ここで[参照] をクリックして、アップロードするファイルを選びます




[参照] ボタンの隣にあるテキストボックスは、パスとファイルの場所を含みます。

7. [Upload] をクリックします。
8. Web サイトで、プロジェクト フォルダーを右クリックして、[最新の情報に更新] をクリックします。
9. UploadedFiles フォルダーを開きます。アップロードされたファイルが、このフォルダーにあります。




## ユーザーに複数のファイルをアップロードさせる

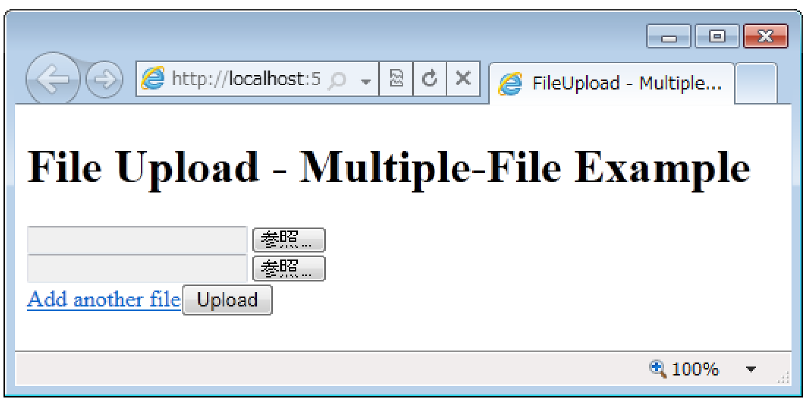
前の例では、ユーザーにひとつのファイルをアップロードさせました。しかし、FileUpload ヘルパーは複数のファイルを一度にアップロードするために使うこともできます。これは、写真をアップロードするような、一度にひとつのファイルをアップロードするのでは面倒なシナリオで便利です。（写真のアップロードについては、「第 9 章　イメージを扱う」を参照してください。）　この例は、一度に 2 つのファイルをアップロードさせる方法を示しています。これは、より多くのファイルをアップロードさせる場合と同じ手法です。

1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。
2. ルートで、新たに FileUploadMultiple.cshtml という名前のファイルを作成します。
3. ページ中のデフォルトのマークアップとコードを以下で置き換えます。

```
@{
    var message = "";
    if (IsPost) {
        var fileName = "";
        var fileSavePath = "";
        int numFiles = Request.Files.Count;
        int uploadedCount = 0;
        for(int i =0; i < numFiles; i++) {
            var uploadedFile = Request.Files[i];
            if (uploadedFile.ContentLength > 0) {
                fileName = Path.GetFileName(uploadedFile.FileName);
                fileSavePath = Server.MapPath("~/App_Data/UploadedFiles/" +
                    fileName);
                uploadedFile.SaveAs(fileSavePath);
                uploadedCount++;
            }
        }
        message = "File upload complete. Total files uploaded: " +
        uploadedCount.ToString();
```

```
    }
}
<!DOCTYPE html>
<html>
    <head><title>FileUpload - Multiple File Example</title></head>
<body>
    <form id="myForm" method="post"
        enctype="multipart/form-data"
        action="">
    <div>
    <h1>File Upload - Multiple-File Example</h1>
    @if (!IsPost) {
        @FileUpload.GetHtml(
            initialNumberOfFiles:2,
            allowMoreFilesToBeAdded:true,
            includeFormTag:true,
            addText:"Add another file",
            uploadText:"Upload")
        }
    <span>@message</span>
    </div>
    </form>
</body>
</html>
```

この例では、ページの本体にある FileUpload ヘルパーは、デフォルトで 2 つのファイルをアップロードさせるよう設定されています。allowMoreFilesToBeAdded が true に設定されているため、ヘルパーはより多くのアップロード ボックスを追加するためのリンクをレンダリングします。




ユーザーがアップロードしたファイルを処理するために、このコードは前の例で使ったものと同じ基本的な手法、つまり Request.Files からファイルを取り出し、保存します。（ここには、正しいファイル名とパスを取得するために必要なさまざまなことが含まれます。） 今回新しいことは、ユーザーが複数のファイルをアップロードしていても、それがわからないことです。これを調べるために、Request.Files.Count を取得します。

この数がわかれば、Request.Files を繰り返し、それぞれのファイルを取り出し、保存できます。コレクションを通じて、既知の数を繰り返したいときは、次のように for ループを使います。

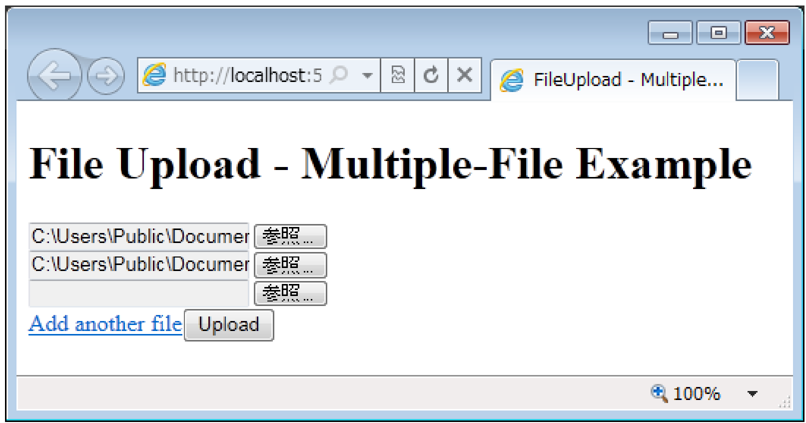
```
for(int i =0; i < numFiles; i++) {
    var uploadedFile = Request.Files[i];
    if (uploadedFile.ContentLength > 0) {
        fileName = Path.GetFileName(uploadedFile.FileName);
```

```
    // etc.
}
```

変数 i は、0 から設定した上限までを繰り返すための一時的なカウンターにすぎません。ここでは、上限はファイルの数です。ただし、ASP.NET における通常のカウントと同じく、カウンターはゼロから始まるので、上限はファイルの数よりも 1 少なくなります。（3 つのファイルがアップロードされた場合、カウントは 0 から 2 までです。）

uploadedCount 変数は、アップロードと保存に成功したすべてのファイルの合計です。このコードは、期待されるファイルがアップロードされなかった場合に対応しています。

4. このページをブラウザーで実行します。ブラウザーは、ページと、2 つのアップロード ボックスを表示します。
5. アップロードする 2 つのファイルを選びます。
6. [Add another file] をクリックします。このページに新しいアップロード ボックスが表示されます。




7. [Upload] をクリックします。
8. Web サイトで、プロジェクト フォルダーを右クリックし、[最新の情報に更新] をクリックします。
9. UploadedFiles フォルダーを開き、ファイルがアップロードされていることを確認します。

## その他のリソース

- 「第 9 章　イメージを扱う」
- CSV ファイルへのエクスポート
- ASP.NET Web Pages with Razor Syntax

# 第９章　イメージを扱う

本章では、Web サイトにおいて、イメージを追加、表示、操作（サイズ変更、反転、ウォーターマークの追加）する方法について示します。

ここでは次のことを学びます。

- ページに動的にイメージを追加する方法
- ユーザーにイメージをアップロードさせる方法
- イメージのサイズを変更する方法
- イメージを反転や回転する方法
- イメージにウォーターマークを追加する方法
- イメージをウォーターマークとして使う方法

本章で紹介する ASP.NET プログラミングの機能は、以下の通りです。

- WebImage ヘルパー
- パスやファイル名を操作する手段を提供する Path オブジェクト

## Web ページに動的にイメージを追加する

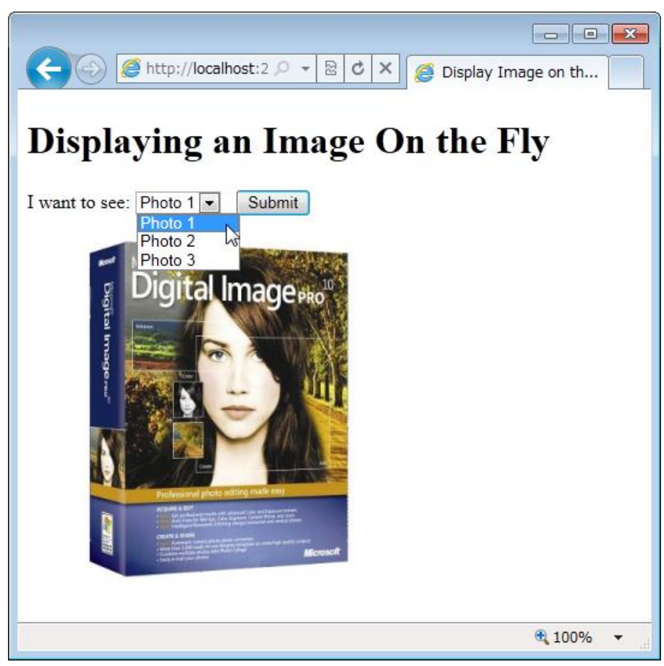
Web サイトを開発しているときに、Web サイトや個々のページにイメージを追加することができます。また、プロファイル用の写真を追加するといった処理のために、ユーザーにイメージをアップロードさせることもできます。

Web サイトにイメージがあり、それをページ上に表示したい場合は、次のように HTML の<img>要素を使います。

```
<img src="images/Photo1.jpg" alt="Sample Photo" />
```

しかし、ページを実行するまでどのイメージを表示べきかわかっていない場合には、動的にイメージを表示する必要があるかもしれません。

このセクションの手順は、イメージの名前リストからユーザーが指定したファイル名のイメージを、その場で表示する方法について示しています。ユーザーは、ドロップダウン リストからイメージの名前を選び、ページを送信したときに、選択したイメージが表示されます。




1. WebMatrix で、新しい Web サイトを作成します。
2. 新たに DynamicImage.cshtml というページを追加します。
3. Web サイトのルート フォルダーで、新しいフォルダーを作成し、images と名付けます。
4. 作成した images フォルダーに、4 つのイメージを追加します。（どんなイメージでも追加しやすいものでかまいませんが、ページに収まるものにしてください。） これらのイメージを、Photo1.jpg、Photo2.jpg、Photo3.jpg、Photo4.jpg という名前に変更します。（今回の手順では、Photo4.jpg は使いませんが、本章の後半で使います。）
5. 4 つのイメージが、読み込み専用として設定されていないことを確認します。
6. このページの既存のマークアップを、以下で置き換えます。

```
@{ var imagePath= "";
    if( Request["photoChoice"] != null){
        imagePath = @"images¥" + Request["photoChoice"];
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Display Image on the Fly</title>
</head>
<body>
    <h1>Displaying an Image On the Fly</h1>
    <form method="post" action="">
        <div>
            I want to see:
            <select name="photoChoice">
                <option value="Photo1.jpg">Photo 1</option>
                <option value="Photo2.jpg">Photo 2</option>
                <option value="Photo3.jpg">Photo 3</option>
            </select>
             
            <input type="submit" value="Submit" />
        </div>
```

```
        <div style="padding:10px;">
            @if(imagePath != ""){
                <img src="@imagePath" alt="Sample Image" width="300px" />
            }
        </div>
    </form>
</body>
</html>
```

ページの本体は、photoChoice という名前のドロップダウン リスト（<select>要素）があります。このリストには 3 つの選択肢があり、各リスト項目の value 属性は、images フォルダーに置いてあるイメージのうちの一つの名前が設定されています。本質的に、このリストはユーザーに"Photo 1"のような分かりやすい名前で選択させ、ページが送信されるときに.jpg ファイル名を渡しています。

このコードでは、Request["photoChoice"]を読み取ることで、リストからユーザーの選択（言い換えると、イメージファイル名）を取得します。まず、選択されているものがあるかどうかを確認します。もしあれば、イメージ用のフォルダー名とユーザーのイメージファイル名からなるイメージのパスを構成します。（パスを構成しようとしたときに、Request["photoChoice"]に何もなければ、エラーになります。）　この結果、以下のような相対パスが得られます。

```
images/Photo1.jpg
```

このパスは、imagePath という名前の変数に保存され、後で、このページで必要になります。

本体には、ユーザーが選択したイメージを表示するために使われる<img>要素もあります。src 属性には、静的な要素を表示するためのようなファイル名や URL は設定されていません。その代りに、@imagePath に設定されており、このコードで設定したパスから値が取り出されます。

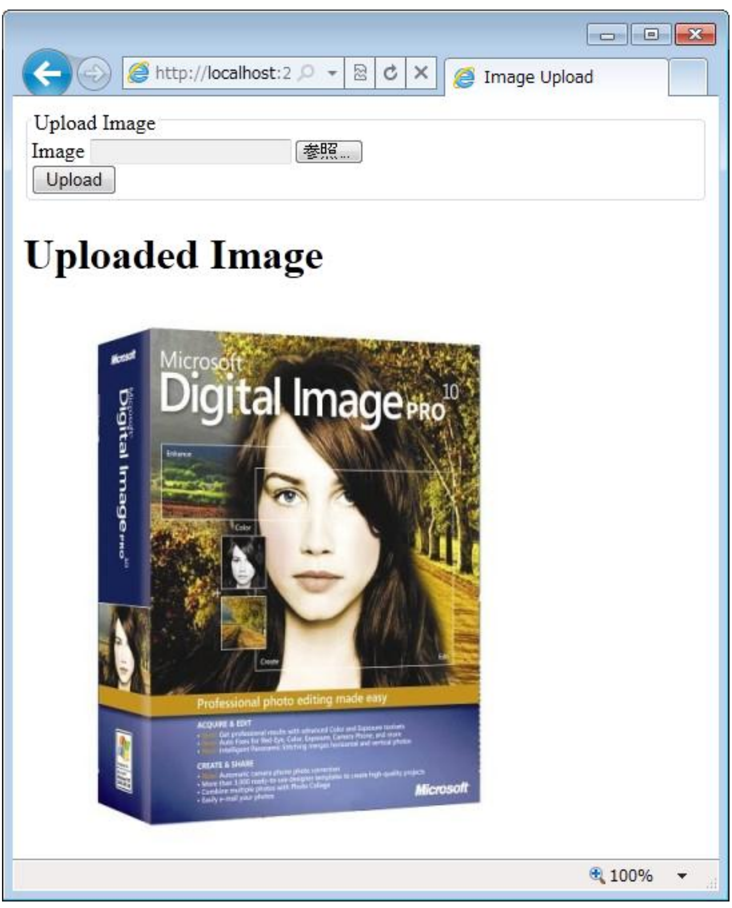
最初にこのページを実行するときは、ユーザーは何も選択していないため、何もイメージは表示されません。通常、これは src 属性が空であり、イメージが赤い"x"（または、ブラウザーがイメージを見つけられない場合にレンダリングされるもの）として表示されることを意味します。これを防ぐため、<img>要素は imagePath 変数が値を持っているかどうかを確認する if ブロックの中に置いています。ユーザーが選択したときには、imagePath はパスを含みます。ユーザーがイメージを選択しないか、最初のページが表示されるときには、<img>要素はレンダリングされません。

7. ファイルを保存して、このページをブラウザーで実行します。（実行する前に、「ファイル」ワークスペースで、このページが選択されていることを確認してください。）

## イメージをアップロードする

前の例では、動的にイメージを表示する方法を示しましたが、イメージはあらかじめ Web サイト上に用意されていたものだけでした。次の手順では、ユーザーにイメージをアップロードさせ、それをページ上に表示する方法を示します。ASP.NET では、イメージを作成、操作、保存するメソッドを持つ WebImage ヘルパーを使って、その場でイ

メージを操作できます。WebImage ヘルパーは、すべての一般的な Web イメージをサポートしています。これには、.jpg、.png、.bmp が含まれます。本章では、.jpg を使いますが、これらのイメージ形式のどれでも使えます。




1. 新しいページを追加し、UploadImage.cshtml と名付けます。
2. このページの既存のマークアップを、以下で置き換えます。

```
@{ WebImage photo = null;
    var newFileName = "";
    var imagePath = "";

    if(IsPost){
        photo = WebImage.GetImageFromRequest();
        if(photo != null){
            newFileName = Guid.NewGuid().ToString() + "_" +
                Path.GetFileName(photo.FileName);
            imagePath = @"images¥" + newFileName;

            photo.Save(@"~¥" + imagePath);
        }
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Image Upload</title>
</head>
<body>
```

```
    <form action="" method="post" enctype="multipart/form-data">
        <fieldset>
            <legend> Upload Image </legend>
            <label for="Image">Image</label>
            <input type="file" name="Image" />
            <br/>
            <input type="submit" value="Upload" />
        </fieldset>
    </form>
    <h1>Uploaded Image</h1>
        @if(imagePath != ""){
        <div class="result">
            <img src="@imagePath" alt="image" />
        </div>
    }
</body>
</html>
```

テキストの本体には、ユーザーにアップロードするファイルを選択させる<input type="file">要素があります。[Submit] をクリックすると、選んだファイルがフォームとともに送信されます。

アップロードされたイメージを取得するために、WebImage ヘルパーを使います。イメージを扱うための、あらゆる種類の便利なメソッドが含まれています。具体的には、WebImage.GetImageFromRequest を使って、アップロードされたイメージ（もしあれば）を取得し、photo という名前の変数に格納します。

この例での多くの作業は、ファイルやパス名を取得したり、設定したりすることです。つまり、ユーザーがアップロードするイメージの名前（名前部分のみ）を取得し、そのイメージを格納するための場所として使う新しいパスを作成するということです。ユーザーは、同じ名前を持つ複数のイメージをアップロードするかもしれないため、一意の名前を作成し、既存のイメージを上書きしてしまわないようにする追加のコードを使います。

実際にイメージがアップロードされた場合（if (photo != null) で判定する）、イメージの FileName プロパティからイメージの名前を取り出します。イメージをアップロードするとき、FileName はユーザー コンピューター上のパスを含む、ユーザー側での元の名前を保持します。これは、次のようなものです。

```
C:¥Users¥Joe¥Pictures¥SamplePhoto1.jpg
```

もっとも、すべてのパス情報は必要ありません。必要なのは実際のファイル名（SamplePhoto1.jpg）だけです。パスからファイル名だけを取り出すには、次のように Path.GetFileName メソッドを使います。

```
Path.GetFileName(photo.FileName)
```

新たに一意のファイル名を作成するには、元の名前に GUID を追加します。（GUID についての詳細は、本章の「GUID について」を参照してください。）　これでイメージを保存するために使えるパスを構成します。保存パスは、新しいファイル名、フォルダー（images）、現在の Web サイトの場所から作成されます。

**注意**　images フォルダーにファイルを保存するコードのために、アプリケーションはこのフォルダーに対する読み書きの権限が必要です。開発用のコンピューターでは、通常問題になることはありません。しかし、Web サイトをホス

ティング プロバイダーの Web サーバーに発行するときには、この権限を明示的に設定する必要があるかもしれません。ホスティング プロバイダーのサーバーでこのコードを実行して、エラーが発生したら、ホスティング プロバイダーに権限を設定する方法について確認してください。

最後に、保存パスを WebImage ヘルパーの Save メソッドに渡します。これは、アップロードされたイメージを新しい名前で保存します。Save メソッドは、photo.Save(@"~¥" + imagePath)のようなものです。完全なパスには、現在の Web サイトの場所をあらわす@"~¥"が追加されます。（~演算子の詳細については、「第 2 章　Razor 構文を使った ASP.NET Web プログラミングの紹介」を参照してください。）

前のサンプルと同じように、ページの本体はイメージを表示するための<img>要素を含みます。imagePath が設定されると、<img>要素がレンダリングされ、src 属性が imagePath の値に設定されます。

3. このページをブラウザーで実行します。

**GUID について**

GUID（グローバル ユニーク ID）は、936DA01F-9ABD-4d9d-80C7-02AF85C822A8 のように見える識別子です。（技術的には、これは 16 バイト/128 ビットの数値です。） GUID が必要な場合、GUID を生成するための特別なコードを呼び出します。GUID の背後には、巨大な数値（$3.4\times10^{38}$）とそれを生成するアルゴリズムがあり、得られる数値は仮想的に唯一のものであることが保証されています。このため、GUID は同じ名前を繰り返さないようにしたい何かの名前を生成するためによい方法です。もちろん、その反面、GUID はわかりやすいものではないので、そのような名前はコード中でのみ使うようにすべきです。

## イメージのサイズを変更する

Web サイトがユーザーからのイメージを受け入れる場合、イメージを表示したり、保存したりする前に、サイズを変更したいことがあります。これを行うためにも、WebImage ヘルパーが使えます。

以下の手順は、サムネイルを作るためにアップロードされたイメージのサイズを変更し、サムネイルと元のイメージを Web サイトに保存する方法を示しています。ページにはサムネイルを表示し、フルサイズのイメージへリダイレクトするハイパーリンクを使います。

1. 新たに Thumbnail.cshtml という名前のページを追加します。
2. images フォルダーに、thumbs という名前のサブフォルダーを作成します。
3. このページの既存のマークアップを、以下で置き換えます。

```
@{ WebImage photo = null;
    var newFileName = "";
    var imagePath = "";
    var imageThumbPath = "";

    if(IsPost){
        photo = WebImage.GetImageFromRequest();
        if(photo != null){
            newFileName = Guid.NewGuid().ToString() + "_" +
                Path.GetFileName(photo.FileName);
            imagePath = @"images¥" + newFileName;
            photo.Save(@"~¥" + imagePath);

            imageThumbPath = @"images¥thumbs¥" + newFileName;
            photo.Resize(width: 60, height: 60, preserveAspectRatio: true,
                preventEnlarge: true);
            photo.Save(@"~¥" + imageThumbPath); }
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Resizing Image</title>
</head>
<body>
<h1>Thumbnail Image</h1>
    <form action="" method="post" enctype="multipart/form-data">
        <fieldset>
            <legend> Creating Thumbnail Image </legend>
            <label for="Image">Image</label>
            <input type="file" name="Image" />
            <br/>
            <input type="submit" value="Submit" />
        </fieldset>
    </form>
    @if(imagePath != ""){
        <div class="result">
            <img src="@imageThumbPath" alt="Thumbnail image" />
            <a href="@Html.AttributeEncode(imagePath)" target="_Self">
                View full size
            </a>
        </div>
    }
</body>
</html>
```

このコードは、前の例のコードに似ています。違いは、このコードはイメージを 2 回保存するということで、ひとつは通常どおり、もうひとつはイメージのサムネイル コピーを作成した後です。まず、アップロードされた

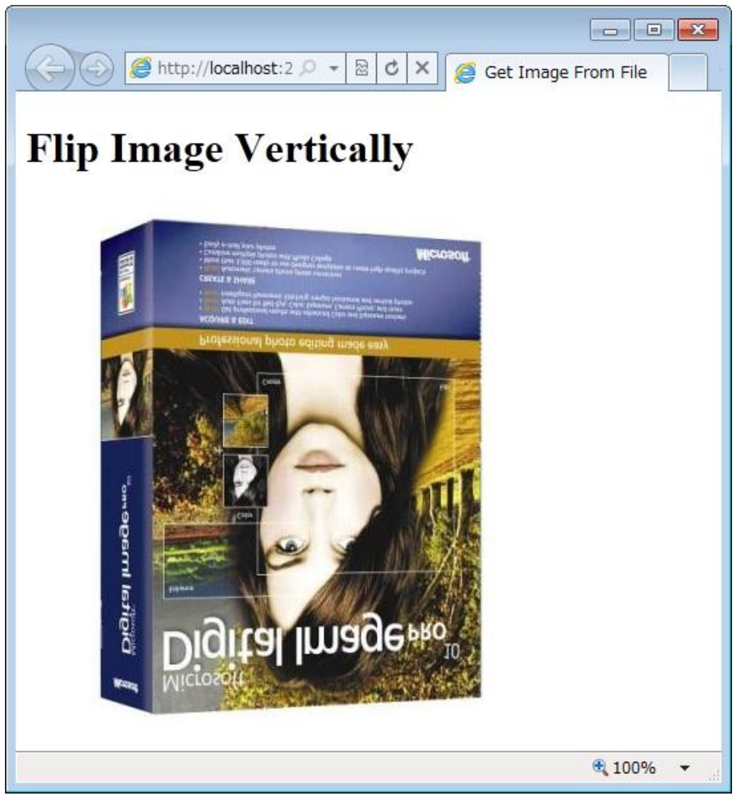
イメージを取得して、それを images フォルダーに保存します。次に、サムネイル イメージのための新しいパスを作成します。実際にサムネイルを作成するため、WebImage ヘルパーの Resize メソッドを呼び出して、60 ピクセル×60 ピクセルのイメージを作成します。この例は、アスペクト比を保存する方法と、イメージの拡大（新しいサイズがイメージを拡大してしまうかもしれないため）を防ぐ方法を示しています。サイズ変更されたイメージは、thumbs サブフォルダーに保存されます。

マークアップの最後では、以前のイメージを条件的に表示する例で見たのと同じ、動的な src 属性を持つ<img>要素を使います。ここでは、サムネイルを表示します。また、<a>要素を使って大きなイメージへのハイパーリンクを作成します。<img>要素の src 属性のように、<a>要素の href 属性を動的に imagePath に設定します。パスが URL として使えるようにするため、imagePath を Html.AttributeEncode メソッドに渡します。これは、パスに含まれる予約文字を URL として使える文字に変換するものです。

4. このページをブラウザーで実行します。

## イメージを回転、反転させる

WebImage ヘルパーでは、イメージの反転や回転もできます。以下の手順では、サーバー上のイメージを取得して、イメージを上下（垂直）に反転し、保存し、反転したイメージをページ上に表示する方法を示しています。この例では、すでにサーバー上にあるファイル（Photo2.jpg）を使います。実際のアプリケーションでは、前の例で行ったように、動的に取得した名前のイメージを反転させたいでしょう。




1. 新たに Flip.cshtml という名前のページを追加します。
2. ファイル中の既存のマークアップを、以下で置き換えます。

```
@{ var imagePath= "";
    WebImage photo = new WebImage(@"~¥Images¥Photo2.jpg");
    if(photo != null){
        imagePath = @"images¥Photo2.jpg";
        photo.FlipVertical();
        photo.Save(@"~¥" + imagePath);
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Get Image From File</title>
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
</head>
<body>
    <h1>Flip Image Vertically</h1>
        @if(imagePath != ""){
        <div class="result">
            <img src="@imagePath" alt="Image" />
        </div>
    }
</body>
</html>
```

　このコードは、サーバーからイメージを取り出すために WebImage ヘルパーを使います。イメージを保存するための前述の例で使ったのと同じ方法で、イメージへのパスを作成し、WebImage を使って作成するときに、このパスを渡します。

```
WebImage photo = new WebImage(@"~¥Images¥Photo2.jpg");
```

　イメージが見つかったら、前述の例でやったように、新しいパスとファイル名を作成します。イメージを反転させるためには、FlipVertical メソッドを呼び出し、このイメージをもう一度保存します。

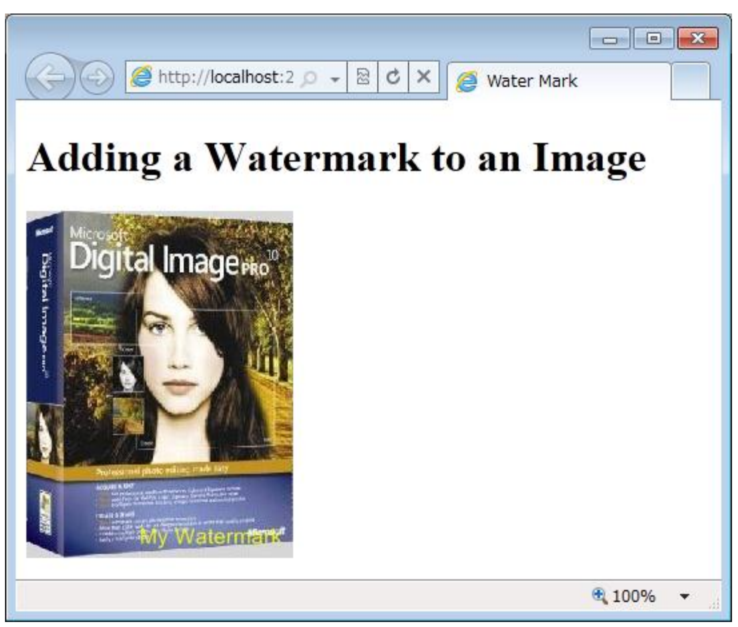
　src 属性が imagePath に設定された<img>要素を使って、ページ上に、もう一度イメージが表示されます。

3. このページをブラウザーで実行します。Photo2.jpg は、上下反転して表示されます。もう一度、このページを表示すると、イメージは再び反転して元通りに表示されます。

　イメージを回転させるには、同じコードを使い、FlipVertical や FlipHorizontal を呼び出す代わりに、RotateLeft や RotateRight を呼び出します。

## イメージにウォーターマークを追加する

　Web サイトにイメージを追加するとき、それを保存したり、ページ上に表示する前に、イメージにウォーターマークを追加したいことがあります。ウォーターマークは、イメージに著作権情報を追加したり、ビジネス名を宣伝したりするために使われます。




1. 新たに Watermark.cshtml という名前のページを追加します。
2. 既存のマークアップを以下で置き換えます。

```
@{ var imagePath= "";
    WebImage photo = new WebImage(@"~¥Images¥Photo3.jpg");
    if(photo != null){
        imagePath = @"images¥Photo3.jpg";
        photo.AddTextWatermark("My Watermark", fontColor:"Yellow", fontFamily:
            "Arial");
        photo.Save(@"~¥" + imagePath); }
    }
<!DOCTYPE html>
<html>
<head>
    <title>Water Mark</title>
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
</head>
<body>
    <h1>Adding a Watermark to an Image</h1>
    @if(imagePath != ""){
        <div class="result">
            <img src="@imagePath" alt="Image" />
        </div>
    }
</body>
</html>
```

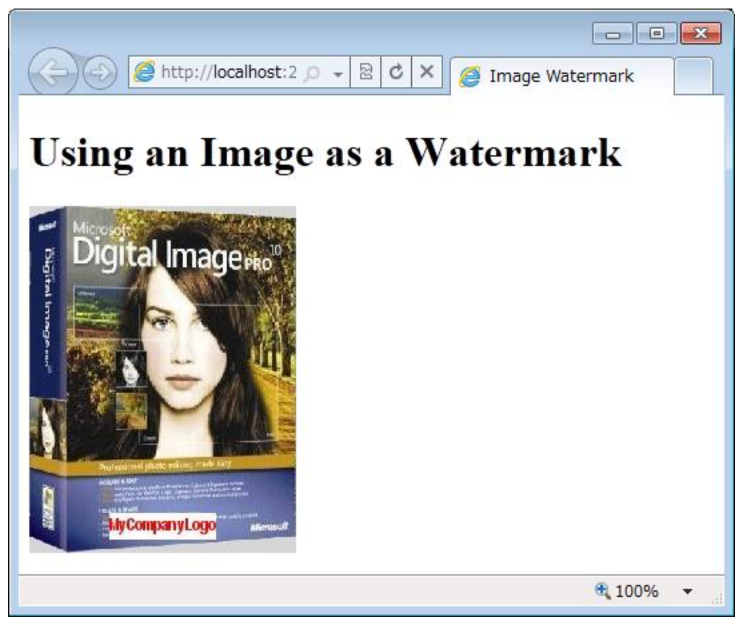
このコードは、前述の Flip.cshtml のコードに似ています。（ただし、今回は Photo3.jpg ファイルを使っています。） ウォーターマークを追加するためには、イメージを保存する前に、WebImage ヘルパーの AddTextWatermark メソッドを呼び出します。AddTextWatermark の呼び出しで、フォントの色を黄色に設定し、フォント ファミリを Arial に設定して、"My Watermark"というテキストを渡します。（ここでは示されていませんが、WebImage ヘルパーは、ウォーターマークテキストの不透明度、フォント ファミリやフォント サイズ、位置を指定することもできます。） イメージを保存するとき、読み込み専用であってはいけません。

以前に見たとおり、src 属性に@imagePath を設定した<img>要素を使って、イメージがページ上に表示されます。

3. このページをブラウザーで実行します。

## イメージをウォーターマークとして使う

ウォーターマークとしてテキストを使う代わりに、他のイメージを使うこともできます。しばしば、社名ロゴをウォーターマークとして使うことや、テキストの著作権情報の代わりにウォーターマークイメージを使うことがあります。




1. 新たに ImageWatermark.cshtml という名前のページを追加します。
2. images フォルダーに、ロゴとして使いたいイメージを追加し、このイメージを MyCompanyLogo.jpg という名前にします。このイメージは、幅 80 ピクセル、高さ 20 ピクセルのサイズに設定されたときに、はっきりわかるイメージにします。
3. 既存のマークアップを以下で置き換えます。

```
@{ var imagePath = "";
    WebImage WatermarkPhoto = new WebImage(@"~¥" +
        @"¥Images¥MyCompanyLogo.jpg");
    WebImage photo = new WebImage(@"~¥Images¥Photo4.jpg");
    if(photo != null){
        imagePath = @"images¥Photo4.jpg";
        photo.AddImageWatermark(WatermarkPhoto, width: 80, height: 20,
            horizontalAlign:"Center", verticalAlign:"Bottom",
            opacity:100, padding:10);
        photo.Save(@"~¥" + imagePath);
    }
}
<!DOCTYPE html>
<html>
```

```
<head>
    <title>Image Watermark</title>
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
</head>
<body>
    <h1>Using an Image as a Watermark</h1>
    @if(imagePath != ""){
        <div class="result">
            <img src="@imagePath" alt="Image" />
        </div>
    }
</body>
</html>
```

これは、前の例のコードを変更したものです。ここでは、イメージを保存する前に、対象となるイメージ（Photo4.jpg）にウォーターマークイメージを追加するため、AddImageWatermark を呼び出します。AddImageWatermark を呼び出すとき、イメージは幅（width）が 80 ピクセル、高さ（height）が 20 ピクセルに設定されます。MyCompanyLogo.jpg イメージは対象イメージにおいて水平方向（horizontalAlign）の中央（Center）、垂直方向（verticalAlign）の下部（Bottom）に配置されます。不透明度（opacity）は 100%に設定し、余白（padding）は 10 ピクセルに設定されます。ウォーターマーク イメージが対象となるイメージよりも大きいときには、何も起きません。ウォーターマーク イメージが対象イメージよりも大きく、ウォーターマーク イメージの余白が 0 に設定されているとき、ウォーターマークは無視されます。

前述のように、動的に src 属性を設定した<img>要素を使ってイメージを表示します。

4. このページをブラウザーで実行します。

## その他のリソース

- 第 8 章　ファイルを扱う
- ASP.NET Web Pages with Razor Syntax Reference

# 第１０章　　ビデオを扱う

本章では、Razor 構文を使う ASP.NET Web ページにおいてビデオを表示する方法について解説します。

Razor 構文を使う ASP.NET Web ページでは、Flash（.swf）、メディア プレイヤー（.wmv）、Silverlight（.xap）のビデオを再生できます。

ここでは次のことを学びます。

- ビデオ プレイヤーを選択する方法
- Web ページにビデオを追加する方法
- ビデオ プレイヤーの属性を設定する方法

本章で紹介する ASP.NET Razor ページの機能は、以下の通りです。

- Video ヘルパー

## ビデオ プレーヤーを選ぶ

ビデオ ファイルには数多くの形式がありますが、それぞれの形式は別々のプレイヤーを必要とし、プレイヤーの設定方法も異なります。ASP.NET Razor ページでは、Video ヘルパーを使うことで、Web ページ上でビデオを再生できます。Video ヘルパーは、通常ページにビデオを追加するために使われる object と embed という HTML 要素を自動的に生成することで、Web ページへのビデオの埋め込みを単純化します。

Video ヘルパーは、以下のメディア プレイヤーをサポートしています。

- Adobe Flash
- Windows Media Player
- Microsoft Silverlight

### Flash プレイヤー

Video ヘルパーの Flash プレイヤーは、Web ページ上で Flash ビデオ（.swf ファイル）を再生できるようにします。最低限、ビデオ ファイルへのパスを提供しなければなりません。パス以外に何も指定しなければ、プレイヤーは Flash の現在使われているバージョンで設定されるデフォルトの値を使います。典型的なデフォルトの設定は、以下の通りです。

- ビデオは、デフォルトの幅と高さを使い、背景色なしで表示されます。
- ビデオは、ページが読み込まれたときに自動的に再生されます。
- ビデオは、明示的に停止されるまで、繰り返し再生されます。
- ビデオは、指定されたサイズに合うよう切り取られるのではなく、全体を表示するようサイズが調整されます。
- ビデオは、ウィンドウの中で再生されます。

## MediaPlayer プレイヤー

Video ヘルパーの MediaPlayer プレイヤーは、Web ページ上で Windows Media ビデオ（.wmv ファイル）、Windows Media オーディオ（.wma ファイル）、MP3（.mp3 ファイル）を再生できるようにします。再生するメディア ファイルのパスを含む必要はありますが、その他のパラメーターはオプションです。パスだけを指定すると、プレイヤーは以下のような Media Player の現在使われているバージョンで設定されるデフォルトの設定を使います。

- ビデオは、デフォルトの幅と高さを使って表示されます。
- ビデオは、ページが読み込まれたときに自動的に再生されます。
- ビデオは、一度だけ再生されます（繰り返しません）。
- プレイヤーは、ユーザー インターフェイスのコントロール セットを表示します。
- ビデオは、ウィンドウの中で再生されます。

## Silverlight プレイヤー

Video ヘルパーの Silverlight プレイヤーは、Web ページ上で Windows Media ビデオ（.wmv ファイル）、Windows Media オーディオ（.wma ファイル）、MP3（.mp3 ファイル）を再生できるようにします。Silverlight ベースのアプリケーション パッケージ（.xap ファイル）を指すパスをパラメーターとして設定しなければなりません。さらに、幅と高さのパラメーターも設定する必要があります。その他のパラメーターは、オプションです。ビデオのために Silverlight プレイヤーを使う場合、必要なパラメーターだけを設定すれば、Silverlight プレイヤーは背景色なしでビデオを表示します。

**注意**　Silverlight についてご存じない方へ。.xap ファイルは、レイアウトの手順を示す.xaml ファイル、マネージコードを持つアセンブリ、その他のリソースから構成される圧縮されたファイルです。.xap ファイルは、Visual Studio で Silverlight アプリケーション オブジェクトとして作成できます。

Silverlight ビデオ プレイヤーは、プレイヤーのために提供した設定と、.xap ファイルで提供される設定の両方を使います。

**MIME の型**

ブラウザーがファイルをダウンロードするとき、ブラウザーは、ファイルの種類がレンダリングされたドキュメントのために指定される MIME 型と一致するかどうかを確認します。MIME 型は、ファイルのコンテンツの種類やメディアの種類をあらわします。Video ヘルパーは、以下の MIME 型を使います。

application/x-shockwave-flash
application/x-mplayer2
application/x-silverlight-2

## Flash（.swf）ビデオを再生する

以下の手順は、sample.swf という名前の Flash ビデオを再生する方法を示します。この手順は、サイトに Media という名前のフォルダーがあり、そのフォルダーに.swf ファイルがあることを想定しています。

1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で述べたように、まだ追加していなければ、Web サイトに ASP.NET Web Helpers Library を追加します。
2. Web サイトで、ページを追加し、FlashVideo.cshtml と名付けます。
3. このページに、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
<head>
    <title>Flash Video</title>
</head>
<body>
    @Video.Flash(path: "Media/sample.swf",
                 width: "400",
                 height: "600",
                 play: true,
                 loop: true,
                 menu: false,
                 bgColor: "red",
                 quality: "medium",
                 scale: "exactfit",
                 windowMode: "transparent")
</body>
</html>
```

4. このページをブラウザーで実行します。（実行する前に、「ファイル」ワークスペースで、このページが選択されていることを確認してください。）　ページは表示され、ビデオは自動的に再生されます。

Flash ビデオの quality パラメーターを low、autolow、autohigh、medium、high、best に設定できます。

```
// Set the Flash video quality
@Video.Flash(path: "Media/sample.swf", quality: "autohigh")
```

scale パラメーターを使って、Flash ビデオを指定したサイズで再生するように変更できます。設定できるものは以下の通りです。

- showall。元のアスペクト比を維持したままビデオ全体を表示します。ただし、上下または左右に縁が表示されるかもしれません。
- noorder。元のアスペクト比を維持したままビデオのサイズが調整されます。ただし、一部が切り取られるかもしれません。
- exactfit。元のアスペクト比を維持せず、ビデオ全体を表示します。ただし、歪みが生じるかもしれません。

scale パラメーターを指定しない場合、ビデオ全体が表示され、一部が切り取られることなく元のアスペクト比が維持されます。以下の例は、scale パラメーターを使う方法を示しています。

```
// Set the Flash video to an exact size
@Video.Flash(path: "Media/sample.swf", width: "1000", height: "100",
    scale: "exactfit")
```

Flash プレイヤーは、windowMode というビデオ モード設定をサポートします。これは window、opaque、transparent に設定できます。デフォルトでは、windowMode は window に設定され、ビデオは Web ページとは独立したウィンドウに表示されます。opaque という設定は、Web ページ上のものをビデオの背後に隠します。Transparent という設定は、ビデオが透明であるかのように、Web ページの背景としてビデオを表示します。

## MediaPlayer（.wmv）ビデオを再生する

以下の手順は、Media フォルダーにある sample.wmv という名前の Window Media ビデオを再生する方法を示しています。

1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で述べたように、まだ追加していなければ、Web サイトに ASP.NET Web Helpers Library を追加します。
2. Web サイトで、ページを追加し、MediaPlayerVideo.cshtml と名付けます。
3. このページに、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
<head>
    <title>MediaPlayer Video</title>
</head>
<body>
    @Video.MediaPlayer(
        path: "Media/sample.wmv",
        width: "400",
        height: "600",
        autoStart: true,
        playCount: 2,
        uiMode: "full",
        stretchToFit: true,
        enableContextMenu: true,
        mute: false,
        volume: 75)
</body>
</html>
```

4. このページをブラウザーで実行します。ビデオが読み込まれて、自動的に再生されます。

playCount に整数値を設定して、ビデオを自動的に再生する回数を指定できます。

```
// Set the MediaPlayer video playCount
@Video.Flash(path: "Media/sample.swf", playCount: 2)
```

uiMode パラメーターを使うと、ユーザー インターフェイスに表示されるコントロールを指定できます。uiMode は invisible、none、mini、full に設定できます。uiMode パラメーターを指定しないと、ビデオはステータス ウィンドウ、シーク バー、コントロール用のボタン、ボリュームコントロールをビデオ ウィンドウに追加して表示します。これらのコントロールは、オーディオ ファイルを再生するためにプレイヤーを使うときにも表示されます。以下の例は、uiMode パラメーターの使い方を示しています。

```
// Set the MediaPlayer control UI
@Video.MediaPlayer(path: "Media/sample.wmv", uiMode: "mini")
```

デフォルトでは、ビデオ再生のときの音声は有効になっています。Mute パラメーターを true に設定すると、音声を無音にできます。

```
// Play the MediaPlayer video without audio
@Video.MediaPlayer(path: "Media/sample.wmv", mute: true)
```

volume パラメーターの値を 0 から 100 の間に設定することで、MediaPlayer ビデオの音量レベルを調節できます。

```
// Play the MediaPlayer video without audio
@Video.MediaPlayer(path: "Media/sample.wmv", volume: 75)
```

## Silverlight ビデオを再生する

以下の手順は、Media という名前のフォルダーにある Silverlight の.xap ページに含まれるビデオを再生する方法を示しています。

1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で述べたように、まだ追加していなければ、Web サイトに ASP.NET Web Helpers Library を追加します。
2. Web サイトで、ページを追加し、SilverlightVideo.cshtml と名付けます。
3. このページに、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
<head>
    <title>Silverlight Video</title>
</head>
<body>
    @Video.Silverlight(
        path: "Media/sample.xap",
        width: "400",
        height: "600",
        bgColor: "red",
        autoUpgrade: true)
</body>
</html>
```

4. このページをブラウザーで実行します。

## その他のリソース

- Silverlight の概要
- OBJECT タグおよび EMBED タグの属性について
- Windows Media Player 11 SDK PARAM Tags
- ASP.NET Web Pages with Razor Syntax Reference

# 第１１章　　Web サイトに電子メールを追加する

本章では、Web サイトから自動的に電子メール メッセージを送信する方法について解説します。

ここでは次のことを学びます。

- Web サイトから電子メール メッセージを送信する方法
- 電子メール メッセージにファイルを添付する方法

本章で紹介する ASP.NET の機能は、以下の通りです。

- WebMail ヘルパー

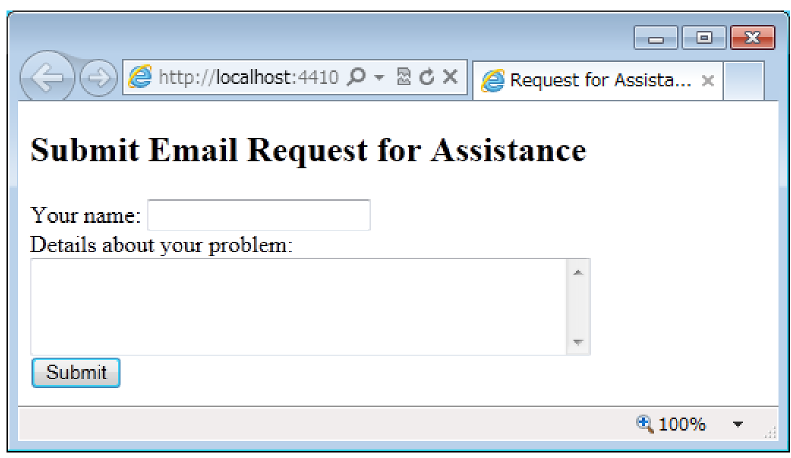
## Web サイトから電子メール メッセージを送信する

Web サイトから電子メールを送信する必要については、さまざまな理由が考えられます。ユーザーに確認メッセージを送信したり、自分自身に（たとえば、新しいユーザーが登録されたといった）通知を送信したりすることがあります。WebMail ヘルパーは、電子メールの送信を容易にします。

WebMail ヘルパーを使うためには、SMTP サーバーにアクセスする必要があります（STMP は、Simple Mail Transfer Protocol の略です）。SMTP サーバーは、メッセージを受信者のサーバーに転送する、つまり電子メールの外向きを行う電子メール サーバーです。Web サイトのためにホスティング プロバイダーを使う場合は、電子メールを設定するために、SMTP サーバー名を知る必要があります。社内ネットワークで作業しているときは、管理者か IT 部門に連絡して、使える SMTP サーバーの情報を入手します。自宅で作業する場合は、通常の電子メール プロバイダーが使えるかどうか確かめて、SMTP サーバー名を調べてください。通常は、以下のものが必要です。

- SMTP サーバー名
- ポート番号（通常は 25 ですが、ISP によってはポート 587 を使う必要があります）
- クレデンシャル（ユーザー名、パスワード）

以下の手順では、2 つのページを作成します。最初のページは、テクニカル サポートの入力フォームを埋めるために使われるような、ユーザーが説明を入力するフォームを持ちます。最初のページは、その情報を 2 つめのページに送ります。2 つめのページでは、コードはユーザーの情報を抽出して、電子メール メッセージを送信します。また、問題レポートを受け取ったことを確認するメッセージを表示します。




**注意**　この例を単純化するため、コードは使われるページの中で WebMail ヘルパーを初期化します。しかし、実運用する Web サイトでは、初期化コードはグローバルなファイルの中に置いておく方がよいでしょう。そうすることで、Web サイトのどのファイルからでも使えるように WebMail ヘルパーは初期化されます。より詳細な情報については、「第 18 章　サイト全体の動作をカスタマイズする」を参照してください。

1. 新しい Web サイトを作成します。
2. 新たに EmailRequest.cshtml という名前のページを追加し、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
<head>
    <title>Request for Assistance</title>
</head>
<body>
    <h2>Submit Email Request for Assistance</h2>
    <form method="post" action="ProcessRequest.cshtml">
        <div>
            Your name:
            <input type="text" name="customerName" />
        </div>

        <div>
            Details about your problem: <br />
            <textarea name="customerRequest" cols="45" rows="4"></textarea>
        </div>

        <div>
            <input type="submit" value="Submit" />
        </div>
    </form>
</body>
</html>
```

フォーム要素の action 属性が ProcessRequest.cshtml に設定されていることに注意してください。これは、このフォームが送信されるのは、そのページであり、今のページに戻るのではないという意味です。

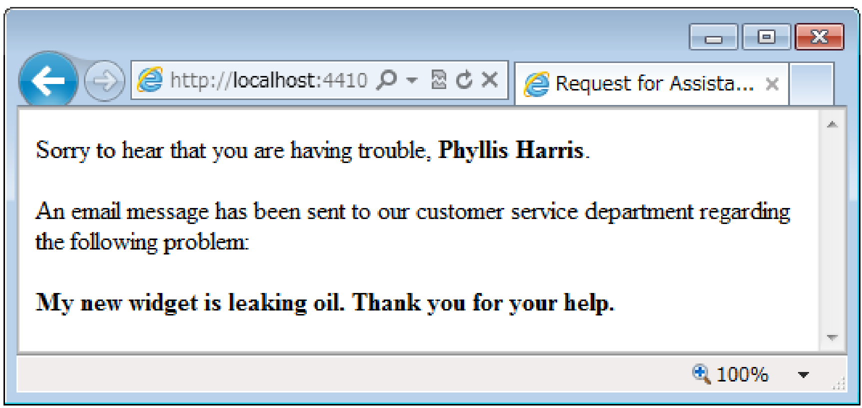
3. Web サイトに、新たに ProcessRequest.cshtml という名前のページを追加し、以下のコードとマークアップを追加します。

```
@{
    var customerName = Request["customerName"];
    var customerRequest = Request["customerRequest"];
    try {
        // Initialize WebMail helper
        WebMail.SmtpServer = "your-SMTP-host";
        WebMail.SmtpPort = 25;
        WebMail.EnableSsl = true;
        WebMail.UserName = "your-user-name-here";
        WebMail.From = "your-email-address-here";
        WebMail.Password = "your-account-password";

        // Send email
        WebMail.Send(to: "target-email-address-here",
            subject: "Help request from - " + customerName,
            body: customerRequest
        );
    }
    catch (Exception ex ) {
        <text>
            <b>The email was <em>not</em> sent.</b>
            The code in the ProcessRequest page must provide an
            SMTP server name, a user name, a password, and
            a "from" address.
        </text>
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Request for Assistance</title>
</head>
<body>
    <p>Sorry to hear that you are having trouble, <b>@customerName</b>.</p>

    <p>An email message has been sent to our customer service
        department regarding the following problem:</p>

    <p><b>@customerRequest</b></p>
</body>
</html>
```

このコードでは、ページに送信されたフォームの項目の値を取得します。次に、WebMail ヘルパーの Send メソッドを呼び出して、電子メール メッセージを作成し、送信します。この場合、使われる値は、フォームから送信された値を連結したテキストを元にしています。

このページのコードは、try/catch ブロックの中にあります。電子メールの送信ができなかった理由が何であれ（たとえば、設定が正しくなかった）、このページが表示されます。<text>タグは、コード ブロックの中のテキストが複数行あることを示すために使われます。（try/catch ブロックや<text>タグの詳細については、「第 2 章　Razor 構文を使った ASP.NET Web プログラミングの紹介」を参照してください。

4. コード中の以下の電子メール関連の設定を修正します。
   - your-SMTP-host に、実際にアクセスできる SMTP サーバーの名前を設定します。
   - your-user-name-here に、SMTP サーバー アカウントのユーザー名を設定します。
   - your-email-address-here に、自分自身の電子メールアドレスを設定します。これは、メッセージの送信元となる電子メール アドレスになります。
   - your-account-password に、SMTP サーバー アカウントのパスワードを設定します。
   - target-email-address-here に、メッセージを送りたい人の電子メール アドレスを設定します。通常、これは受信者の電子メール アドレスです。ただし、テストのときは、自分自身にメッセージを送れます。このためには、ここに自分自身の電子メール アドレスを設定します。ページを実行すると、メッセージを受け取ります。
5. EmailRequest.cshtml ページをブラウザーで実行します。（実行する前に、「ファイル」ワークスペースでこのページが選択されていることを確認してください。）
6. 自分の名前と問題の説明を入力し、[Submit] ボタンをクリックします。ProcessRequest.cshtml ページにリダイレクトされ、メッセージを確認し、電子メールメッセージが送られます。




## 電子メールを使ってファイルを送信する

電子メール メッセージにファイルを添付して送信することもできます。以下の手順では、テキスト ファイルと 2 つの HTML ページを作成します。テキストファイルは、電子メールの添付として使います。

1. Web サイトで、新たにテキストファイルを作成し、MyFile.txt と名付けます。
2. 以下のテキストをコピーし、このファイルに貼り付けます。

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
```

3. 新たに SendFile.cshtml という名前のページを作成し、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Attach File</title>
</head>
<body>
    <h2>Submit Email with Attachment</h2>
    <form method="post" action="ProcessFile.cshtml">
        <div>
            Your name:
            <input type="text" name="customerName" />
        </div>
        <div>
            Subject line: <br />
            <input type="text" size= 30 name="subjectLine" />
        </div>
        <div>
            File to attach: <br />
            <input type="text" size=60 name="fileAttachment" />
        </div>
        <div>
            <input type="submit" value="Submit" />
        </div>
    </form>
</body>
</html>
```

4. ProcessFile.cshtml という名前のページを作成し、以下のマークアップを追加します。

```
@{
    var customerName = Request["customerName"];
    var subjectLine = Request["subjectLine"];
    var fileAttachment = Request["fileAttachment"];

    try {
        // Initialize WebMail helper
        WebMail.SmtpServer = "your-SMTP-host";
        WebMail.SmtpPort = 25;
        WebMail.EnableSsl = true;
        WebMail.UserName = "your-user-name-here";
        WebMail.From = "your-email-address-here";
        WebMail.Password = "your-account-password";

        // Create array containing file name
        var filesList = new string [] { fileAttachment };

        // Attach file and send email
        WebMail.Send(to: "target-email-address-here",
            subject: subjectLine,
            body: "File attached. <br />From: " + customerName,
            filesToAttach: filesList);
    }
    catch (Exception ex) {
        <text>
            <b>The email was <em>not</em> sent.</b>
            The code in the ProcessFile page must provide an
            SMTP server name, a user name, a password, and
            a "from" address.
        </text>
```

```
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Request for Assistance </title>
</head>
<body>
    <p><b>@customerName</b>, thank you for your interest.</p>

    <p>An email message has been sent to our customer service
    department with the <b>@fileAttachment</b> file attached.</p>
</body>
</html>
```

5. コード中の以下の電子メール関連の設定を修正します。
   - your-SMTP-host に、実際にアクセスできる SMTP サーバーの名前を設定します。
   - your-user-name-here に、SMTP サーバー アカウントのユーザー名を設定します。
   - your-email-address-here に、自分自身の電子メールアドレスを設定します。これは、メッセージの送信元となる電子メール アドレスになります。
   - your-account-password に、SMTP サーバー アカウントのパスワードを設定します。
   - target-email-address-here に、自分の電子メールアドレスを設定します。（前述のとおり、通常は、誰か他の人に電子メールを送りますが、テストのため、自分自身に送信します。）
6. SendFile.cshtml ページをブラウザーで実行します。
7. 名前、タイトル行、添付するテキストファイル名（MyFile.txt）を入力します。
8. [Submit] ボタンをクリックします。前述のとおり、ProcessFile.cshtml ページへリダイレクトされ、メッセージを確認し、添付ファイル付きの電子メールメッセージが送られます。

## その他のリソース

- 第 18 章　サイト全体の動作をカスタマイズする
- Simple Mail Transfer Protocol
- ASP.NET Web Pages with Razor Syntax Reference

# 第１２章　　Web サイトに検索を追加する

本章では、検索エンジン Bing を使って Web サイトに検索機能を追加する方法について解説します。
ここでは次のことを学びます。

- Web サイトに、Web サイト（自分自身を含む）の検索機能を追加する方法

本章で紹介する ASP.NET の機能は、以下の通りです。

- Bing ヘルパー

## Web サイトを検索する

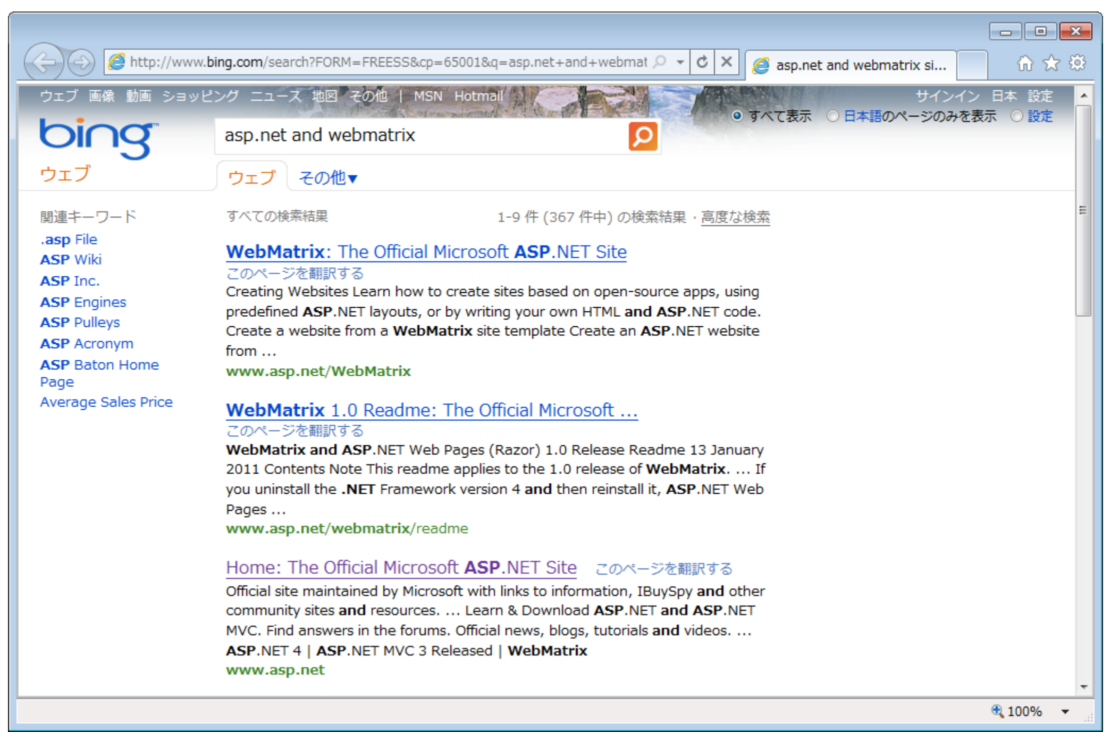
Web サイトから Web 全体を検索する機能を追加することで、自分のサイトを離れることなくインターネットの検索結果を取り込むことができます。サイトに検索を追加するのは、以下の方法が便利です。

- 自分自身のサイト（つまり、現在のサイト）を検索する"Search this site"ボックスを追加します。これは、サイト上のコンテンツを見つけやすくします。
- 関連サイトを検索しやすくするボックスを追加します。たとえば、自身のサイトが学校のスポーツチームに関係するものなら、学校の Web サイトも検索する検索ボックスを追加できます。
- Web 全体を検索させるボックスを追加します。ただし、サイトを離れなくて済むように、別ウィンドウで検索を開きます。

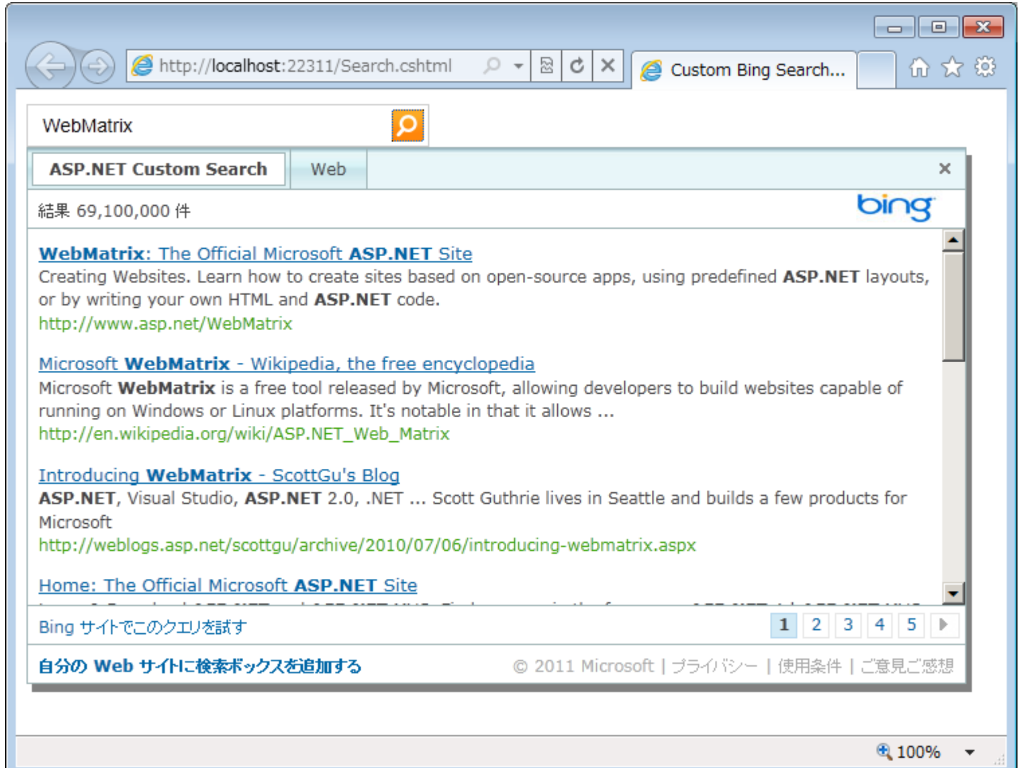
サイトに検索を追加するためには、Bing ヘルパーを使い、（オプションで）検索するサイトの URL を指定します。Bing ヘルパーは、ユーザーが検索語を入力できるテキストボックスをレンダリングします。
検索の設定には、"simple"オプションと、"advanced"オプションの 2 つの方法があります。simple オプションでは、ヘルパーはユーザーが検索を実行するときにクリックできる Bing 検索アイコンを含むボックスをレンダリングします。

Search the ASP.NET site:
asp.net and webmatrix
Search Site　Search Web

検索するサイトを指定すると、ヘルパーはユーザーが指定したサイトだけを検索するか、Web 全体を検索するかを指定するためのラジオボタンをレンダリングします。ユーザーが検索を送信すると、simple オプションは検索結果を Bing サイト（http://bing.com/）にリダイレクトします。結果は、ユーザーが Bing ホームページで検索語を入力したのと同じように、新たなブラウザーのウィンドウに表示されます。




advanced オプションは、ラジオボタンなしで検索ボックスをレンダリングします。ただし、Bing サイトにリダイレクトする代わりに、ヘルパーは検索結果を取り込み、ページ中にそれらを書式化して表示します。




検索結果をどのように書式化するかをオプションで指定できます。単純なサイトで、検索するサイトを指定すると、結果は（Web のサイトを示す）タブで表示され、ユーザーは指定した検索と Web 全体の検索を切り替えて行き来できます。

以下の手順では、simple オプションと advanced オプションの両方を使う方法を示す Web ページを作成します。

1. 新たな Web サイトを作成します。
2. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。
3. 新たに Search.cshtml という名前のページを作成し、以下のマークアップを追加します。

```
@{
    Bing.SiteUrl = "www.asp.net";
    Bing.SiteTitle = "ASP.NET Custom Search";
}
<!DOCTYPE html>
<html>
    <head>
        <title>Custom Bing Search Box</title>
    </head>
    <body>

    <div>
        <h1>Simple Search</h1>
        <p>The simple option displays results by opening a new browser window that shows
the Bing home page.</p>
        Search the ASP.NET site: <br/>
        @Bing.SearchBox()
    </div>

    <div>
        <h1>Advanced Search Option</h1>
        <p>The advanced option shows search results directly in this page. You can specify
options to format the results.</p>
        Search the ASP.NET site: <br/>
        @Bing.AdvancedSearchBox(
            boxWidth: "300px",
            resultWidth: 600,
            resultHeight: 900,
            themeColor: "Green",
            locale: "ja-JP")
        </div>
    </body>
</html>
```

コードでは、Bing ヘルパーを 2 回呼び出しています。最初は SearchBox メソッド（"simple"オプション）を使い、2 回目は AdvancedSearchBox メソッド（"advanced"オプション）を使います。どちらのメソッドも、オプションの siteUrl パラメーターで、検索するサイトを指定できます。（URL を指定しないと、Bing は Web 全体を検索します。）　ここでは、www.asp.net を検索します。自分自身のサイトを検索させたい場合は、www.asp.net をその URL で置き換えてください。

AdvancedSearchBox メソッドでは、検索するサイトの名前を表示する結果ペインで siteName パラメーターが使われます。検索ボックスのサイズは、boxWidth パラメーターを使って設定します。結果ペインのサイズは、resultWidth と resultHeight パラメーターで設定でき、結果ペインの色は themeColor パラメーターで設定できます。さらに、検索や結果を表示するときにヘルパーが使う言語は、locale パラメーターを設定することで指定

できます。themeColor パラメーターで指定できるテーマ カラーのオプションを含め、Bing の検索オプションに関する詳細については Bing ボックスのドキュメントを参照してください。

ロケールは、en-US（英語、アメリカ）、en-GB（英語、イギリス）、es-MX（スペイン語、メキシコ）、fr-CA（フランス語、カナダ）のように、2 つの単語を使って指定します。このリストは、「ロケール コード」を検索すれば見つかります。

| **注意**　ロケールは、ヘルパーがユーザー インターフェイスや結果を表示するときに使われる国/地域と言語を参照するものです。特定の言語に限定したページだけを検索するように指定するものではありません。 |
|---|

4. ブラウザーで、Search.cshtml ページを実行します。（実行する前に、このページが「ファイル」ワークスペースで選択されていることを確認してください。）
5. "simple"ボックスに検索語を入力し、ボタンをクリックします。結果が、新しいブラウザーのウィンドウに表示されます。
6. "advanced"ボックスに検索語を入力します。ページに検索結果を含むペインが表示されます。

| **注意**　Bing ヘルパーが結果を返すために、サイトは公開された状態にあり、そのコンテンツは Bing によって検査（「クロール」）されていなければなりません。"Search this site"ボックスを追加し、Bing ヘルパーで自分自身のサイトを検索させるよう設定するときは、検索エンジンがそのサイトを見つけるまでのしばらくの間はテストできません。言い換えると、WebMatrix の中で検索機能を直接テストすることはできません。 |
|---|

## その他のリソース

- Make your Website SEO friendly
- Locale ID (LCID) Chart
- ASP.NET Web Pages with Razor Syntax
- Bing API documentation
- Bing Box documentation

# 第１３章　Web サイトにソーシャル ネットワーキングを追加する

本章では、サイトにソーシャル ネットワーキング サービスを統合する方法について解説します。

本章では、Facebook や Digg のようなサイト上で、どのように Web サイトをブックマークしたり、リンクしたりするか、さらにサイトに Twitter フィードを追加し、Gravatar のイメージと Xbox ゲーマー カードを使って装飾することを学びます。

ここでは次のことを学びます。

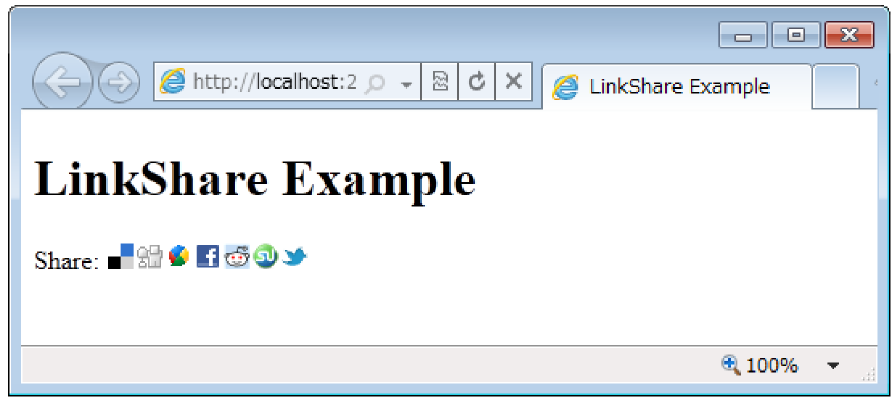
- サイトをブックマークしたり、リンクしたりする方法
- Twitter フィードを追加する方法
- Gravatar.com のイメージをレンダリングする方法
- Xbox ゲーマー カードをサイト上に表示する方法
- Facebook の“Like”ボタンをページに追加する方法

本章で紹介する ASP.NET のプログラミング コンセプトは、以下の通りです。

- LinkShare ヘルパー
- Twitter ヘルパー
- Gravatar ヘルパー
- GamerCard ヘルパー
- Facebook ヘルパー

## ソーシャル ネットワーキング サイトに Web サイトをリンクする

自分のサイトを気に入った人は、それを知り合いと共有したいと思うことがあります。Digg や Reddit、Facebook、Twitter、その他同様のサイト上でページを共有するためにクリックできるグリフ（アイコン）を表示すれば、これを簡単に実現できます。こうしたグリフを表示するために、ページに LinkShare ヘルパーを追加します。ページを見た人は、個々のグリフをクリックできます。ソーシャル ネットワーキング サイトに結びついたアカウントを持っていれば、サイト上のページへのリンクを投稿できます。



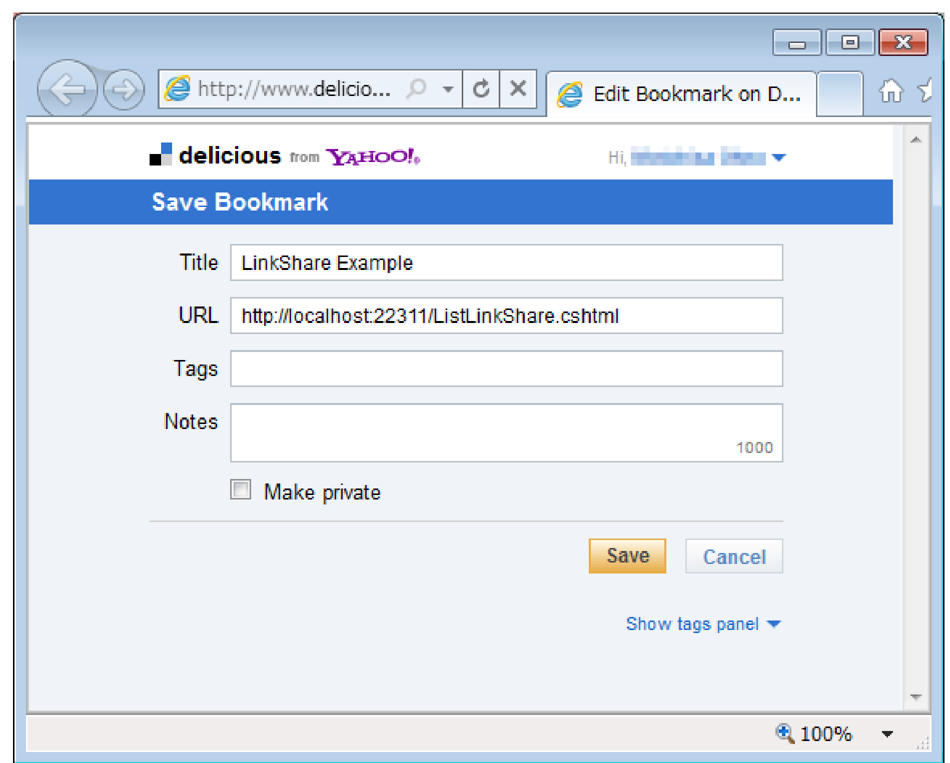

1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。

2. 新たに ListLinkShare.cshtml という名前のページを作成し、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
    <head>
        <title>LinkShare Example</title>
    </head>
    <body>
        <h1>LinkShare Example</h1>
        Share: @LinkShare.GetHtml("LinkShare Example")
    </body>
</html>
```

この例では、LinkShare ヘルパーを実行するとき、ページのタイトルがパラメーターとして渡され、それがソーシャル ネットワーキング サイトへのページタイトルとして渡されます。

3. ListLinkShare.cshtml ページをブラウザーで実行します。（実行前に、「ファイル」ワークスペースで、このページが選択されていることを確認してください）。
4. サイン アップしたサイトの中からひとつ選んでグリフをクリックしてください。そのリンクから、リンクを共有できるソーシャル ネットワーク サイトのページを表示します。たとえば、del.icio.us リンクをクリックすると、Delicious の Web サイト上の Save Bookmark ページが表示されます。




## Twitter フィードを追加する

ASP.NET は、ページに Twitter フィードを追加するためのヘルパーを提供しています。コードで Twitter.Profile メソッドを使うと、特定の Twitter ユーザーの Twitter フィードを自分の Web ページ上に表示できます。コードで

Twitter.Search メソッドを使うと、Twitter 検索（単語、ハッシュタグ、その他の検索可能なテキスト）を指定でき、ページ上に結果を表示できます。どちらのヘルパーも、幅や高さ、スタイルの設定を変更できます。

Twitter 情報へのアクセスは公開されているため、ページ上で Twitter ヘルパーを使うために Twitter アカウントは必要ありません。

以下の手順は、2 つの Twitter ヘルパーをデモンストレーションする Web ページを作成する方法を示しています。

1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。
2. 新たに Twitter.cshtml という名前のページを追加します。
3. ページに、以下のコードとマークアップを追加します。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Twitter Example</title>
    </head>
    <body>
        <table>
            <tr>
                <td>Twitter profile helper</td>
                <td>Twitter search helper</td>
            </tr>
            <tr>
                <td>@Twitter.Profile("<Insert User Name>")</td>
                <td>@Twitter.Search("<Insert search criteria here>")</td>
            </tr>
        </table>
    </body>
</html>
```

4. Twitter.Profile のコード文で、<Insert User Name>を、表示したいフィードのアカウント名に置き換えてください。
5. Twitter.Search のコード文で、<Insert search criteria here>を、検索したいテキストに置き換えてください。
6. このページをブラウザーで実行します。

## Gravatar イメージをレンダリングする

Gravatar ("globally recognized avatar") は、あなたのアバターとして複数の Web サイトで使えるイメージです。つまり、あなたを表すイメージです。たとえば、Gravatar は、フォーラムの投稿、ブログのコメントなどで個人を識別できます。(自分の Gravatar は、http://www.gravatar.com/ にある Gravatar の Web サイトで登録できます。) Web サイト上に、名前や電子メールアドレスの隣にイメージを表示したいときは、Gravatar ヘルパーを使えます。

以下の例では、あなた自身をあらわす単一の Gravatar を使います。Gravatar を使うもう一つの方法は、サイトに登録された人々の Gravatar アドレスを指定できるようにすることです。（登録の方法については、「第 16 章　セキュリティとメンバーシップの追加」で学びます。）　これにより、ユーザーの情報を表示するときはいつでも、ユーザー名の場所に Gravatar を追加できます。

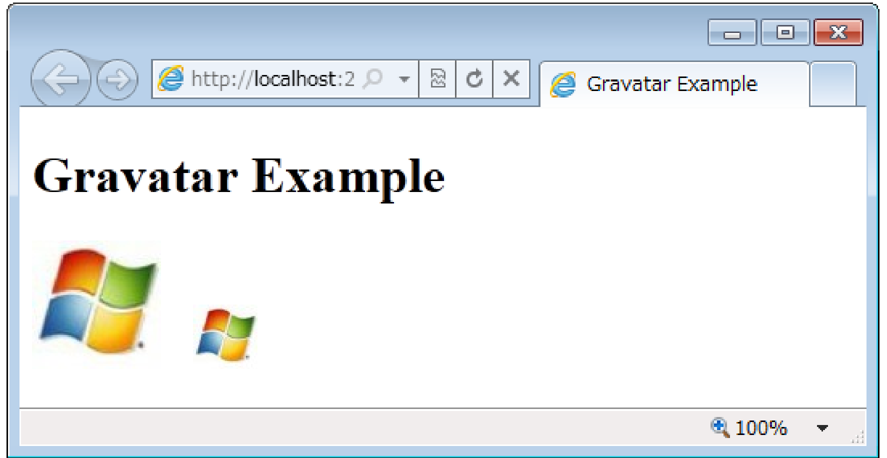
1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。
2. 新たに Gravatar.cshtml という名前の Web ページを追加します。
3. ファイルに、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Gravatar Example</title>
    </head>
    <body>
        <h1>Gravatar Example</h1>
        @Gravatar.GetHtml("<Your Gravatar account here>")
           
        @Gravatar.GetHtml("<Your Gravatar account here>", 40)
    </body>
</html>
```

Gravatar.GetHtml メソッドは、ページ上に Gravatar イメージを表示します。イメージの大きさを変更するには、2 番目のパラメーターで数値を指定します。デフォルトの大きさは 80 です。80 より小さな数値を指定すると、イメージは小さくなります。80 より大きな数値を指定すると、イメージは大きくなります。

4. 2 つの Gravatar.GetHtml メソッド中で、<Your Gravatar account here>を、あなたの Gravatar アカウントで使っている電子メールアドレスに置き換えてください。（もし、Gravatar アカウントを持っていなければ、アカウントを持っている他の人の電子メールアドレスが使えます。）

5. このページをブラウザーで実行します。ページは、指定した電子メールアドレスに対する 2 つの Gravatar イメージを表示します。2 番目のイメージは、最初のイメージよりも小さく表示されます。




## XBOX ゲーマー カードを表示する

Microsoft Xbox のゲームをオンラインで遊ぶ人は、どのユーザーも一意の ID を持っています。どのプレイヤーもゲーマー カードの形式で情報が蓄積され、評判、ゲーマーのスコア、最近遊んだゲームを表示します。Xbox ゲーマーならば、GamerCard ヘルパーを使ってサイトのページ上にゲーマー カードを表示できます。

1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。
2. 新たに XboxGamer.cshtml という名前のページを作成し、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Xbox Gamer Card</title>
    </head>
    <body>
        <h1>Xbox Gamer Card</h1>
        @GamerCard.GetHtml("major nelson")
    </body>
</html>
```

GamerCard.GetHtml メソッドを使って、表示するゲーマー カードのエリアスを指定します。

3. このページをブラウザーで実行します。ページは、指定したゲーマー カードを表示します。

## Facebook の"Like" ボタンを表示する

Facebook ヘルパーの LikeButton メソッドを使えば、Facebook の友人とコンテンツを共有することが容易になります。

Facebook ヘルパーは、“Like”ボタンと、そのページの“Like”を何人の人がクリックしたかを示すカウンタを（Facebook から読みだして）レンダリングします。

サイト上で Facebook の“Like”ボタンをクリックすると、そのページを“Like”と示すユーザーの Facebook フィード上にリンクが表示されます。

RECENT ACTIVITY

likes WebMatrix: The Official Microsoft ASP.NET Site.

デフォルトでは、Facebook ヘルパーの LikeButton メソッドは、現在のページを指す“Like”ボタンを生成します。これは、もっとも一般的なシナリオです。“Like”ボタンを見ると、そのとき読んでいるもの何であれ、Facebook リンクを作成する機会が与えられます。あるいは、LikeButton メソッドを使う Facebook ヘルパーに、URL を渡すこともできます。これは、ページが他のサイトの一覧を表示し、それぞれのサイトに個別に“Like”ボタンを提供したいときに便利です。

LikeButton メソッドでは、“Like”ボタンをどのように表示するかを決めるオプションを指定できます。

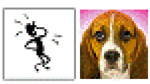
- リンクが、“Like”リンクまたは“Recommend”リンクのどちらを表示するか
- ページの“Like”をクリックした回数をどのように表示するか

Standard: Like 25 people like this.

Button count: Like 25

Box count: 25 Like

- そのページを“Like”とクリックした人の Facebook プロファイル画像を表示するかどうか



- “Like”ボタンを表示する幅と色（明るい、または暗い）

以下の例では、2 つの“Like”ボタンを作成します。ひとつは現在のページを指し、もうひとつは指定した URL（ASP.NET WebMatrix の Web サイト）を指しています。この例を試すためには、Facebook アカウントを持っている必要があります。

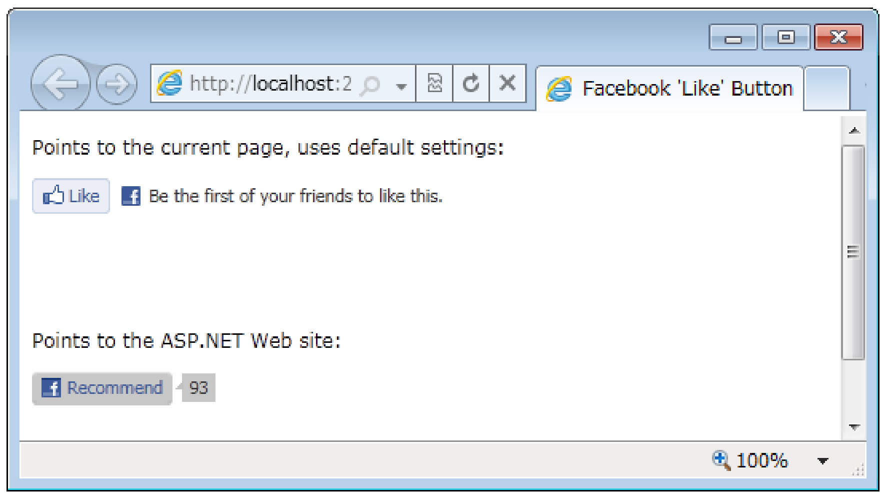
1. 「第１章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに Facebook.Helper Library を追加します（追加していない場合。Facebook ヘルパーは、他の多くのヘルパーと異なるライブラリにあることに注意してください。）。
2. 新たに FacebookLikeBtn.cshtml という名前のページを作成し、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Facebook 'Like' Button</title>
        <style>body {font-family:verdana;font-size:9pt;}</style>
    </head>
    <body>
    <p>Points to the current page, uses default settings:</p>
        @Facebook.LikeButton()

    <p>Points to the ASP.NET Web site:</p>
        @Facebook.LikeButton(
            href: "http://www.asp.net/webmatrix",
            action: "recommend",
            width: 250,
            buttonLayout: "button_count",
            showFaces: true,
            colorScheme: "dark")
    </body>
</html>
```

Facebook.LikeButton メソッドの最初のインスタンスは、デフォルトの設定を使うため、現在のページを指しています。2 番目のインスタンスはオプションを含みます。url パラメーターを使って、“Like”の指す URL を指定

します。“Like”を“Recommend”に変更するには、action パラメーターを"recommend"（デフォルトは"like"）に設定します。カウンタの“ボタン”スタイルを指定するには、layout パラメーターを"button_count"に設定します（その他の設定は"standard"または"box_count"）。“Like”ボタンの下に Facebook のプロファイル画像を表示するには、showFaces パラメーターを true に設定します。最後に、色の設定を変えるには、colorScheme パラメーターを"dark"に設定します（デフォルトは"light"）。

3. この Web ページをブラウザーで実行します。ページは、指定したとおりに Facebook の“Like”ボタンを表示します。




4. ASP.NET の Web サイトを指す“Recommend”ボタンをクリックしてください。Facebook にログインしていない場合は、ログインが促されます。

　WebMatrix でこのページをテストする場合、最初のリンク（現在のページを指す“Like”ボタン）をテストすることはできません。ローカル コンピューター上で（localhost URL を使って）実行しているため、Facebook はリンクをたどれません。しかし、サイトをインターネットに公開すれば、リンクは正しく動作します。

## その他のリソース

- ASP.NET Web Pages with Razor Syntax

# 第１４章　　トラフィックの解析

Web サイトを稼働させた後、Web サイトのトラフィックを解析したいことがあります。

ここでは次のことを学びます。

- Web サイトのトラフィックに関する情報を解析プロバイダーに送信する方法

本章で紹介する ASP.NET のプログラミング機能は、以下の通りです。

- Analytics ヘルパー

## ビジター情報の追跡（解析）

解析とは、Web サイトのトラフィックを測定する技術のための一般的な用語であり、サイトがどのように使われているかを理解できます。Google、Yahoo、StatCounter など、多くの解析サービスがあります。

解析するには、解析プロバイダーのアカウントをサインアップし、追跡したいサイトを登録します。プロバイダーから、アカウントのための ID を含む JavaScript コードの断片（スニペット）が送られます。追跡したいサイト上の Web ページに、この JavaScript スニペットを追加します。（通常、解析スニペットは、サイト上のすべてのページで使われているフッター、レイアウトページ、あるいは他の HTML マークアップに追加します。）　こうした JavaScript スニペットを含むページがリクエストされると、スニペットは現在のページに関する情報を解析プロバイダーに送信し、ページに関する詳細を記録します。

サイトの統計を見たい場合は、解析プロバイダーの Web サイトにログインします。すると、サイトに関する以下のようなさまざまな種類のレポートを見ることができます。

- 個々のページごとの、ページビューの数。とくに、何人の人がサイトを訪問し、サイトのどのページが最も利用されているかが（おおざっぱに）わかります。
- 指定したページの滞在時間。ホームページが訪問者の関心を惹いているかどうかといったことがわかります。
- サイトを訪問する前に、どのサイトにいたか。トラフィックがどのリンクやサーチなどから生じたものかを理解する手助けになります。
- サイトを訪問した人々が、どれだけ滞在しているか。
- どの国からアクセスしているか。
- どんなブラウザーやオペレーティングシステムが使われているか。

ASP.NET は、いくつかの解析ヘルパーを含みます（Analytics.GetGoogleHtml、Analytics.GetYahooHtml、そして Analytics.GetStatCounterHtml）。このため、解析のための JavaScript スニペットを管理するのは容易です。JavaScript コードをどのように、どこに配置するかを考える代わりに、しなければならないのはページにヘルパーを追加することだけです。渡す必要がある情報は、アカウント名だけです。（StatCounter については、いくつか追加の値を渡す必要があります。）

以下の手順では、GetGoogleHtml ヘルパーを使うレイアウトページを作成します。他の解析プロバイダーのアカウントを持っている場合は、そのアカウントを使うこともできます。

**注意**　解析用のアカウントを作成するとき、追跡したいサイトの URL を登録します。ローカル コンピューター上のものをテストしようとすると、実際のトラフィックは追跡できないため（あなたのトラフィックだけです）、記録したり、サイトの統計を表示することはできません。以下の手順は、ページに解析ヘルパーを追加する方法を示しているだけです。サイトを公開すると、実際のサイトが解析プロバイダーに情報を送ります。

1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。
2. Google Analytics でアカウントを作成し、アカウント名を記録します。
3. 新たに Analytics.cshtml という名前のレイアウト ページを作成し、以下のマークアップを追加します。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Analytics Test</title>
    </head>
    <body>
        <h1>Analytics Test Page</h1>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
        sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. </p>
        <div id="footer">
            &copy; 2010 MySite
        </div>
```

```
        @Analytics.GetGoogleHtml("myaccount")
    </body>
</html>
```

**注意**　Analytics ヘルパーの呼び出しは、Web ページの本体の中（</body>タグの前）に配置しなければなりません。そうしないと、ブラウザーはスクリプトを実行しません。

他の解析プロバイダーを使う場合は、代わりに以下のヘルパーを使ってください。

- (Yahoo) @Analytics.GetYahooHtml("myaccount")
- (StatCounter) @Analytics.GetStatCounterHtml("project", "security")

4. Myaccount を、ステップ 2 で作成したアカウント名で置き換えてください。
5. このページをブラウザーで実行します。（実行する前に、「ファイル」ワークスペースでこのページが選択されていることを確認してください。）
6. ブラウザーで、ページのソースを表示します。以下のような解析コードがレンダリングされていることがわかります。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Analytics Test</title>
    </head>
    <body>
        <h1>Analytics Test Page</h1>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
        sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
        <div id="footer">
            &copy; 2010 MySite
        </div>
        <script type="text/javascript">
            var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." : 
"http://www.");
            document.write(unescape("%3Cscript src='" + gaJsHost + 
"google-analytics.com/ga.js' type='text/javascript'%3E%3C/script%3E"));
        </script>
        <script type="text/javascript">
            try{
            var pageTracker = _gat._getTracker("myaccount");
            pageTracker._trackPageview();
            } catch(err) {}
        </script>
    </body>
</html>
```

7. Google Analytics サイトにログオンし、サイトの統計を確認します。サイトを実運用している場合、ページの訪問者に関する記録を確認できます。

# 第１５章　Web サイトのパフォーマンスを改良するためのキャッシング

記憶領域を持つことで Web サイトをスピードアップできます。つまり、データをキャッシュすることで、頻繁に変更されないデータの抽出や処理にかかる時間を大幅に節約できます。

ここでは次のことを学びます。

- Web サイトのレスポンスを改善するためにキャッシュを使う方法

本章で紹介する ASP.NET のプログラミング機能は、以下の通りです。

- WebCache ヘルパー

## Web サイトの応答を改良するためのキャッシング

サイトのページがリクエストされるときはいつでも、Web サーバーはそのリクエストを実行する作業をしなければなりません。ページによっては、サーバーはデータベースからデータを抽出するような（比較的）長い時間かかる処理を実行することになるかもしれません。1 回の処理にはさほど長い時間がかかるものでなくても、サイトに多くのトラフィックがあると、個々のリクエストが全体として大量の処理を要求することになり、サーバーへの実行を複雑化や低速化させることになります。

このような状況で Web サイトのパフォーマンスを改善するひとつの方法が、データをキャッシュすることです。サイトが同じ情報を繰り返し取得する場合で、個人ごとに情報を変更する必要がなく、時間にも影響されないのであれば、再取得や再計算する代わりに、データを一度だけ取得して、結果を保存しておけます。その情報に対する次回のリクエストでは、キャッシュから取り出すだけです。

一般に、キャッシュする情報は頻繁に変更しません。キャッシュに配置する情報は、Web サーバーのメモリーに保存されます。キャッシュがどれくらいの期間保存されるかは秒単位から日単位で指定できます。キャッシュ期間が過ぎると、自動的に情報はキャッシュから削除されます。

**注意**　キャッシュに置かれたものは、期間が過ぎる以外の理由で削除されることもあります。たとえば、一時的に Web サーバーのメモリが少なくなり、キャッシュから取り出すことでメモリを再構成するかもしれません。以下の通り、キャッシュに情報を置く場合は、必要なときにまだ情報があるかどうかを確認する必要があります。

Web サイトが現在の温度や天気予報を表示するページを持っていると考えてみてください。この種の情報を取得するには、外部のサービスにリクエストを送ります。そうした情報はあまり変更されることはないので（たとえば、2 時間ごと）、外部呼出しにかかる時間や帯域のために、キャッシュを使うよい候補です。

ASP.NET は、サイトにキャッシング機能を追加し、キャッシュにデータを追加しやすくするため、WebCache ヘルパーを提供しています。以下の手順では、現在時刻をキャッシュするページを作成しています。本来、現在時刻は頻繁に変更されるものであり、複雑な計算でもないため、これは実用的な例ではありません。しかし、キャッシュがどのように機能するかを見るにはよい方法です。

1. Web サイトに新たに WebCache.cshtml という名前のページを追加します。
2. 以下のコードとマークアップをページに追加します。

```
@{
    var cacheItemKey = "Time";
    var cacheHit = true;
    var time = WebCache.Get(cacheItemKey);

    if (time == null) {
        cacheHit = false;
    }

    if (cacheHit == false) {
        time = @DateTime.Now;
        WebCache.Set(cacheItemKey, time, 1, false);
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>WebCache Helper Sample</title>
</head>
<body>
    <div>
        @if (cacheHit) {
            @:Found the time data in the cache.
        } else {
            @:Did not find the time data in the cache.
        }
    </div>
    <div>
        This page was cached at @time.
    </div>
</body>
</html>
```

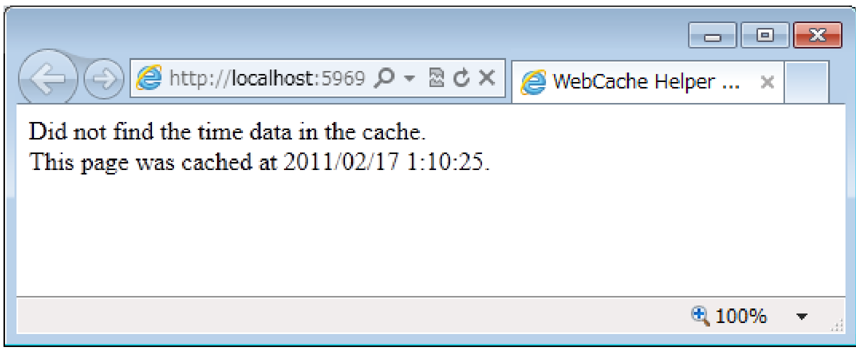
データをキャッシュするときは、Web サイトを通じて一意の名前を使ってキャッシュに保存します。ここでは、Time という名前のキャッシュ エントリを使います。これは、コード中の cacheItemKey です。

このコードは、まず Time キャッシュ エントリを読み込みます。値が返されると（つまり、キャッシュ エントリが空でない）、コードはキャッシュ データから time 変数の値を設定します。

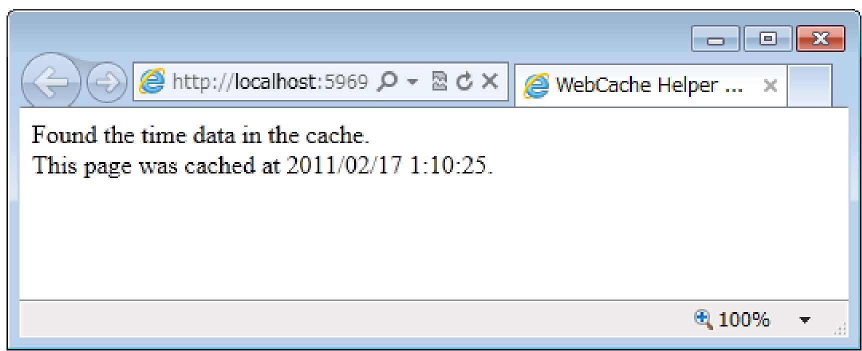
しかし、キャッシュが存在しない場合（つまり、空）、コードは time の値を設定し、キャッシュに追加し、失効値を 1 分に設定します。ページが 1 分以内にリクエストされなければ、キャッシュ エントリは破棄されます。（キャッシュの項目に対するデフォルトの失効値は、20 分です。）

このコードは、データをキャッシュするときに使うパターンを示しています。キャッシュから何かを取り出す前に、必ず WebCache.Get メソッドが null を返すかどうかを確認します。キャッシュ エントリが失効するか、何かの理由で削除されているかもしれないことを覚えておいてください。与えられたエントリがキャッシュにあることは保証されていません。

3. WebCache.cshtml をブラウザーで実行します。（実行する前に、「ファイル」ワークスペースでこのページが選択されていることを確認してください。） ページを最初にリクエストするとき、時刻データはキャッシュになく、コードは時刻の値をキャッシュに追加します。




4. WebCache.cshtml をブラウザーで再読み込みしてください。今回は、時刻データがキャッシュされています。時刻は最後にページを見たときから変更されていません。




5. キャッシュが空になるまで 1 分間待って、ページを再読み込みしてください。ページは、キャッシュに時刻データがないことを示し、更新された時刻がキャッシュに追加されます。

## その他のリソース

- 第 7 章　データをグラフで表示する

# 第１６章　セキュリティとメンバーシップの追加

本章では、ページをログインした人々にしか見せないようにして、Web サイトを安全にする方法を示します。（誰もがアクセスできるページを作成する方法についても紹介します。）

ここでは次のことを学びます。

- 登録ページとログインページを持ち、いくつかのページがメンバーに対してのみアクセスできる Web サイトを作成する方法
- パブリックとメンバー専用ページの作成方法
- メンバー アカウントを作成する際に自動化プログラム（Bot）を防ぐための CAPTCHA を使う方法

本章で紹介する ASP.NET 機能は、以下の通りです。

- WebSecurity ヘルパー
- SimpleMembership ヘルパー
- ReCaptcha ヘルパー

## Web サイト メンバーシップの紹介

ユーザーがログインできるような Web サイト、つまりメンバーシップをサポートするサイトを作成できます。これが有用なさまざまな理由があります。たとえば、サイトにメンバーだけが利用できる機能を持たせることができます。あるいは、フィードバックを送信したり、コメントを残すために、ユーザーにログインを要求できます。

Web サイトがメンバーシップをサポートする場合でも、そのサイトではユーザーのログインを必要としないページを使うことができます。ログインしていないユーザーは、匿名ユーザーと呼ばれます。

ユーザーは、Web サイトで登録でき、サイトにログインできるようになります。Web サイトは、ユーザーが誰であるかを確認できるようなユーザー名（しばしば電子メールアドレスが使われます）とパスワードを要求します。ログインのプロセスや、ユーザー アイデンティティの確認は、認証と呼ばれます。

WebMatrix は、「スターター サイト」テンプレートを使って、以下を含む Web サイトを作成します。

- メンバーのユーザー名とパスワードを保持するために使われるデータベース
- 匿名の（新しい）ユーザーが登録できる場所としての登録ページ
- ログインとログアウト ページ
- パスワードの回復とリセット用のページ

**注意**　「スターター サイト」テンプレートは自動的にこれらのページを作成しますが、本章では ASP.NET のセキュリティとメンバーシップの基礎を学ぶために、これらの簡易バージョンを手作業で作成します。

## 登録とログインページを持つ Web サイトの作成

1. WebMatrix を起動します。
2. 「クイック スタート」ページで、「テンプレートからサイトを作成する」を選びます。
3. 「スターター サイト」テンプレートを選び、[OK] をクリックします。WebMatrix は、新しいサイトを作成します。
4. 左ペインで、「ファイル」ワークスペースをクリックします。
5. Web サイトのルート フォルダーで、グローバルな設定を保持するために使われている_AppStart.cshtml ファイルを開きます。これは、//文字を使ってコメントアウトされている、いくつかの文を含みます。

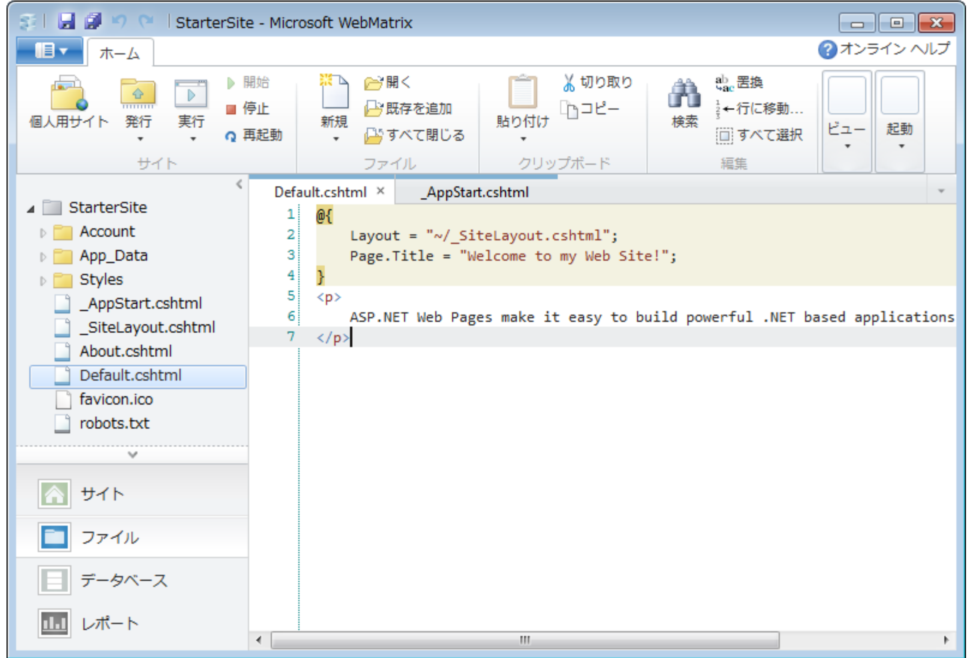
```
@{
    WebSecurity.InitializeDatabaseConnection("StarterSite", "UserProfile", "UserId",
        "Email", true);
    // WebMail.SmtpServer = "mailserver.example.com";
    // WebMail.EnableSsl = true;
    // WebMail.UserName = "username@example.com";
    // WebMail.Password = "your-password";
    // WebMail.From = "your-name-here@example.com";
}
```

電子メールを送信できるようにするには、WebMail ヘルパーが使えます。これは、「第 11 章　Web サイトで電子メールを使う」で説明したように、SMTP サーバーへのアクセスを必要とします。その章では、ひとつのページにさまざまな SMTP の設定を使う方法を示しました。本章では、これらの設定のいくつかを使いますが、これを中心となるファイルに配置するため、それぞれのページにコードを書く必要はありません。（登録用のデータベースを設定するために SMTP の設定を構成する必要はありません。電子メールのエリアスからユーザーが正しいかどうかを検証したい場合やユーザーが忘れたパスワードをリセットする場合に、SMTP の設定が必要です。）

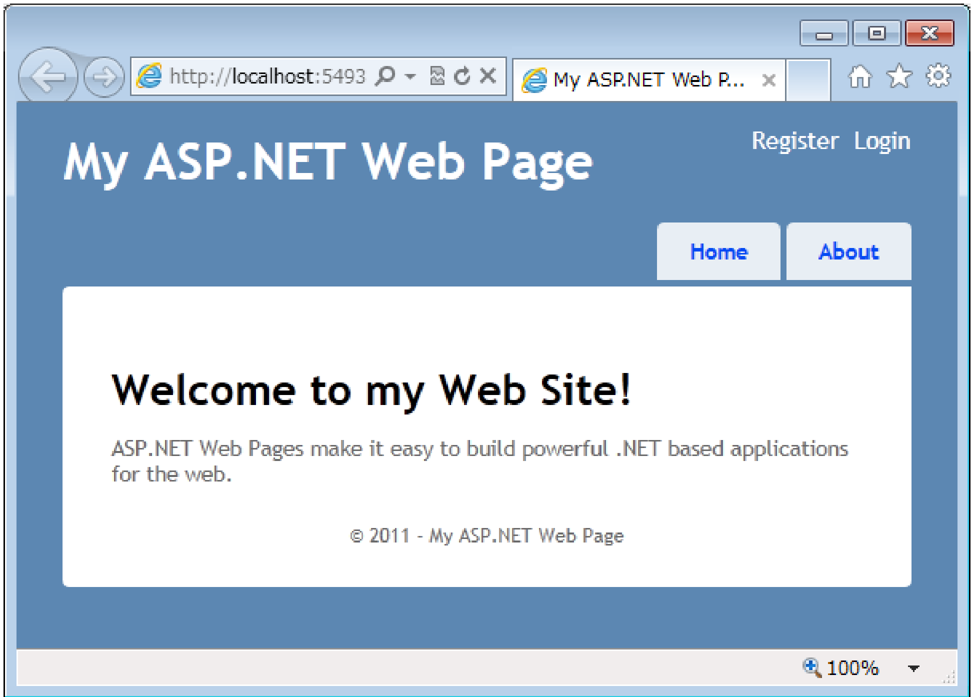
6. これらの文をコメントでなくします。（各行の先頭にある//を削除します）
7. コード中で、以下の電子メール関連の設定を修正します。

- WebMail.SmtpServer を、実際にアクセスできる SMTP サーバーの名前に設定します。
- WebMail.EnableSsl を true にしておきます。この設定は、SMTP サーバーへのクレデンシャルの送信を暗号化によって安全にします。
- WebMail.UserName を、SMTP サーバー アカウントのユーザー名に設定します。
- WebMail.Password を、SMTP サーバー アカウントのパスワードに設定します。
- WebMail.From を、自分自身の電子メールアドレスに設定します。これは、メッセージの送信元の電子メールアドレスになります。

8. _AppStart.cshtml を保存して、閉じます。
9. Default.cshtml ファイルを開きます。



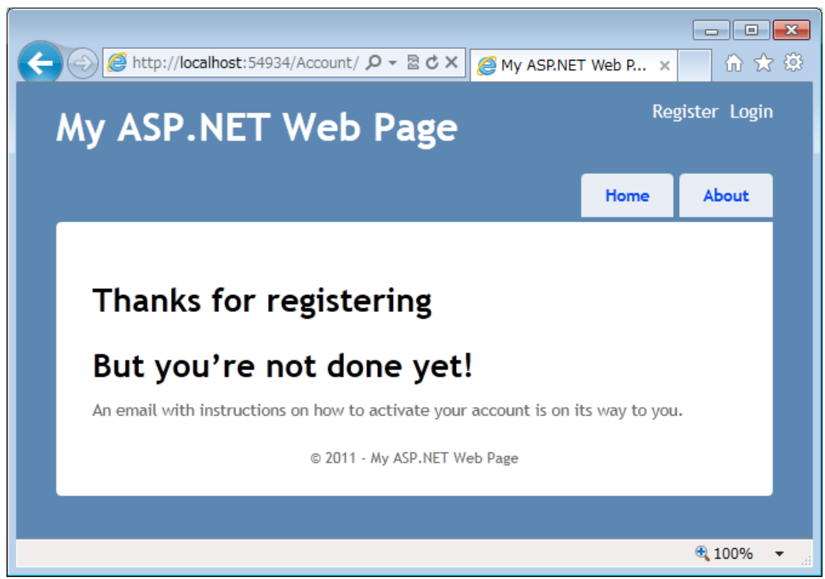

10. ブラウザーで、Default.cshtml ページを実行します。



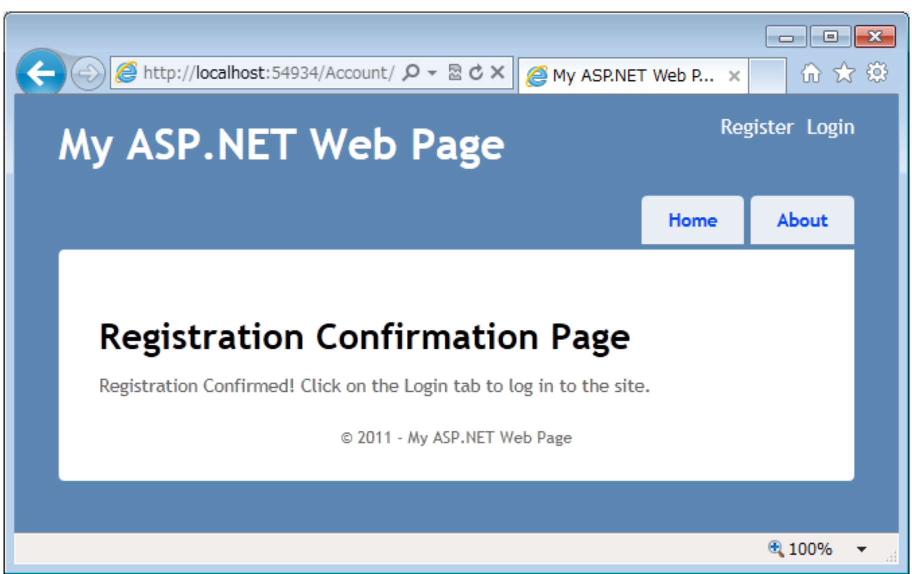

11. ページの右上の隅で、[Register]リンクをクリックします。

12. ユーザー名とパスワードを入力し、[Register]をクリックします。

「スターター サイト」テンプレートから Web サイトを作成した場合、StarterSite.sdf という名前のデータベースがサイトの App_Data フォルダー中に作成されます。登録すると、ユーザー情報がデータベースに追加されます。登録が完了すると、メッセージが使った電子メールアドレスに送信されます。

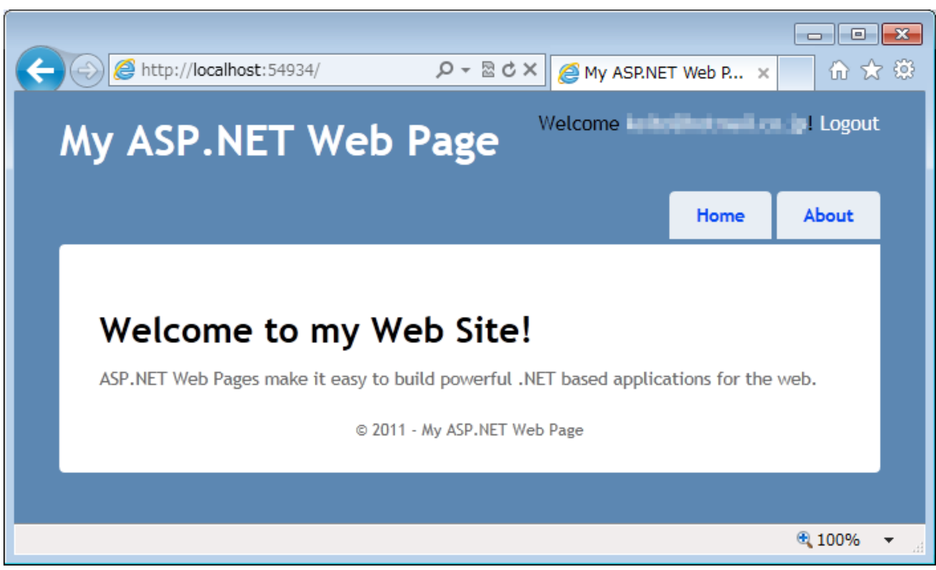



13. 電子メールプログラムを呼び出して、確認コードとサイトへのハイパーリンクを持つメッセージを見つけてください。
14. アカウントをアクティベートするハイパーリンクをクリックします。確認用のハイパーリンクは、登録用の確認ページを開きます。




15. [Login] リンクをクリックし、登録したアカウントを使ってログインします。

ログインすると、[Login] と [Register] リンクが [Logout] リンクに置き換えられます。




16. [About] リンクをクリックしてください。

About.cshtml ページが表示されます。ここで、ログインしたときの外観の変更は、ログイン状態の変化です。（「Welcome Joe」というメッセージと「Logout」リンク。）

**注意**　デフォルトでは、ASP.NET Web ページは、サーバーにクレデンシャルを平文（人間が読めるテキスト）で送信します。実運用するサイトでは、サーバーと機密情報を交換する場合には、安全な HTTP（https://、Secure Sockets Layer または SSL として知られる）を使うべきです。前述の例のとおり、WebMail.EnableSsl=true を設定することで機密情報を暗号化できます。SSL に関する詳細な情報は、「Securing Web Communications: Certificates, SSL, and https://」を参照してください。

## メンバー限定ページの作成

さしあたり、誰もが Web サイトのどのページも閲覧できるとします。しかし、ログインした人々（つまり、メンバー）だけが使えるページを持ちたいことがあります。ASP.NET は、ログイン メンバーだけがアクセスできるページを設定できます。通常、匿名ユーザーはメンバー限定ページをアクセスしようとすると、ログインページにリダイレクトします。

以下の手順では、About ページ（About.cshtml）へのアクセスを制限して、ログイン ユーザーだけがアクセスできるようにします。

1. About.cshtml ファイルを開きます。これは、_SiteLayout.cshtml ページをレイアウト ページとして使うコンテンツ ページです。（レイアウト ページについての詳細は、「第 3 章　一貫性のある外観の作成」を参照してください。）
2. About.cshtml ファイルのすべてのコードを以下のコードに置き換えます。このコードは、WebSecurity オブジェクトの IsAuthenticated プロパティを検査し、ユーザーがログインしている場合は true を返します。そうでなければ、コードは Response.Redirect を呼び出して、ユーザーを Account フォルダーにある Login.cshtml ページにリダイレクトします。以下は、About.cshtml ファイルの全体です。

```
@if (!WebSecurity.IsAuthenticated) {
    Response.Redirect("~/Account/Login");
}

@{
    Layout = "~/_SiteLayout.cshtml";
    Page.Title = "About My Site";
}

<p>
This web page was built using ASP.NET Web Pages. For more information,
visit the ASP.NET home page at <a href="http://www.asp.net" tar-
get="_blank">http://www.asp.net</a>
</p>
```

**注意**　この例の URL（~/Account/Login）は、.cshtml ファイル拡張子を含みません。ASP.NET は、.cshtml ページを指す URL のファイル拡張子を必要としません。詳細については、「第 18 章　サイト全体の動作をカスタマイズする」のルーティングのセクションを参照してください。

3. ブラウザーで、Default.cshtml を実行します。サイトにログインしている場合は、[Logout] リンクをクリックします。
4. [About] リンクをクリックします。ログインしていないので、Login.cshtml ページにリダイレクトされます。

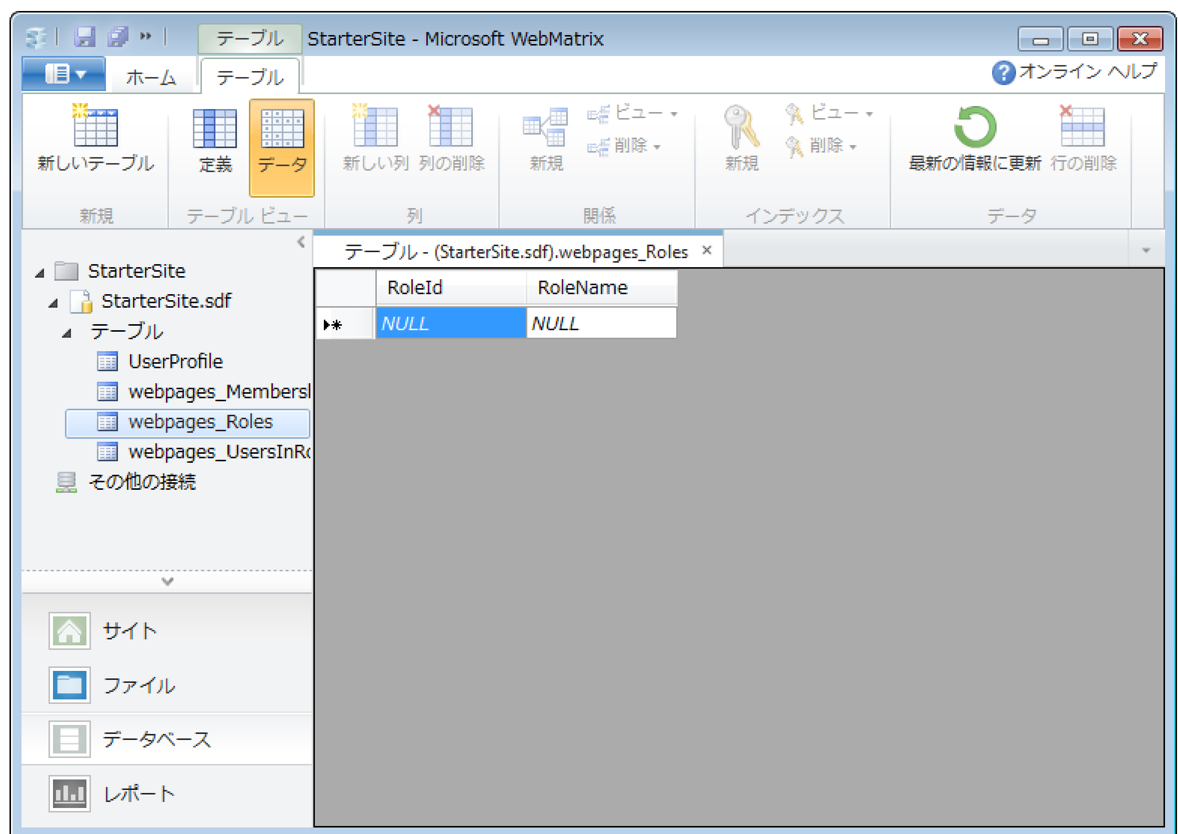
複数のページを安全にアクセスするため、それぞれのページにセキュリティ チェックを追加するか、_SiteLayout.cshtml のようなレイアウト ページがセキュリティ チェックを含むように作成できます。サイトの他の

ページも、今_SiteLayout.cshtml を参照している Default.cshtml と同じように、セキュリティ チェックのためにレイアウト ページを使えるようになります。

## ユーザー（ロール）のグループに対するセキュリティの作成

サイトが多くのメンバーを持つ場合、個々のユーザーごとにページを見せるかどうかの許諾を確認するのは効率的ではありません。その代りに、個々のメンバーが所属するグループやロールを作成できます。これで、ロールに基づく許諾を確認できます。このセクションでは、"admin"ロールを作成して、このロールに所属するユーザーだけがアクセスできるページを作成します。

1. WebMatrix で、「データベース」ワークスペースをクリックします。
2. 左ペインで、StarterSite.sdf ノードを開き、「テーブル」ノードを開き、webpages_Roles テーブルをダブルクリックします。




3. "admin"という名前のロールを追加します。RoleId 項目は自動的に入力されます。（これは、「第 5 章　データを扱う」で説明したプライマリー キーであり、アイデンティティ項目として設定されています。）
4. RoleId 項目の値をメモしておきます。（最初に定義するロールであれば、1 になります。）

テーブル - (StarterSite.sdf).webpages_Roles

| | RoleId | RoleName |
|---|---|---|
| | 1 | admin |
| ▸* | NULL | NULL |

5. Webpages_Roles テーブルを閉じます。
6. UserProfile テーブルを開きます。
7. このテーブルにあるユーザーから一人以上の UserId の値をメモして、このテーブルを閉じます。
8. webpages_UserInRoles テーブルを開き、UserID と RoleID の値をテーブルに入力します。たとえば、3 番目のユーザーを"admin"ロールに設定するためには、以下のように値を入力します。

テーブル - (StarterSite.sdf).UserProfile

| | UserId | RoleId |
|---|---|---|
| | 5 | 1 |
| ▶* | NULL | NULL |

9. webpages_UserInRoles テーブルを閉じます。

ロールを定義したので、このロールに属するユーザーがアクセスできるページを設定できます。

10. Web サイトのルート フォルダーで、新たに AdminError.cshtml という名前のページを作成し、既存のコンテンツを以下のコードで置き換えます。これは、ユーザーがページへのアクセスを許可されていないときにリダイレクトさせるページです。

```
@{
    Layout = "~/_SiteLayout.cshtml";
    PageData["Title"] = "Admin-only Error";
}
<p>You must log in as an admin to access that page.</p>
```

11. Web サイトのルート フォルダーで、新たに AdminOnly.cshtml という名前のページを作成し、既存のコードを以下のコードで置き換えます。

```
@{
    Layout = "~/_SiteLayout.cshtml";
    PageData["Title"] = "Administrators only";
}

@if ( Roles.IsUserInRole("admin")) {
    <span> Welcome <b>@WebSecurity.CurrentUserName</b>! </span>
}
else {
    Response.Redirect("~/AdminError");
}
```

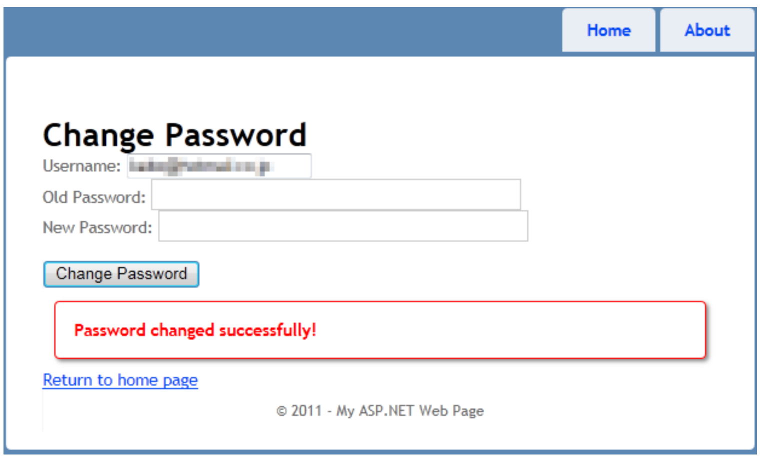
現在のユーザーが"admin"ロールのメンバーのときは、Roles.IsUserInRole メソッドは true を返します。

12. ブラウザーで Default.cshtml を実行します。ただしログインはしません。（ログイン中の場合は、ログアウトします。）

13. ブラウザーのアドレスバーで、URL の"Default"を"AdminOnly"に変更します。（言い換えると、AdminOnly.cshtml ファイルを呼び出します。） "admin"ロールのユーザーでログインしていないため、AdminError.cshtml ページにリダイレクトされます。
14. Default.cshtml に戻って、"admin"ロールに追加したユーザーとしてログインします。
15. AdminOnly.cshtml ページを閲覧します。今度は、ページが見られます。

## パスワード変更ページの作成

パスワード変更ページを作成することで、ユーザーはパスワードを変更できるようになります。以下の例は、この処理の基本部分を示しています。（「スターター サイト」テンプレートは、以下の手順で作成するページよりもさらに複雑なエラーチェックを行う ChangePassword.cshtml ファイルを含みます。）



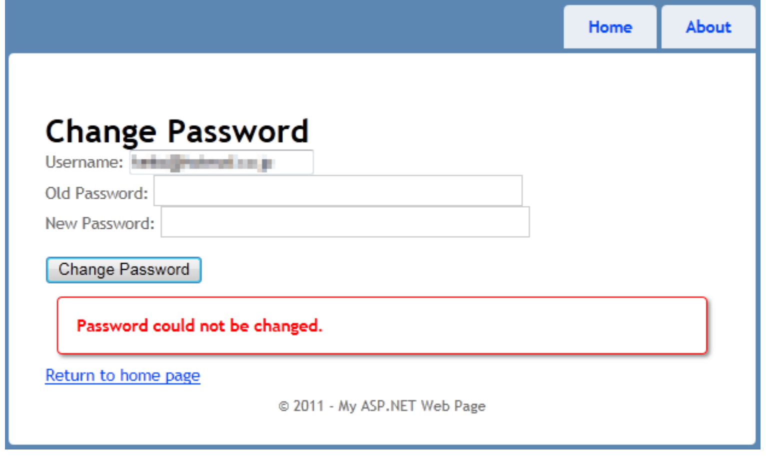

1. Web サイトの Account フォルダーで、ChangePassword2.cshtml という名前のページを作成します。
2. コンテンツを以下のコードで置き換えます。

```
@{
    Layout = "~/_SiteLayout.cshtml";
    PageData["Title"] = "Change Password";

    var message = "";
    if(IsPost) {

        string username = Request["username"];
        string newPassword = Request["newPassword"];
        string oldPassword = Request["oldPassword"];

        if(WebSecurity.ChangePassword(username, oldPassword, newPassword)) {
            message="Password changed successfully!";
        }
        else
        {
            message="Password could not be changed.";
        }
    }
}
```

```
<style>
    .message {font-weight:bold; color:red; margin:10px;}
</style>
<form method="post" action="">
    Username: <input type="text" name="username"
        value="@WebSecurity.CurrentUserName" />
    <br/>
    Old Password: <input type="password" name="oldPassword" value="" />
    <br/>
    New Password: <input type="password" name="newPassword" value="" />
    <br/><br/>
    <input type="submit" value="Change Password" />
    <div class="message">@message</div>
    <div><a href="Default.cshtml">Return to home page</a></div>
</form>
```

ページ本体は、ユーザーがユーザー名と新旧のパスワードを入力するテキストボックスを含みます。コードでは、WebSecurity ヘルパーの ChangePassword メソッドを呼び出して、ユーザーが入力した値を渡します。

3. ブラウザーでページを実行します。すでにログインしていれば、このページにユーザー名が表示されます。
4. 間違った旧パスワードを入力してみてください。正しいパスワードを入力しないと、WebSecurity.ChangePassword メソッドが失敗し、メッセージが表示されます。




5. 正しい値を入力して、もう一度パスワードの変更を試してください。

## ユーザーに新しいパスワードを生成させる

ユーザーがパスワードを忘れてしまったときは、新しいものを生成させます（これは、前述のパスワードの変更とは違います）。ユーザーに新しいパスワードを渡すために、WebSecurity ヘルパーの GeneratePasswordResetToken メソッドを使って、トークンを生成します。トークンとは、暗号化された安全な文字列で、ユーザーに送信され、パスワードをリセットするといった目的をユーザーに確認するために使われます。以下の手順は、この処理の一般的な方法を示しています。つまり、トークンを生成し、電子メールでユーザーに送信し、トークンを読み取りユーザーが

新しいパスワードを入力するページへリンクします。このリンクは、電子メール中でユーザーが次のように見えるものです。

```
http://localhost:36916/Account/PasswordReset2?PasswordResetToken=08HZGH0ALZ3CGz3
```

URL の末尾にあるランダムに見える文字が、トークンです。

（「スターター サイト」テンプレートは、以下に示す例よりも複雑なエラーチェックを行う ForgotPassword.cshtml ファイルを含みます。）

Forgot your password?

Enter your email address:

Get New Password

1. Web サイトの Account フォルダーで、新たに ForgetPassword2.cshtml という名前のページを作成します。
2. 既存のコンテンツを以下のコードで置き換えます。

```
@{
    Layout = "~/_SiteLayout.cshtml";
    PageData["Title"] = "Forgot your password?";

    var message = "";
    var username = "";

    if (WebMail.SmtpServer.IsEmpty() ){
        // The default SMTP configuration occurs in _start.cshtml
        message = "Please configure the SMTP server.";
    }

    if(IsPost) {
        username = Request["username"];
        var resetToken = WebSecurity.GeneratePasswordResetToken(username);

        var portPart = ":" + Request.Url.Port;
        var confirmationUrl = Request.Url.Scheme
            + "://"
            + Request.Url.Host
            + portPart
            + VirtualPathUtility.ToAbsolute(
              "~/Account/PasswordReset2?PasswordResetToken="
            + Server.UrlEncode(resetToken));

        WebMail.Send(
            to: username,
            subject: "Password Reset",
            body: @"Your reset token is:<br/><br/>"
                + resetToken
```

```
                + @"<br/><br/>Visit <a href="""
                + confirmationUrl
                + @""">"
                + confirmationUrl
                + @"</a> to activate the new password."
        );

        message = "An email has been sent to " + username
            + " with a password reset link.";
    }
}
<style>
    .message {font-weight:bold; color:red; margin:10px;}
</style>
<form method="post" action="">
    @if(!message.IsEmpty()) {
        <div class="error">@message</div>
    } else{
    <div>
    Enter your email address: <input type="text" name="username" /> <br/>
    <br/><br/>
    <input type="submit" value="Get New Password" />
    </div>
    }
</form>
```

ページの本体は、ユーザーに電子メールアドレスを尋ねるテキストボックスを含みます。ユーザーがフォームを送信すると、まず、ページのこの部分で電子メール メッセージを送信するため、SMTP メールの設定が行われているかを確認します。

ページの核心は、パスワード リセット用のトークンを生成する部分で、以下のようにユーザーが入力した電子メールアドレス（ユーザー名）を渡します。

```
string resetToken = WebSecurity.GeneratePasswordResetToken(username);
```

コードの残りの部分は、電子メール メッセージを送るためのものです。そのほとんどは、テンプレートから作られたサイトの一部である Register.cshtml ファイルから持ってきたものです。

実際に電子メールを送信するには、WebMail ヘルパーの Send メソッドを呼び出します。電子メールの本体は、テキストと HTML 要素の両方を含む文字列の変数が連結されることで作成されます。ユーザーが、電子メールを受け取ると、本体は以下のように見えます。

Your reset token is:

facxpyrXRQc7HV[illegible]V+tw==

Visit http://localhost:54934/Account/PasswordReset2?PasswordResetToken=facxpyrXRQc7HV[illegible]V%2btw%3d%3d to activate the new password.

3. Account フォルダーで、新たに PasswordReset2.cshtml という名前のページを作成し、コンテンツを以下のコードで置き換えます。

```
@{
    Layout = "~/_SiteLayout.cshtml";
    PageData["Title"] = "Password Reset";

    var message = "";
    var passwordResetToken = "";

    if(IsPost) {
        var newPassword = Request["newPassword"];
        var confirmPassword = Request["confirmPassword"];
        passwordResetToken = Request["passwordResetToken"];

        if( !newPassword.IsEmpty() &&
            newPassword == confirmPassword &&
            WebSecurity.ResetPassword(passwordResetToken, newPassword)) {
            message = "Password changed!";
        }
        else {
            message = "Password could not be reset.";
        }
    }
}
<style>
    .message {font-weight:bold; color:red; margin:10px;}
</style>
<div class="message">@message</div>
<form method="post" action="">
    Enter your new password: <input type="password" name="newPassword" /> <br/>
    Confirm new password: <input type="password" name="confirmPassword" /><br/>
    <br/>
    <input type="submit" value="Submit"/>
</form>
```

　このページは、ユーザーがパスワードをリセットするために電子メール中のリンクをクリックしたときに実行されるものです。本体は、ユーザーがパスワードを入力し、確認するためのテキストボックスを含みます。

　URL からパスワード トークンを取り出すには、Request["PasswordResetToken"]を読み込みます。この URL が以下のような形式だったことを思い出してください。

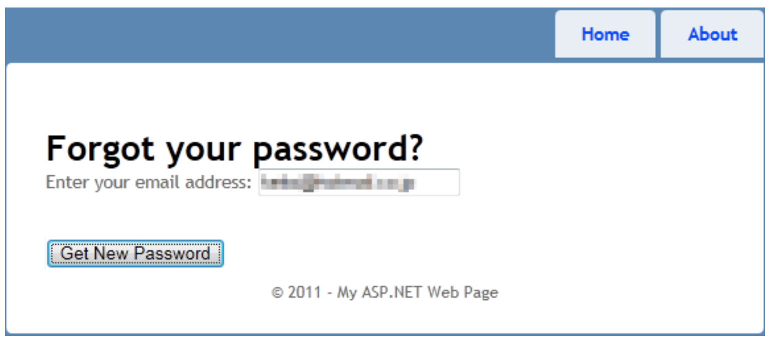
```
http://localhost:36916/Account/PasswordReset2?PasswordResetToken=08HZGH0ALZ3CGz3
```

　コードは、トークン（ここでは、08HZGH0ALZ3CGz3）を取得し、WebSecurity ヘルパーの ResetPassword メソッドを呼び出して、トークンと新しいパスワードを渡します。トークンが正しければ、ヘルパーは電子メールのトークンから判別したユーザーのパスワードを更新します。リセットが成功したら、ResetPassword メソッドは true を返します。

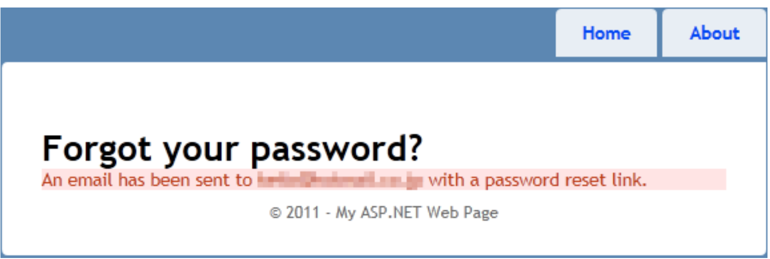
以下の例では、ResetPassword の呼び出しは、&&（論理積）演算子を使った正当性の検証と組み合わせられています。ここでは、リセットは以下の条件を満たす場合に成功します。

- newPassword テキストボックスが空でない（!演算子が偽になる）
- newPassword と confirmPassword の値が一致する
- ResetPassword メソッドが成功する

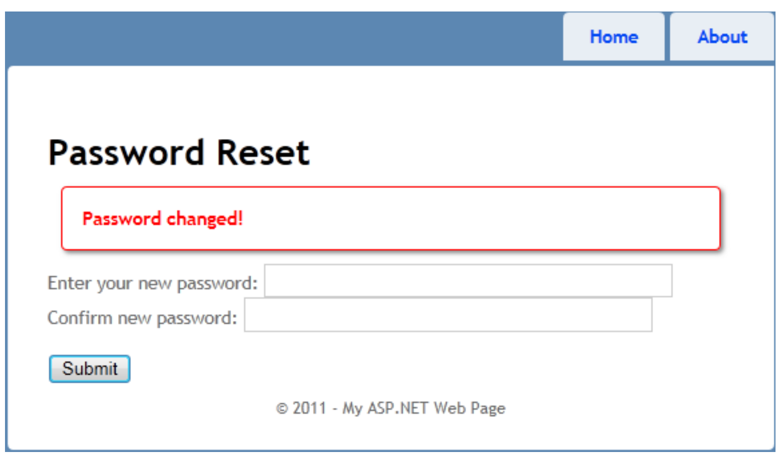
4. ブラウザーで ForgetPassword2.cshtml を実行します。




5. 電子メールアドレスを入力し、[Get New Password] をクリックします。ページは電子メールを送信します。（この処理に少し遅延が生じるかもしれません。）




6. 電子メールを確認し、"Password Reset."という件名のメッセージを探します。
7. この電子メールで、リンクをクリックします。このトークンは、PasswordReset2.cshtml ページに移動します。
8. 新しいパスワードを入力して、[Submit] をクリックします。



## Web サイトに参加する自動プログラムを防止する

ログイン ページは、Web サイトへの登録から自動プログラム（Web ロボットやボットと呼ばれます）を防止したいでしょう。（このようなボットをグループに参加させる一般的な目的は、製品販売の URL を投稿することにあります。） ユーザーが本当の人であり、コンピューター プログラムでないことを確認するために、CAPTCHA という入力検証を使うことができます。（CAPTCHA は、Completely Automated Public Turing test to tell Computers and Humans Apart の略称です。）

ASP.NET ページでは、ReCaptcha ヘルパーを使って、reCAPTCHA サービス（http://recaptcha.net）に基づく CAPTCHA テストをレンダリングできます。ReCaptcha ヘルパーは、2 つのゆがんだ単語をイメージとして表示し、ページを検証するために、ユーザーが正しく入力しなければなりません。ユーザーの応答は、ReCaptcha.NET サービスで検証されます。

1. Web サイトを ReCaptcha.net（http://recaptcha.net）に登録します。登録が完了すると、公開鍵と秘密鍵を取得できます。
2. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。
3. Account フォルダーで、Register.cshtml という名前のファイルを開きます。
4. captchaMessage 変数の//コメント文字を削除します。
5. PRIVATE_KEY 文字列を、取得した秘密鍵に置き換えます。
6. ReCaptcha.Validate 呼び出しを含む文から、//コメント文字を削除します。

   以下の例は、完全なコードです。（取得したキーを user-key-here と置き換えてください。）

   ```
   // Validate the user's response
   if (!ReCaptcha.Validate("user-key-here")) {
       captchaMessage = "Response was not correct";
       isValid = false;
   }
   ```

   Register.cshtml ページの末尾で、PUBLIC_KEY を取得した公開鍵に置き換えます。

7. ReCaptcha 呼び出しを含む行からコメント文字を削除します。以下の例は、完全なコードを示しています。（ただし、user-key-here は取得したキーに置き換えてください。）

```
@ReCaptcha.GetHtml("user-key-here", theme: "white")
```

8. ブラウザーで、Default.cshtml を実行します。サイトにログインして、[Logout] リンクをクリックします。
9. [Register] リンクをクリックして、CAPTCHA テストを使った登録を試します。

**注意**　お使いのコンピューターがプロキシ サーバーを使うドメイン上にある場合は、Web.Config ファイルの defaultproxy 要素を構成する必要があるかもしれません。以下の例は、Web.Config ファイルの defaultproxy プロパティを修正して、reCAPTCHA サービスを使えるようにする例を示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.net>
        <defaultProxy>
            <proxy
                usesystemdefault = "false"
                proxyaddress="http://myProxy.MyDomain.com"
                bypassonlocal="true"
                autoDetect="False"
            />
        </defaultProxy>
    </system.net>
</configuration>
```

## その他のリソース

- 第 18 章　サイト全体の動作をカスタマイズする
- Securing Web Communications: Certificates, SSL, and https://

# 第１７章　　デバッグの紹介

デバッグは、作成したコード ページにある間違いを見つけて修正する作業です。本章では、サイトのデバッグや解析するために使えるツールや技法について示しています。

ここでは次のことを学びます。

- ページの解析やデバッグを手助けする情報を表示する方法
- Web ページを解析するための Internet Explorer Developer Tools や Firebug のようなデバッグ ツールを使う方法

本章で紹介する ASP.NET 機能と WebMatrix（および他の）ツールは、以下の通りです。

- ServerInfo ヘルパー
- ObjectInfo ヘルパー
- Internet Explorer Developer Tools と Firebug デバッグ ツール

作成したコードにおける間違いや問題の解決における重要な面は、まず、それらを回避することです。これは、エラーを引き起こすコードを try/catch ブロックに置くことで実現できます。詳細については、「第 2 章　Razor 構文を使った ASP.NET Web プログラミングの紹介」のエラー処理に関するセクションを参照してください。ASP.NET Razor ページをデバッグするための Visual Studio の統合デバッガの使い方については、「Program ASP.NET Web Pages in Visual Studio」を参照してください。

## サーバー情報を表示するために ServerInfo ヘルパーを使う

ServerInfo ヘルパーは、ページをホストする Web サーバーの環境についての情報の概要を提供する診断ツールです。また、ブラウザーがページを要求するときに送出された HTTP リクエスト情報も表示します。ServerInfo ヘルパーは、現在のユーザー アイデンティティ、リクエスト元のブラウザーの種類などを表示します。この種の情報は、一般的な問題を解決する手助けとなります。

1. 新たに ServerInfo.cshtml という名前の Web ページを作成します。
2. ページの末尾で、閉じる </body> タグの直前に、@ServerInfo.GetHtml() を追加します。

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        @ServerInfo.GetHtml()
    </body>
</html>
```

ServerInfo コードは、ページ中のどこにでも追加できます。ただし、最後に追加することで、他のページ コンテンツと分けて出力できるため、読みやすくなります。

**注意**　Web ページを実運用サーバーに移動する前に、Web ページから診断用のコードを削除してください。これは、ServerInfo ヘルパーだけでなく、ページにコードを追加するような、本章で説明するその他の診断手法についても同じです。Web サイトのビジターには、サーバー名、ユーザー名、サーバーのパスといった詳細な情報を提供すべきではありません。この種の情報は、悪用するような人に便利なものだからです。

3. このページを保存して、ブラウザーで実行します。（「ファイル」ワークスペースで、このページが選択されていることを確認してください。）

ServerInfo ヘルパーは、情報についての 4 つの表をページ中に表示します。

- **サーバーの設定**。このセクションは、ホスティングしている Web サーバーに関するコンピューター名や実行中の ASP.NET のバージョン、ドメイン名、サーバー時間などの情報を表示します。
- **ASP.NET サーバー変数**。このセクションは、Web ページ リクエストの一部となる HTTP プロトコルの詳細（HTTP 変数と呼ばれます）と値の詳細を表示します。
- **HTTP ランタイム情報**。このセクションは、Web ページを実行中の Microsoft .NET Framework のバージョン、パス、キャッシュの詳細などの情報を表示します。（「第 2 章　Razor 構文を使った ASP.NET Web プログラミングの紹介」で学んだとおり、Razor 構文を使った ASP.NET Web ページは、Microsoft の ASP.NET Web サーバー技術に基づいて構築されます。これは、.NET Framework と呼ばれるソフトウェア開発ライブラリ上で構築されています。）
- **環境変数**。このセクションは、Web サーバーにおけるすべてのローカル環境変数と値の一覧を表示します。

サーバーとリクエストの情報に関するすべてを記述するのは本章の範囲を超えますが、ServerInfo ヘルパーが、多くの診断情報を提供していることはわかります。ServerInfo が返す値の情報については、Microsoft TechNet の Web サイトにある「認識される環境変数」と MSDN の Web サイトにある「IIS Server Variables」を参照してください。

## ページの値を表示するために出力式を埋め込む

コード中で何が発生したかを確認するための方法として、ページ中に出力式を埋め込むことがあります。ご存じのとおり、@myVariable や@(subTotal * 12)のようなコードをページに追加することで、変数の値を直接出力できます。デバッグのために、こうした出力式をコード中の主要な部分に配置できます。デバッグが終了したら、この式を削除するかコメントアウトします。以下の手順は、ページをデバッグするために埋め込み式を使う典型的な方法を示しています。

1. 新たに OutputExpression.cshtml という名前の WebMatrix ページを作成します。
2. ページのコンテンツを以下で置き換えます。

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>

    @{
        var weekday = DateTime.Now.DayOfWeek;
        // As a test, add 1 day to the current weekday.
        if(weekday.ToString() != "Saturday") {
            // If weekday is not Saturday, simply add one day.
            weekday = weekday + 1;
        }
        else {
            // If weekday is Saturday, reset the day to 0, or Sunday.
            weekday = 0;
        }
        // Convert weekday to a string value for the switch statement.
        var weekdayText = weekday.ToString();

        var greeting = "";

        switch(weekdayText)
        {
            case "Monday":
                greeting = "Ok, it's a marvelous Monday.";
                break;
            case "Tuesday":
                greeting = "It's a tremendous Tuesday.";
                break;
            case "Wednesday":
                greeting = "Wild Wednesday is here!";
                break;
            case "Thursday":
                greeting = "All right, it's thrifty Thursday.";
                break;
            case "Friday":
                greeting = "It's finally Friday!";
                break;
            case "Saturday":
                greeting = "Another slow Saturday is here.";
```

```
                break;
            case "Sunday":
                greeting = "The best day of all: serene Sunday.";
                break;
            default:
                break;
        }
    }

    <h2>@greeting</h2>

    </body>
</html>
```

この例は、weekday 変数の値を調べるために switch 文を使い、曜日ごとに異なる出力メッセージを表示しています。例の中で、最初のコード ブロックの if ブロックは現在の曜日の値に 1 日追加することで、わざと曜日を変更しています。これは、間違いを示すためのものです。

3. ページを保存して、ブラウザーで実行します。

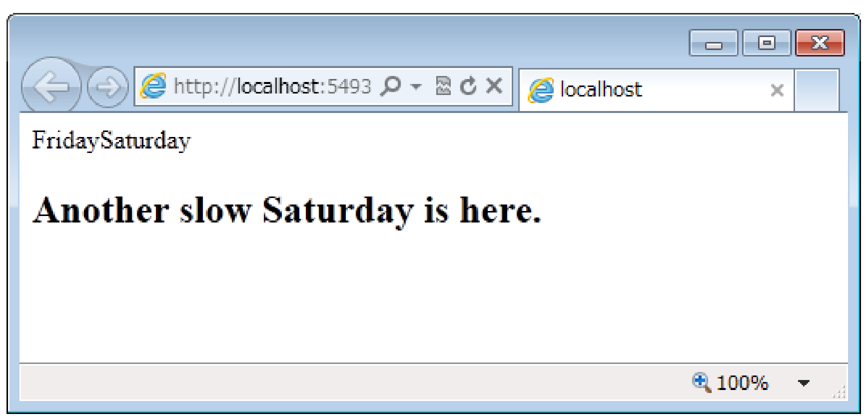
このページは、間違った曜日のメッセージを表示します。実際の曜日が何であれ、1 日ずれたメッセージが表示されます。ここではメッセージが間違っている理由が分かっていますが（このコードは、わざと間違った日付の値を設定しているので）、現実にはコードのどこに問題があるのかを把握するのはたいてい困難です。デバッグのためには、weekday のような主要なオブジェクトや変数の値に何が起きているのかを知る必要があります。

4. コード中のコメントで示された 2 つの場所に@weekday を挿入して、出力式を追加します。この出力式は、コード実行中の位置における変数の値を表示します。

```
var weekday = DateTime.Now.DayOfWeek;
// Display the initial value of weekday.
@weekday

// As a test, add 1 day to the current weekday.
if(weekday.ToString() != "Saturday") {
    // If weekday is not Saturday, simply add one day.
    weekday = weekday + 1;
}
else {
    // If weekday is Saturday, reset the day to 0, or Sunday.
    weekday = 0;
}

// Display the updated test value of weekday.
@weekday

// Convert weekday to a string value for the switch statement.
var weekdayText = weekday.ToString();
```

5. ページを保存して、ブラウザーで実行します。

ページは、まず実際の曜日を表示し、1 日追加されたことで曜日が更新され、switch 文で結果メッセージが表示されます。2 つの変数式（@weekday）による出力は、出力に HTML の<p>タグを追加しなかったため、間にスペースはありません。この式は、たんにテスト用のものです。



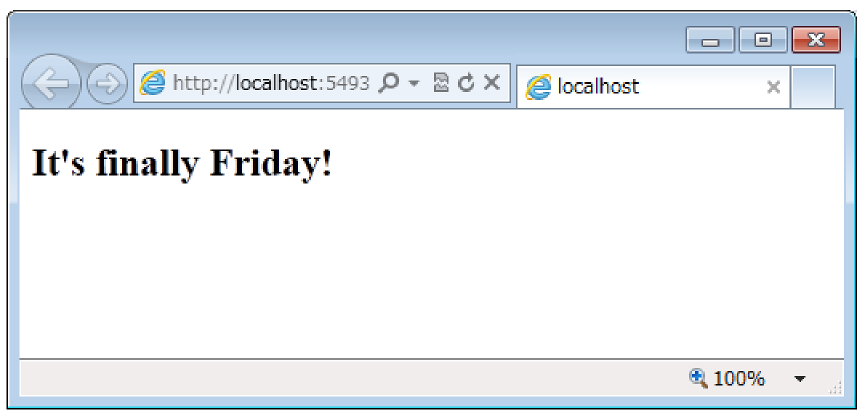

ここで、どこで間違いが発生したかがわかります。最初にコード中の weekday 変数を表示するときは、正しい日を表示します。if ブロックの後で 2 回目に表示するときは、1 日ずれた日が表示されます。つまり、最初と 2 回目の weekday 変数を表示している間に問題が起きたことがわかります。これが本当のバグであれば、このようなアプローチで、問題を発生させているコードの位置を絞り込むことができます。

6. 追加した 2 つの出力式を削除し、曜日を変更したコードを削除することで、コードを修正してください。残りは、以下の例のような完全なコードのブロックです。

```
@{
    var weekday = DateTime.Now.DayOfWeek;
    var weekdayText = weekday.ToString();

    var greeting = "";

    switch(weekdayText)
    {
        case "Monday":
            greeting = "Ok, it's a marvelous Monday.";
            break;
        case "Tuesday":
            greeting = "It's a tremendous Tuesday.";
            break;
        case "Wednesday":
            greeting = "Wild Wednesday is here!";
            break;
        case "Thursday":
            greeting = "All right, it's thrifty Thursday.";
            break;
        case "Friday":
            greeting = "It's finally Friday!";
            break;
        case "Saturday":
            greeting = "Another slow Saturday is here.";
```

```
            break;
        case "Sunday":
            greeting = "The best day of all: serene Sunday.";
            break;
        default:
            break;
    }
}
```

7. ブラウザーでページを実行します。今回は、実際の曜日に対する正しいメッセージが表示されます。




## オブジェクトの値を表示するために ObjectInfo ヘルパーを使う

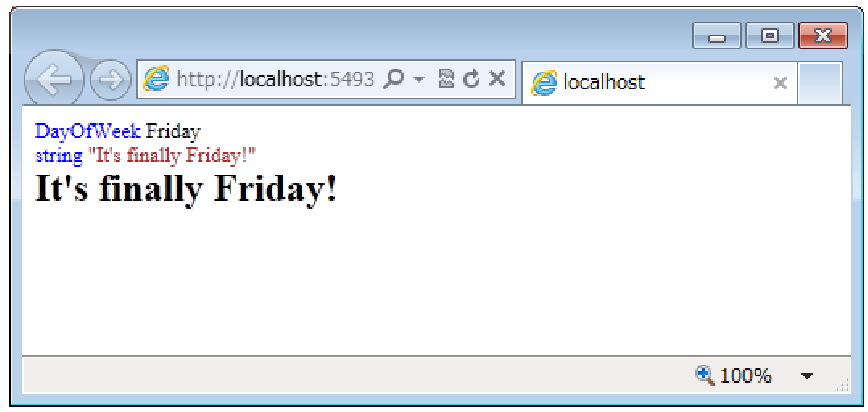
ObjectInfo ヘルパーは、渡される個々のオブジェクトの型と値を表示します。これにより、コード中で変数やオブジェクトの値を表示でき（前述の例における出力式に似ています）、さらにオブジェクトのデータ型情報も表示できます。

1. 先に作成した OutputExpression.cshtml という名前のファイルを開きます。
2. ページ中のすべてのコードを、以下のコードブロックで置き換えます。

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>
    @{
        var weekday = DateTime.Now.DayOfWeek;
        @ObjectInfo.Print(weekday)
        var weekdayText = weekday.ToString();

        var greeting = "";

        switch(weekdayText)
        {
            case "Monday":
                greeting = "Ok, it's a marvelous Monday.";
                break;
```

```
            case "Tuesday":
                greeting = "It's a tremendous Tuesday.";
                break;
            case "Wednesday":
                greeting = "Wild Wednesday is here!";
                break;
            case "Thursday":
                greeting = "All right, it's thrifty Thursday.";
                break;
            case "Friday":
                greeting = "It's finally Friday!";
                break;
            case "Saturday":
                greeting = "Another slow Saturday is here.";
                break;
            case "Sunday":
                greeting = "The best day of all: serene Sunday.";
                break;
            default:
                break;
        }
    }
    @ObjectInfo.Print(greeting)
    <h2>@greeting</h2>

    </body>
</html>
```

3. ページを保存して、ブラウザーで実行します。




この例では、ObjectInfo ヘルパーは、2 つの項目を表示します。

- **型**。最初の変数は、型は DayOfWeek です。2 番目の変数は、型は string です。
- **値**。ここでは、ページ中に挨拶用の変数の値として表示していますが、ObjectInfo に渡した変数の値がふたたび表示されます。

複雑なオブジェクトにおいても、ObjectInfo ヘルパーは詳細な情報を表示します。基本的に、オブジェクトのすべてのプロパティの型と値を表示します。

## デバッグ ツールを使う

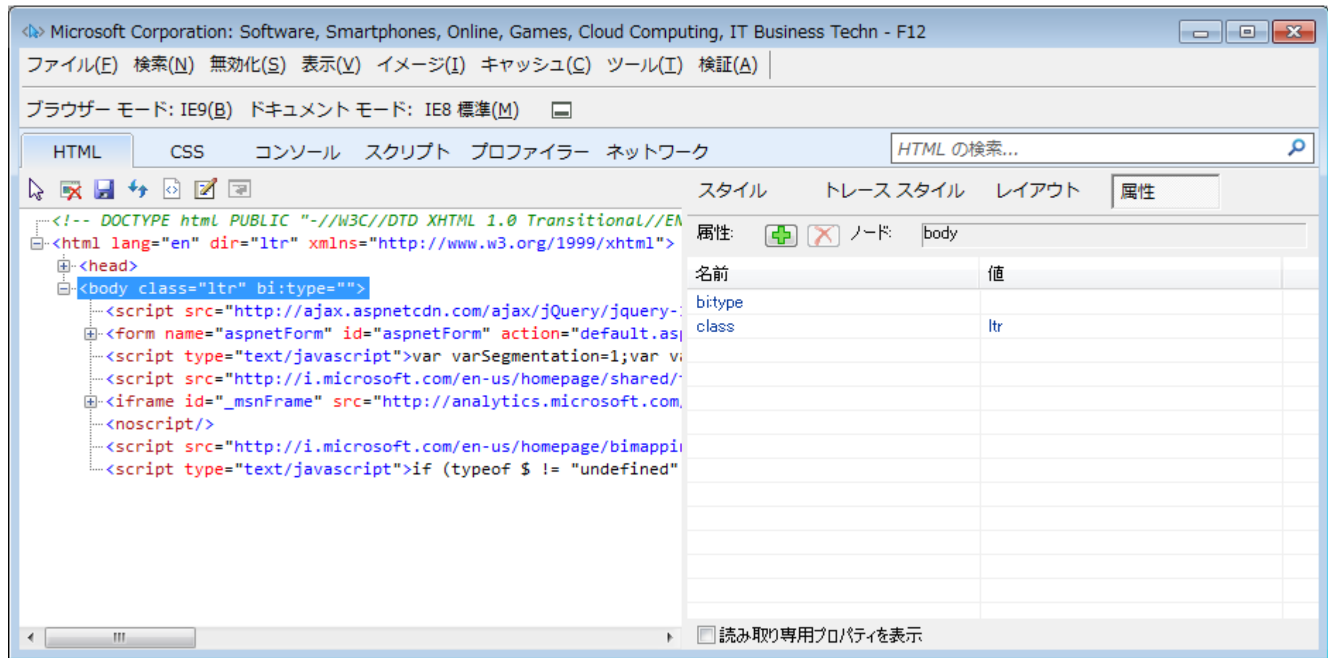
デバッグのために、ページ中に情報を表示するだけでなく、ページがどのように実行されているかについての情報を得るためのツールが使えます。このセクションでは、Web ページのためによく使われている診断ツールの使い方について示し、作成するサイトをデバッグするために WebMatrix 上でツールを使う方法についても示します。

### Internet Explorer Developer Tools

Internet Explorer Developer Tools は、Internet Explorer 8 に組み込まれた Web ツールのパッケージです。（以前の Internet Explorer をお使いの場合は、Microsoft のダウンロード センターにある「Internet Explorer Developer Toolbar」ページからツールをインストールできます。） このツールは、ASP.NET コードをデバッグすることに特化したものではありませんが、ASP.NET が動的に生成したマークアップやスクリプトを含め、HTML、CSS、スクリプトをデバッグするのに大変便利なものです。

以下の手順は、Internet Explorer Developer Tools をどのように使うかを示しています。ここでは、Internet Explorer 8 を使うことを想定しています。

1. Internet Explorer で、www.microsoft.com のような公開されている Web ページを閲覧します。
2. 「ツール」メニューで「F12 開発者ツール」をクリックします。
3. 「HTML」タブをクリックして、<html> 要素を開き、さらに <body> 要素を開きます。ウィンドウは、<body> 要素のすべてのタグを表示します。
4. 右ペインで、「属性」タブをクリックし、<body> タグの属性を確認します。



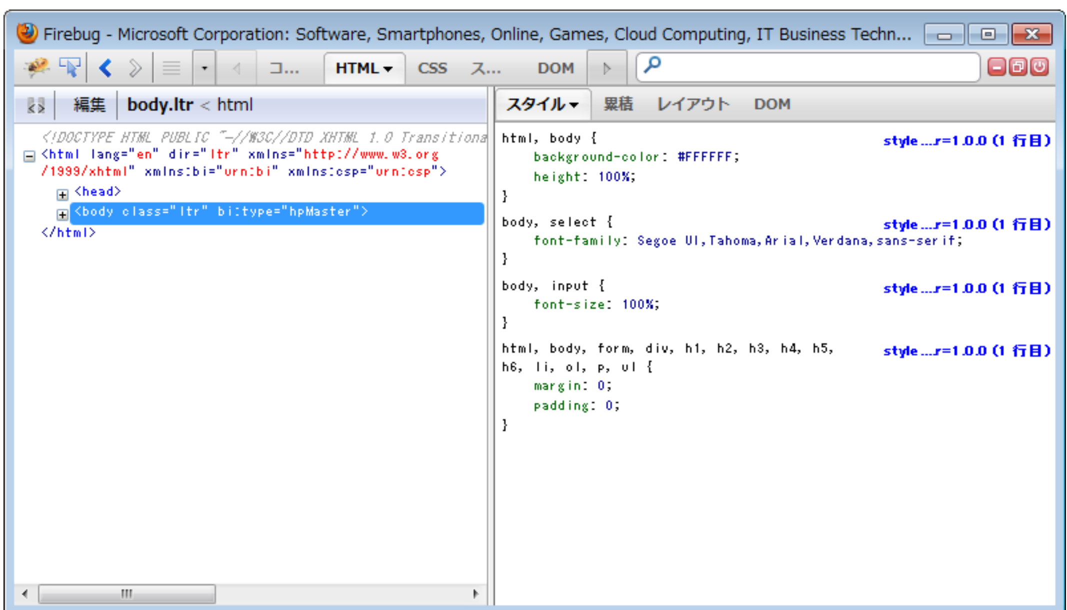

5. 右ペインで、「スタイル」をクリックし、ページの本体部分に割り当てられている CSS スタイルを確認します。Internet Explorer Developer Tools についての詳細は、MSDN の Web サイトにある「Internet Explorer 開発者ツールを理解する」を参照してください。

## Firebug

Firebug は Mozilla Firefox のアドオンで、HTML マークアップや CSS を調べたり、クライアント スクリプトをデバッグしたり、クッキーや他のページの情報を表示できます。Firebug は、「Firebug の Web サイト」（http://getfirebug.com/）からインストールできます。Internet Explorer デバッグツールと同じく、ASP.NET コードのデバッグに特化したツールではありませんが、ASP.NET が動的に生成するものを含め、HTML や他のページ要素をテストするために大変便利なものです。

以下の手順は、Firebug をインストール後にできることを示しています。

1. Firefox で、www.microsoft.com を閲覧します。
2. [ツール] メニューで [Firebug] をクリックし、さらに [Firebug を新しいウィンドウで開く] をクリックします。
3. Firebug のメイン ウィンドウで、左ペインの「HTML」タブをクリックし、<html> ノードを展開します。
4. <body> タグを選び、右ペインの「スタイル」タブをクリックします。Firebug は、Microsoft のサイトのスタイル情報を表示します。




Firebug は、HTML や CSS スタイルの編集や検証についての多くのオプションがあり、スクリプトのデバッグや改良に役立ちます。「Net」タブでは、サーバーと Web ページ間のネットワーク トラフィックを解析できます。たとえば、ページをプロファイルして、ブラウザーにすべてのコンテンツがダウンロードするまでの時間を確認できます。Firebug についての詳細は、「Firebug の公式サイト」や「Firebug Documentation Wiki」を参照してください。

## その他のリソース

MSDN オンライン ドキュメント

- IIS Server Variables

Visual Studio のデバッグ

- Program ASP.NET Web Pages in Visual Studio

TechNet オンライン ドキュメント

- 認識される環境変数

Internet Explorer Developer Tools

- Internet Explorer 開発者ツールを理解する
- Internet Explorer Developer Toolbar のダウンロード（Internet Explorer 8 よりも前のバージョン）
- 開発者ツールでの HTML および CSS のデバッグ
- 開発者ツールを使用したスクリプトのデバッグ

Web 開発者のための Firebug Add-on

- Firebug の公式サイト
- Firebug Documentation Wiki

# 第１８章　サイト全体の動作をカスタマイズする

本章では、個々のページではなく、Web サイト全体またはフォルダー全体をどのように設定するかについて説明します。

ここでは次のことを学びます。

- サイトのすべてのページに対する値（グローバル値またはヘルパーの設定）を設定するコードを実行する方法
- フォルダーのすべてのページに対する値を設定するコードを実行する方法
- ページが読み込まれる前後にコードを実行する方法
- 中心となるエラー ページにエラーを送る方法
- フォルダーのすべてのページに認証を追加する方法
- ASP.NET が、どのように読みやすく検索されやすい URL を使うようルーティングするか

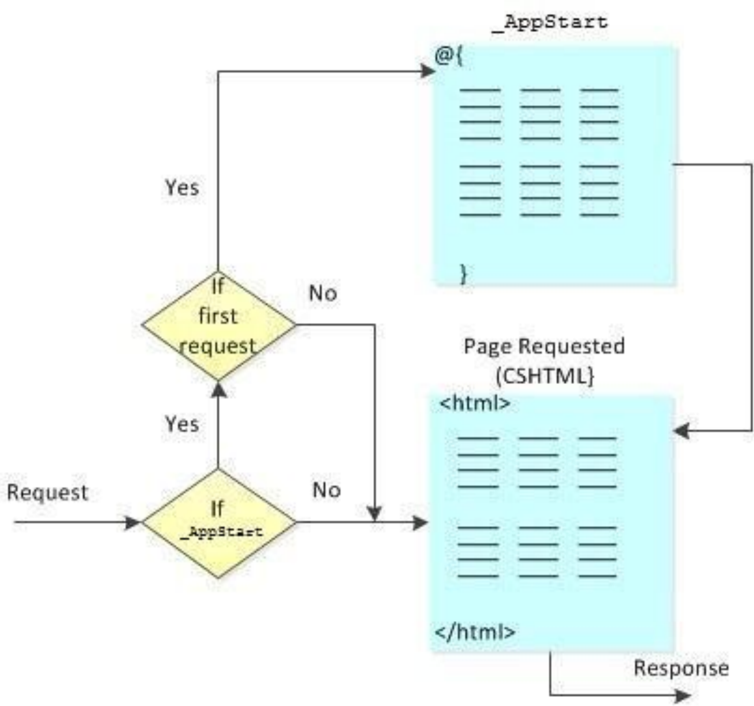
## Web サイトのスタートアップ コードを追加する

記述するコードや設定のほとんどは、個々のページにあります。たとえば、電子メール メッセージを送信するページは、電子メールの送信（つまり、SMTP サーバー）の設定を初期化し、電子メール メッセージを送信するために必要なすべてのコードを含んでいます。

一方、状況によっては、サイト上でページを実行する前にコードを実行したいことがあるかもしれません。これは、サイトのどこからでも使えるようにしたい設定値（グローバル値と呼ばれます）のために役立ちます。いくつかのヘルパーは、たとえば電子メールの設定やアカウントキーのような値を提供しなければならず、こうした設定をグローバル値として保持しておくとよいでしょう。

このために、サイトのルートに_AppStart.cshtml という名前のページを作成します。このページがあれば、サイト中のどのページがリクエストされたときにも、まずこのページが実行されます。このため、グローバル値を設定するためのコードを実行する場所に適しています。（_AppStart.cshtml は先頭にアンダースコア文字が付いているため、直接このページがリクエストされた場合でも、ASP.NET はブラウザーに送信しません。）

以下の図は、_AppStart.cshtml ページがどのように動作するかを示しています。ページのリクエストがあり、それがサイトのすべてのページの最初のリクエストであれば、まず ASP.NET は_AppStart.cshtml ページが存在するかを確認します。もしあれば、_AppStart.cshtml ページのコードが実行され、リクエストされたページが実行されます。




## Web サイトのグローバル値を設定する

1. WebMatrix Web サイトのルート フォルダーで、_AppStart.cshtml という名前のファイルを作成します。このファイルは、サイトのルートになければなりません。
2. デフォルトのマークアップを以下のコードで置き換えます。

```
@{
    AppState["customAppName"] = "Application Name";
}
```

このコードは、サイトのすべてのページから自動的に利用できる AppState ディクショナリに値を保持します。

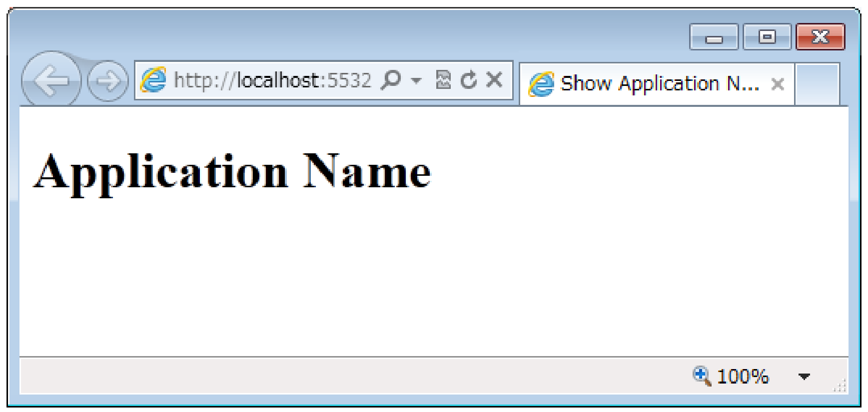
**注意** _AppStart.cshtml ファイルにコードを書くときには注意が必要です。_AppStart.cshtml ファイル中のコードでエラーが発生すると、Web サイトは起動しません。

3. ルート フォルダーで、新たに AppName.cshtml という名前のページを作成します。
4. デフォルトのマークアップを、以下で置き換えます。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Show Application Name</title>
    </head>
    <body>
        <h1>@AppState["customAppName"]</h1>
    </body>
</html>
```

このコードは、_AppStart.cshtml ページで設定した AppState オブジェクトから値を取り出しています。

5. ブラウザーで AppName.cshtml ページを実行します（実行する前に、「ファイル」ワークスペースでこのページが選択されていることを確認してください）。ページはグローバル値を表示します。




## ヘルパーの値を設定する

_AppStart.cshtml ファイルのよい使い方は、サイト中で使われ、初期化しなければならないヘルパーのための値を設定することです。よい例が ReCaptcha ヘルパーです。これは、reCAPTCHA アカウントのための公開鍵と秘密鍵を指定する必要があります。ReCaptcha ヘルパーを使いたい個々のページで、こうしたキーを設定する代わりに、_AppStart.cshtml で一度だけ設定しておけば、サイトのすべてのページで設定されていることになります。_AppStart.cshtml で設定されるその他の値として、「第 16 章　セキュリティとメンバーシップの追加」の SMTP サーバーで使われる電子メール送信の設定があります。

以下の手順は、ReCaptcha のキーをグローバルに設定する方法を示しています。（ReCaptcha ヘルパーについての詳細は、「第 16 章　セキュリティとメンバーシップの追加」を参照してください。

1. 「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」で説明したように、Web サイトに ASP.NET Web Helpers Library を追加します（追加していない場合）。
2. Web サイトを ReCaptcha.net（http://recaptcha.net）に登録します（登録していない場合）。登録が完了すると、公開鍵と秘密鍵を取得できます。
3. Web サイトのルート フォルダーに、新たに_AppStart.cshtml というファイルを作成します（作成していない場合）。
4. _AppStart.cshtml の既存のコードを以下のコードで置き換えます。

```
@{
    // Add the PublicKey and PrivateKey strings with your public
    // and private keys. Obtain your PublicKey and PrivateKey
    // at the ReCaptcha.Net (http://recaptcha.net) website.
    ReCaptcha.PublicKey = "";
    ReCaptcha.PrivateKey = "";
}
```

5. 入手した公開鍵と秘密鍵を使って、PublicKey と PrivateKey プロパティを設定します。
6. _AppStart.cshtml ファイルを保存して、閉じます。
7. Web サイトのルート フォルダーで、新たに Recaptcha.cshtml という名前のページを作成します。
8. デフォルトのマークアップを以下のコードで置き換えます。

```
@{
    var showRecaptcha = true;
    if (IsPost) {
        if (ReCaptcha.Validate()) {
            @:Your response passed!
            showRecaptcha = false;
        }
        else{
            @:Your response didn't pass!
        }
    }
}
<!DOCTYPE html>
<html>
    <head>
        <title>Testing Global Recaptcha Keys</title>
    </head>
    <body>
    <form action="" method="post">
    @if(showRecaptcha == true){
        if(ReCaptcha.PrivateKey != ""){
            <p>@ReCaptcha.GetHtml()</p>
            <input type="submit" value="Submit" />
        }
        else {
            <p>You can get your public and private keys at
            the ReCaptcha.Net website (http://recaptcha.net).
            Then add the keys to the _AppStart.cshtml file.</p>
        }
    }
    </form>
    </body>
</html>
```

9. ブラウザーで Recaptcha.cshtml ページを実行します。PrivateKey の値が正しければ、ページは reCAPTCHA コントロールとボタンを表示します。キーが_AppStart.cshtml でグローバルに設定されていなければ、ページはエラーを表示します。

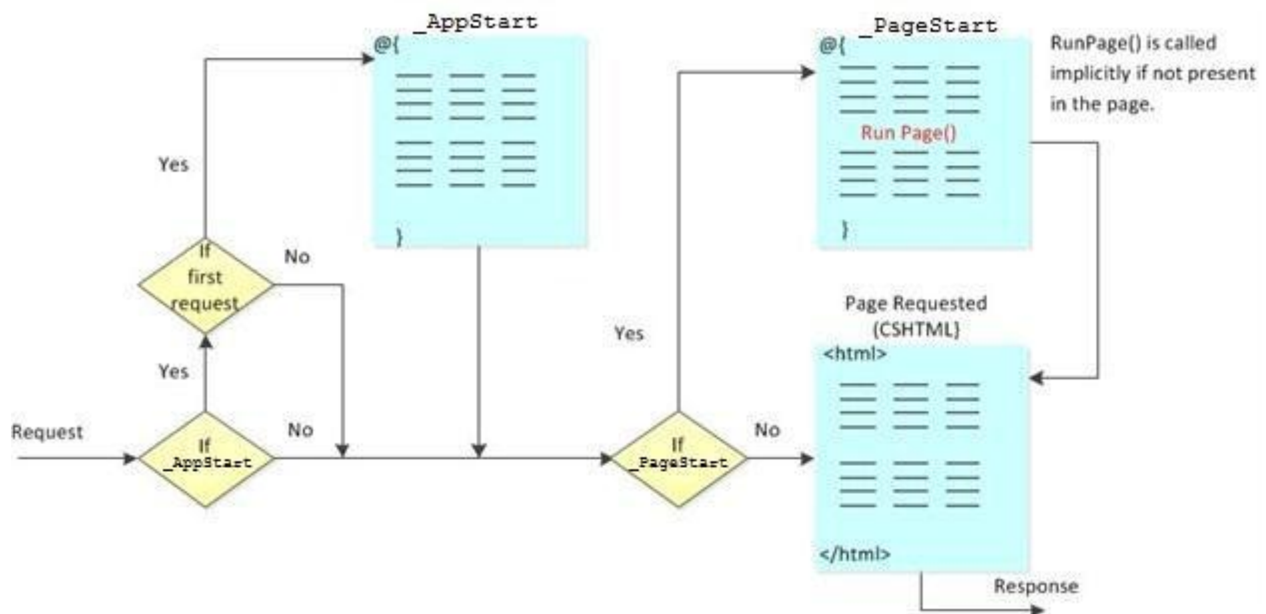
10. 判別用の単語を入力します。reCAPTCHA の判定にパスすれば、そのことを示すメッセージが表示され、そうでなければエラーメッセージと reCAPTCHA コントロールが再び表示されます。

## フォルダーのファイルの前後にコードを実行する

サイト中のページを実行する前に_AppStart.cshtml を使えるのと同じように、特定のフォルダーにあるページを実行する前に（および後に）実行されるコードを書くこともできます。これは、フォルダー中のすべてのページに対して同じレイアウト ページを設定したり、フォルダー中のページを実行する前にユーザーがログインしているかを確認したりするような場合に役立ちます。

特定のフォルダーにあるすべてのページのために、_PageStart.cshtml という名前のファイルにコードを作成できます。以下の図は、_PageStart.cshtml ページがどのように働くかを示しています。ページのリクエストがあると、ASP.NET はまず_AppStart.cshtml ページを確認して、実行します。次に、_PageStart.cshtml ページがあるかどうかを確認して、もしあれば、これを実行します。その後で、リクエストされたページを実行します。

_PageStart.cshtml ページの中で、リクエストされた実行ページに、RunPage メソッドを含むことで、どこで処理するかを指定できます。これにより、ページを実行する前と後にコードを実行することもできます。




ASP.NET は、_PageStart.cshtml ファイルの階層を作成します。_PageStart.cshtml ファイルは、サイトのルート フォルダーやサブ フォルダーに配置します。ページがリクエストされると、最上位（サイトのルートに近い）_PageStart.cshtml ファイルが実行され、続いて次のサブ フォルダーの_PageStart.cshtml ファイルが実行され、リクエストされたページを含むフォルダーにたどり着くまでサブ フォルダー構造を下っていきます。すべての_PageStart.cshtml ファイルが実行された後で、リクエストされたページが実行されます。

たとえば、以下では_PageStart.cshtml ファイルと default.cshtml ファイルを組み合わせています。

```
@* ~/_PageStart.cshtml *@
@{
    PageData["Color1"] = "Red";
```

```
    PageData["Color2"] = "Blue";
}

@* ~/myfolder/_PageStart.cshtml *@
@{
    PageData["Color2"] = "Yellow";
    PageData["Color3"] = "Green";
}

@* ~/myfolder/default.cshtml *@
@PageData["Color1"]
<br/>
@PageData["Color2"]
<br/>
@PageData["Color3"]
```

default.cshtml を実行すると、以下のようになります。

```
Red

Yellow

Green
```

## フォルダー中のすべてのページのために初期化コードを実行する

_PageStart.cshtml ファイルのよい使い方として、単一フォルダーのすべてのファイルのために同じレイアウト ページを初期化することがあります。

1. ルート フォルダーで、新たに InitPages という名前のフォルダーを作成します。
2. Web サイトの InitPages フォルダーで、_PageStart.cshtml という名前のファイルを作成し、デフォルトのマークアップを以下のコードで置き換えます。

```
@{
    // Sets the layout page for all pages in the folder.
    Layout = "~/Shared/_Layout1.cshtml";
    // Sets a variable available to all pages in the folder.
    PageData["MyBackground"] = "Yellow";
}
```

3. Web サイトのルートで、Shared という名前のフォルダーを作成します。
4. Shared フォルダーで、_Layout1.cshtml という名前のファイルを作成し、デフォルトのマークアップを以下のコードで置き換えます。

```
@{
    var backgroundColor = PageData["MyBackground"];
}
<!DOCTYPE html>
```

```
<html>
<head>
    <title>Page Title</title>
    <link type="text/css" href="/Styles/Site.css" rel="stylesheet" />
</head>
<body>
    <div id="header">
        Using the _PageStart.cshtml file
    </div>
    <div id="main" style="background-color:@backgroundColor">
        @RenderBody()
    </div>
    <div id="footer">
        &copy; 2010 Contoso. All rights reserved
    </div>
</body>
</html>
```

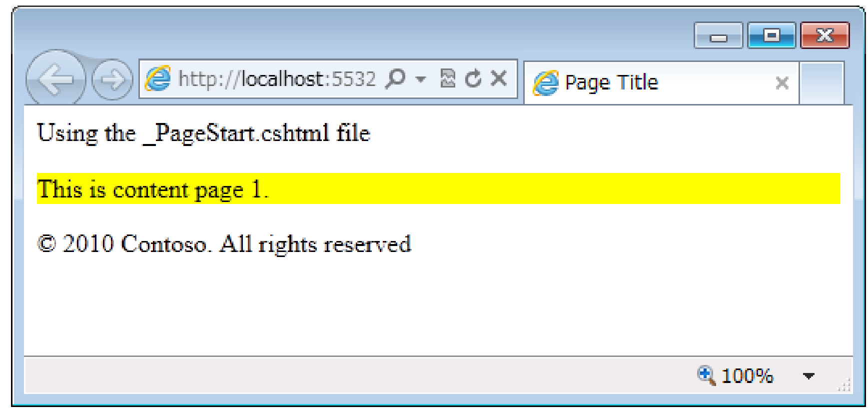
5. InitPages フォルダーで、Content1.cshtml という名前のファイルを作成し、デフォルトのマークアップを以下で置き換えます。

```
<p>This is content page 1.</p>
```

6. InitPages フォルダーで、もうひとつ Content2.cshtml という名前のファイルを作成し、デフォルトのマークアップを以下で置き換えます。

```
<p>This is content page 2.</p>
```

7. ブラウザーで Content1.cshtml を実行します。




Content1.cshtml ページを実行するとき、_PageStart.cshtml ファイルは Layout を設定し、PageData["MyBackground"]を色として設定します。Content1.cshtml では、このレイアウトと色が適用されます。

8. ブラウザーで、Content2.cshtml を表示します。

どちらのページも_PageStart.cshtml でレイアウト ページと色が初期化されるため、レイアウトは同じです。

## エラーを処理するために_PageStart.cshtml を使う

もうひとつの_PageStart.cshtml のよい使い方として、フォルダーの.cshtml ページ中で発生するかもしれないプログラミング エラー（例外）を処理することがあります。以下の例は、これを行う方法を示しています。

1. ルート フォルダーで、InitCatch という名前のフォルダーを作成します。
2. Web サイトの InitCatch フォルダーで、_PageStart.cshtml という名前のファイルを作成し、既存のマークアップを以下のコードで置き換えます。

```
@{
    try
    {
        RunPage();
    }
    catch (Exception ex)
    {
        Response.Redirect("~/Error.cshtml?source=" +
            HttpUtility.UrlEncode(Request.AppRelativeCurrentExecutionFilePath));
    }
}
```

このコードでは、明示的に try ブロックの中で RunPage ブロックを呼び出すように、リクエスト ページを実行しています。リクエスト ページでプログラミング エラーが発生すると、catch ブロック中のコードが実行されます。この場合、コードはページ（Error.cshtml）にリダイレクトされ、URL の一部としてエラーを発生させたファイルの名前が渡されます（このページはすぐに作成されます。）

3. Web サイトの InitCatch フォルダーで、Exception.cshtml という名前のファイルを作成し、既存のマークアップを以下のコードで置き換えます。

```
@{
    var db = Database.Open("invalidDatabaseFile");
}
```

この例のため、このページでは、存在しないデータベースを開こうとすることでわざとエラーを発生させています。

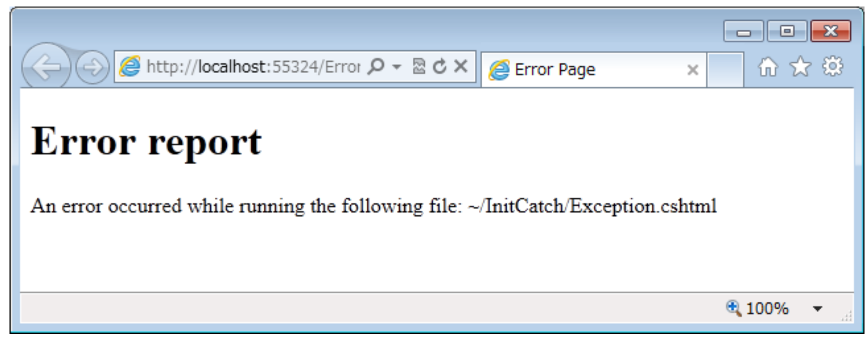
4. ルート フォルダーで、Error.cshtml という名前のファイルを作成し、既存のマークアップを以下のコードで置き換えます。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Error Page</title>
    </head>
    <body>
```

```
        <h1>Error report</h1>
        <p>An error occurred while running the following file: @Request["source"]</p>
    </body>
</html>
```

このページでは、@Request["source"]という式が URL から値を取り出して表示します。

5. ツールバーで、「保存」をクリックします。
6. ブラウザーで、Exception.cshtml を実行します。




Exception.cshtml でエラーが発生するため、_PageStart.cshtml ページはメッセージを表示する Error.cshtml ファイルにリダイレクトされます。

例外に関する細については、「第 2 章　Razor 構文を使った ASP.NET Web プログラミングの紹介」を参照してください。

## フォルダーへのアクセスを制限するために_PageStart.cshtml を使う

フォルダーのすべてのファイルに対するアクセスを制限するために_PageStart.cshtml を使うこともできます。

1. 「テンプレートからサイトを作成する」を使って、新しい Web サイトを作成します。
2. 利用可能なテンプレートから、「スターター サイト」を選びます。
3. ルート フォルダーで、AuthenticatedContent という名前のフォルダーを作成します。
4. AuthenticatedContent フォルダーで、_PageStart.cshtml という名前のファイルを作成し、既存のマークアップを以下のコードで置き換えます。

```
@{
    Response.CacheControl = "no-cache";

    if (!WebSecurity.IsAuthenticated) {
        Response.Redirect("~/Account/Login");
    }
}
```

コードの最初は、フォルダー中のすべてのファイルがキャッシュされることを防ぐことです。（これは、公開されているコンピューターで、次のユーザーにキャッシュされたページが使われないようにするシナリオで必要

なことです。）　次に、コードは、フォルダー中のどのページについても表示する前にユーザーがサイトにサイン インしているかどうかを確認します。ユーザーがサイン インしていなければ、コードはログイン ページにリダイレクトします。

5. AuthenticatedContent フォルダーで、新たに Page.cshtml という名前のページを作成します。
6. デフォルトのマークアップを、以下で置き換えます。

```
@{
    Layout = "~/_SiteLayout.cshtml";
    Page.Title = "Authenticated Content";
}
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
    </head>
    <body>
        Thank you for authenticating!
    </body>
</html>
```

7. ブラウザーで、Page.cshtml を実行します。コードは、ログイン ページにリダイレクトします。ログインする前に登録しなければなりません。登録して、ログインすると、このページに移動して、コンテンツを表示できます。

## 読みやすく検索しやすい URL を作成する

サイト中のページの URL は、そのサイトがどのように働くかに影響します。「親しみやすい」URL ならば、人々がサイトを利用しやすくなります。さらに、親しみやすい URL を使うことは ASP.NET の Web サイトに対する検索エンジン最適化（SEO）を自動的に手助けします。

### ルーティングについて

ASP.NET は、サーバー上のファイルを指し示すだけでなく、ユーザーの動作を表現する意味を持つ URL を作成できます。架空のブログのために用意された以下の URL を比べてみてください。

```
http://www.contoso.com/Blog/blog.cshtml?categories=hardware
http://www.contoso.com//Blog/blog.cshtml?startdate=2009-11-01&enddate=2009-11-30

http://www.contoso.com/Blog/categories/hardware/
http://www.contoso.com/Blog/2009/November
```

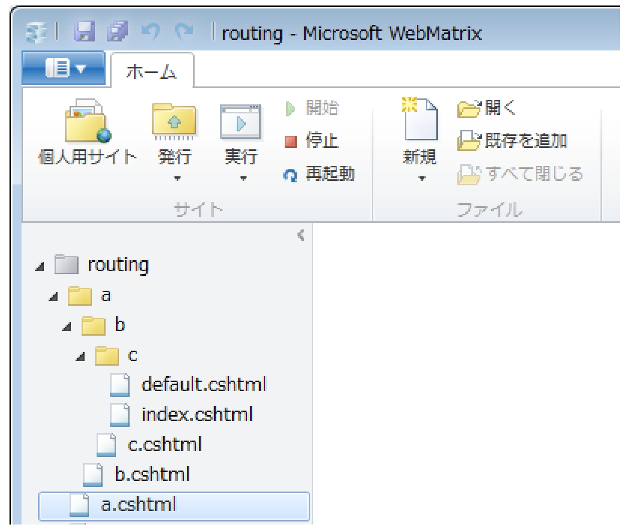
最初の 2 つの場合、ユーザーはブログを表示するために blog.cshtml ページを使っていることを知らなければならず、正しい分類や日付の範囲を得るための問い合わせ文字列を構築しなければなりません。後の 2 つの例では、作成が容易です。

最初の例における URL は、直接特定のファイル（blog.cshtml）を指し示しています。何らかの理由で、サーバー上のフォルダーを移動することになるか、ブログを異なるページを使うように書き直すことになれば、リンクは使えなくなります。後ろの例では、URL は特定のページを指し示しておらず、ブログの実装や場所が変わっても、URL は正しく使えます。

ASP.NET では、ルーティングを使うことで、上記の例のような親しみやすい URL を作成できます。ルーティングは、URL から、リクエストに対応するページ（または複数のページ）への論理マッピングを作成します。マッピングは論理的（物理的でない、つまり特定のファイルでない）に行われるため、サイトの URL を定義する方法には高い柔軟性があります。

## どのようにルーティングが働くか

ASP.NET がリクエストを処理するとき、URL を読み込んでどのようにルーティングするかを決定します。ASP.NET は、ディスク上のファイルへの URL の個々の断片を、左から右へ一致させようとします。一致すれば、URL の残りの部分はパス情報としてページに渡されます。たとえば、Web サイトにおいて以下のフォルダー構造を考えてみてください。




さらに、以下の URL を使ってリクエストされると想定します。

```
http://www.contoso.com/a/b/c
```

この場合、次のように探されます。

1. /a/b/c.cshtml というパスと名前のファイルがあれば、実行して、何の情報も渡しません。
   そうでなければ…
2. /a/b.cshtml というパスと名前のファイルがあれば、これを使い、情報として c を渡します。
   そうでなければ…
3. /a.cshtml というパスと名前のファイルがあれば、このページを実行して、情報として b/c を渡します。

指定したフォルダーに正確に一致する.cshtml ファイルが見つからなければ、ASP.NET は代わりに以下のファイルを探そうとします。

4. /a/b/c/default.cshtml（パス情報はありません）。
5. /a/b/c/index.cshtml（パス情報はありません）。

**注意** 明確にしておくと、特定のページのリクエスト（つまり、.cshtml ファイル拡張子を含めたリクエスト）については期待通りに動作します。http://www.contoso.com/a/b.cshtml のようなリクエストは、b.cshtml ページを実行します。

ページ中で、ディレクトリ情報を持つページの UrlData プロパティを通じてパス情報を取得できます。ViewCustomers.cshtml という名前のファイルがあり、サイトがこのリクエストを取得すると考えてみてください。

```
http://mysite.com/myWebSite/ViewCustomers/1000
```

上記の決まりで説明したとおり、リクエストはページに送られます。ページの中で、以下のようなコードを使うことで、パス情報（ここでは、"1000"という値）を取得して表示できます。

```
<!DOCTYPE html>
<html>
    <head>
        <title>URLData</title>
    </head>
    <body>
        Customer ID: @UrlData[0].ToString()
    </body>
</html>
```

**注意** ルーティングは、ファイル名全体を取り込まないため、同じファイル名で拡張子が違うページがあると、あいまいになります（たとえば、MyPage.cshtml と MyPage.html）。ルーティングの問題を回避するためには、拡張子だけが違う名前をサイトに置かないようにしてください。

## その他のリソース

- ASP.NET Web Pages with Razor Syntax Reference

## 付録 – ASP.NET クイック API リファレンス

このページは、Razor 構文を使う ASP.NET Web ページのプログラミングにおいて、もっとも一般的に使われるオブジェクトやプロパティ、メソッドの簡潔な例を一覧にしたものです。

API のリファレンス ドキュメントについては、MSDN 上の「ASP.NET Web Pages Reference Documentation」を参照してください。

この付録には、以下のリファレンス情報が含まれます。

- クラス
- データ
- ヘルパー

## クラス

<table>
<tr><td>AsBool(), AsBool(true|false)<br>文字列値を論理値（true/false）に変換します。文字列が true/false を表していない場合は、false または指定された値を返します。<br>bool b = stringValue.AsBool();</td></tr>
<tr><td>AsDateTime(), AsDateTime(value)<br>文字列値を日付/時間に変換します。文字列が日付/時間を表していない場合は、DateTime.MinValue または指定された値を返します。<br>DateTime dt = stringValue.AsDateTime();</td></tr>
<tr><td>AsDecimal(), AsDecimal(value)<br>文字列値を decimal 値に変換します。文字列が decimal 値を表していない場合は、0.0 または指定された値を返します。<br>decimal d = stringValue.AsDecimal();</td></tr>
<tr><td>AsFloat(), AsFloat(value)<br>文字列値を浮動小数値（float）に変換します。文字列が浮動小数値を表していない場合は、0.0 または指定された値を返します。<br>float d = stringValue.AsFloat();</td></tr>
<tr><td>AsInt(), AsInt(value)<br>文字列値を整数値（integer）に変換します。文字列が整数値を表していない場合は、0 または指定された値を返します。<br>int i = stringValue.AsInt();</td></tr>
<tr><td>Href(path [, param1 [, param2]])<br>ローカル ファイル名からブラウザー互換の URL を作成します。オプションとして、パス部分も追加できます。<br><a href="@Href("~/Folder/File")">Link to My File</a><br><a href="@Href("~/Product", "Tea")">Link to Product</a></td></tr>
</table>

Html.Raw(value)

value を HTML エンコードされた出力としてレンダリングする代わりに、HTML マークアップとしてレンダリングします。

```
@* Inserts markup into the page. *@
@Html.Raw("<div>Hello <em>world</em>!</div>")
```

---

IsBool(), IsDateTime(), IsDecimal(), IsFloat(), IsInt()

値が文字列から指定された方に変換できる場合に true を返します。

```
var isint = stringValue.IsInt();
```

---

IsEmpty()

オブジェクトや変数が値を持たないときに true を返します。

```
if (Request["companyname"].IsEmpty()) {
    @:Company name is required.<br />
}
```

---

IsPost

リクエストが POST のときに true を返します。（初期リクエストは、通常 GET です。）

```
if (IsPost) { Response.Redirect("Posted"); }
```

---

Layout

このページに適用するレイアウトページのパスを指定します。

```
Layout = "_MyLayout.cshtml";
```

---

PageData[key], PageData[index], Page

現在のリクエスト中のページ、レイアウト ページ、部分ページ間で共有するデータを含みます。以下の例のように、同じデータをアクセスする動的な Page プロパティを使うこともできます。

```
PageData["FavoriteColor"] = "red";
PageData[1] = "apples";
Page.MyGreeting = "Good morning";
// Displays the value assigned to PageData[1] in the page.
@Page[1]
ASP.NET Web Pages Using The Razor Syntax
Appendix – ASP.NET Quick API Reference 212
// Displays the value assigned to Page.MyGreeting.
@Page.MyGreeting
```

| |
|---|
| RenderBody()<br>（レイアウト ページ）どの名前付きセクションでもないコンテンツ ページのコンテンツをレンダリングします。<br>@RenderBody() |
| RenderPage(path, values)<br>RenderPage(path[, param1 [, param2]])<br>指定されたパスと、オプションの追加データを使って、コンテンツ ページをレンダリングします。PageData からの追加パラメータは、位置（以下の例 1）またはキー（以下の例 2）を使って取得できます。<br>RenderPage("_MySubPage.cshtml", "red", 123, "apples")<br>RenderPage("_MySubPage.cshtml", new { color = "red", number = 123, food = "apples" }) |
| RenderSection(sectionName [, required = true\|false])<br>（レイアウト ページ）名前を持つコンテンツ セクションをレンダリングします。Required を false に設定すると、セクションはオプションになります。<br>@RenderSection("header") |
| Request.Cookies[key]<br>HTTP クッキーの値を取得または設定します。<br>var cookieValue = Request.Cookies["myCookie"].Value; |
| Request.Files[key]<br>現在のリクエストでアップロードされたファイルを取得します。<br>Request.Files["postedFile"].SaveAs(@"MyPostedFile"); |
| Request.Form[key]<br>フォーム中でポストされたデータを（文字列として）取得します。Request[Key] は、Request.Form と Request.QueryString コレクションの両方を調べます。<br>var formValue = Request.Form["myTextBox"];<br>// This call produces the same result.<br>var formValue = Request["myTextBox"]; |
| Request.QueryString[key]<br>URL の問い合わせ文字列で指定されたデータを取り出します。Request[Key]は、Request.Form と Request.QueryString コレクションの両方を調べます。<br>var queryValue = Request.QueryString["myTextBox"];<br>// This call produces the same result.<br>var queryValue = Request["myTextBox"]; |

Request.Unvalidated(key)
Request.Unvalidated().QueryString|Form|Cookies|Headers[key]

フォームの要素、問い合わせ文字列の値、クッキー、またはヘッダーの値について、リクエストの検証を個別に無効化します。リクエストの検証は、デフォルトで有効になっており、マークアップや他の潜在的に危険なコンテンツを防止します。

```
// Call the method directly to disable validation on the specified item from one of the Request collections.
Request.Unvalidated("userText");
// You can optionally specify which collection the value is from.
var prodID = Request.Unvalidated().QueryString["productID"];
var richtextValue = Request.Unvalidated().Form["richTextBox1"];
var cookie = Request.Unvalidated().Cookies["mostRecentVisit"];
```

---

Response.AddHeader(name, value)

レスポンスに HTTP サーバー ヘッダーを追加します。

```
// Adds a header that requests client browsers to use basic authentication.
Response.AddHeader("WWW-Authenticate", "BASIC");
```

---

Response.OutputCache(seconds [, sliding] [, varyByParams])

指定された時間、ページの出力をキャッシュします。オプションとして sliding を設定し、個々のページアクセスごとにタイムアウトをリセットできます。また、varyByParams は、ページ リクエストでそれぞれの問い合わせ文字列に対して、異なるバージョンとしてページをキャッシュします。

```
Response.OutputCache(60);
Response.OutputCache(3600, true);
Response.OutputCache(10, varyByParams : new[] {"category","sortOrder"});
```

---

Response.Redirect(path)

ブラウザーのリクエストを新しい場所にリダイレクトします。

```
Response.Redirect("~/Folder/File");
```

---

Response.SetStatus(httpStatusCode)

ブラウザーに送信する HTTP のステータスコードを設定します。

```
Response.SetStatus(HttpStatusCode.Unauthorized);
Response.SetStatus(401);
```

---

Response.AddHeader(name, value)

レスポンスに HTTP サーバー ヘッダーを追加します。

```
// Adds a header that requests client browsers to use basic authentication.
Response.AddHeader("WWW-Authenticate", "BASIC");
```

---

Response.WriteBinary(data [, mimetype])

レスポンスへ、data をコンテンツとして出力します。オプションとして MIME タイプを指定できます。

```
Response.WriteBinary(image, "image/jpeg");
```

Response.WriteFile(file)

レスポンスへ、ファイルをコンテンツとして出力します。

```
Response.WriteFile("file.ext");
```

@section(sectionName) { content }

（レイアウト）名前を持つコンテンツ セクションを定義します。

```
@section header { <div>Header text</div> }
```

Server.HtmlDecode(htmlText)

HTML エンコードされた文字列をデコードします。

```
var htmlDecoded = Server.HtmlDecode("&lt;html&gt;");
```

Server.HtmlEncode(text)

文字列をエンコードして HTML マークアップ中にレンダリングします。

```
var htmlEncoded = Server.HtmlEncode("<html>");
```

Server.MapPath(virtualPath)

指定された仮想パスに対応するサーバーの物理パスを返します。

```
var dataFile = Server.MapPath("~/App_Data/data.txt");
```

Server.UrlDecode(urlText)

URL からテキストをデコードします。

```
var urlDecoded = Server.UrlDecode("url%20data");
```

Session[key]

ユーザーがブラウザーを閉じるまで使える値を取得または設定します。

```
Session["FavoriteColor"] = "red";
```

ToString()

オブジェクトの値に対応する文字列を表示します。

```
<p>It is now @DateTime.Now.ToString()</p>
```

UrlData[index]

URL から追加データを取得します（たとえば、/MyPage/ExtraData）。

```
var pathInfo = UrlData[0];
```

WebSecurity.ChangePassword(userName, currentPassword, newPassword)

指定されたユーザーのためのパスワードを変更します。

```
var success = WebSecurity.ChangePassword("my-username","current-password", "new-password");
```

WebSecurity.ConfirmAccount(accountConfirmationToken)

ユーザー確認用のトークンを使ってアカウントを確認します。

```
var confirmationToken = Request.QueryString["ConfirmationToken"];
if(WebSecurity.ConfirmAccount(confirmationToken)) {
    //...
}
```

| |
|---|
| WebSecurity.CreateAccount(userName, password [, requireConfirmationToken = true\|false])<br>指定されたユーザー名とパスワードで新たなユーザー アカウントを作成します。確認用のトークンを要求する場合は、requireConfirmationToken を true にします。<br>WebSecurity.CreateAccount("my-username", "secretpassword"); |
| WebSecurity.CurrentUserId<br>現在、ログインしているユーザーID を取得します。<br>var userId = WebSecurity.CurrentUserId; |
| WebSecurity.CurrentUserName<br>現在、ログインしているユーザーの名前を取得します。<br>var welcome = "Hello " + WebSecurity.CurrentUserName; |
| WebSecurity.GeneratePasswordResetToken(username [, tokenExpirationInMinutesFromNow])<br>パスワードをリセットできるようにするために、ユーザーに電子メールで送信するパスワード リセット用のトークンを生成します。<br>var resetToken = WebSecurity.GeneratePasswordResetToken("my-username");<br>var message = "Visit http://example.com/reset-password/" + resetToken +<br>" to reset your password";<br>WebMail.Send(..., message); |
| WebSecurity.GetUserId(userName)<br>ユーザー名からユーザーID を返します<br>var urlDecoded = Server.UrlDecode("url%20data"); |
| WebSecurity.IsAuthenticated<br>現在のユーザーがログインしている場合に true を返します。<br>if(WebSecurity.IsAuthenticated) {...} |
| WebSecurity.IsConfirmed(userName)<br>ユーザーが（たとえば、確認用の電子メールを通じて）確認されている場合に、true を返します。<br>if(WebSecurity.IsConfirmed("joe@contoso.com")) { ... } |
| WebSecurity.Login(userName, password[, persistCookie])<br>クッキー中の認証トークンを設定することで、そのユーザーをログインします。<br>if(WebSecurity.Login("username", "password")) { ... } |
| WebSecurity.Logout()<br>認証トークン クッキーを削除することで、ユーザーをログアウトします。<br>WebSecurity.Logout(); |
| WebSecurity.RequireAuthenticatedUser()<br>ユーザーが認証されていなければ、HTTP ステータスを 401（認証の失敗）に設定します。<br>WebSecurity.RequireAuthenticatedUser(); |

| WebSecurity.RequireRoles(roles)<br>現在のユーザーが、指定されたロールのメンバーでなければ、HTTP ステータスを 401（認証の失敗）に設定します。<br>WebSecurity.RequireRoles("Admin", "Power Users"); |
|---|
| WebSecurity.RequireUser(userId)<br>WebSecurity.RequireUser(userName)<br>現在のユーザーが指定されたユーザーでない場合は、HTTP ステータスを 401（認証の失敗）に設定します。<br>WebSecurity.RequireUser("joe@contoso.com"); |
| WebSecurity.ResetPassword(passwordResetToken, newPassword)<br>パスワード リセット用のトークンが正しければ、ユーザーのパスワードが新しいパスワードに変更されます。<br>WebSecurity.ResetPassword( "A0F36BFD9313", "new-password") |

## データ

| Database.Execute(SQLstatement [, parameters]<br>INSERT、DELETE または UPDAET のような SQLstatement（およびオプションのパラメーター）を実行し、影響されたレコードの数を返します<br>db.Execute("INSERT INTO Data (Name) VALUES ('Smith')");<br>db.Execute("INSERT INTO Data (Name) VALUES (@0)", "Smith"); |
|---|
| Database.GetLastInsertId()<br>最後に挿入された行の ID 列を返します。<br>db.Execute("INSERT INTO Data (Name) VALUES ('Smith')");<br>var id = db.GetLastInsertId(); |
| Database.Open(filename)<br>Database.Open(connectionStringName)<br>指定されたデータベース ファイルか、Web.config ファイルの名前付き接続文字列を使って指定されたデータベースを開きます。<br>// Note that no filename extension is specified.<br>var db = Database.Open("SmallBakery"); // Opens SmallBakery.sdf in App_Data<br>// Opens a database by using a named connection string.<br>var db = Database.Open("SmallBakeryConnectionString"); |
| Database.OpenConnectionString(connectionString)<br>接続文字列を使ってデータベースを開きます。（接続文字列名を使う、Database.Open と対照的です。）<br>var db = Database.OpenConnectionString("Data Source=\|DataDirectory\|¥SmallBakery.sdf"); |

| Database.Query(SQLstatement[, parameters])<br>SQLstatement（およびオプションのパラメーター）を使ってデータベースに問い合わせをかけ、結果をコレクションとして返します。<br>foreach (var result in db.Query("SELECT * FROM PRODUCT")) {<p>@result.Name</p>}<br>foreach (var result = db.Query("SELECT * FROM PRODUCT WHERE Price > @0", 20))<br>{ <p>@result.Name</p> } |
|---|
| Database.QuerySingle(SQLstatement [, parameters])<br>SQLstatement（およびオプションのパラメーター）を実行し、単一レコードを返します。<br>var product = db.QuerySingle("SELECT * FROM Product WHERE Id = 1");<br>var product = db.QuerySingle("SELECT * FROM Product WHERE Id = @0", 1); |
| Database.QueryValue(SQLstatement [, parameters])<br>SQLstatement（およびオプションのパラメーター）を実行し、単一の値を返します。<br>var count = db.QueryValue("SELECT COUNT(*) FROM Product");<br>var count = db.QueryValue("SELECT COUNT(*) FROM Product WHERE Price > @0", 20); |

## ヘルパー

| Analytics.GetGoogleHtml(webPropertyId)<br>指定された ID に対する Google Analytics 用の JavaScript コードをレンダリングします。<br>@Analytics.GetGoogleHtml("MyWebPropertyId") |
|---|
| Analytics.GetStatCounterHtml(project, security)<br>指定されたプロジェクトに対する StatCounter Analytics 用の JavaScript コードをレンダリングします。<br>@Analytics.GetStatCounterHtml(89, "security") |
| Analytics.GetYahooHtml(account)<br>指定されたアカウントに対する Yahoo Analytics 用の JavaScript コードをレンダリングします。<br>@Analytics.GetYahooHtml("myaccount") |
| Bing.SearchBox([boxWidth])<br>検索を Bing に渡します。検索するタイトルと検索ボックスのタイトルを指定するために、Bing.SiteUrl と Bing.SiteTitle プロパティを設定できます。通常、これらのプロパティは_AppStart ページで設定します。<br>@Bing.SearchBox() @* Searches the web.*@<br>@{<br>    Bing.SiteUrl = "www.asp.net"; @* Limits search to the www.asp.net site. *@<br>}<br>@Bing.SearchBox() |

| |
|---|
| Bing.AdvancedSearchBox([, boxWidth] [, resultWidth] [, resultHeight] [, themeColor] [, locale])<br>ページ中に Bing の検索結果を表示します。オプションで書式を指定できます。検索するサイトと検索ボックスのタイトルを指定するために、、Bing.SiteUrl と Bing.SiteTitle プロパティを設定できます。通常、これらのプロパティは_AppStart ページで設定します。<br>@{<br>    Bing.SiteUrl = "www.asp.net";<br>    Bing.SiteTitle = "ASP.NET Custom Search";<br>}<br>@Bing.AdvancedSearchBox(<br>    boxWidth: "250px",<br>    resultWidth: 600,<br>    resultHeight: 900,<br>    themeColor: "Green",<br>    locale: "en-US") |
| Chart(width, height [, template] [, templatePath])<br>グラフを初期化します。<br>@{<br>    var myChart = new Chart(width: 600, height: 400);<br>} |
| Chart.AddLegend([title] [, name])<br>グラフに凡例を追加します。<br>@{<br>var myChart = new Chart(width: 600, height: 400)<br>    .AddLegend("Basic Chart")<br>    .AddSeries(<br>        name: "Employee",<br>        xValue: new[] { "Peter", "Andrew", "Julie", "Mary", "Dave" },<br>        yValues: new[] { "2", "6", "4", "5", "3" })<br>    .Write();<br>} |

Chart.AddSeries([name] [, chartType] [, chartArea] [, axisLabel] [, legend] [, markerStep] [, xValue] [, xField] [, yValues] [, yFields] [, options])

グラフに値の系列を追加します。

```
@{
var myChart = new Chart(width: 600, height: 400)
    .AddSeries(
        name: "Employee",
        xValue: new[] { "Peter", "Andrew", "Julie", "Mary", "Dave" },
        yValues: new[] { "2", "6", "4", "5", "3" })
    .Write();
}
```

---

Crypto.Hash(string [, algorithm])

Crypto.Hash(bytes [, algorithm])

指定されたデータのハッシュを返します。デフォルトのアルゴリズムは sha256 です。

```
@Crypto.Hash("data")
```

---

Facebook.LikeButton(href [, buttonLayout] [, showFaces] [, width] [, height] [, action] [, font] [, colorScheme] [, refLabel])

Facebook ユーザーがページに接続できるようにします。

```
@Facebook.LikeButton("www.asp.net")
```

---

FileUpload.GetHtml([initialNumberOfFiles] [, allowMoreFilesToBeAdded] [, includeFormTag] [, addText] [, uploadText])

ファイルをアップロードする UI をレンダリングします。

```
@FileUpload.GetHtml(initialNumberOfFiles:1, allowMoreFilesToBeAdded:false,
includeFormTag:true, uploadText:"Upload")
```

---

GamerCard.GetHtml(gamerTag)

指定された Xbox のゲーマー タグをレンダリングします。

```
@GamerCard.GetHtml("joe")
```

---

Gravatar.GetHtml(email [, imageSize] [, defaultImage] [, rating] [, imageExtension] [, attributes])

指定された電子メールアドレスに対する Gravatar イメージをレンダリングします。

```
@Gravatar.GetHtml("joe@contoso.com")
```

---

Json.Encode(object)

データ オブジェクトを JavaScript Object Notation（JSON）形式の文字列に変換します。

```
var myJsonString = Json.Encode(dataObject);
```

---

Json.Decode(string)

JSON エンコードされた入力文字列を、繰り返し使うか、データベースに挿入できるデータ オブジェクトに変換します。

```
var myJsonObj = Json.Decode(jsonString);
```

<table>
<tr><td>LinkShare.GetHtml(pageTitle [, pageLinkBack] [, twitterUserName] [, additionalTweetText] [, linkSites])<br>指定されたタイトルとオプションの URL を使ってソーシャル ネットワークへのリンクをレンダリングします。<br>@LinkShare.GetHtml("ASP.NET Web Pages Samples")<br>@LinkShare.GetHtml("ASP.NET Web Pages Samples", "http://www.asp.net")</td></tr>
<tr><td>ModelStateDictionary.AddError(key, errorMessage)<br>フォーム項目にエラーメッセージを関連付けます。このメンバーにアクセスするには、ModelState ヘルパーを使います。<br>ModelState.AddError("email", "Enter an email address");</td></tr>
<tr><td>ModelStateDictionary.AddFormError(errorMessage)<br>フォームにエラーメッセージを関連付けます。このメンバーにアクセスするには、ModelState ヘルパーを使います。<br>ModelState.AddFormError("Password and confirmation password do not match.");</td></tr>
<tr><td>ModelStateDictionary.IsValid<br>検証エラーがないときに true を返します。このメンバーにアクセスするには、ModelState ヘルパーを使います。<br>if (ModelState.IsValid) { // Save the form to the database }</td></tr>
<tr><td>ObjectInfo.Print(value [, depth] [, enumerationLength])<br>オブジェクトとすべての子オブジェクトのプロパティと値をレンダリングします。<br>@ObjectInfo.Print(person)</td></tr>
<tr><td>Recaptcha.GetHtml([, publicKey] [, theme] [, language] [, tabIndex])<br>reCAPTCHA 検証テストをレンダリングします。<br>@ReCaptcha.GetHtml()</td></tr>
<tr><td>ReCaptcha.PublicKey<br>ReCaptcha.PrivateKey<br>reCAPTCHA サービスのための公開鍵と秘密鍵を設定します。通常、これらのプロパティは_AppStart ページで設定します。<br>ReCaptcha.PublicKey = "your-public-recaptcha-key";<br>ReCaptcha.PrivateKey = "your-private-recaptcha-key";</td></tr>
<tr><td>ReCaptcha.Validate([, privateKey])<br>reCAPTCHA 検証テストの結果を返します。<br>if (ReCaptcha.Validate()) {<br>    // Test passed.<br>}</td></tr>
<tr><td>ServerInfo.GetHtml()<br>ASP.NET Web ページに関するステータス情報をレンダリングします。<br>@ServerInfo.GetHtml()</td></tr>
</table>

| |
|---|
| Twitter.Profile(twitterUserName)<br>指定されたユーザーの Twitter ストリームをレンダリングします。<br>@Twitter.Profile("billgates") |
| Twitter.Search(searchQuery)<br>指定された検索テキストに対する Twitter ストリームをレンダリングします。<br>@Twitter.Search("asp.net") |
| Video.Flash(filename [, width, height])<br>指定されたファイルに対応する Flash ビデオ プレイヤーをレンダリングします。オプションで幅と高さを指定できます。<br>@Video.Flash("test.swf", "100", "100") |
| Video.MediaPlayer(filename [, width, height])<br>指定されたファイルに対応する Windows Media プレイヤーをレンダリングします。オプションで幅と高さを指定できます。<br>@Video.MediaPlayer("test.wmv", "100", "100") |
| Video.Silverlight(filename, width, height)<br>指定された.xap ファイルに対応する Silverlight プレイヤーを、幅と高さを指定してレンダリングします。<br>@Video.Silverlight("test.xap", "100", "100") |
| WebCache.Get(key)<br>key で指定されたオブジェクトを返すか、オブジェクトが見つからない場合は null を返します。<br>var username = WebCache.Get("username") |
| WebCache.Remove(key)<br>key で指定されたオブジェクトをキャッシュから削除します。<br>WebCache.Remove("username") |
| WebCache.Set(key, value [, minutesToCache] [, slidingExpiration])<br>Key で指定した名前の下で、キャッシュに value を配置します。<br>WebCache.Set("username", "joe@contoso.com ") |
| WebGrid(data)<br>問い合わせのデータを使って新たな WebGrid オブジェクトを作成します。<br>var db = Database.Open("SmallBakery");<br>var grid = new WebGrid(db.Query("SELECT * FROM Product")); |
| WebGrid.GetHtml()<br>HTML テーブル中に、データを表示するマークアップをレンダリングします。<br>@grid.GetHtml()// The 'grid' variable is set when WebGrid is created. |
| WebGrid.Pager()<br>WebGrid オブジェクトのページングコントロールをレンダリングします。<br>@grid.Pager() // The ‘grid’ variable is set when WebGrid is created. |

| |
|---|
| WebImage(path)<br>指定されたパスからイメージを読み込みます。<br>var image = new WebImage("test.png"); |
| WebImage.AddImagesWatermark(image)<br>指定したイメージをウォーター マークとして追加します。<br>WebImage photo = new WebImage("test.png");<br>WebImage watermarkImage = new WebImage("logo.png");<br>photo.AddImageWatermark(watermarkImage); |
| WebImage.AddTextWatermark(text)<br>指定したテキストをイメージに追加します。<br>image.AddTextWatermark("Copyright") |
| WebImage.FlipHorizontal()<br>WebImage.FlipVertical()<br>水平または垂直方向にイメージを反転させます。<br>image.FlipHorizontal();<br>image.FlipVertical(); |
| WebImage.GetImageFromRequest()<br>ファイルをアップロード中にイメージをページに送信する際、イメージを読み込みます。<br>var image = WebImage.GetImageFromRequest(); |
| WebImage.Resize(width, height)<br>イメージのサイズを変更します。<br>image.Resize(100, 100); |
| GamerCard.GetHtml(gamerTag)<br>指定された Xbox のゲーマー タグをレンダリングします。<br>@GamerCard.GetHtml("joe") |
| WebImage.RotateLeft()<br>WebImage.RotateRight()<br>イメージを左または右に回転します。<br>image.RotateLeft();<br>image.RotateRight(); |
| WebImage.Save(path [, imageFormat])<br>指定されたパスにイメージを保存します。<br>image.Save("test.png"); |
| WebMail.Password<br>SMTP サーバーのためのパスワードを設定します。通常、このプロパティは_AppStart ページで設定します。<br>WebMail.Password = "password"; |

| WebMail.Send(to, subject, body [, from] [, cc] [, filesToAttach] [, isBodyHtml] [, additionalHeaders])<br>電子メール メッセージを送信します。<br>WebMail.Send("touser@contoso.com", "subject", "body of message", "fromuser@contoso.com"); |
| --- |
| WebMail.SmtpServer<br>SMTP サーバー名を設定します。通常、このプロパティは_AppStart ページで設定します。<br>WebMail.SmtpServer = "smtp.mailserver.com"; |
| WebMail.UserName<br>SMTP サーバー名のユーザー名を設定します。通常、このプロパティは_AppStart ページで設定します。<br>WebMail.UserName = "Joe"; |

# 付録 - ASP.NET Web ページ Visual Basic

この付録は、Visual Basic による Razor 構文を使う ASP.NET Web ページ プログラミングの概要を提供します。

本書では、Razor 構文を使う ASP.NET のコード例は、C#を使っています。しかし、Razor 構文は、Visual Basic もサポートしています。Visual Basic で ASP.NET Web ページをプログラムするには、.vbhtml ファイル拡張子を持つ Web ページを作成し、Visual Basic のコードを追加します。この付録は、ASP.NET Web ページを作成するための Visual Basic 言語と構文の機能の概要を提供します。

ここでは次のことを学びます。

- 8 つのプログラミング Tips
- Visual Basic 言語と構文

## 8 つのプログラミング Tips

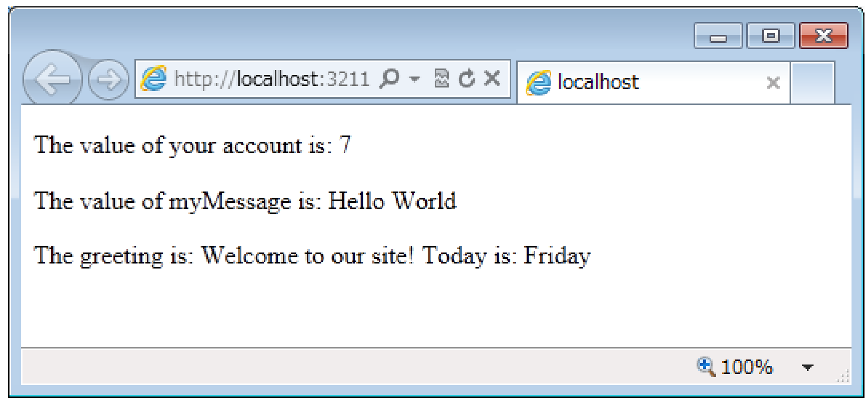
このセクションは、Razor 構文を使う ASP.NET サーバー コードを記述するために必ず知っておくべきいくつかの Tips について紹介します。

### ページには、@文字を使ってコードを追加する

@文字で、インライン（埋め込み）式、単一文のブロック、複数文のブロックを開始します。

```
<!-- Single statement blocks -->
    @Code Dim total = 7 End Code
    @Code Dim myMessage = "Hello World" End Code

<!-- Inline expressions -->
<p>The value of your account is: @total </p>
<p>The value of myMessage is: @myMessage</p>
<!-- Multi-statement block -->
@Code
    Dim greeting = "Welcome to our site!"
    Dim weekDay = DateTime.Now.DayOfWeek
    Dim greetingMessage = greeting & " Today is: " & weekDay.ToString()
End Code
<p>The greeting is: @greetingMessage</p>
```

この結果、ブラウザーには次のように表示されます。




**HTML エンコーディング**

前述のように@文字を使ってページ中にコンテンツを表示するとき、ASP.NET は出力を HTML エンコードします。これは、予約された HTML 文字（<、>、&など）を、HTML タグやエンティティとしてでなく、それらの文字が Web ページ中の文字として表示されるコードに置き換えるということです。HTML エンコードをしないと、サーバーからの出力を正しく表示することはできず、セキュリティ リスクを持つことになりかねません。

マークアップのタグとして解釈させるための HTML マークアップを出力したいのであれば（たとえば、パラグラフのための<p></p>やテキストを強調するための<em></em>）、本章の後半にある「コード ブロックにおけるテキスト、マークアップ、コードの連結」セクションを参照してください。

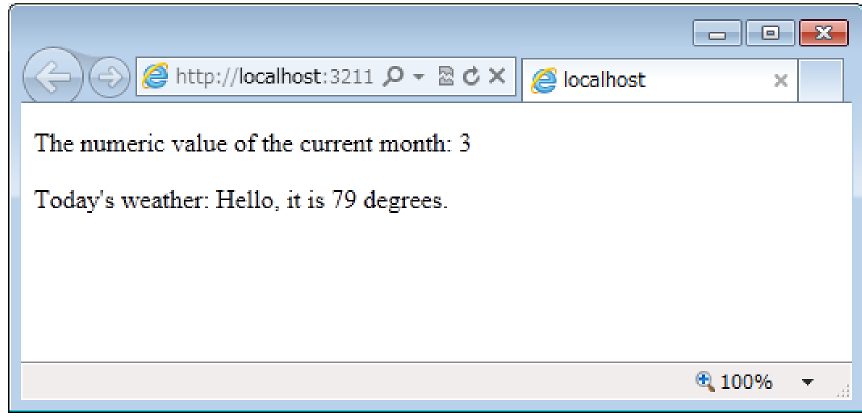
HTML エンコードについての詳細は、「第 4 章　フォームを扱う」を参照してください。

## Code…End Code でコード ブロックを囲む

コードブロックは 1 つ、または複数の文を含むもので、予約語 Code と End Code で囲まれます。予約語 Code は、@文字の直後に置き、間にスペース文字を入れることはできません。

```
<!-- Single statement block. -->
@Code
    Dim theMonth = DateTime.Now.Month
End Code

<p>The numeric value of the current month: @theMonth</p>
<!-- Multi-statement block. -->
@Code
    Dim outsideTemp = 79
    Dim weatherMessage = "Hello, it is " & outsideTemp & " degrees."
End Code
<p>Today's weather: @weatherMessage</p>
```

ブラウザーでは、次のように表示されます。




## ブロックの中で、各コード文を改行で終わらせる

Visual Basic のコードブロックの中では、それぞれの文は改行で終わります。（本章の後半で、必要に応じて 1 つの長い文を複数の行に折り返して記述する方法を説明します。）

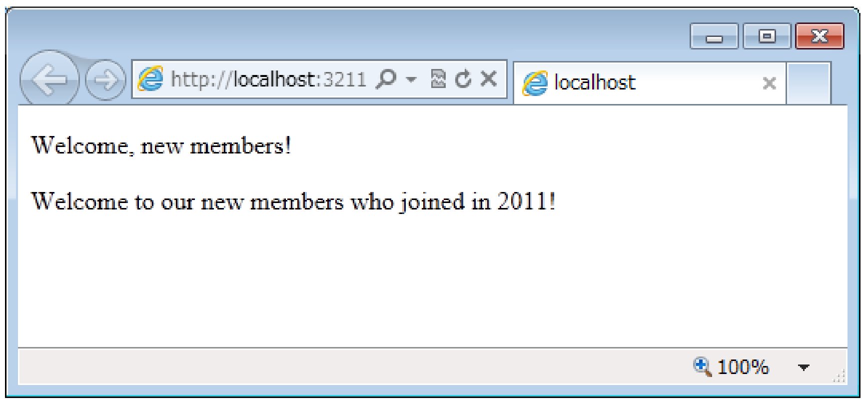
```
<!-- Single statement block. -->
@Code
    Dim theMonth = DateTime.Now.Month
End Code

<!-- Multi-statement block. -->
@Code
    Dim outsideTemp = 79
    Dim weatherMessage = "Hello, it is " & outsideTemp & " degrees."
End Code

<!-- An inline expression, so no line break needed. -->
<p>Today's weather: @weatherMessage</p>
```

## 値を保存するために変数を使う

変数を使って値を保存できます。これには文字列、数値、日付などが含まれます。新しい変数は予約語 Dim を使って作成します。変数は@を使って直接ページに挿入できます。

```
<!-- Storing a string -->
@Code
    Dim welcomeMessage = "Welcome, new members!"
End Code
<p>@welcomeMessage</p>

<!-- Storing a date -->
@Code
    Dim year = DateTime.Now.Year
End Code

<!-- Displaying a variable -->
<p>Welcome to our new members who joined in @year!</p>
```

ブラウザーでは、次のように表示されます。




## ダブル クォーテーション記号でリテラル文字列値を囲む

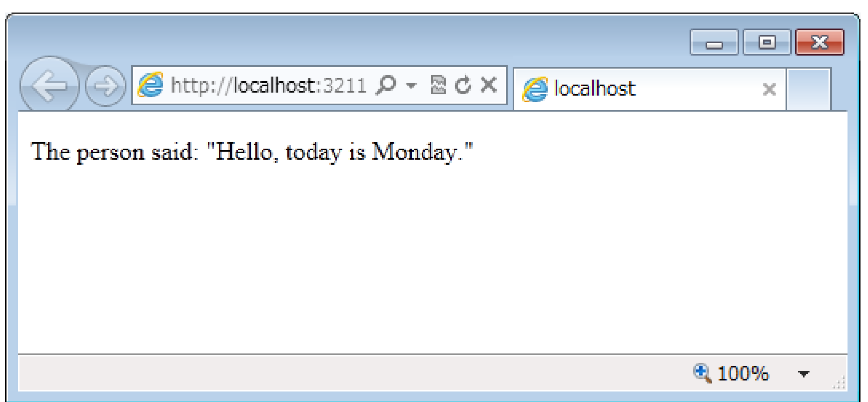
文字列は、テキストとして扱われる文字のつながりです。文字列を指定するには、ダブル クォーテーション記号で囲みます。

```
@Code
    Dim myString = "This is a string literal"
End Code
```

ダブル クォーテーション記号を埋め込む場合は、2 つのダブル クォーテーション記号を挿入します。ダブル クォーテーション文字を一度だけページ出力に表示させたいときは、囲まれた文字列として""を入力します。2 つのダブル クォーテーション文字を表示させたいときは、囲まれた文字列として""""を入力します。

```
<!-- Embedding double quotation marks in a string -->
@Code
    Dim myQuote = "The person said: ""Hello, today is Monday."""
End Code
<p>@myQuote</p>
```

結果は、次のようにブラウザーに表示されます。



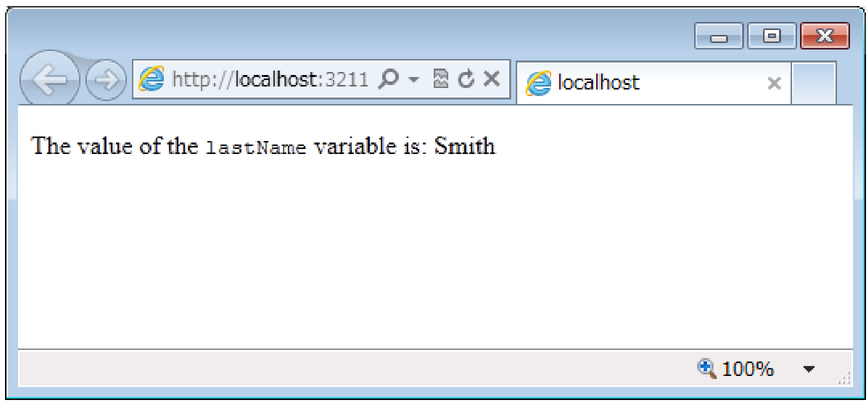
## コードは大小文字を区別しない

Visual Basic 言語は、大文字と小文字を区別しません。プログラミング用の予約語（Dim、If、True など）や変数名（myString、subTotal など）は、大文字と小文字のどちらで書いても同じです。

以下のコードは、小文字を使った lastname という変数名に値を代入し、大文字を使ってページに変数の値を出力します。

```
@Code
    Dim lastName = "Smith"
    ' Keywords like dim are also not case sensitive.
    DIM someNumber = 7
End Code
<p>The value of the <code>lastName</code> variable is: @LASTNAME</p>
```

結果は、次のようにブラウザーに表示されます。




## コードのほとんどが関係するオブジェクト

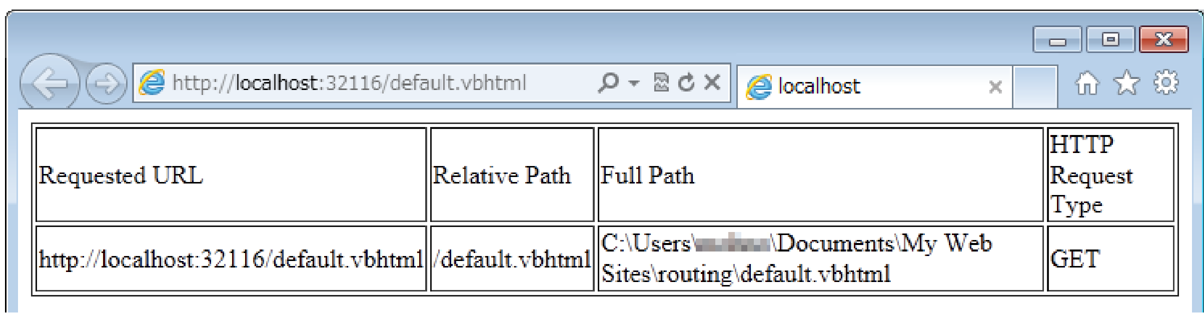
オブジェクトは、プログラムで扱うページ、テキストボックス、ファイル、イメージ、Web リクエスト、電子メールのメッセージ、顧客レコード（データベースの行）といったものをあらわします。オブジェクトは、その特徴をあらわすプロパティを持ちます。たとえば、テキストボックス オブジェクトは、Text プロパティを、リクエスト オブジェクトは Url プロパティを、電子メールのメッセージは From プロパティを、顧客オブジェクトは FirstName プロパティなどを持っています。また、オブジェクトは実行できる「動詞」としてメソッドを持ちます。たとえば、ファイル オブジェクトは Save メソッドを、イメージ オブジェクトは Rotate メソッドを、電子メール オブジェクトは Send メソッドなどを持っています。

ページ上のフォーム フィールドの値（テキストボックスなど）、リクエストを発生したブラウザーの種類、ページの URL、ユーザー アイディンティティなどの情報を得るために、しばしば Request オブジェクトを使います。次の例は、Request オブジェクトにアクセスする方法と、サーバー上のページの絶対パスを得るために Request オブジェクトの MapPath メソッドを呼び出す方法を示しています。

```
<table border="1">
<tr>
    <td>Requested URL</td>
    <td>Relative Path</td>
```

```
        <td>Full Path</td>
        <td>HTTP Request Type</td>
    </tr>
    <tr>
        <td>@Request.Url</td>
        <td>@Request.FilePath</td>
        <td>@Request.MapPath(Request.FilePath)</td>
        <td>@Request.RequestType</td>
    </tr>
    </table>
```

ブラウザーには次のように表示されます。



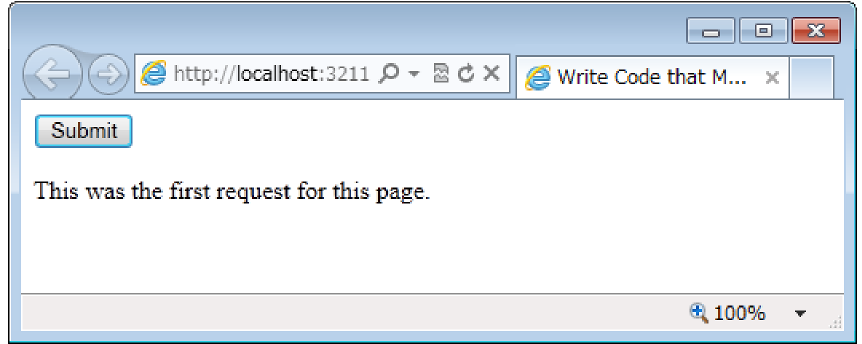

## 意思決定用のコードを書く

動的な Web ページの重要な機能は、条件に応じて何をするかを決められるようにすることです。この処理のもっとも典型的な方法は、if 文（必要に応じて else 文）を使います。

```
@Code
    Dim result = ""
    If IsPost Then
        result = "This page was posted using the Submit button."
    Else
        result = "This was the first request for this page."
End If
End Code
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Write Code that Makes Decisions</title>
    </head>
<body>

    <form method="POST" action="" >
        <input type="Submit" name="Submit" value="Submit"/>
        <p>@result</p>
    </form>

</body>
</html>
```

「If IsPost」という文は、「If IsPost = true」という記述の短縮版です。If 文には、条件を判断するいくつかの方法やコードブロックの繰り返しなどがあります。これらは本章で後述します。

この結果はブラウザーに次のように表示されます（[Submit] ボタンをクリックした後）。




**HTTP の GET と POST メソッドと IsPost プロパティ**

Web ページで使われるプロトコル（HTTP）は、サーバーにリクエストを送るために限られた数のメソッド（動詞）しかサポートしていません。もっとも典型的な 2 つが、ページを読むために使われる GET と、ページを送出するための POST です。一般に、ユーザーが最初にページをリクエストするとき、そのページは GET を使ってリクエストされます。ユーザーがフォームを入力して Submit をクリックすると、ブラウザーはサーバーに POST リクエストを送ります。

Web プログラミングでは、ページが GET としてまたは POST としてリクエストされたかどうかを判別することは便利で、ページをどのように処理するかを知ることができます。ASP.NET Web ページでは、IsPost プロパティを使ってリクエストが GET か POST かを区別できます。リクエストが POST であれば、IsPost プロパティは true を返し、フォーム上のテキストボックスの値を読むといった処理を実行できます。本書の多くの例は、IsPost の値に応じたページの処理方法について示しています。

## 単純なコード サンプル

以下の手順は、基本的なプログラミング テクニックを示すページを作成する方法です。この例では、ユーザーに 2 つの数字を入力させ、それらを加算した結果を表示します。

1. エディターで、ファイルを作成し、AddNumbers.vbhtml という名付けます。
2. 以下のコードとマークアップをページ中にコピーして、ページの既存の内容と置き換えます。

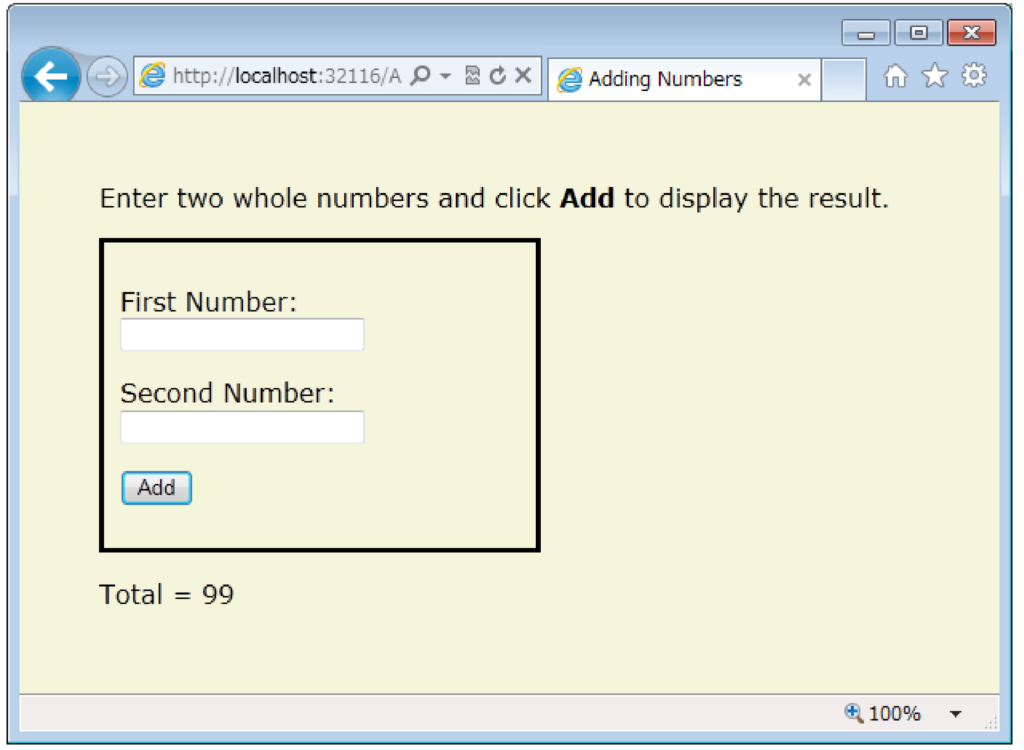
```
@Code
    Dim total = 0
    Dim totalMessage = ""
    if IsPost Then
        ' Retrieve the numbers that the user entered.
        Dim num1 = Request("text1")
        Dim num2 = Request("text2")
        ' Convert the entered strings into integers numbers and add.
        total = num1.AsInt() + num2.AsInt()
        totalMessage = "Total = " & total
    End If
End Code
<!DOCTYPE html>
<html lang="en">
```

```
    <head>
        <meta charset="utf-8" />
        <title>Adding Numbers</title>
        <style type="text/css">
            body {background-color: beige; font-family: Verdana, Ariel;
                margin: 50px;
                }
            form {padding: 10px; border-style: solid; width: 250px;}
        </style>
    </head>
<body>
    <p>Enter two whole numbers and click <strong>Add</strong> to display the result.</p>
    <p></p>
    <form action="" method="post">
    <p><label for="text1">First Number:</label>
    <input type="text" name="text1" />
    </p>
    <p><label for="text2">Second Number:</label>
    <input type="text" name="text2" />
    </p>
    <p><input type="submit" value="Add" /></p>
    </form>
    <p>@totalMessage</p>
</body>
</html>
```

ここでは、以下の点に注意しておいてください。

- @文字は、このページの最初のコードブロックを開始し、ページの最後の方に埋め込まれている totalMessage 変数に先行しています。
- ページの先頭にあるブロックは、Code…End Code で囲まれています。
- 変数 total、num1、num2、totalMessage は、いくつかの数字とひとつの文字列を保持します。
- totalMessage 変数に代入されるリテラル文字列値は、ダブル クォーテーション記号で囲まれています。
- Visual Basic のコードは大小文字を区別しないため、ページの最後の方で totalMessage 変数が使われるとき、この変数の名前は先頭のものとスペルが一致しているだけで十分です。
- 式「num.AsInt() + num2.AsInt()」は、オブジェクトとメソッドがどのように動作するかを示しています。それぞれの変数の AsInt メソッドはユーザーが入力した文字列を数（整数値）に変換するため、算術演算できるようになります。
- <form> タグは、「method="post"」という属性を含みます。これは、ユーザーが [Add] をクリックしたときに、このページが HTTP POST メソッドを使ってサーバーに送られることを指定するものです。ページが送信されるとき、「If(IsPost)」が真だと評価され、条件コードが実行されると、数値を加算した結果が表示されます。

3. ページを保存して、ブラウザーで実行します。（実行する前に、「ファイル」ワークスペースでこのページが選択されていることを確認してください。） 2 つの数字を入力して [Add] ボタンをクリックします。




## Visual Basic 言語と構文

「第 1 章　はじめての WebMatrix と ASP.NET Web ページ」では、ASP.NET Web ページの作成方法の基本的な例や、HTML マークアップへのサーバー コードの追加方法について示しました。ここでは、Razor 構文を使う ASP.NET サーバー コードで Visual Basic を使う基本、つまりプログラミング言語の規則について学びます。

これまでにプログラミング（とくに、C、C++、C#、Visual Basic または JavaScript）の経験があれば、ここに書かれていることの多くは親しみ深いでしょう。必要なことは、どのように.vbhtml ファイルに WebMatrix のコードを追加することだけかもしれません。

## 基本的な構文

### コード ブロックにおけるテキスト、マークアップ、コードの連結

サーバー コード ブロックでは、しばしばページにテキストやマークアップ（またはその両方）を出力したいことがあります。サーバー コード ブロックが、コードではない、そのままレンダリングすべきテキストを含む場合、ASP.NET はコードとテキストを区別できる必要があります。このために、いくつかの方法があります。

- <p></p> や <em></em> のような HTML 要素中にテキストを囲む。

```
@If IsPost Then
    ' This line has all content between matched <p> tags.
    @<p>Hello, the time is @DateTime.Now and this page is a postback!</p>
Else
    ' All content between matched tags, followed by server code.
    @<p>Hello, <em>Stranger!</em> today is: </p> @DateTime.Now
```

```
End If
```

HTML 要素はテキストや、追加の HTML 要素、サーバー コード式を含むことができます。ASP.NET は、開いた HTML タグを見つけると、その要素とそのコンテンツをすべてレンダリングし、ブラウザーに送ります。（かつ、サーバー コード式を解決します）

- @:演算子か、<text> 要素を使う。@: は、プレーン テキストや不一致の HTML タグを含む単一行のコンテンツを出力します。<text> 要素は、複数行をまとめて出力します。これらのオプションは、出力の一部を HTML 要素にレンダリングしたくない場合に便利です。

```
@If IsPost Then
    ' Plain text followed by an unmatched HTML tag and server code.
    @:The time is: <br /> @DateTime.Now
    ' Server code and then plain text, matched tags, and more text.
    @DateTime.Now @:is the <em>current</em> time.
End If
```

以下の例は、前述の例と同じですが、レンダリングするテキストを囲むために<text>タグを使っています。

```
@If IsPost Then
    @<text>
    The time is: <br /> @DateTime.Now
    @DateTime.Now is the <em>current</em> time.
    </text>
End If
```

以下の例は、含まれないテキスト、不一致の HTML タグ（<br />）、そしてサーバー コードと一致する HTML タグを持つ3行を<text>と</text>タグを使って囲んでいます。個々の行の先頭に@:を付けることもできます。どちらでも動作します。

```
@{
    var minTemp = 75;
    <text>It is the month of @DateTime.Now.ToString("MMMM"), and
    it's a <em>great</em> day! <br /><p>You can go swimming if it's at
    least @minTemp degrees. </p></text>
}
```

**注意**　HTML 要素、@:演算子、<text>要素を使って、このセクションで示したテキストを出力する場合、ASP.NET は出力を HTML エンコードしません。（前述したとおり、ASP.NET は、@が先行するサーバー コード式やサーバー コード ブロックの出力を、このセクションで示した特別な場合を除いてエンコードします。）

### 空白

文中の余分な空白（文字列リテラルの外にあるもの）は、文には影響しません。

```
@Code Dim personName = "Smith" End Code
```

## 長い文を複数の行に分ける

アンダースコア文字（_、Visual Basic では継続文字と呼ばれます）を使うことで、長いコード文を複数の行に分けることができます。文を次の行に分けるには、文の終わりにスペースと継続文字を追加します。文が次の行につながります。読みやすさを改善する必要があれば、文を複数の行に折り返すことができます。以下の文は、同じです。

```
@Code
    Dim familyName _
    = "Smith"
End Code

@Code
    Dim _
    theName _
    = _
    "Smith"
End Code
```

ただし、文字列リテラルの途中で行を折り返すことはできません。以下の例は、正しく動作しません。

```
@Code
    ' Doesn't work.
    Dim test = "This is a long _
        string"
End Code
```

上記のコードのように、長い文字列を複数行に折り返して連結したい場合は、後述の連結演算子（&）を使う必要があります。

## コードのコメント

コメントを使うと、自分自身や他人のためのメモを残しておけます。Razor 構文のコメントは、@*で始まり、*@で終わります。

```
@* A single-line comment is added like this example. *@

@*
    This is a multiline code comment.
    It can continue for any number of lines.
*@
```

コード ブロックの中では、Razor 構文のコメントを使うか、標準的な Visual Basic のコメント文字であるシングルクォート文字を各行の最初で使うことができます。

```
@Code
    ' You can make comments in blocks by just using ' before each line.
```

```
End Code

@Code
    ' There is no multi-line comment character in Visual Basic.
    ' You use a ' before each line you want to comment.
End Code
```

## 変数

変数は、データを保存するために使う名前付きのオブジェクトです。変数にはどんな名前を付けることもできますが、名前は英字で始まり、スペースや予約記号を含むことはできません。Visual Basic では、前述のとおり、変数名の大文字・小文字は区別されません。

### 変数とデータ型

変数は、その変数に保存されたデータの種類で示される、特定のデータ型を持ちます。（"Hello world" のような）文字列値を保持する文字列変数や、（3 や 79 のような）整数値を保持する整数変数、（4/12/2010 や March 2009 のような）さまざまな形式の日付値を保持する日付変数が使えます。このほかにも使えるデータ型があります。ただし、通常は変数の型を指定する必要はありません。ほとんどの場合、ASP.NET が、変数の中のデータの使われ方に基づいて、その型を決定できます。（ときには、本書の例にみられるとおり、型を指定しなければならないこともあります。）

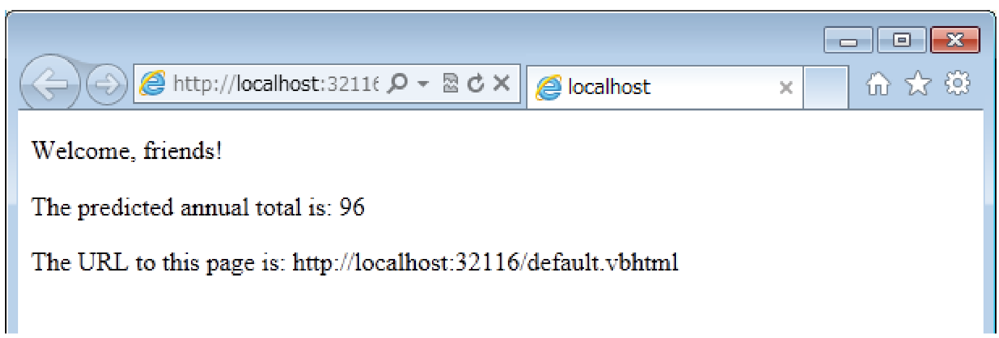
変数を宣言するためには、（型を指定したくない場合は）予約語 Dim と変数名（たとえば、Dim myVar）を使います。変数を型付きで宣言するためには、Dim と変数名に続いて As と型名を使います（たとえば、Dim myVar As String）。

```
@Code
    ' Assigning a string to a variable.
    Dim greeting = "Welcome"

    ' Assigning a number to a variable.
    Dim theCount = 3

    ' Assigning an expression to a variable.
    Dim monthlyTotal = theCount + 5

    ' Assigning a date value to a variable.
    Dim today = DateTime.Today

    ' Assigning the current page's URL to a variable.
    Dim myPath = Request.Url

    ' Declaring variables using explicit data types.
    Dim name as String = "Joe"
    Dim count as Integer = 5
    Dim tomorrow as DateTime = DateTime.Now.AddDays(1)
End Code
```

以下の例は、Web ページにおける典型的な変数の使い方を示しています。

```
@Code
    ' Embedding the value of a variable into HTML markup.
    ' Precede the markup with @ because we are in a code block.
    @<p>@greeting, friends!</p>
End Code

<!-- Using a variable with an inline expression in HTML. -->
<p>The predicted annual total is: @( monthlyTotal * 12)</p>

<!-- Displaying the page URL with a variable. -->
<p>The URL to this page is: @myPath</p>
```

結果は、次のようにブラウザーに表示されます。




## データ型の変換とテスト

通常、ASP.NET はデータ型を自動的に決定できますが、そうでない場合もあります。このような場合、明示的な変換を実行して、ASP.NET を手助けする必要があります。型を変換する必要がない場合でも、どのような型で処理されるかを確認するためにテストすることは役に立ちます。

もっともよくあるケースは、文字列から整数や日付のような他の型への変換しなければならないものです。以下の例は、文字列から数値へ変換しなければならない典型的な場合です。

```
@Code
    Dim total = 0
    Dim totalMessage = ""
    if IsPost Then
        ' Retrieve the numbers that the user entered.
        Dim num1 = Request("text1")
        Dim num2 = Request("text2")
        ' Convert the entered strings into integers numbers and add.
        total = num1.AsInt() + num2.AsInt()
        totalMessage = "Total = " & total
    End If
End Code
```

規則として、ユーザー入力は文字列として渡されます。ユーザーに数値を入力するよう促して、数字が入力された場合でも、ユーザー入力が送られてコード中で読み取る場合は、そのデータは文字列形式です。このため、文字列を

数値に変換しなければなりません。この例で、変換せずに値を計算しようとすると、ASP.NET は 2 つの文字列を加算できないため、以下のようなエラーになります。

```
型 'string' を型 'int' に暗黙的に変換できません。
```

値を整数値に変換するには、AsInt メソッドを呼び出します。変換が成功すれば、値を加算できます。
以下の表は、変数に対する一般的な変換とテスト用メソッドの一覧です。

| **メソッド** | **説明** | **例** |
|---|---|---|
| AsInt(), IsInt() | （"593" のような）整数をあらわす文字列を整数値に変換する。 | `Dim myIntNumber = 0`<br>`Dim myStringNum = "539"`<br>`If myStringNum.IsInt() Then`<br>`    myIntNumber = myStringNum.AsInt()`<br>`End If` |
| AsBool(), IsBool() | "true" や "false" のような文字列を、論理型に変換する。 | `Dim myStringBool = "True"`<br>`Dim myVar = myStringBool.AsBool()` |
| AsFloat(), IsFloat() | "1.3" や "7.439" のような 10 進数をあらわす文字列を浮動小数値に変換する。 | `Dim myStringFloat = "41.432895"`<br>`Dim myFloatNum = myStringFloat.AsFloat()` |
| AsDecimal(), IsDecimal() | "1.3" や "7.439" のような 10 進数をあらわす文字列を decimal 型に変換する。（ASP.NET では、decimal 型は浮動小数型よりも高精度。） | `Dim myStringDec = "10317.425"`<br>`Dim myDecNum = myStringDec.AsDecimal()` |
| AsDateTime(), IsDateTime() | 日付と時間をあらわす文字列を ASP.NET の DateTime 型に変換する。 | `Dim myDateString = "12/27/2010"`<br>`Dim newDate = myDateString.AsDateTime()` |
| ToString() | 文字列以外の型を文字列に変換する。 | `Dim num1 As Integer = 17`<br>`Dim num2 As Integer = 76`<br>`' myString is set to 1776`<br>`Dim myString as String = num1.ToString() & num2.ToString()` |

## 演算子

　演算子は、式の中でどんな種類のコマンドを実行するかを ASP.NET に伝えるための予約語や文字です。Visual Basic 言語（およびそれに基づく Razor 構文）は、多くの演算子をサポートしますが、最初に覚えておく必要があるのはわずかです。以下の表は、よく使われる演算子をまとめたものです。

| 演算子 | 説明 | 例 |
|---|---|---|
| +<br>-<br>*<br>/ | 数式における算術演算子。 | @(5 + 13)<br>Dim netWorth = 150000<br>Dim newTotal = netWorth * 2<br>@(newTotal / 2) |
| = | 代入および等価。文脈によって、右辺の値を左辺のオブジェクトに代入するか、値の等価性を判断する。 | Dim age = 17<br>Dim income = Request("AnnualIncome") |
| <> | 不等。値が等しくなければ true を返す。 | Dim theNum = 13<br>If theNum <> 15 Then<br>    ' Do something.<br>End If |
| <<br>><br><=<br>>= | より小さい、<br>より大きい、<br>より小さいか等しい、<br>より大きいか等しい。 | If 2 < 3 Then<br>    ' Do something.<br>End If<br>Dim currentCount = 12<br>If currentCount >= 12 Then<br>    ' Do something.<br>End If |
| & | 連結、文字列をつなぐ。 | ' The displayed result is "abcdef".<br>@("abc" & "def") |
| +=<br>-= | 変数から値を加算または減算する。 | Dim theCount As Integer = 0<br>theCount += 1 ' Adds 1 to count |
| . | ドット。オブジェクトとプロパティやメソッドを区別するために使う。 | Dim myUrl = Request.Url<br>Dim count = Request("Count").AsInt() |
| () | カッコ。式をグループ化したり、メソッドにパラメーターを渡したり、配列やコレクションのメンバーをアクセスするために使う。 | @(3 + 7)<br>@Request.MapPath(Request.FilePath) |
| Not | 否定。true 値を false に（またはその逆）に逆転させる。通常、false（つまり、true でない）を判断する完結表現として使う。 | Dim taskCompleted As Boolean = False<br>' Processing.<br>If Not taskCompleted Then<br>    ' Continue processing<br>End If |
| AndAlso<br>OrElse | 論理積と論理和。条件を結びつけるために使う。 | Dim myTaskCompleted As Boolean = false<br>Dim totalCount As Integer = 0<br>' Processing.<br>If (Not myTaskCompleted) AndAlso total-Count < 12 Then<br>    ' Continue processing.<br>End If |

## コード中でファイルやフォルダー パスを扱う

しばしば、コード中でファイルやフォルダー パスを使うことがあります。以下は、開発コンピューターにおける Web サイトの物理フォルダーの構造の例です。

```
C:¥WebSites¥MyWebSite
    default.cshtml
    datafile.txt
    ¥images
        Logo.jpg
    ¥styles
        Styles.css
```

Web サーバー上で、Web サイトは、そのサイトの物理フォルダーに対応（マップ）する仮想フォルダー構造を持っています。（仮想パスについて思いつくひとつは、ドメイン名に続く URL の部分を構成するものです。） デフォルトでは、仮想フォルダー名は物理フォルダー名と同じです。仮想ルートは、コンピューターの C:ドライブのルートフォルダーをバックスラッシュ（＼、文字セットによっては円記号の¥）であらわすのと同じように、スラッシュ（/）であらわされます。（仮想フォルダーのパスは、常に/を使います。） 以下に、この構造における Styles.css ファイルの物理パスと仮想パスを示します。

- 物理パス: C:¥WebSites¥MyWebSiteFolder¥styles¥Styles.css
- （仮想ルートパスからの）仮想パス: /styles/Styles.css

コード中でファイルやフォルダーを扱う場合、処理するオブジェクトによって、あるときは物理パスを、あるときは仮想パスを参照する必要があり、ASP.NET はコード中でファイルやフォルダーを処理するための機能として、~演算子、Server.MapPath メソッド、Href メソッドを提供しています。

### ~演算子：仮想ルートを取得する

サーバー コードにおいて、フォルダーやファイルの仮想ルート パスを指定するために、~演算子を使います。これは、Web サイトを異なるフォルダーや場所に移動しても、コード中のパスを変更する必要がないため便利なものです。

```
@Code
    Dim myImagesFolder = "~/images"
    Dim myStyleSheet = "~/styles/StyleSheet.css"
End Code
```

### Server.MapPath メソッド：仮想パスから物理パスへ変換する

Server.MapPath メソッドは、（/default.cshtml のような）仮想パスを（C:¥WebSites¥MyWebSiteFolder¥default.cshtml のような）絶対的な物理パスへ変換します。Web サーバー上でテキストファイルを読み書きする場合など、完全な物理パスを必要とする処理のために、このメソッドを使います（通

常、ホスティング サイトのサーバー上に置かれたサイトの絶対的な物理パスはわかりません）。ファイルやフォルダーの仮想パスを渡すと、物理パスを返します。

```
@Code
    Dim dataFilePath = "~/dataFile.txt"
End Code

<!-- Displays a physical path C:¥Websites¥MyWebSite¥datafile.txt -->
<p>@Server.MapPath(dataFilePath)</p>
```

### Href メソッド: サイト リソースへのパスを作成する

WebPage オブジェクトの Href メソッドは、サーバー コード（~演算子を含む）で作成したパスを、ブラウザーが理解できるパスへ変換します。（~演算子は ASP.NET の演算子なので、ブラウザーは理解できません。） Href メソッドは、イメージ ファイルや他の Web ページ、CSS ファイルのようなリソースへのパスを作成するために使います。たとえば、<img>要素、<link>要素、<a>要素の属性を指定するために、HTML マークアップ中でこのメソッドを使います。

```
@Code
    Dim myImagesFolder = "~/images"
    Dim myStyleSheet = "~/styles/StyleSheet.css"
End Code

<!-- This code creates the path "../images/Logo.jpg" in the src attribute. -->
<img src="@Href(myImagesFolder)/Logo.jpg" />

<!-- This produces the same result, using a path with ~ -->
<img src="@Href("~/images")/Logo.jpg" />

<!-- This creates a link to the CSS file. -->
<link rel="stylesheet" type="text/css" href="@Href(myStyleSheet)" />
```

## 条件文と繰り返し

ASP.NET のサーバーコードは、条件に基づく処理を実行したり、指定された数だけ文を繰り返したりするコード（ループを実行するコード）を書くことができます。

### 条件判定

単純な条件判定には If…Then 文を使い、指定した判定が True か False を返します。

```
@Code
    Dim showToday = True
    If showToday Then
        DateTime.Today
    End If
End Code
```

予約語 If はブロックを開始します。実際の判定（条件）は、予約語 If の後ろに続き、True か False を返します。If 文は Then で終わります。判定が True の場合、If と End If で囲まれた文が実行されます。If 文は、Else ブロックを含むことができ、条件が False の場合にそこに指定された文が実行されます。

```
@Code
    Dim showToday = False
    If showToday Then
        DateTime.Today
    Else
        @<text>Sorry!</text>
    End If
End Code
```

If 文でコード ブロックをはじめるときは、ブロックを囲むために使う Code…End Code 文は必要ありません。たんにブロックに@を追加すれば機能します。If に関するこの動作は、コード ブロックが続く、他の Visual Basic のプログラミング予約語、For、For Each、Do While などでも同じです。

```
@If showToday Then
    DateTime.Today
Else
    @<text>Sorry!</text>
End If
```

一つ以上の ElseIf ブロックを使えば、複数の条件を追加できます。

```
@Code
    Dim theBalance = 4.99
    If theBalance = 0 Then
        @<p>You have a zero balance.</p>
    ElseIf theBalance > 0 AndAlso theBalance <= 5 Then
        ' If the balance is above 0 but less than
        ' or equal to $5, display this message.
        @<p>Your balance of $@theBalance is very low.</p>
    Else
        ' For balances greater than $5, display balance.
        @<p>Your balance is: $@theBalance</p>
    End If
End Code
```

この例では、If ブロックの最初の条件が真でなければ、ElseIf の条件が調べられます。その条件が真なら、ElseIf ブロック中の文が実行されます。どの条件も合わなければ、Else ブロックの文が実行されます。ElseIf ブロックはいくつでも追加でき、さらに「そのほかのすべて」という条件として Else ブロックを追加できます。

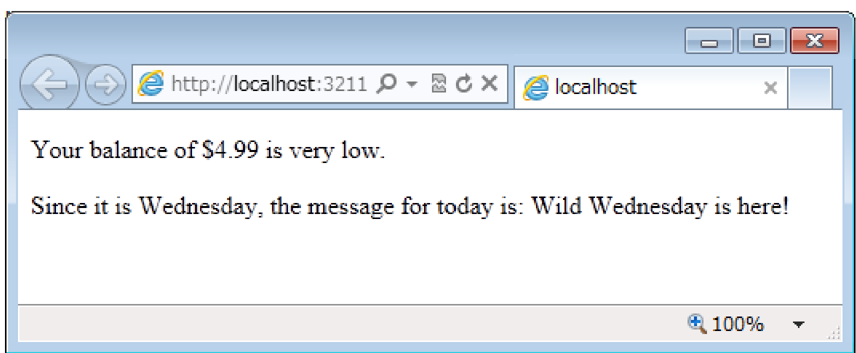
多数の条件を判定するためには、Select Case ブロックを使えます。

```
@Code
    Dim weekday = "Wednesday"
    Dim greeting = ""
```

```
    Select Case weekday
        Case "Monday"
            greeting = "Ok, it's a marvelous Monday."
        Case "Tuesday"
            greeting = "It's a tremendous Tuesday."
        Case "Wednesday"
            greeting = "Wild Wednesday is here!"
        Case Else
            greeting = "It's some other day, oh well."
        End Select
    End Code
<p>Since it is @weekday, the message for today is: @greeting</p>
```

判定する値は、カッコの中にあります（この例では、weekday 変数）。個々の判定値は、Case 文に書きます。テストする値が Case 文の値に一致する場合、その Case ブロックのコードが実行されます。

上記 2 つの条件ブロックは、ブラウザーで次のように表示されます。




## コードのループ

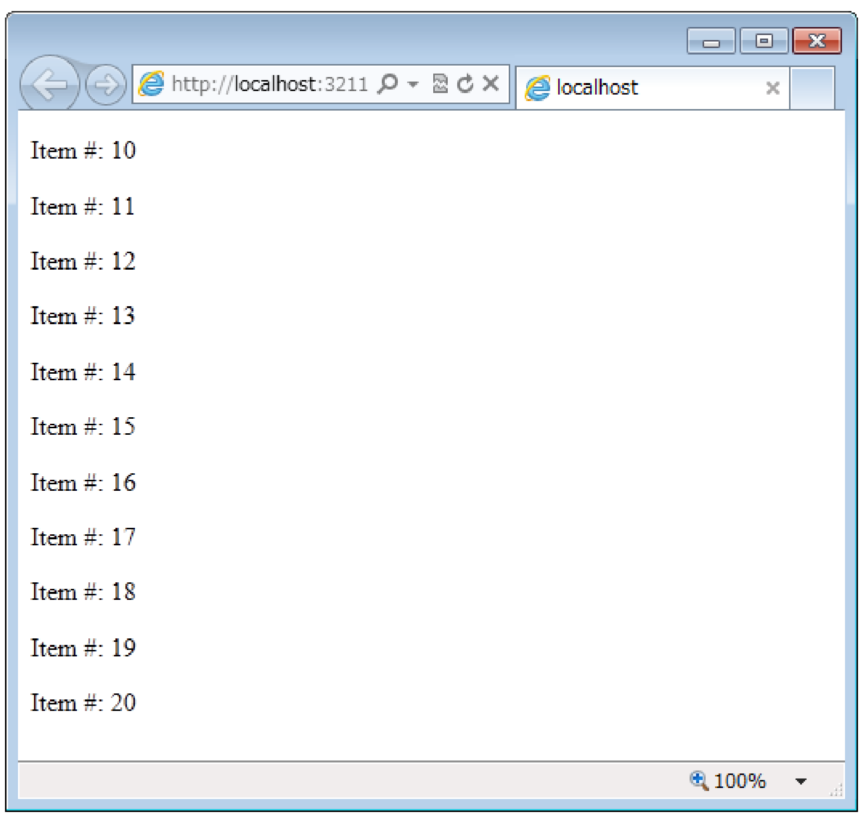
しばしば、同じ文を繰り返し実行する必要が生じます。これはループで実現します。たとえば、データの集まり（コレクション）のそれぞれの項目ごとに同じ文を実行することがあります。ループしたい回数が正確にわかっていれば、For ループを使えます。この種のループは、とくにカウントアップやカウントダウンに役立ちます。

```
@For i = 10 To 20
    @<p>Item #: @i</p>
Next i
```

このループは、予約語 For ではじまり、3 つ要素が続きます。

- For 文の直後では、カウンター用の変数を宣言し（Dim を使う必要はありません）、続いて i = 10 to 20 のように範囲を指定します。これは、変数 i が 10 から始まり、20 になるまで繰り返されることを意味します。
- For と Next 文の間は、ブロックの本体です。繰り返しごとに実行される、ひとつ以上のコード文を含むことができます。
- Next i 文は繰り返しの終わりです。カウンターを 1 加算し、次の繰り返しを始めます。

For 文と Next 文の間のコードは、各繰り返しで実行されるコードです。このマークアップは、新しいパラグラフ（<p>要素）を作成し、毎回出力に、i（カウンター）の値を表示する行を追加しています。このページを実行すると、この例は各行が項目番号を示しているテキストを出力する 11 行を作成します。




コレクションや配列を処理する場合は、たいてい For Each ループを使います。コレクションは、似たようなオブジェクトの集まりであり、For Each ループはそうしたコレクションの個々の要素に対して処理を適用できます。この種のループは、For ループと違ってカウンターをインクリメントしたり、上限を設定する必要がないので、コレクションを扱うのに便利です。その代り、For Each ループではコレクション全体を通じて最後まで順に処理します。

次の例では、Request.ServerVariables コレクション（Web サーバーに関する情報が含まれているオブジェクト）の項目を返します。ここでは For Each ループを使って、それぞれの要素の名前を、HTML の箇条書きリストの中に新しい<li>要素を作成して表示しています。

```
<ul>
@For Each myItem In Request.ServerVariables
    @<li>@myItem</li>
Next myItem
</ul>
```

予約語 For Each に続いてコレクションの 1 つの項目をあらわす変数があり（この例では、myItem）、予約語 In が続き、さらに繰り返しで処理したいコレクションが続きます。For Each ループの本体では、ここで宣言した変数を使って現在の項目にアクセスできます。

http://localhost:3211 localhost

- ALL_HTTP
- ALL_RAW
- APPL_MD_PATH
- APPL_PHYSICAL_PATH
- AUTH_TYPE
- AUTH_USER
- AUTH_PASSWORD
- LOGON_USER
- REMOTE_USER
- CERT_COOKIE
- CERT_FLAGS
- CERT_ISSUER
- CERT_KEYSIZE
- CERT_SECRETKEYSIZE
- CERT_SERIALNUMBER
- CERT_SERVER_ISSUER
- CERT_SERVER_SUBJECT
- CERT_SUBJECT
- CONTENT_LENGTH
- CONTENT_TYPE
- GATEWAY_INTERFACE
- HTTPS

より汎用的なループを作成するためには、Do While 文が使えます。

```
@Code
    Dim countNum = 0
    Do While countNum < 50
        countNum += 1
        @<p>Line #@countNum: </p>
    Loop
End Code
```

このループは、予約語 Do While ではじまり、条件が続き、その後ろのブロックが繰り返されます。ループは、通常、カウント用の変数やオブジェクトをインクリメント（加算）またはデクリメント（減算）します。この例では、+=演算子がループを実行するたびに countNum に 1 加算しています。（カウントダウン用にループ中で変数をデクリメントするときは、デクリメント演算子-=を使います。）

## オブジェクトとコレクション

ASP.NET Web サイトのほとんどすべてのものは、Web ページ自身を含めてオブジェクトです。このセクションでは、コード中でしばしば処理する重要なオブジェクトについて説明します。

### ページ オブジェクト

ASP.NET におけるもっとも基本的なオブジェクトは、ページです。ページ オブジェクトは、特別なオブジェクトの限定をすることなく、プロパティにアクセスできます。次のコードは、ページの Request オブジェクトを使って、そのページのファイル パスを取得しています。

```
@Code
    Dim path = Request.FilePath
End Code
```

Page オブジェクトのプロパティを使って、以下のような多くの情報を取得できます。

- Request。すでにご覧のとおり、これは現在のリクエストに関する情報のコレクションで、リクエストを発生したブラウザーの種類、ページの URL、ユーザー アイデンティティなどを含みます。
- Response。これは、サーバー コードの実行が完了したときにブラウザーに送出されるレスポンス（ページ）に関する情報のコレクションです。たとえば、このプロパティを使ってレスポンスへの情報を書き込むことができます。

```
@Code
    ' Access the page's Request object to retrieve the URL.
    Dim pageUrl = Request.Url
End Code
    <a href="@pageUrl">My page</a>
```

## コレクション オブジェクト（配列とディクショナリ）

コレクションは、同じ型のオブジェクトの集まりで、データベースの Customer オブジェクトのコレクションのようなものを言います。ASP.NET は、Request.Files コレクションのような多くのビルトイン コレクションを持っています。

しばしばコレクションのデータを扱います。2 つの汎用的なコレクションの種類に、配列とディクショナリがあります。配列は、同じ項目のコレクションを保持したい場合、かつそれぞれの項目のために異なる変数を作りたくない場合に役立ちます。

```
<h3>Team Members</h3>
@Code
    Dim teamMembers() As String = {"Matt", "Joanne", "Robert", "Nancy"}
    For Each name In teamMembers
        @<p>@name</p>
    Next name
End Code
```

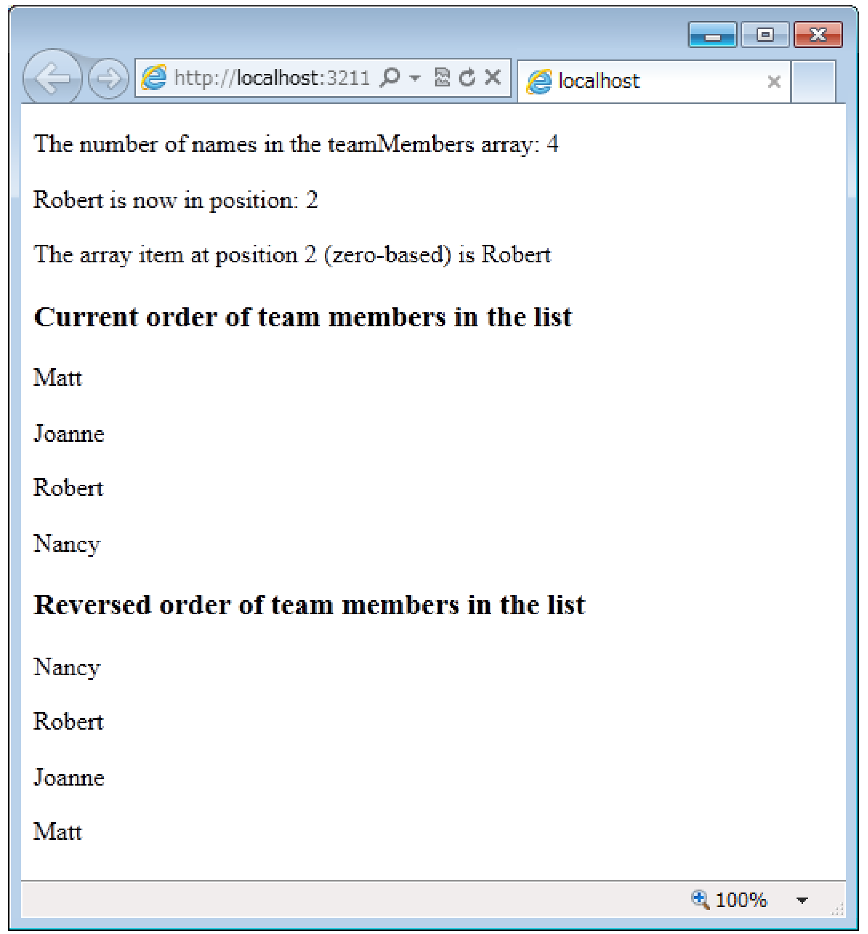
配列を使って、String や Integer、DateTime のような指定した型を宣言できます。配列を含む変数を指定する場合、宣言には（Dim myVar() As String のように）変数名にカッコを追加します。配列の要素にアクセスするときは、その位置（インデックス）を使うか、For Each 文を使います。配列のインデックスはゼロ ベース、つまり先頭の要素は 0 番目の位置、2 番目の要素は 1 番目の位置にあります。

```
@Code
    Dim teamMembers() As String = {"Matt", "Joanne", "Robert", "Nancy"}
    @<p>The number of names in the teamMembers array: @teamMembers.Length </p>
    @<p>Robert is now in position: @Array.IndexOf(teamMembers, "Robert")</p>
    @<p>The array item at position 2 (zero-based) is @teamMembers(2)</p>
```

```
    @<h3>Current order of team members in the list</h3>
    For Each name In teamMembers
        @<p>@name</p>
    Next name
    @<h3>Reversed order of team members in the list</h3>
    Array.Reverse(teamMembers)
    For Each reversedItem In teamMembers
    @<p>@reversedItem</p>
    Next reversedItem
End Code
```

配列の要素の数は、その Length プロパティを取得することでわかります。配列中の指定した要素の場所を得るには（配列の検索）、Array.IndexOf メソッドを使います。また、配列の要素を逆順にして同じことをすることも（Array.Reverse メソッド）、コンテンツをソートすることも（Array.Sort メソッド）できます。

文字列配列のコードの出力は、ブラウザーに次のように表示されます。




ディクショナリは、キー/値の組み合わせのコレクションで、設定用のキー（または名前）を指定したり、対応する値を取り出すことができます。

```
@Code
    Dim myScores = New Dictionary(Of String, Integer)()
    myScores.Add("test1", 71)
    myScores.Add("test2", 82)
    myScores.Add("test3", 100)
```

```
        myScores.Add("test4", 59)
    End Code
    <p>My score on test 3 is: @myScores("test3")%</p>
    @Code
        myScores("test4") = 79
    End Code
    <p>My corrected score on test 4 is: @myScores("test4")%</p>
```

ディクショナリを作成するためには、予約語 new を使って、新たに作成するディクショナリ オブジェクトを指定します。ディクショナリは、予約語 var を使って変数に代入できます。ディクショナリの項目のデータ型は、カッコ（( )）を使って指定します。宣言の最後には、カッコのペアを追加しなければなりません。これが新たなディクショナリを作るためのメソッドです。

ディクショナリに新たな項目を追加するには、ディクショナリ変数（ここでは myScores）の Add メソッドを呼び出して、キーと値を指定します。あるいは、キーを指定する角カッコを使い、以下の例のような単純な代入文としても処理できます。

```
    @Code
        myScores("test4") = 79
    End Code
```

ディクショナリから値を取り出すには、カッコにキーを指定します。

```
    @myScores("test4")
```

## パラメーター付きでメソッドを呼び出す

本章で前述したように、プログラムで使うオブジェクトはメソッドを持てます。たとえば、Database オブジェクトは Database.Connect メソッドを持ちます。メソッドの多くは、1 つ以上のパラメーターを持ちます。パラメーターは、メソッドが処理を完了できるように渡される値です。たとえば、Request.MapPath メソッドの宣言を見ると、どの Web ページのパスを使うかがわかります。たとえば、Request.MapPath メソッドの宣言を見ると、次のように 3 つのパラメーターを受け取ります。

```
    Public Overridable Function MapPath (virtualPath As String, _
        baseVirtualDir As String, _
        allowCrossAppMapping As Boolean)
```

このメソッドは、指定された仮想パスに対応するサーバー上の物理パスを返します。このメソッドの 3 つのパラメーターは、VirtualPath、baseVirtualDir、allowCrossAppMapping です。（宣言においては、パラメーターは受け取るデータのデータ型付きで並べられていることに注意してください。） このメソッドを呼び出すときには、3 つすべてのパラメーターを渡さなければなりません。

Razor 構文を使う Visual Basic では、メソッドにパラメーターを渡すために、位置パラメーターと名前パラメーターの 2 つの選択肢を提供しています。位置パラメーターを使ってメソッドを呼び出すには、メソッド宣言で指定された正確な順番をパラメーターに渡します。（この順序を知るには、通常メソッドのドキュメントを読みます。） 順

序にしたがうためには、どのパラメーターもスキップできません。必要に応じて、値を持たない位置パラメーターには空文字列（""）か null を渡します。

以下の例は、Web サイトに scripts という名前のフォルダーがあることを想定しています。コードは、Request.MapPath メソッドを呼び出し、指定された順序で 3 つのパラメーターに値を渡しています。これで、結果としてマップされたパスを表示します。

```
@Code
    ' Pass parameters to a method using positional parameters.
    Dim myPathPositional = Request.MapPath("/scripts", "/", true)
End Code
<p>@myPathPositional</p>
```

メソッドが多くのパラメーターを持つ場合は、名前パラメーターを使うことでコードの可読性を維持できます。名前パラメーターを使ってメソッドを呼び出すには、パラメーター名に続いてコロン（:）と値を指定します。名前パラメーターの利点は、好きな順序で値を渡せることです。（不利な点は、メソッド呼び出しが小さく収まらないことです。）

以下の例は、上記と同じメソッドを呼び出していますが、名前パラメーターをつかって値を渡しています。

```
@Code
    ' Pass parameters to a method using named parameters.
    Dim myPathNamed = Request.MapPath(baseVirtualDir:= "/", allowCrossAppMapping:=
true, virtualPath:= "/scripts")
End Code
<p>@myPathNamed</p>
```

ご覧のとおり、パラメーターは異なる順序で渡されています。しかし、前述の例もこの例も、同じ値を返します。

## エラー処理

### Try-Catch 文

コードの中の文には、しばしば制御できない理由によって失敗する者が含まれます。たとえば、次のようなものです。

- コードがファイルを開いたり、作成したり、書き込む場合、さまざまな種類のエラーが発生するおそれがあります。ファイルが存在しない、ロックされている、コードが権限を持っていない、など。
- 同様に、コードがデータベースのレコードを更新しようとすると、権限の問題が起きるおそれがあります。データベースへの接続が落ちてしまったり、保存するデータが不正な場合、など。

プログラミング用語では、こうした状況を「例外」と呼びます。コードが例外に遭遇すると、エラーメッセージを生成（スロー）します。これは、よくても、ユーザーをいらだたせるものです。

コードがこのような状況に遭遇した場合、この種のエラーメッセージを避けるために、Try/Catch 文を使えます。Try 文では、チェックしたいコードを実行します。1 つ以上の Catch 文では、発生したものが指定したエラー（指定した例外の型）かどうかを調べます。ここでは、予想できるだけエラーに対応する Catch 文を追加できます。

**注意**　Try/Catch 文で Response.Redirect メソッドを使うことは避けることをお勧めします。ページ中で例外を起こす恐れがあります。

以下の例は、最初のリクエストでテキストファイルを作成し、ユーザーにファイルを開くボタンを表示します。この例は、意図的に誤ったファイル名を使っているため、例外を発生します。起きうる 2 つの例外のためのコードは Catch 文を含んでいます。ひとつはファイル名が間違っている場合の FileNotFoundException で、もうひとつは ASP.NET がフォルダーを発見できない場合に発生する DirectoryNotFoundException です。（すべてが正常に動作するようすを確認するために、この例のコメントを外すこともできます。）

コードが例外を処理しないと、さきほどの画面ショットのようなエラー ページが表示されます。しかし、Try/Catch セクションは、この種のエラーが発生しても、そのような画面が表示されることを防ぎます。

```
@Code
    Dim dataFilePath = "~/dataFile.txt"
    Dim fileContents = ""
    Dim physicalPath = Server.MapPath(dataFilePath)
    Dim userMessage = "Hello world, the time is " + DateTime.Now
    Dim userErrMsg = ""
    Dim errMsg = ""
    If IsPost Then
        ' When the user clicks the "Open File" button and posts
        ' the page, try to open the file.
        Try
            ' This code fails because of faulty path to the file.
            fileContents = File.ReadAllText("c:¥batafile.txt")
```

```
            ' This code works. To eliminate error on page,
            ' comment the above line of code and uncomment this one.
            ' fileContents = File.ReadAllText(physicalPath)

        Catch ex As FileNotFoundException
            ' You can use the exception object for debugging, logging, etc.
            errMsg = ex.Message
            ' Create a friendly error message for users.
            userErrMsg = "The file could not be opened, please contact " _
                & "your system administrator."
        Catch ex As DirectoryNotFoundException
            ' Similar to previous exception.
            errMsg = ex.Message
            userErrMsg = "The file could not be opened, please contact " _
                & "your system administrator."
        End Try
    Else
        ' The first time the page is requested, create the text file.
        File.WriteAllText(physicalPath, userMessage)
    End If
End Code
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Try-Catch Statements</title>
    </head>
    <body>
    <form method="POST" action="" >
        <input type="Submit" name="Submit" value="Open File"/>
    </form>

    <p>@fileContents</p>
    <p>@userErrMsg</p>

    </body>
</html>
```

## その他のリソース

- ASP.NET
- Visual Basic 言語

# 付録 – Visual Studio における ASP.NET Web ページのプログラミング

この付録は、Razor 構文を使う ASP.NET Web ページをプログラムするために、どのように Visual Studio 2010 や Visual Web Developer 2010 Express を使うかを示します。

ここでは次のことを学びます。

- Visual Web Developer 2010 Express および ASP.NET Razor Tools（ASP.NET MVC3 RTM リリースを含む）のインストール方法
- IntelliSense とデバッガーを含む、ASP.NET Razor ページを扱う Visual Studio の機能の使い方

## なぜ Visual Studio を使うのか

WebMatrix や他のコード エディターを使って、Razor 構文を使う ASP.NET Web ページをプログラムできます。（Web サイトに限らず）多様なアプリケーションを作成する強力なツール セットを持つ包括的な機能を備えた統合開発環境である Microsoft Visual Studio 2010 を使うこともできます。ASP.NET Razor ページを扱うには、Visual Studio のいずれかのエディションや無料の Visual Web Developer が使えます。

Visual Studio には、ASP.NET Razor Web ページのプログラミングにおいて特に役立つ 2 つの機能があります。

- **IntelliSense**。これは、エディター中で文を補完したり、扱いたいクラスやメソッドについての情報を一覧表示したりすることで、プログラミングの生産性を改善します。（Webmatrix は、HTML や CSS のようないくつかのプログラミング要素についての IntelliSense を備えていますが、C#や Visual Basic のプログラミング コードでは使えません。）
- **デバッガー**。デバッガーを使うことで、実行中のプログラムを停止させたり、変数を調べたり、コードを 1 行 1 行実行することで、コードの問題に対処できます。

現在、これらの機能は Visual Studio だけで使えるものです。

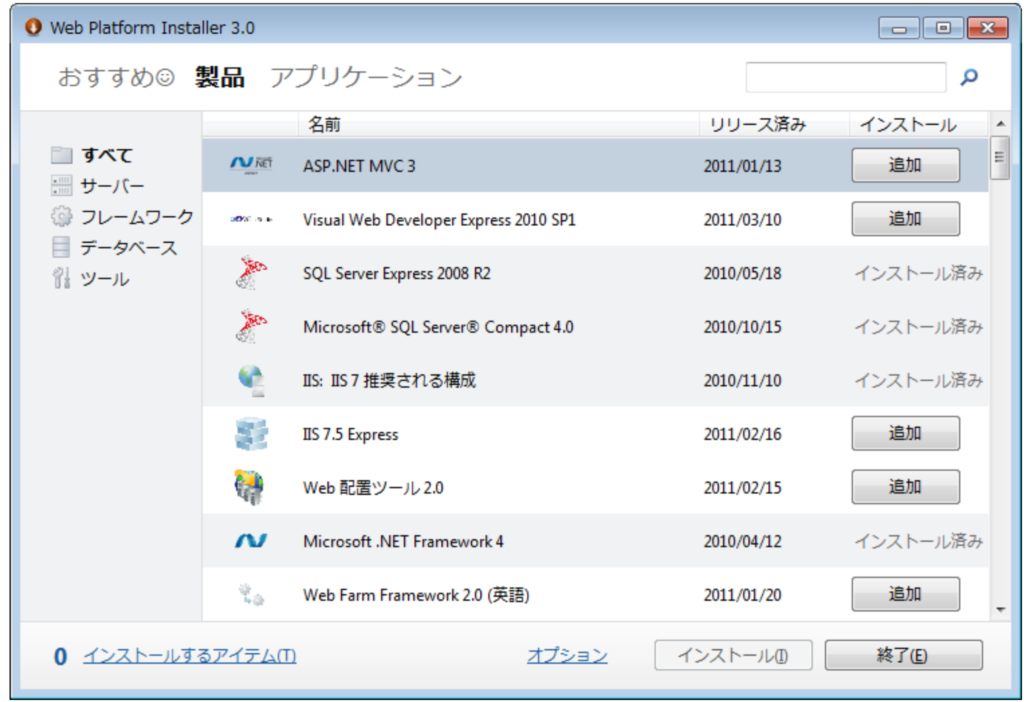
## ASP.NET Razor Tools のインストール

このセクションでは、無料の Visual Web Developer 2010 Express Edition と ASP.NET Web Pages Tools for Visual Studio のインストール方法について示します。

1. まだ Web Platform Installer を持っていなければ、以下の URL からダウンロードします。

   http://www.microsoft.com/web/downloads/platform.aspx

2. Web Platform Installer を実行し、「製品」を選び、ASP.NET MVC 3 を見つけて、[追加] をクリックします。この製品は、ASP.NET Razor Web サイトを構築するための Visual Studio Tools を含みます。

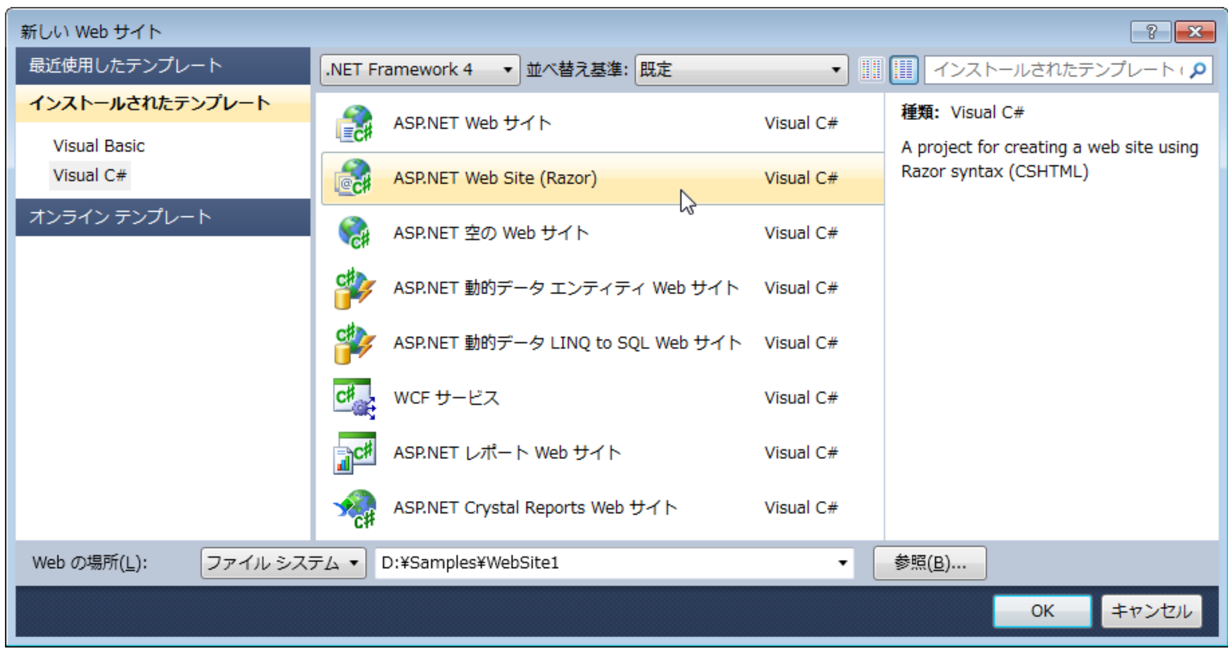



3. Visual Studio または Visual Web Developer Express をインストールしていない場合は、[Visual Web Developer Express 2010]を見つけて、[追加] ボタンをクリックします。
4. [インストール] をクリックして、インストールを完了させます。

## Visual Studio 用 ASP.NET Razor Tools を使う

IntelliSense やデバッガーを使うために、Visual Studio で ASP.NET Razor Web サイトを作成します。

1. Visual Studio または Visual Web Developer を起動します。
2. [ファイル] メニューで、[新しい Web サイト] をクリックします。
3. [新しい Web サイト] ダイアログ ボックスで、使う言語を選びます（Visual C#または Visual Basic）。
4. [ASP.NET Web Site (Razor)] テンプレートを選びます。
5. 「Web の場所」の隣にあるドロップダウン リストで「ファイル システム」を選び、パスとして、ローカル フォルダーを入力します。

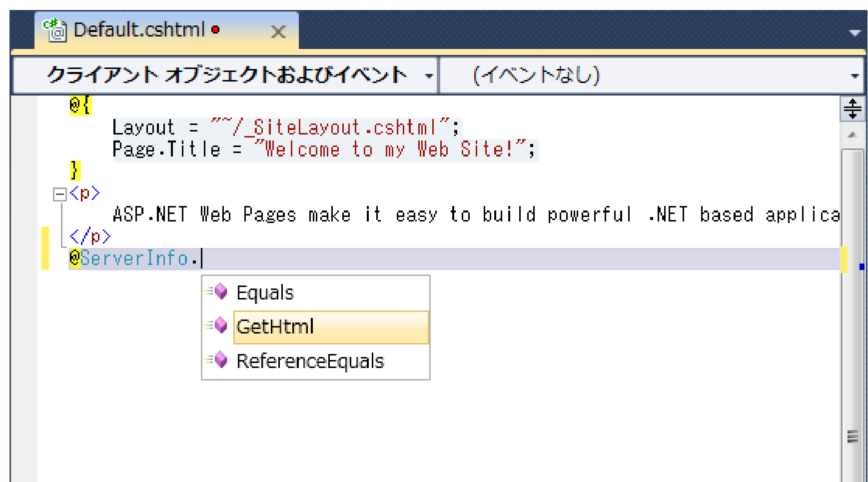



6. [OK] をクリックします。

## IntelliSense を使う

これでサイトが作成できたので、Visual Studio でどのように IntelliSense が機能するかを確認できます。

1. 作成した Web サイトで、Default.cshtml ページを開きます。ウィンドウの下部で、[ソース] タブが選ばれていることを確認します。
2. ページ中の閉じる </p> タグの直後に、@ServerInfo. と入力します（ピリオド記号を含む）。すると、IntelliSense が ServerInfo ヘルパーで使えるメソッドをどのようにドロップダウン リストに表示するかがわかります。




3. リストから GetHtml メソッドを選び、[Enter] を押します。IntelliSense は、自動的にメソッド名を挿入します。（C#の多くのメソッドと同じく、メソッドの直後に () 文字を追加しなければなりません。）

GetHtml メソッドの最終的なコードは、次の例のようになります。

```
@Server.GetHtml()
```

4. [Ctrl]+[F5] を押して、このページを実行します。ブラウザーには、このページは次のように表示されます。

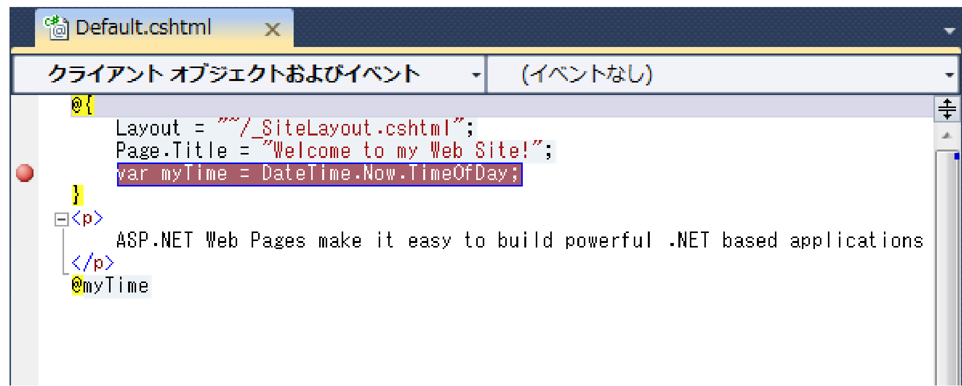
5. ブラウザーを閉じ、更新した Default.cshtml ページを保存します。

## デバッガーを使う

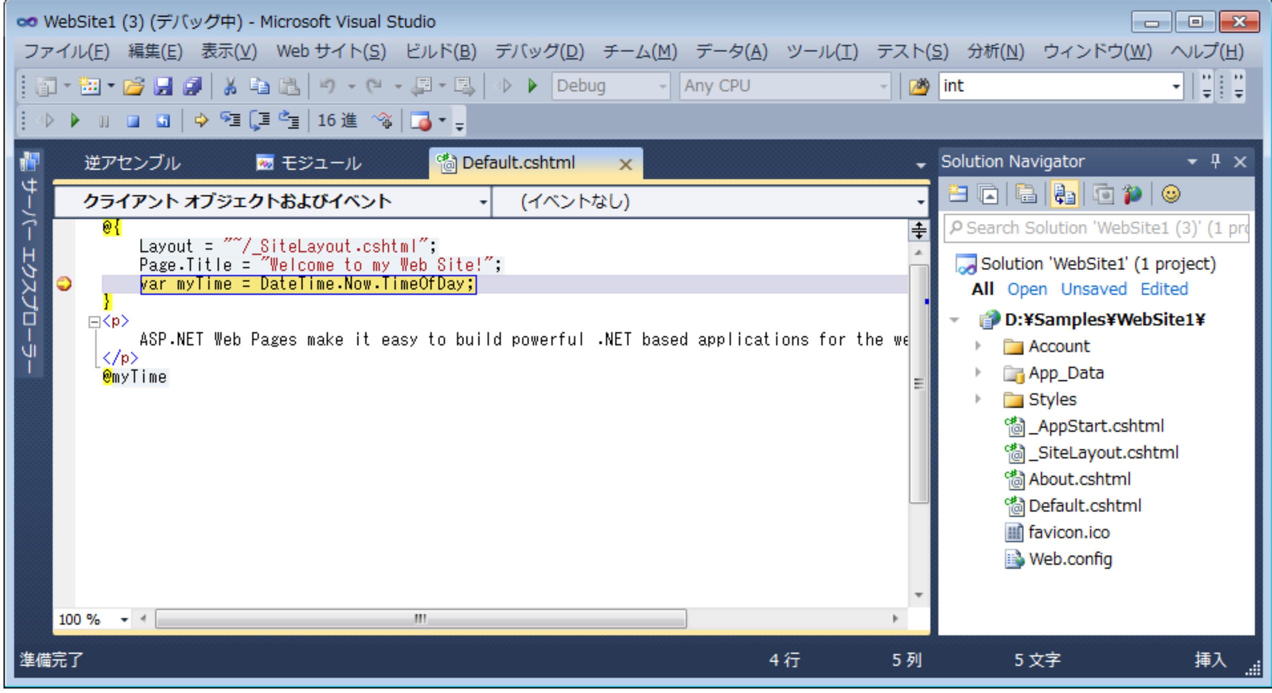
1. Default.cshtml ページの先頭で、Page.Title で始まる行の下に、次のようなコード行を追加します。

```
var myTime = DateTime.Now.TimeOfDay;
```

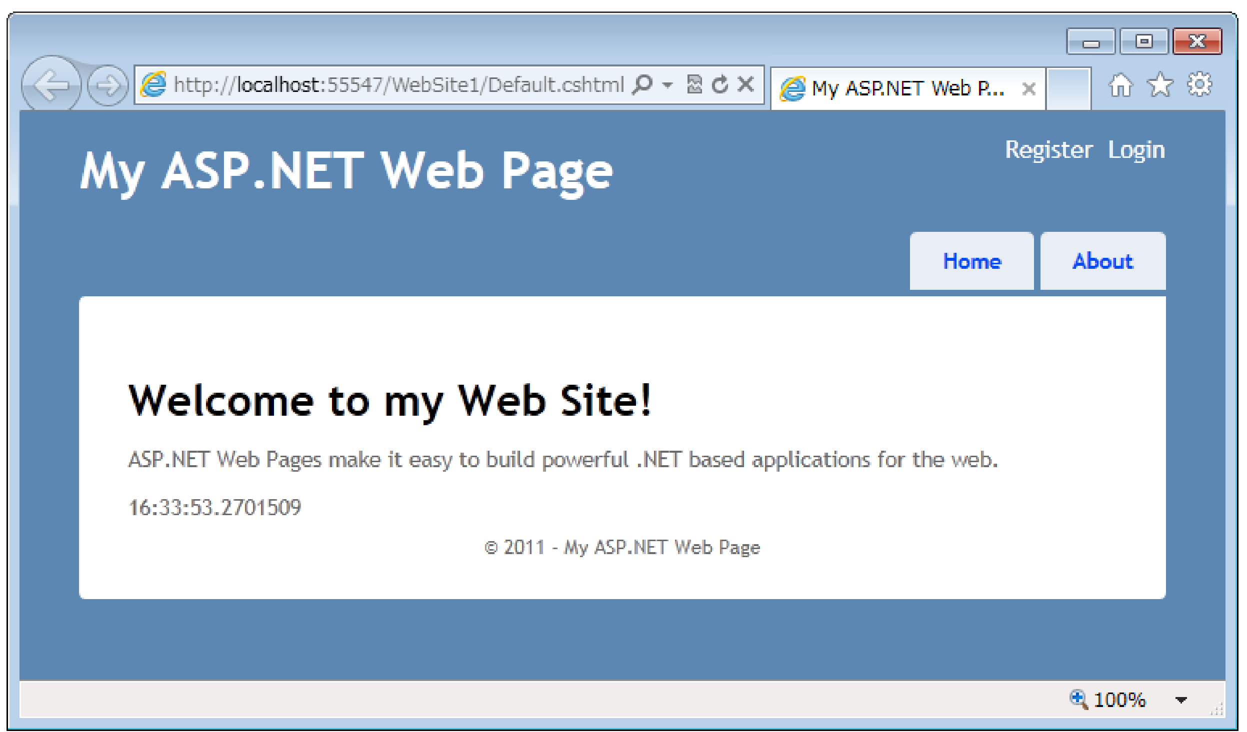
2. コードの左側にある灰色の余白部分で、この行をクリックしてブレークポイントを追加します。ブレークポイントは、プログラムの実行中に何が起きているかを確認したい場所で、一時的に停止させることをデバッガーに伝えます。
3. ServerInfo.GetHtml メソッドの呼び出しを削除し、同じ場所に@myTime 変数の呼び出しを追加します。この呼び出しは、新たに追加したコードが返す現在時刻の値を表示します。
   2 行のコードとブレークポイントが更新されたページは、次のようになります。




4. [F5]を押して、このページをデバッガーで実行します。ページは、設定したブレークポイントで停止します。エディターでは、以下のイメージが示すように（黄色い）ブレークポイントが表示されます。




5. [ステップ イン] ボタン（または [F11]）をクリックします。これでコードの次の行まで実行します。再び [F11] を押すと、次の実行可能行まで実行され、これが繰り返されます。
6. マウス ポインターを myTime 変数上で止めておくか、[ローカル] や [呼び出し履歴] ウィンドウで表示させることで、値を確認できます。
7. 変数のテストやコードのステップ実行が終わったら、[F5] を押して停止させることなく実行します。これで、ページは次のようにブラウザーに表示されます。




デバッガーについての詳細や、Visual Studio におけるデバッグ手法については、「チュートリアル：Visual Web Developer での Web ページのデバッグ」を参照してください。

## 免責

免責事項: このドキュメントの内容は情報提供のみを目的としており、明示または黙示に関わらず、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。このドキュメントに記載されている情報 (URL 等のインターネット Web サイトに関する情報を含む) は、将来予告なしに変更することがあります。お客様がこのドキュメントを運用した結果の影響については、お客様が負うものとします。別途記載されていない場合、このドキュメントで例として挙げられている企業、組織、製品、ドメイン名、電子メール アドレス、ロゴ、人物、地名、およびイベントは、架空のものです。それらが、いずれかの実際の企業、組織、製品、ドメイン名、電子メール アドレス、ロゴ、人物、地名、あるいはイベントを指していることはなく、そのように解釈されるべきではありません。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。

© 2011 Microsoft Corporation. All Rights Reserved.

Microsoft is a trademark of the Microsoft group of companies.  All other trademarks are property of their respective owners.