第 4 章

ASP.NET 的 Web 页面语法

ASP.NET 使用以 .aspx 作为后缀的网页，这种网页又称为 ASPX 页或者 Web 窗体页。.NET Framework 中的 Page 类是所有 ASPX 页的基类，也就是说，每个 Web 窗体都是 Page 类的实例。本章简单了解 ASP.NET Web 窗体的结构，包括页面运行机制和常用指令等，在介绍 Web 窗体页之前，将分别创建 Web 窗体应用程序和网站，并比较它们之间的异同点。

本章学习要点

- ◎ 掌握 Web 应用程序的创建
- ◎ 掌握 Web 网站的创建
- ◎ 熟悉 Web 应用程序与网站的异同点
- ◎ 了解 Web 窗体页的特点
- ◎ 熟悉 Web 窗体页的元素
- ◎ 了解 Web 窗体页的运行过程
- ◎ 掌握 @Page 和 @Control 指令
- ◎ 掌握 @Register 和 @Master 指令
- ◎ 了解 ASP.NET 的其他页面指令

扫一扫，下载
本章视频文件

4.1 Web 应用程序和网站

C/S 和 B/S 是应用程序的两种模式，C/S 是客户端 / 服务器端程序，而 B/S 是浏览器端 / 服务器端应用程序，这类应用程序一般借助于 IE、Chrome 和 Firefox 等浏览器来运行。Web 应用程序一般是 B/S 模式，它是基于 Web 的，而不是采用传统方法运行的。简单地说，Web 应用程序是典型的浏览器 / 服务器架构的产物。

4.1.1 Web 应用程序

在第 3 章已经介绍过如何创建一个基于窗体的 Web 应用程序，基于窗体的 Web 应用程序创建完毕后，会自动生成一些目录和文件。如图 4-1 所示为基于窗体的 Web 应用程序。

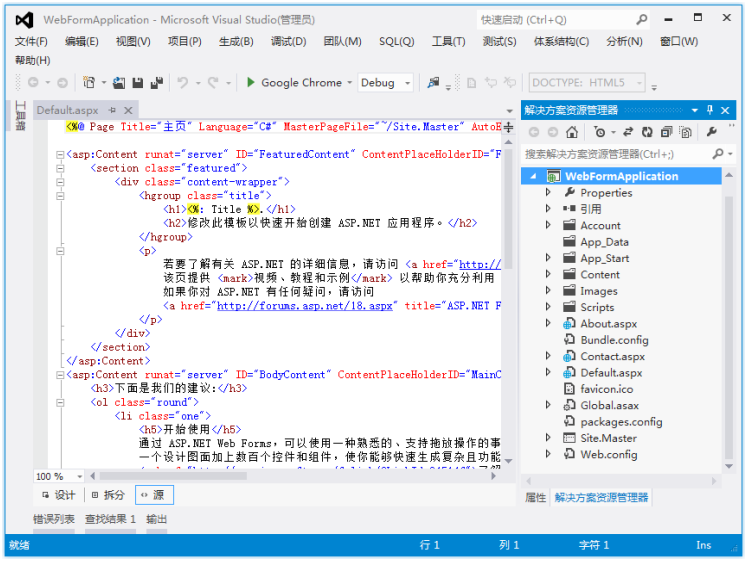

图 4-1 基于窗体的 Web 应用程序

在图 4-1 所示界面中，包含多个目录和文件，常用的目录和文件说明如下。

- Properties 目录：该目录中包含一个 AssemblyInfo.cs 文件，这是一个包含程序版本、版权等信息的属性文件。
- Account 目录：该目录包含多个 Web 窗体，这是基于窗体创建 Web 应用程序时生成的一个目录，包含用户登录和注册等页面。
- App_Data 目录：包含 Microsoft Office Access 和 SQL Expression 文件以及 XML 文件或者其他数据存储文件。
- Images 目录：包含图像文件。
- Scripts 目录：包含脚本文件。
- Global.asax 文件：这是一个可选文件，通常被称为 ASP.NET 应用程序文件。该文件包含响应 ASP.NET 或 HTTP 模块所引发的应用程序级别和会话级别事件的代码。如果文件不存在，进行创建时，必须将其放在应用程序的根目录下。
- Web.config 文件：该文件用来存储 ASP.NET Web 应用程序的配置信息，这是一个 XML 文件。

在图 4-1 所示的界面中，Content、Images、Scripts、Abount.aspx 等目录和文件都是基于窗体的 Web 应用程序生成的。开发者也可以创建不带窗体的 Web 应用程序，创建时只需要在弹出的对话框中选择【ASP.NET 空 Web 应用程序】选项即可。如图 4-2 所示为创建空 Web 应用程序时的结构。

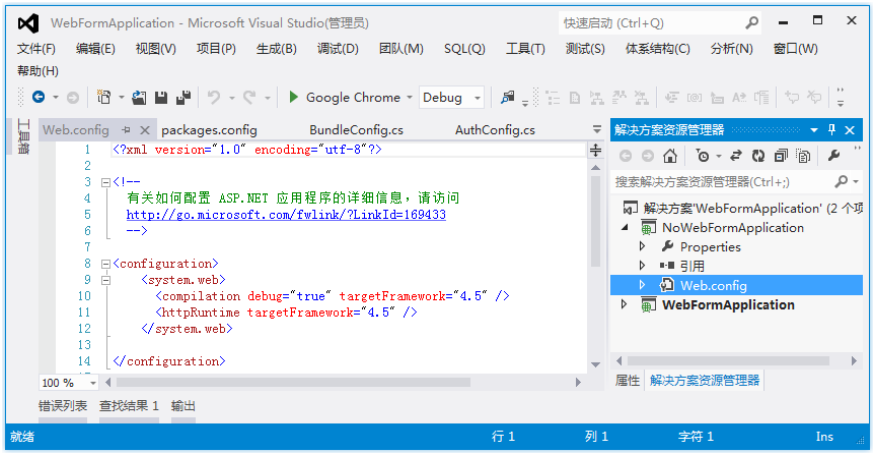

图 4-2　不带窗体的 Web 应用程序

比较图 4-1 和图 4-2 可以看出，不带窗体的 Web 应用程序很简单，只包含 Properties 目录、引用文件目录和 Web.config 文件。

4.1.2　Web 网站

开发者可以通过创建 Web 窗体应用程序的方式创建 Web 程序，还可以通过创建 Web 网站的方式创建 Web 程序。选择【文件】|【新建】|【网站】菜单命令打开【新建网站】对话框，如图 4-3 所示。

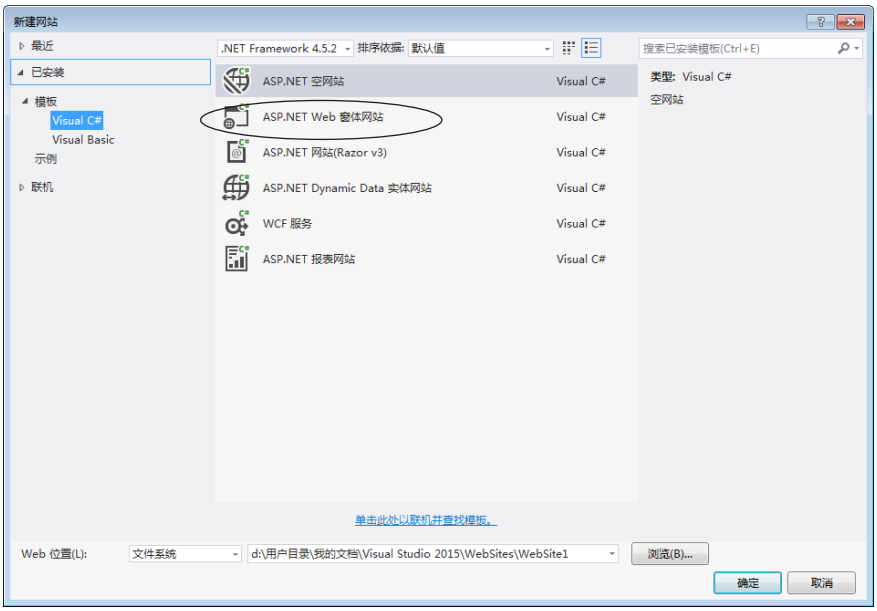

图 4-3　【新建网站】对话框

在如图 4-3 所示的对话框中，选择【ASP.NET Web 窗体网站】后，输入或选择网站位置，然后单击【确定】按钮，即可创建基于窗体的 Web 网站，结果如图 4-4 所示。

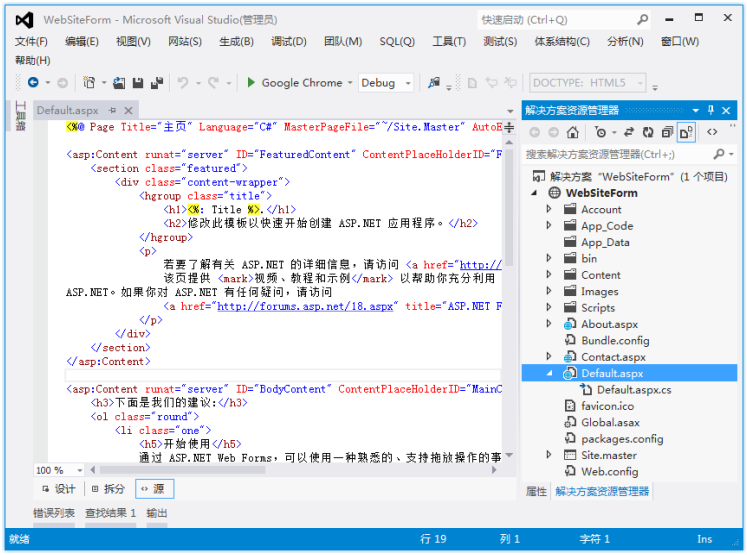

图 4-4　基于窗体的 Web 网站

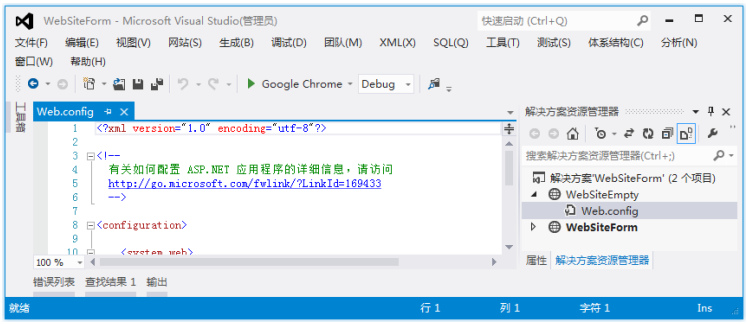

图 4-5　不带窗体的 Web 网站的结构

由图 4-4 所示的界面可知，基于窗体的 Web 网站也会生成多个目录和文件，其中许多目录和文件的说明都与基于窗体的 Web 应用程序相似，这里只介绍 App_Code 和 bin 目录。

- App_Code 目录：包含作为应用程序一部分编译的类的源文件。当页面被请求时，ASP.NET 编译该目录中的代码，该目录中的代码在应用程序中自动地被引用。
- bin 目录：包含应用程序所需的任何预生成的程序集。

开发者也可以创建不带窗体的 Web 网站，在图 4-3 所示的对话框中选择【ASP.NET 空网站】即可。图 4-5 为不带窗体的 Web 网站的结构。从图 4-5 中可以看出，创建空网站时，只生成一个 Web.config 文件。

4.1.3　比较 Web 应用程序和 Web 网站

VS 2012 中既可以创建 Web 应用程序，也可以创建 Web 网站。一般来说，Web 应用程序适合相对较大的系统，而 Web 网站比较适合中小型企业网站。下面分别从相同点和不同点进行说明。

1. 相同点

Web 应用程序和 Web 网站有两个相同点：都用来设计和实现 ASP.NET 网页；都可以添加 ASP.NET 文件夹，如都包括 App_Browsers、App_Data、App_GlobalResources、App_LocalResources 和 App_Themes。

2. 不同点

Web 应用程序和 Web 网站存在多个不同点，说明如下：

- Web 应用程序 Default.aspx 显示有两个原有文件，分别指 Default.aspx.cs 和 Default.aspx.designer.cs；Web 网站 Default.aspx 有一个原有文件，即 Default.aspx.cs。

- Web 应用程序有重新生成和发布网站两项；Web 网站只有发布网站一项。
- Web 应用程序和一般的 WinForm 没有什么区别，如都引用的是命名空间；Web 网站在引用后出现一个 bin 目录，该目录存在后缀名为“.dll”和“.pdb”的文件。
- Web 应用程序可以作为类库被引用；Web 网站则不可以作为类库被引用。
- Web 应用程序可以添加 ASP.NET 文件夹，但是不包括 bin 和 App_Code；Web 网站可以添加 ASP.NET 文件夹，但是包括 bin 和 App_Code。
- Web 应用程序可以添加组件和类；Web 网站则不能。
- 源文件虽然都是Default.aspx.cs文件，但是 Web 应用程序多了对 System.Collections 命名空间的引用。

4.2　Web 窗体页

无论是创建 Web 应用程序还是 Web 网站，都可以为其添加 Web 窗体页。使用 Web 窗体页可以创建可编程的 Web 页面，这些 Web 页面用作 Web 应用程序的用户界面。

Web 窗体页可以在任何浏览器或者客户端设备中向用户提供信息，并使用服务器端代码来实现应用程序逻辑。Web 窗体页几乎可以包含任何支持 HTTP 的语言，如 HTML、XML、WML、JScript 和 JavaScript 等。

4.2.1　Web 窗体页的特点

Web 窗体页基于 ASP.NET 技术，它是创建 ASP.NET 网站和 Web 应用程序编程模式常用的一种。Web 窗体页是整合了 HTML、服务器控件和服务器代码的事件驱动网页，特点如下：

- 兼容所有的浏览器和设备。Web 窗体页自动为样式和布局等功能呈现正确的、符合浏览器的 HTML。开发者还可以选择将 Web 窗体页设计为在特定浏览器 (如 IE 5) 上运行并利用多样式浏览器客户端的功能。
- 兼容 .NET 公共语言运行时所支持的任何语言，其中包括 Visual Basic、C# 和 JScript。
- 基于 Microsoft .NET Framework 生成，它提供了该框架的所有优点，包括托管环境、类型安全性和继承。
- 在 VS 中为快速应用程序开发提供支持，该工具用于对窗体进行设计和编程。
- 可使用为 Web 开发提供应用程序功能的控件进行扩展，从而使开发者能够快速地创建多样式的用户界面。
- 具有灵活性，开发者可以添加用户创建的控件和第三方控件。

4.2.2　Web 窗体页的元素

在 Web 窗体页中，用户界面编程被分为两个不同的部分：可视元素和逻辑。可视元素称作 Web 窗体“页”，这种页通常由一个包含静态 HTML 和 ASP.NET 服务器控件的文件组成。Web 窗体页的逻辑由代码组成，这些代码由开发者创建，与窗体进行交互。

针对可视元素和逻辑，ASP.NET 提供了两个用于管理它们的模型，即单文件页模型和代码隐藏页模型。这两个模型的功能相同，可以使用相同的控件和代码。

1. 单文件页模型

在单文件页模型中，页的标记及其编程代码位于一个物理文件 (即“.aspx”文件) 中，

编程代码位于脚本块中，该块包含 runat=server 属性，用于标记该块或控件在服务器端执行。

使用单文件页模型有以下几个优点：

- 可以方便地将代码和标记保留在同一个文件中。
- 更容易部署或发送给其他程序员。
- 由于文件之间没有相关性，更容易对单文件页进行重命名。
- 更易于管理源码文件。

2. 代码隐藏页模型

通过代码隐藏页模型，可以在一个文件 (即“.aspx”文件) 中保存标记，并在另一个文件 (即“.aspx.cs”) 中保存编程代码，该文件被称为“代码隐藏”文件或“页面后台”文件，代码文件的名称会根据所使用的编程语言而有所变化。

使用代码隐藏页模型包括以下几个优点：

- 可以清晰地区分界面中的标记控件和程序代码。
- 代码并不会向界面设计人员或其他人员公开。
- 代码可以在多个页面中进行重用。

4.2.3 认识 Web 窗体页

Web 窗体页以“.aspx”结尾，当创建基于窗体的 Web 应用程序或网站时，会自动生成一些 Web 窗体页。当然，开发者也可以亲自动手创建 Web 窗体页。

【例 4-1】

首先创建一个基于窗体的 Web 应用程序，然后在该程序中创建全称是 FirstTest.aspx 的 Web 窗体页。创建完毕后打开页面，【源】窗口中的代码如图 4-6 所示。

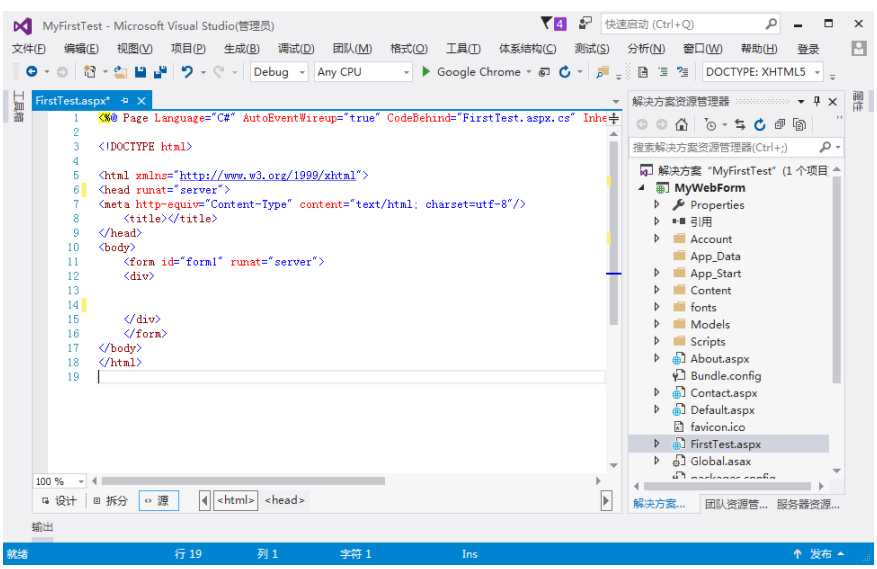

图 4-6 Web 窗体页【源】窗口中的代码

在图 4-6 所示窗口中，单击【设计】按钮可打开设计窗口，单击【拆分】按钮可同时查看页面的源代码和设计效果。观察图 4-6 中的代码可以发现，系统首先通过 @Page 指令定义 ASP.NET 页解析器和编译器所使用的特定页面的属性，然后添加了一段 HTML 代码。开发者可以在 body 元素中添加 HTML 服务器控件、Web 服务器控件或其他内容。

运行页面很简单，在该页面右击，在弹出的快捷菜单中选择【在浏览器中查看】命令即可。或者选中程序中的 Web 窗体页右击，然后同样选择【在浏览器中查看】命令。

4.2.4　高手带你做——了解 ASPX 页面的处理过程

一个 ASP.NET 的 ASPX 页从请求到处理，再到浏览器呈现的过程如下。

01 用户通过客户端浏览器请求页面，页面第一次运行。如果开发人员通过编程让页面执行初步处理，如对页面进行初始化操作等，可以在 Page_load 事件中实现。

02 Web 服务器在其磁盘中定位所请求的页面。

03 如果 Web 页面的扩展名为 .aspx，就把这个文件交给 aspnet-isapi.dll 进行处理。如果以前没有执行过这个页面，那么就由 CLR 进行编译并执行，得到 HTML 结果；如果已经执行过，那么就直接执行编译好的程序并得到 HTML 结果。

04 把 HTML 返回到客户端浏览器。浏览器解释并执行 HTML 页面，显示 Web 页面的内容。

05 当用户在页面中输入信息，选择内容，或者单击按钮后，页面可能会再次被发送到 Web 服务器，这在 ASP.NET 中被称为“回发”。更确切地说，页面发送回其自身。例如，如果用户正在访问 default.aspx 页面，则单击该页面上的某个按钮，可以将该页面发送回服务器，发送的目标还是 default.aspx。

06 在 Web 服务器上，该页面再次被运行，并执行后台代码指定的操作。

07 服务器将执行操作后的页面以 HTML 的形式发送到客户端浏览器。

只要用户访问同一个页面，该循环过程就会继续。用户每次单击某个按钮时，页面中的信息就会发送到 Web 服务器，然后该页面再次运行。每个循环称为一次“往返行程”。由于页面处理发生在 Web 服务器上，因此页面执行的每个步骤都需要一次到服务器的往返行程。

提示

有时可能需要代码仅在首页请求页面时执行，而不是每次回发都执行。这时就可以使用 Page 对象的 IsPostBack 属性来避免对往返过程执行不必要的处理。

4.2.5　页面的生命周期

每个 ASP.NET 页面从请求到呈现到浏览器中都会经历一个生命周期，并在生命周期中执行一系列处理步骤。这些步骤包括初始化、实例化控件、还原和维护状态、运行事件处理程序代码以及呈现给用户。

对于初学者来说，了解 ASP.NET 页面的生命周期非常重要。因为这样做就能在生命周期的合适阶段编写相应的代码，以达到预期的效果。此外，如果要开发自定义控件，更应该熟悉页面的生命周期，以便正确地进行控件的初始化，使用视图状态数据填充控件属性以及运行所有控件的行为代码。

ASP.NET 页面的生命周期顺序如下。

01 页面请求阶段。请求发生在生命周期开始之前。当用户请求页面时，ASP.NET 将确定是否需要分析和编译页面 (从而开始页面的生命周期)；或者是否可以在不运行页面的情况下发送页面的缓存版本以进行响应。

02 开始阶段。在开始阶段，将设置页面属性，如 Request 和 Response 对象。在此阶段，页面还将确定请示是回发请求还是新请求，并设置 IsPostBack 属性。

03 初始化阶段。在页面初始化期间，可以使用页面中的控件，并设置每个控件的 UniqueId 属性。此外，任何主题都将应用于页面。如果当前请求是回发请求，则回发数据尚未加载，并且控件属性尚未还原为视图状态中的值。

04 加载阶段。在页面加载期间，如果当前请求是回发请求，则将使用从视图状态和控件状态恢复的信息加载控件属性。

05 验证阶段。在验证期间将调用所有验证控件的 Validate() 方法，此方法将设置各个验证控件和页面的 IsValid 属性。

06 回发事件处理阶段。在此阶段如果请求的是回发请求，则将调用所有事件处理程序。

07 呈现阶段。在呈现之前，会针对该页面和所有控件保存视图状态。在呈现阶段，页面针对每个控件调用其 Render() 方法，它会提供一个文本编写器，用于将控件的输出写入页面 Response 对象的 OutputStream 属性中。

08 卸载阶段。在页面完全呈现并已发送到客户端时，准备丢弃页面。此时将卸载页面属性并执行清理操作。

4.2.6 页面生命周期事件

在 ASP.NET 页面生命周期的每个阶段，都将引发相应的处理事件。表 4-1 列出了页面生命周期的常用事件。

表 4-1 页面生命周期事件

事件名称	事件说明
Page_PreInit	检查 IsPostBack 属性，确定是不是第一次处理该页。创建或重新创建动态控件。动态设置主控页。动态设置 Theme 属性。读取或设置配置文件属性值
Page_Init	读取或初始化控件属性
Page_Load	读取和更新控件属性
控件事件	使用这些事件来处理特定控件事件，如 Button 控件的 Click 事件
Page_PreRender	该事件对页面或者控件的内容进行最后的更改
Page_Unload	使用该事件来执行最后的清理工作。如关闭打开的文件和数据库连接

1. Page_PreInit 事件

每当页面被发送到服务器时，页面就会重新被加载，启动 Page_PreInit 事件，执行 Page_PreInit 事件代码块。需要对页面中的控件进行初始化时，可以使用此事件。示例代码如下：

```
//Page_PreInit 事件
protected void Page_PreInit(object sender,
EventArgs e)
{
   Label1.Text = "OK" ;
}
```

在上述代码中，当触发 Page_PreInit 事件时，就会执行该事件的代码。上述代码将 Label1 的初始化文本设置为“OK”。 Page_PreInit 事件能够使用户在页面处理中，让服务器加载只执行一次，而当页面被返回给客户端时不被执行。在 Page_PreInit 中可以使用 IsPostBack 来实现当第一次加载时 IsPostBack 属性为 false；当页面再次被加载时，IsPostBack 属性将被设置为 true。

2. Page_Init 事件

Page_Init 事件与 Page_PreInit 事件基本相同，其区别在于 Page_Init 不能保证完全加载各个控件。示例如下：

```
//Page_Init 事件
protected void Page_Init(object sender,
EventArgs e)
{
```

```
    if (!IsPostBack)      // 判断是否为第一次加载
    {
      Label1.Text = "OK";
    }else
    {
      Label1.Text = "IsPostBack";
    }
  }
```

3. Page_Load 事件

大多数初学者会认为 Page_Load 事件是当页面第一次访问时触发的事件。其实不然，在 ASP.NET 页面生命周期内，Page_Load 远远不是第一个触发的事件。通常情况下，ASP.NET 事件的发生顺序如下：

① Page_Init()
② Load ViewState
③ Load Postback Data
④ Page_Load()
⑤ Handle Control Events
⑥ Page_PreRender()
⑦ Unload Event
⑧ Dispose Method Called

Page_Load 事件是在网页加载时一定会被执行的事件。在 Page_Load 事件中，一般都需要使用 IsPostBack 属性来判断用户是否进行了操作。因为 IsPostBack 属性会指示该页是否为响应客户端而加载，或者它是否正被第一次访问而加载。示例代码如下：

```
//Page_Load 事件
protected void Page_Load (object sender,
EventArgs e)
{
  if (!IsPostBack)      // 判断是否为第一次加载
  {
    Label1.Text = "OK";
  }else
  {
    Label1.Text = "IsPostBack";
  }
}
```

上述代码使用了 Page_Load 事件，在页面被创建时，系统会自动在代码隐藏页模型的页面中增加此方法。当用户执行了操作，页面响应客户端回发时，则 IsPostBack 属性为 true，于是 else 块中的代码被执行。

4. Page_UnLoad 事件

在页面被执行完毕后，可以通过 Page_UnLoad 事件来执行页面卸载时的清除工作，当页面被卸载时，执行此事件。在如下情况中都会触发 Page_Unload 事件：

- 页面被关闭时。
- 数据库连接关闭时。
- 对象被关闭时。
- 完成日志记录或者其他程序的请求时。

4.3　页面指令

ASP.NET 页面中通常包含一些类似 <%@ %> 这样的代码，被称为页面指令。这些指令允许相应指定一些属性和配置信息。当使用指令时，标准的做法是将指令放在文件的开头，当然也可以将它们置于“.aspx”或“.ascx”文件中的任何位置。本节介绍 ASP.NET 中提供的一些基本指令，每个指令都可以包含一个或者多个特定于该指令的属性。

4.3.1　@Page 指令

在 ASP.NET 里，使用最多的就是 Web 窗体。每个 Web 窗体里都必须有 @Page 指令，所以 @Page 指令也是使用最为频繁的 ASP.NET 指令。

@Page 指令定义 Web 窗体使用的属性，这些属性将被 Web 窗体页分析器和编译器使用。只能包含在 .aspx 文件中。

我们每新建一个 Web 页面时，系统会自动为该 Web 页面头部创建一个 @Page 指令，来指明页面最基本的属性。初始代码如下：

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

其中 <%@ Page [Attribute=Value]... %> 是系统默认的指令格式。后面由空格隔开的一组组的数据是该指令的属性和值。Language="C#" 指该页面的默认语言为 C#；AutoEventWireup="true" 指系统自动绑定页面服务器端控件的事件；CodeFile="Default.aspx.cs" 指定页面使用的代码文件；Inherits="_Default" 指该页面继承自代码文件中的哪个类。

当然 @Page 指令不只这几个属性。具体属性罗列如表 4-2 所示。

表 4-2　@Page 指令属性

属　性	说　明
AutoEventWireup	用于指示页面上的服务器端控件的事件是否自动绑定
Buffer	用于确定是否启用了 HTTP 响应缓冲
ClassName	用于指定在请求页时将自动进行动态编译的页的类名，其值可以是任何有效的类名，并且可以包括类的完整命名空间；如果未指定该属性的值，则已编译页的类名将基于页的文件名
ClientTarget	指示 ASP.NET 服务器控件应该为其呈现内容的目标用户代理 (通常是 Web 浏览器)，该值可以是应用程序配置文件 <clientTarget> 节中定义的任何有效别名
CodeFile	用于指定指向页引用的代码隐藏文件的路径。此属性与 Inherits 属性一起使用可以将代码隐藏源文件与网页相关联。该属性仅对编译的页有效
CodeFileBaseClass	指定页的基类及其关联的代码隐藏类的路径。该属性是可选的，如果使用该属性，必须同时使用 CodeFile 属性
CodePage	指示用于响应的编码方案的值，该值是一个用作编码方案 ID 的整数
ContentType	将响应的 HTTP 内容类型定义为标准的 MIME 类型
Debug	指示是否应使用调试符号编译该页
Description	提供该页的文本说明。ASP.NET 分析器忽略该值
EnableSessionState	定义页的会话状态要求。如果启用了会话状态，则为 true；如果可以读取会话状态但不能进行更改，则为 ReadOnly；否则为 false。默认值为 true。这些值是不区分大小写的
ErrorPage	定义在出现未处理页异常时用于重定向的目标 URL
Inherits	定义供页继承的代码隐藏类。它可以是从 Page 类派生的任何类。它与 CodeFile 属性 (包含指向代码隐藏类的源文件的路径) 一起使用
Src	指定包含链接到页的代码的源文件的路径。在链接的源文件中，可以选择将页的编程逻辑包含在类中或代码声明块中

Inherits 属性用来设置页面与后台代码中相关联的类。打开 CodeFile 属性所指的文件，会找到该属性所指的类名。但是这里存放的只是用户定义的事件处理程序，并没有任何服务器端对象的声明。

页面的后台关联类是一个 partial(部分) 类，这说明系统还隐藏着一个文件，存放着该类的另一部分。而在这个隐藏的文件里有着所有服务器端控件的声明和属性设置代码。这两个部分类已经完整地声明了一个继承自 Page 类的子类。

因为页面最终也要被系统解释成一个类，所以页面和后台代码类的关系是继承关系。所以，Inherits 属性设置的是被继承的类。

4.3.2　@Control 指令和 @Register 指令

ASP.NET 用户控件的页面指令是 @Control，和 Web 窗体的 @Page 指令用法一样，用来定义用户控件的属性，供分析器的编辑器检查使用。

新建一个用户控件时，系统会自动在该用户控件的头部加入一个 @Control 指令。默认代码如下：

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="WebUserControl.ascx.cs"
 Inherits="Web UserControl" %>
```

这里的属性和 @Page 指令一样，Language 是指定服务器端语言；AutoEventWireup 说明是否自动绑定事件；CodeFile 指定用户控件后台的代码文件；Inherits 同样是指定该用户控件继承自哪个类。

@Control 指令具体的属性如表 4-3 所示。

表 4-3　@Control 指令的属性

属　性	说　明
AutoEventWireup	用于指示控件上的服务器端控件的事件是否自动绑定
ClassName	一个字符串，用于指定需在请求时进行动态编译的控件的类名。此值可以是任何有效的类名，并且可以包括类的完整命名空间 (一个完全限定的类名)。如果没有为此属性指定值，已编译控件的类名将基于该控件的文件名
CodeFile	指定所引用的控件代码隐藏文件的路径。此属性与 Inherits 属性一起使用，将代码隐藏源文件与用户控件相关联
Debug	指示是否应使用调试符号编译控件
EnableViewState	指示是否跨控件请求维护视图状态。如果维护，则为 true；否则为 false。默认值为 true
Inherits	定义提供控件继承的代码隐藏类。它可以是从 UserControl 类派生的任何类。与包含代码隐藏类源文件的路径的 CodeFile 属性一起使用
Src	指定包含链接到控件的代码的源文件的路径。在所链接的源文件中，可选择在类中或在代码声明块中包括控件的编程逻辑

在声明用户控件的时候，要用到 @Control 指令；而在使用用户控件的时候，则需要用到 @Register 指令。

@Register 指令的语法如下：

```
<%@ Register src="WebUserControl.ascx" tagname="WebUserControl" tagprefix="uc1" %>
```

其具体属性说明如表 4-4 所示。

表 4-4　@Register 指令的属性

属　性	说　明
tagprefix	一个任意别名，提供对包含指令中所使用标记命名空间的短引用
tagname	与类关联的任意别名。此属性只用于用户控件
namespace	正在注册的自定义控件的命名空间
src	与 tagprefix:tagname 对关联的声明性 ASP.NET 用户控件文件的位置 (相对的或绝对的)
assembly	与 tagprefix 属性关联的命名空间所驻留的程序集

用户控件可以使用在 Web 窗体或其他用户控件中，但是不能自己嵌套自己。所以 @Register 指令可以出现在用户控件或 Web 窗体内。

4.3.3　@Master 指令

@Master 指令用于指示当前页面标识为 ASP.NET 母版页。简单地说，@Master 指令用于创建母版页。在使用 @Master 指令时，要指定和站点上的内容页面一起使用的模板页面的属性，内容页面 (使用 @Page 指令创建) 可以继承母版页上的所有内容。

@Master 指令与 @Page 指令类似，但是它的属性要比 @Page 指令的属性少。创建一个母版页时，生成的 @Master 指令的代码如下：

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.cs"
Inherits="WorkTest.SiteMaster" %>
```

4.3.4　@MasterType 指令

@MasterType 指令把一个类名关联到 ASP.NET 页面，以获取特定母版页中包含的强类型化的引用或成员。@MasterType 指令支持 TypeName 和 VirtualPath 两个属性。

- TypeName 属性：设置从中获取强类型化的引用或成员的派生类的名称。
- VirtualPath 属性：设置从中检索强类型化的引用或成员的页面地址。

如下代码简单演示了 @MasterType 指令的使用：

```
<%@ MasterType VirtualPath="~/MyWork.master" %>
```

4.3.5　@Import 指令

@Import 指令在页面或用户控件中显式地引入一个命名空间，以便所有已定义类型可以在页面访问，而不必使用完全限定名。例如，创建 ASP.NET 中 DataSet 类的实例时可以导入 System.Data 命名空间，也可以使用完全限定名。使用完全限定名的代码如下：

```
System.Data.DataSet ds = new System.Data.DataSet();
```

@Import 指令可以在页面主体中多次使用，它相当于 C# 中的 using 语句。如下所示为它的简单例子：

```
<%@ Import Namespace="System.Data" %>
```

4.3.6　@Implements 指令

@Implements 指令允许在页面或用户控件中实现一个 .NET Framework 接口，该指令只支持 Interface 属性。Interface 属性直接指定 .NET Framework 接口。当 ASP.NET 页面或用户控件实现接口时，可以直接访问其中所有的方法和事件。

如下所示是使用 @Implements 指令的例子：

```
<%@ Implements Interface="System.Web.UI.IValidator" %>
```

4.3.7　@Reference 指令

@Reference 指令用来识别当前页面在运行时应该动态编译和链接的页面或用户控件。该指令在跨页通信方面发挥着重大作用，可以通过它来创建用户控件的强类型实例。

@Reference 指令包含 3 个属性：Page、Control 和 VirtualPath，并可以多次出现在页面中。

- Page 属性：该属性用于指向某个“.aspx”源文件。
- Control 属性：该属性包含“.ascx”用户控件的路径。
- VirtualPath 属性：设置从中引用活动页面的页面或用户控件的位置。

4.3.8　@Assembly 指令

@Assembly 指令将程序集引入到当前页面或用户控件中，以便它所包含的类和接口能够适用于页面中的代码。@Assembly 指令支持 Name 和 Src 两个属性。

- Name 属性：允许指定用于关联页面文件的程序集名称。程序集名称应只包含文件名，不包含文件的扩展名。假设文件是 MyAssembly.vb，那么 Name 属性的值是 MyAssembly。
- Src 属性：允许指定编译时所使用的程序集文件源。

4.3.9　@OutputCache 指令

@OutputCache 指令对页面或用户控件在服务器上如何输出高速缓存进行控制。该指令的常用属性如表 4-5 所示。

表 4-5　@OutputCache 指令的常用属性

属性名称	说　明
Duration	ASP.NET 页面或用户控件高速缓存的持续时间。单位为秒
CacheProfile	允许使用集中式方法管理应用程序的调整缓存配置文件。该属性用于指定在 Web.config 文件中详细说明的调整缓存配置文件名
Location	位置枚举值，只对“.aspx”页面有效，不能用于用户控件。其值包括 Any (默认值)、Client、Downstream、None、Server 和 ServerAndClient

（续表）

属性名称	说　明
NoStore	指定是否随页面发送没有存储的标题
Shared	指定用户控件的输出是否可以在多个页面中共享。默认值为 false
SqlDependency	支持页面使用 SQL Server 高速缓存禁用功能
VeryByControl	用分号分隔开的字符串列表，用于改变用户控件的输出高速缓存
VeryByParam	用分号分隔开的字符串列表，用于改变输出高速缓存
VeryByCustom	一个字符串，指定定制的输出高速缓存需求
VerByHeader	用分号分隔开的 HTTP 标题列表，用于改变输出高速缓存

4.3.10　@PreviousPageType 指令

@PreviousPageType 指令用于指定跨页面的传送过程起始于哪个页面。@PreviousPageType 是一个新指令，用于处理 ASP.NET 提供的跨页面传送新功能。这个简单的指令只包含两个属性：TypeName 和 VirtualPath。

- TypeName 属性：设置回送时派生类的名称。
- VirtualPath 属性：设置回送时所传送页面的地址。

注意

在前面介绍的指令中，除了 @Page、@Control、@Master、@MasterType 和 @PreviousPageType 外，所有指令都可以在页面和控件中声明。@Page 和 @Control 是互斥的：@Page 只能用在“.aspx”文件中；@Control 只能用在“.ascx”文件中。

4.4　高手带你做——允许页面提交 HTML 标签

有时候，如果需要让我们提交的文本展示出来的效果非常美观，通常会向服务器提交一些 HTML 标签来控制文本或内容的样式。

HTML 标签可能包含很多不安全的因素，所以向服务器提交 HTML 标签通常会被服务器认为是不安全的操作。

ASP.NET 作为一个功能强大的应用系统，已经默认地把这类操作给过滤掉了。但是如果我们自己可以确保提交内容的安全性，当然可以通过设置关闭该过滤选项。这就要用到 ASP.NET 页面的 Page 指令来对页面的安全性进行设置了。

假设在一个 ASP.NET 页面中有如下代码：

```
<asp:TextBox ID="txtContent" runat="server" Width="234px"></asp:TextBox>
<asp:Button ID="btnSubmit" runat="server" Text="Submit" onclick="btnSubmit_Click" />
<br />
<asp:Label ID="lblShow" runat="server" Text="Label"></asp:Label>
```

在单击 Button 按钮时会将 TextBox 中的内容显示到其后的 Label 控件中，实现代码如下：

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    this.lblShow.Text = this.txtContent.Text;
}
```

运行页面，首先输入普通的文本，再单击 Submit 按钮，运行效果如图 4-7 所示。如果输入的内容带有 HTML 标签，例如输入“<h3>About</h3>”，再单击 Submit 按钮，页面会报错并提示检测到有潜在的危险，如图 4-8 所示。

解决的办法就是将页面中 @Page 指令的 ValidateRequest 属性值设置为 false。再次运行，输出的效果如图 4-9 所示。

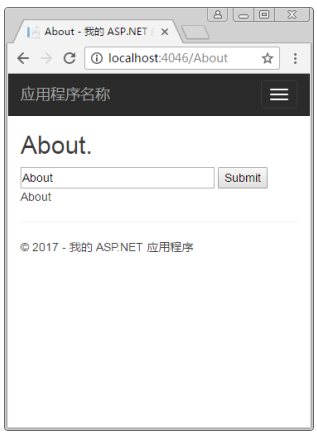

图 4-7　普通文本效果

图 4-8　报错信息

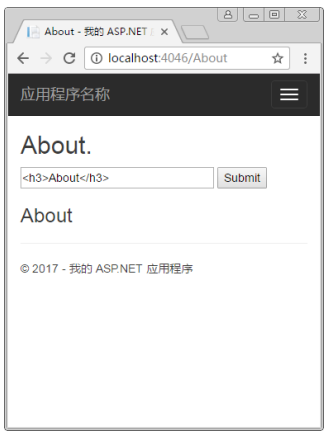

图 4-9　带 HTML 标签的效果

本实例创建一个 TextBox 控件、一个 Button 控件和一个 Label 控件，在单击 Button 控件提交表单的时候，将 TextBox 控件里的值用 Label 控件展示出来。

4.5　成长任务

成长任务 1：测试页面缓存

在 ASP.NET 中使用页面的 @OutputCache 指令可以实现对整个页面进行缓存。假设要在页面上输出当前时间，并让页面每秒钟刷新一次，则每一次刷新打印的时间都会不一样。在这个前提下，我们将该页面缓存 5 秒，如果缓存成功，则 5 秒内页面数据不会变化，在第 6 秒时才更改为新的时间值。

页面显示时间的代码如下：

```
<%= DateTime.Now.ToString() %>
```

```
<script>setTimeout("location.href=location.href",1000)</script>
```

页面头部添加缓存的代码如下：

```
<%@ OutputCache Duration="5" VaryByControl="none" %>
```

在浏览器中运行并观察运行效果。