# ASP.NET编程指南

极客学院出版

# 前言

ASP.NET 是一个由 Microsoft 开发和送入市场的网页应用程序框架，它允许程序员构建动态网站。它也允许你使用功能齐全的程序设计语言比如 C# 或者 VB.NET 来简单构建网页应用程序。

本指南包含了所有初学者将需要使用的 ASP.NET 的基本元素。

## 适用人群

本指南已为初学者准备好来帮助他们理解基本的 ASP.NET 编程。在读完这个指南后你将发现自己处于 ASP.NET 编程中的中等专业知识水平，你可以从那将自己带入下一个等级。

## 学习前提

在进行本指南之前，你应该有一个基本的有关 .NET 编程语言的理解。我们将使用 ASP.NET 网页应用程序框架来开发基于网页的应用程序，如果你有其它的网页技术比如 HTML，CSS，AJAX 等，那将会很好。

# 目录

# 1 简介

ASP.NET 是一个 web 开发平台，它提供编程模型、软件基础程序以及多种服务来帮助开发者搭建健壮的网络应用程序。

ASP.NET 工作于 HTTP 协议之上，并使用 HTTP 命令和政策来建立浏览器到用户之间双向的交流与合作。

ASP.NET 是 Microsoft.NET 平台的一部分。ASP.NET 应用程序是编译后的代码，运行在 .Net framework 中，利用可扩展和可重用的组件和对象编写的。

ASP.NET 应用程序编码可以用以下语言编写：

- C#
- Visual Basic.Net
- Jscript
- J#

ASP.NET 用于产生互动的、通过互联网数据驱动的 web 应用程序。它由大量的控件组成，例如：文本框，按钮，组装标签，构形，以及操控编码来创建 HTML 页面。

## ASP.NET web 表单模型

ASP.NET web 表单延伸了交互作用对 web 应用程序的事件驱动模型。浏览器提交给 web 服务器一个 web 表单，然后服务器返回一个完整的标记页面或 HTML 页面作为回应。

所有客户端用户活动转发到服务器进行有状态的处理。服务器处理客户端动作的输出并触发反馈。

现在，HTTP 是一种无状态协议。ASP.NET 框架帮助储存有关应用程序状态的信息，由以下组成：

- 页状态
- 会话状态

页状态是客户端状态，例如：在 web 表单中不同输入领域的内容。会话状态是由用户浏览和使用的不同页面中获得的集合信息，例如：整体会话状态。为了更清楚地了解这个概念，我们拿购物手推车作为例子。

用户在购物手推车中添加项目。项目是在一个页面中所选，叫做项目页面，而项目这个集合的总数和价格就会在不同的页面所显示，叫做购物车页面。只有 HTTP 是不能记录来自不同页面的信息。ASP.NET 会话状态以及服务器基础设施通过一个会话记录全球所收集的信息。

ASP.NET 在生成 ASP.NET 运行时间编码的时候，ASP.NET 运行时间通过页面请求在服务器终端送去或拿回页面状态，并且与隐藏领域内的服务端组件状态合并。

ASP.NET 用这个方法，服务器会意识到整体应用程序状态并以双层的连接方式进行操作。

## ASP.NET 组件模型

ASP.NET组件模型提供了 ASP.NET 页面的不同的组成部件。基本上它是一个对象模型，描述为：

- 几乎所有的 HTML 元素或标签，例如 <form> 和 <input> 。
- 服务器控件，帮助开发复杂的用户界面。例如：日历控件或者网络视图控件。

ASP.NET 是一项技术，工作于 .Net 框架，包括所有与网络相关的功能。.NET 框架由一个面向对象的等级组成。一个 ASP.NET 的 web 应用是由页面组成。当一个用户请求一个 ASP.NET 页面，IIS 委派页面到 ASP.NET 的运行时系统。

ASP.NET 运行时把 .aspx 页面转化成为一个类的实例，继承了 .Net 框架的基本类页面。因此，每一个 ASP.NET 页面是一个对象，并且其所有组件如服务器端控件也是对象。

## .Net Framework 3.5 的组件

在进入到下一个关于 Visual Studio.Net 的部分，我们先来浏览一下 .Net Framework 3.5 的不同组件。以下表格描述了.Net Framework 3.5 的组件和它们所执行的工作：

| 组件及描述 |
| --- |
| （1）通用语言运行环境或者 CLR<br>它执行内存管理、异常处理、调试、安全检查、线程执行、代码执行、代码安全、验证和编译。由 CLR 直接管理的代码被称为管理代码。但通过管理代码被编译，编译器将源代码转化到一个 CPU 独立的中介语言（IL）代码。实时（JIT）编译器将 IL 代码编译成源代码，特定于 CPU。 |
| （2）.Net Framework 类库<br>它包含一个有可重复使用类型的巨大的库。类、接口、结构、和枚举值，它们统称为类型。 |
| （3）通用语言规范<br>它包含对于 .NET所支持的语言和集成语言实现的规范。 |
| （4）通用类型系统<br>它提供了对于在运行时声明、使用和管理类型和跨语言交流的指导。 |
| （5）元数据和程序集<br>元数据是描述程序的二进制信息，或者储存在一个可执行文件内（PE），或者储存在内存里。程序集是一个逻辑单元，由程序集清单、元数据类型、IL代码、和一组资源，比如图片文件组成。 |
| （6）Windows 窗体<br>Windows 窗体包括在应用程序中展示的任何窗口的图形表示。 |
| （7）ASP.NET 和 ASP.NET AJAX<br>ASP.NET 是 web 开发模型，AJAX 是对于 ASP.NET 开发和执行 AJAX 功能的一个延伸。ASP.NET AJAX 包括组件，可以允许开发者更新网页上的数据，并且不用对页面完整地重新下载。 |

| 组件及描述 |
| --- |
| （8）ADO.NET<br>它是与数据和数据库工作相关的技术。它为数据源提供通道，例如：SQL 服务器、OLE DB、XML 等等。ADO.NET 允许连接源数据进行检索、操作及数据更新。 |
| （9）Windows工作流基础（WF）<br>它帮助构建在 Windows 中以工作流为基础的应用程序。它包括活动、工作流运行时、工作流设计和规则引擎。 |
| （10）Windows 描述基础<br>它提供了一个对于用户界面和业务逻辑之间的分离。它通过使用文档、媒体、两维和三维图形、动画等等，从而帮助开发具有极强的视觉冲击力的界面。 |
| （11）Windows 交流基础（WCF）<br>它是用于构建和执行连接系统的技术。 |
| （12）Windows CardSpace<br>它为网络上访问资源和分享私人信息提供安全保障。 |
| （13）LINQ<br>它赋予 .NET 语言数据查询功能，利用一个类似于传统查询语言 SQL 的句法。 |

# 2

# 环境设置

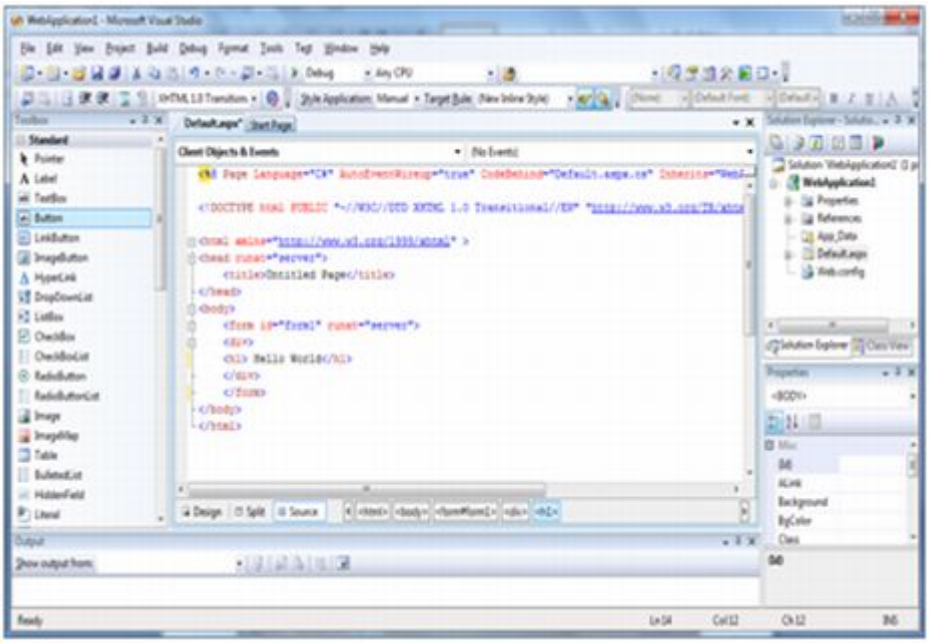
ASP.NET 在 HTTP 的顶部提供了一个抽象层，即应用程序所构建的地方。它提供了高层次的实体，例如一个面向对象的范例中的类和组件。

构建 ASP.NET 应用程序的关键开发工具及前端是 Visual Studio。本教程中，我们主要讲 Visual Studio 2008。

Visual Studio 是一个整合的开发环境，用于编写，编译和调试代码。它为构建 ASP.NET web 应用程序、web 服务、桌面应用程序和移动应用程序提供了一组完整的开发工具。

## Visual Studio IDE

新项目窗口允许从可用模板中选择一个应用程序模板。



图片 2.1 image

当你打开一个新的网站，ASP.NET 提供启动文件夹和网站的文件，包括站点中的第一个 web 表单的两个文件。

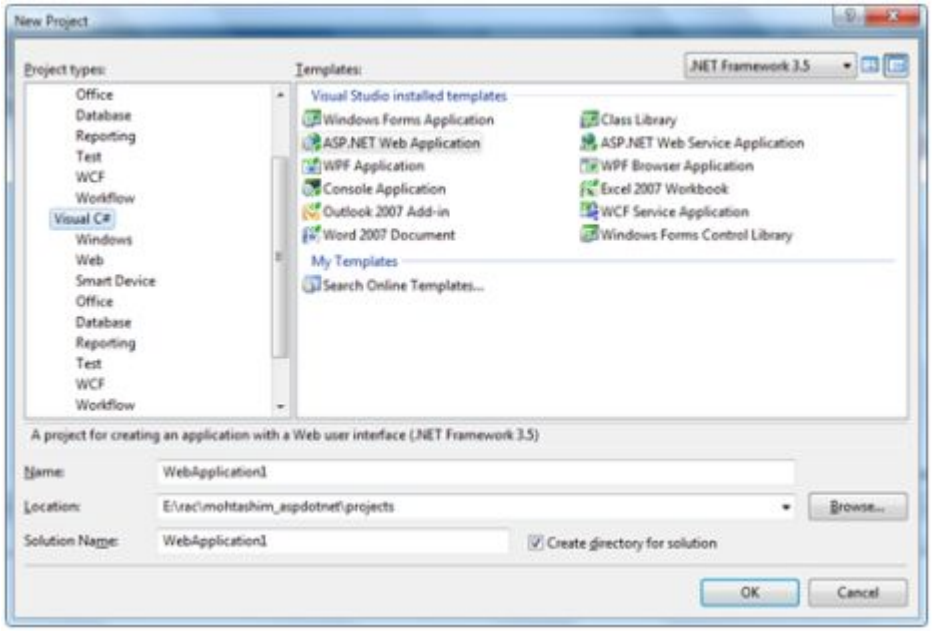
名称为 Default.aspx 的文件包括 HTML 和 asp 代码，可以定义表单，名称为 Default.aspx.cs（为 C# 编码）的文件或者名称为 Default.aspx.vb（为 VB 编码）的文件包含你所选择的语言的代码，并且此代码为一个表单中所执行的动作负责。

在 Visual Studio IDE 中的初始窗口是 Web Forms Designer 窗口。其他支持的窗口有工具箱、解决方案资源管理器以及内容窗口。设计师设计一个 web 表单，在表单上添加代码进行控制，根据你所需使用代码编辑程序。

## 使用视图和窗口

你可以用以下方式使用窗口：

- 把 Web 表单设计器从一个视图改变到另一个视图，点击 Design 或者源按钮。
- 要关闭一个窗口，点击右上角的关闭按钮，重新显示的话，从视图菜单中选择。
- 要隐藏一个窗口，点击自动隐藏按钮。窗口就会变成一个标签。再次显示的话，再次点击自动隐藏按钮。
- 要改变窗口的大小，拖拽窗口即可。




图片 2.2 image

## 在你的网站中添加文件夹和文件

当创造了一个新的 web 表单，Visual Studio 自动为表单生成启动 HTML，并且在 web 表单设计器上显示出源视图。解决方案资源管理器被用于添加其他任何文件，文件夹或者在 web 站点的现有项目。

- 要增加一个标准的文件夹，右键点击你将要在解决方案管理器中添加的项目或文件夹，选择新文件夹。
- 要增加一个 ASP.NET 文件夹，右键点击在解决方案管理器中的项目，在列表中选择文件夹。
- 要在站点中添加一个现有项，右键点击你将要在解决方案管理器中添加的项目，在对话窗口中选择。

## 项目和解决方案

一个典型的 ASP.NET 应用程序由许多的项目组成：web 内容文件（.aspx），源文件（.cs 文件），程序集（.dll 和 .exe 文件），数据源文件（.mgd 文件），引用，图标，用户控件和其他杂项文件和文件夹。所有组成网址的这些文件包含在一个解决方案中。

当创造了一个新的网站，.VB2008 自动创造了解决方案，并且在解决方案管理器中显示。

解决方案可能包含一个或多个项目。一个项目包含内容文件，源文件，以及其他文件比如说数据源和图片文件。一般来说，一个项目的内容可以编译成一个程序集作为一个可执行文件（.exe）或者一个动态链接库（.dll）文件

一般来说一个项目包含以下内容文件：

- 页面文件（.aspx）
- 用户控件（.ascx）
- Web 服务器（.asmx）
- 主版页（.master）
- 网站导航（.sitemap）
- 网站配置文件（.config）

## 建立和运行项目

你可以执行一个应用程序，通过：

- 选择启动
- 选择启动而不从调试菜单中调试
- 按 F5
- Ctrl-F5

程序构建的基本思路是 .exe 或 .dll 文件从 Build 菜单中选择一个命令而生成。

# 3

# 生命周期

ASP.NET 生命周期指定如何：

- ASP.NET 处理页面生成动态输出
- 应用程序及其页面进行实例化和处理
- ASP.NET 动态编译页面

ASP.NET 生命周期可以被分为两组：

- 应用程序生命周期
- 页面生命周期

## ASP.NET 应用程序生命周期

应用程序生命周期有以下阶段：

- 用户请求访问应用程序的资源，即一个页面。浏览器发送此请求到 web 服务器。
- 一个统一管道接收第一个请求，并发生以下事件：
    - 一个 ApplicationManager 类的对象创建。
    - 一个 HostingEnvironment 类的对象创建从而提供信息资源。
- 创建响应对象。应用程序对象如 HttpContext，HttpRequest 和 HttpResponse 被创建并初始化。
- 一个 HttpApplication 对象的实例被创建并被分配到请求。
- 请求由 HttpApplication 类所处理。不同的事件引发此类处理请求。

## ASP.NET 页面生命周期

当请求一个页面时，页面被加载到服务器内存，然后处理并送达到浏览器中。然后再从内存中卸载。在这些步骤中的每一步，方法和事件都是可用的，可以根据应用程序所需进行改写。换言之，你可以编写自己的代码从而去置换缺省代码。

页面类创建了页面上所有控件的等级树。页面上所有的组件，除了指令，其余都是控件树的一部分。你可以通过在页面指令上添加 trace = "true" 来看到控制树。我们会覆盖页面指令，然后在 'directives' 和 'event handling' 下追踪。

页面生命周期阶段为：

- 初始化
- 页面控件实例化
- 状态恢复和维护
- 事件处理代码的执行
- 页面显示

理解页面周期帮助我们编写代码从而使一些具体的事情可以在页面生命周期中任何阶段发生。它同样帮助编写自定义控件并且在合适的时间将其初始化，利用视图状态下的数据填充其属性，并且运行控件行为的代码。

以下是一个 ASP.NET 页面的不同阶段：

- **页面请求** – 当 ASP.NET 得到一个页面请求，它决定是否解析和编译页面，或者会有一个页面的缓存版本；相应地进行回应。
- **页面生命周期的开启** – 在这个阶段，设置请求和回应对象。如果是一个旧的请求或者是回发的，页面 IsPostBack 属性设置为正确。页面 ULCulture 属性同样也被设置。
- **页面初始化** – 在此阶段，页面上的控件通过设置 UniqueID 属性被分配到独特的 ID 并应用主题。对于一个新的请求，加载回发数据并且控件属性被重新储存到视图状态下的值。
- **页面加载** – 在此阶段，设置控件属性，使用视图状态和控件状态值。
- **验证** – 调用验证控件的校验方法并成功执行，页面的 IsValid 属性设置为 true。
- **回发事件处理** – 如果请求是一个回发（旧请求），相关事件处理程序被调用。
- **页面显示** – 在此阶段，页面的视图状态和所有控件被保存。页面为每一个控件调用显示方法，并且呈现的输出被写入页面响应属性中的 OutputStream 类。
- **卸载** – 显示过的页面被送达客户端以及页面属性，例如响应和请求，被卸载并全部清除完毕。

## ASP.NET 页面生命周期事件

在页面生命周期的每一阶段，页面引发一些时间，会被编码。一个事件处理程序基本上是一个函数或子程序，绑定到事件，使用声明式如 OnClick 属性或处理。

以下是页面生命周期事件：

- PreInit – PreInit 是页面生命周期的第一个事件。它检查 IsPostBack 属性并决定页面是否是回发。它设置主题及主版页，创建动态控件，并且获取和设置值配置文件属性值。此事件可通过重载 OnPreInit 方法或者创建一个 Page_PreInit 处理程序进行处置。

- Init – Init 事件初始化控件属性，并且建立控件树。此事件可通过重载 OnInit 方法或者创建一个 Page_Init 处理程序进行处置。
- InitComplete – InitComplete 事件允许对视图状态的跟踪。所有的控件开启视图状态下的跟踪。
- LoadViewState – LoadViewState 事件允许加载视图状态信息到控件中。
- LoadPostData – 在此阶段期间，对所有由 \

<

form> 标签定义的输入字段的内容进行处理。 – PreLoad – PreLoad 在回发数据加载在控件中之前发生。此事件可以通过重载 OnPreLoad 方法或者创建一个 Pre_Load 处理程序进行处置。 – Load – Load 事件被页面最先引发，然后递归地引发所有子控件。控件树中的控件就被创建好了。此事件可通过重载 OnLoad 方法或者创建一个 Page_Load 处理程序来进行处置。 – LoadComplete – 加载进程完成，控件事件处理程序运行，页面验证发生。此事件可通过重载 OnLoad 方法或者创建一个 Page_LoadComplete 处理程序来进行处置。 – PreRender – PreRender 事件就在输出显示之前发生。通过处理此事件，页面和控件可以在输出显示之前执行任何更新。 – PreRenderComplete – 当 PreRender 事件为所有子控件被循环引发，此事件确保了显示前阶段的完成。 – SaveStateComplete – 页面控件状态被保存。个性化、控件状态以及视图状态信息被保存。 – UnLoad – UnLoad 阶段是页面生命周期的最后一个阶段。它为所有控件循环引发 UnLoad 事件，最后为页面自身引发。最终完成清理和释放所有资源和引用，比如数据库连接。此事件可通过调整 OnLoad 方法或者创建一个 Page_UnLoad 处理程序来进行处置。

# 4 实例

ASP.NET 页面是由大量的服务器控件以及 HTML 控件、文本和图像组成的。页面的敏感数据和页面上的不同控件状态被储存在隐藏字段中，组成了页面请求的配置指令。

ASP.NET 运行时控制一个页面实例和其状态的关联。一个 ASP.NET 页面是一个页面的对象或者从之继承而来。

页面上所有的控件同样也是从一个父类控件继承而来的相关控件类的对象。当一个页面运行时，对象页面的一个实例就随其内容控件一起被创建。

一个 ASP.NET 页面同样也是储存在 .aspx 延伸的服务器端文件。

它在本质上是模块化的，并且可被分成以下几个核心部分：

- 网页指令
- 编码区段
- 页面布局

## 页面指令

页面指令为页面设置运行环境。@Page 指令定义了使用 ASP.NET 页面解析器和编译器的特殊页面属性。页面指令指定应该如何处理页面，并指定对页面需要采取的假设。

它允许导入命名空间、加载程序集和注册新的控件，包括自定义标记名称和命名空间前缀。

## 编码区段

编码区段为页面和控件即其他所需功能提供处理程序。我们提到，ASP.NET 遵从对象模型。现在，当一些事件在用户界面发生，这些对象会激发事件，比如说一个用户点击了一个按钮或者移动了光标。这些事件需要往复的这类响应是在事件处理程序功能里编码的。事件处理程序除了绑定到空间上的功能就没什么了。

编码区段或者文件后的编码提供了对于所有这些事件处理程序的路线，以及其他开发者使用的功能。页面代码可以预编译和以二进制汇编的形式进行部署。

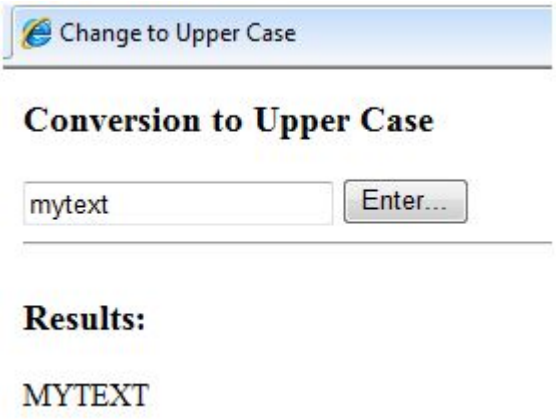
## 页面布局

页面布局提供了页面的界面。它包含服务器控件、文本和内联的 JavaScript 和 HTML 标签。

下面的代码片段提供了一个 ASP.NET 页面的示例，解释了用 C# 编写的页面指令、代码区段和页面布局：

```
<!-- directives -->
<% @Page Language="C#" %>

<!-- code section -->
<script runat="server">

  private void convertoupper(object sender, EventArgs e)
  {
    string str = mytext.Value;
    changed_text.InnerHtml = str.ToUpper();
  }
</script>

<!-- Layout -->
<html>
  <head>
    <title> Change to Upper Case </title>
  </head>

  <body>
    <h3> Conversion to Upper Case </h3>

    <form runat="server">
      <input runat="server" id="mytext" type="text" />
      <input runat="server" id="button1" type="submit" value="Enter..." OnServerClick="convertoupper"/>

      <hr />
      <h3> Results: </h3>
      <span runat="server" id="changed_text" />
    </form>

  </body>

</html>
```

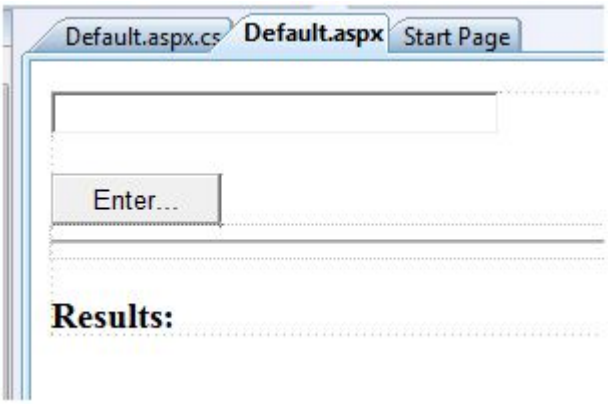
复制此文件到 web 服务器的根目录。一般的是 c:\iNETput\wwwroot。从浏览器中打开文件然后执行，它就会生成以下结果：




图片 4.1 image

## 使用 Visual Studio IDE

让我们用 Visual Studio IDE 展开同样的例子。你可以直接拖拽控件到设计视图，而不用输入代码。




图片 4.2 image

内容文件会自动生成。你只需添加的是 Button1_Click 代码，即如下所示：

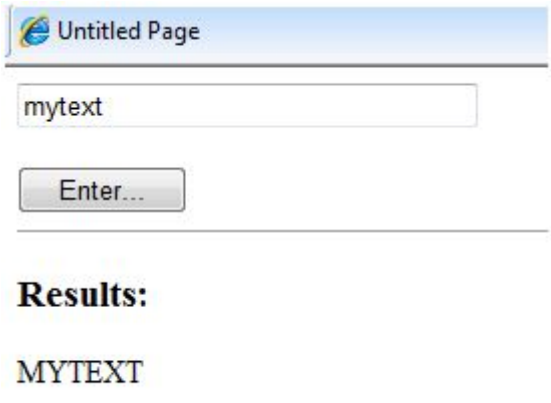
```
protected void Button1_Click(object sender, EventArgs e)
{
  string buf = TextBox1.Text;
  changed_text.InnerHtml = buf.ToUpper();
}
```

内容文件代码已给出：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>

    <form id="form1" runat="server">
      <div>

        <asp:TextBox ID="TextBox1" runat="server" style="width:224px">
        </asp:TextBox>

        <br />
        <br />

        <asp:Button ID="Button1" runat="server" Text="Enter..." style="width:85px" onclick="Button1_Click" />
        <hr />

        <h3> Results: </h3>
        <span runat="server" id="changed_text" />

      </div>
    </form>

  </body>

</html>
```

通过右键点击设计视图来执行此示例，并且从弹出菜单中选择 ‘View in Browser’。这将生成以下结果：




图片 4.3 image

# 5

# 事件处理

事件是一个动作或发生的事，如鼠标单击、按键、鼠标移动，或任何系统生成的通知。一个进程通过事件进行沟通。例如，中断是系统生成的事件。当事件发生，应用程序也能够回应和管理。

ASP.NET 上的事件在用户机器上引发，在服务器上处理。例如，一个用户点击了在浏览器中显示的一个按钮。一个点击事件被引发。浏览器通过把它发送给服务器从而处理这个客户端事件。

服务器有一个子程序来描述当事件被引发时该做什么；这个被称为事件处理程序。因此，当事件信息被传递给服务器，它会检查点击事件是否与事件处理程序有关联。如果有关联的话，事件处理程序就会被执行。

## 事件参数

ASP.NET 事件处理程序一般采用两个参数并返回空。第一个参数代表了对象激发事件，第二个参数是事件参数。

一个事件的一般句法是：

```
private void EventName (object sender, EventArgs e);
```

## 应用程序和会话事件

最重要的应用程序事件是：

- Application_Start – 当开启应用程序或者网页时被引发。
- Application_End – 当停止应用程序或者网页时被引发。

同样的，最常使用的会话事件是：

- Session_Start – 当用户最开始从应用程序上请求一个页面被引发。
- Session_End – 当会话结束后被引发。

## 页面和控件事件

常见的页面和控件事件有：

- DataBinding – 当一个控件绑定到一个数据源时被引发。
- Disposed – 当释放页面或者控件时被引发。
- Error – 它是一个页面事件，当有未处理的异常时发生。

- Init - 当初始化页面或者控件时被引发。
- Load - 当加载页面或者控件时被引发。
- PreRender - 当显示页面或者控件时被引发。
- Unload - 当从内存中卸载页面或者控件时被引发。

## 使用控件处理事件

所有的 ASP.NET 控件作为类而实现，并且当用户对其执行一个特定的动作时，它们会引发事件。比如说，当一个用户点击了一个按钮，那就生成了 'Click' 事件。对于处理事件来说，有内置属性和事件处理程序。事件处理应用程序被编码作为一个事件的回应，并且对其采取适当的行动。

默认情况下，Visual Studio 创建一个事件处理程序，包括处理条款的子程序。这个子句命名程序处理的控件和事件。

button 控件的 ASP 标签：

```
<asp:Button ID="btnCancel" runat="server" Text="Cancel" />
```

Click 事件的事件处理应用程序：

```
Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e As System.EventArgs)

  Handles btnCancel.Click

End Sub
```

一个事件同样可以在没有 Handles 子句的前提下被编码。然后，处理程序必须根据适合控件属性的适当事件进行命名。

button 控件的 ASP 标签：

```
<asp:Button ID="btnCancel" runat="server" Text="Cancel" Onclick="btnCancel_Click" />
```

Click 事件的事件处理应用程序：

```
Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e As System.EventArgs)

End Sub
```

常见的控件事件有：

| 事件 | 属性 | 控件 |
| --- | --- | --- |
| Click | OnClick | 按钮，图像按钮，链接按钮，图像导位图 |
| Command | OnCommand | 按钮，图像按钮，链接按钮 |
| TextChanged | OnTextChanged | 文本框 |
| SelectedIndexChanged | OnSelectedIndexChanged | 下拉菜单，列表框，单选按钮列表，带复选框的列表框 |
| CheckedChanged | OnCheckedChanged | 复选框，单选按钮 |

一些事件导致表单立即发回到服务器，这些被称为回调事件。例如，单击事件像 Button.Click。

一些事件则不会被立即发回到服务器，这些被称为非回调事件。

例如，改变事件或者选择事件，像 TextBox.TextChanged 或者 CheckBox.CheckedChanged。这些非回调事件可以通过设置它们的 AutoPostBack 属性为 true 便可立即使它们回调。

## 默认事件

页面对象的默认事件是加载事件。相似地，每一个控件都有一个默认的事件。比如，按钮控件的默认事件就是 Click 事件。

默认事件处理程序可以在 Visual Studio 中创建，仅通过双击设计视图中的控件。以下表格展示了一写常见控件的默认事件：

| 控件 | 默认事件 |
| --- | --- |
| 广告控件（AdRotator） | AdCreated |
| 项目列表（BulletedList） | Click |
| 按钮（Button） | Click |
| 日历控件（Calender） | SelectionChanged |
| 复选框（CheckBox） | CheckedChanged |
| 复选列表（CheckBoxList) | SelectedIndexChanged |
| 数据表格（DataGrid） | SelectedIndexChanged |
| 数据列表（DataList） | SelectedIndexChanged |
| 下拉列表（DropDownList） | SelectedIndexChanged |
| 超链接（HyperLink） | Click |
| 图像按钮（ImageButton） | Click |
| 热点（ImageMap） | Click |
| 超链接按钮（LinkButton） | Click |
| 单选或多选的下拉列表（ListBox ） | SelectedIndexChanged |
| 菜单（Menu） | MenuItemClick |
| 单选按钮（RadioButton） | CheckedChanged |

| 控件 | 默认事件 |
| --- | --- |
| 单选按钮组（RadioButtonList） | SelectedIndexChanged |

示例

这个例子包括一个简单页面，上面有控件标签和一个按钮控件。当页面事件，例如 Page_Load，Page_Init，Page_PreRender 等等事件发生的时候，它就会发送一条信息，会由标签控件显示。当点击一个按钮，Button_Click 事件被引发，同样发送一条由标签展示的信息。

创建一个新的网站，从控件工具框中拖拽一个标签控件和按钮控件。使用窗口属性，相应地设置控件的 ID 为 .lblmessage. and .btnclick。设置按钮控件的文本属性为“Click”。

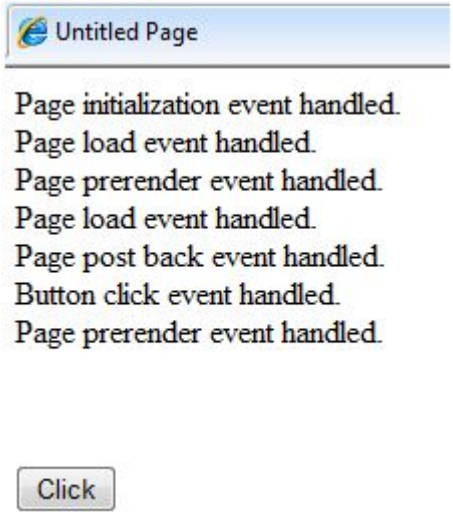
标记文件（.aspx）：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>Untitled Page</title>
  </head>

  <body>
    <form id="form1" runat="server">
      <div>
        <asp:Label ID="lblmessage" runat="server" >

        </asp:Label>

        <br />
        <br />
        <br />

        <asp:Button ID="btnclick" runat="server" Text="Click" onclick="btnclick_Click" />
      </div>
    </form>
  </body>

</html>
```

双击设计视图并移动至文件后的代码。Page_Load 事件是自动创建的，其中没有任何的代码。写下以下的自我解释的代码行：

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace eventdemo {

  public partial class _Default : System.Web.UI.Page {

    protected void Page_Load(object sender, EventArgs e) {
      lblmessage.Text += "Page load event handled. <br />";

      if (Page.IsPostBack) {
        lblmessage.Text += "Page post back event handled.<br/>";
      }
    }

    protected void Page_Init(object sender, EventArgs e) {
      lblmessage.Text += "Page initialization event handled.<br/>";
    }

    protected void Page_PreRender(object sender, EventArgs e) {
      lblmessage.Text += "Page prerender event handled. <br/>";
    }

    protected void btnclick_Click(object sender, EventArgs e) {
      lblmessage.Text += "Button click event handled. <br/>";
    }
  }
}
```

执行页面。标签显示页面加载，页面初始化以及页面预览事件。点击按钮查看效果：


Untitled Page
Page initialization event handled.
Page load event handled.
Page prerender event handled.
Page load event handled.
Page post back event handled.
Button click event handled.
Page prerender event handled.
Click

图片 5.1 image

# 服务器端

我们已经研究了页面生命周期和一个页面如何包含不同的控件。页面本身作为一个控制对象被实例化。所有的 web 表单基本上是 ASP.NET 页面类的实例。页面类有以下极其有用的属性，与内部对象所对应：

- 会话
- 应用程序
- 缓存
- 请求
- 响应
- 服务器
- 用户
- 跟踪

我们会在适当的时间里讨论每一个对象。在本教程中我们将会探索 Server 对象，Request 对象和 Response 对象。

## Server 对象

ASP.NET 中的服务器对象是 System.Web.HttpServerUtility 类的一个实例。The HttpServerUtility 类提供了大量的属性和方法来执行不同的工作。

### Server 对象的属性和方法

HttpServerUtility 类的方法和属性通过由 ASP.NET 提供的内部服务器对象公开的。

以下表格提供了 HttpServerUtility 类一系列的属性。

| 属性 | 描述 |
| --- | --- |
| MachineName | 服务器电脑的名称 |
| ScriptTimeOut | 以秒为单位获取和设置请求超时的值 |

以下表格提供了一些重要的方法：

| 方法 | 描述 |
| --- | --- |
| CreateObject(String) | 创建一个 COM 对象的实例，由其 ProgID 验证。 |
| CreateObject(Type) | 创建一个 COM 对象的实例，由其 Type 验证。 |
| Equals(Object) | 决定具体的对象是否和现有对象一致。 |
| Execute(String) | 在当前请求的上下文中执行处理应用程序指定的虚拟路径。 |

| 方法 | 描述 |
| --- | --- |
| Execute(String, Boolean) | 在当前请求的上下文中执行处理程序指定的虚拟路径，指定是否清除 QueryString 及表单集合。 |
| GetLastError | 返回之前的异常。 |
| GetType | 获取现有实例的类型。 |
| HtmlEncode | 将一个普通的字符串变成合法的 HTML 字符串。 |
| HtmlDecode | 将一个 Html 字符串转化成一个普通的字符串。 |
| ToString | 返回一个表示当前对象的字符串。 |
| Transfer(String) | 对于当前请求，终止当前页面的执行并通过指定页面的 URL 路径，开始执行一个新页面。 |
| UrlDecode | 将一个 URL 字符串转化成一个普通的字符串。 |
| UrlEncodeToken | 与 UrlEncode 作用相同，但是在一个字节数组中，包含以 Base64 编码的数据。 |
| UrlDecodeToken | 与 UrlDecode 工作相同，但是在一个字节数组中，包含以 Base64 编码的数据。 |
| MapPath | 返回与指定的虚拟服务器上的文件路径相对应的物理路径。 |
| Transfer | 在当前应用程序上转移执行到另一个 web 页面。 |

## Request 对象

请求对象是 System.Web.HttpRequest 类的一个实例。它代表了 HTTP 请求的值和属性，使页面加载到浏览器中。

此对象所呈现的信息被封装在更高级别的抽象中（web 控件模型）。然而，这个对象可以帮助检查一些信息，例如客户端浏览器和信息记录程序。

### Request 对象的属性和方法

下表提供了请求对象一些值得注意的属性：

| 属性 | 描述 |
| --- | --- |
| AcceptTypes | 获取一个用户支持的 MIME 接受类型的字符串数组。 |
| ApplicationPath | 在服务器上获取 ASP.NET 应用程序的真实应用程序根路径。 |
| Browser | 获取或设置关于请求用户浏览器能力的信息。 |
| ContentEncoding | 获取或设置字符集的实体。 |
| ContentLength | 指定由客户端发送的内容的长度以字节为单位。 |
| ContentType | 获取或设置传入请求的 MIME 内容类型。 |
| Cookies | 获取客户端发送的 cookies 集合。 |
| FilePath | 获取当前请求的真实路径。 |
| Files | 以多部分的 MIME 格式获取客户端上传文件的集合。 |

| 属性 | 描述 |
| --- | --- |
| Form | 获取表单变量的集合。 |
| Headers | 获取 HTTP 标题的集合。 |
| HttpMethod | 获取用户使用的 HTTP 数据转移方法（如 GET，POST，或者 HEAD） |
| InputStream | 获取传入的 HTTP 的实体内容。 |
| IsSecureConnection | 获取一个值，该值指示 HTTP 连接是否使用安全套接字（即HTTPS）。 |
| QueryString | 获取 HTTP 询问字符串变量的集合。 |
| RawUrl | 获取当前请求的原始 URL。 |
| RequestType | 获取或设置由用户使用的 HTTP 数据转移方法（GET 或者 POST）。 |
| ServerVariables | 获取 Web 服务器变量的集合。 |
| TotalBytes | 获取现有输入流的字节数。 |
| Url | 获取关于现有请求的 URL 的信息。 |
| UrlReferrer | 获取关于与现有 URL 相链接的客户端之前的请求的 URL 信息。 |
| UserAgent | 获取客户端浏览器的原始用户代理字符串。 |
| UserHostAddress | 获取远程客户机的 IP 主机地址。 |
| UserHostName | 获取远程客户机的 DNS 名称。 |
| UserLanguages | 获取客户端语言首选项的排序字符串数组。 |

下表提供了一些重要的方法：

| 方法 | 描述 |
| --- | --- |
| BinaryRead | 从当前的输入流中执行一个指定字节数的二进制读数。 |
| Equals(Object) | 决定指定对象是否等同于现有对象。（继承自对象） |
| GetType | 获取现有实例的类型。 |
| MapImageCoordinates | 将传入的象场表单参数绘制成适当的 x 坐标和 y 坐标值。 |
| MapPath(String) | 将指定的真实路径绘制成一个物理路径。 |
| SaveAs | 在硬盘中存为一个 HTTP 请求。 |
| ToString | 返回一个代表现有对象的字符串。 |
| ValidateInput | 导致验证发生，通过访问 Cookies，Form，QueryString 属性的集合。 |

## Response 对象

响应对象代表了服务器对于用户请求的响应。它是 System.Web.HttpResponse 类的一个实例。

在 ASP.NET 中，响应对象在给用户发送 HTML 文本的过程中不扮演任何重要的角色，因为服务器端控件有嵌套的、面向对象的方法来自我呈现。

然而，HttpResponse 对象提供了一些重要的功能，比如 cookie 特点和 Redirect() 方法。 Response.Redirect() 方法允许将用户转移到另一个页面，在应用程序内部或应用程序外部均可。它需要一个往返过程。

### Response 对象的属性和方法

下表提供了一些响应对象值得注意的属性：

| 属性 | 描述 |
| --- | --- |
| Buffer | 获取或设置一个值，表明是否缓冲输出，并在完整的响应程序结束后将其发送。 |
| BufferOutput | 获取或设置一个值，表名是否缓冲输出，并在完整页面结束进城后将其发送。 |
| Charset | 获取或设置输出流的 HTTP 字符集。 |
| ContentEncoding | 获取或设置输出流的 HTTP 字符集。 |
| ContentType | 获取或设置输出流的 HTTP MIME 类型。 |
| Cookies | 获取相应 cookie 集合。 |
| Expires | 获取或设置一个浏览器上缓存的页面在到期前的分钟数。 |
| ExpiresAbsolute | 获取或设置从缓存中移除缓存信息的绝对日期和时间。 |
| HeaderEncoding | 获取或设置一个编码对象，代表现有标题输出流的编码。 |
| Headers | 获取响应标题的集合。 |
| IsClientConnected | 获取一个值，表明用户是否仍和服务器相连。 |
| Output | 使输出的文本到输出的 HTTP 响应流。 |
| OutputStream | 使二进制输出到输出的 HTTP 内容本体。 |
| RedirectLocation | 获取或设置 Http 标题位置的值。 |
| Status | 设置状态栏，返回给客户端。 |
| StatusCode | 获取或设置返回到客户端的 HTTP 输出状态码。 |
| StatusDescription | 获取或设置返回给客户端的 HTTP 输出状态字符串。 |
| SubStatusCode | 获取或设置一个值限制响应的状态码。 |
| SuppressContent | 获取或设置一个值，表明是否发送 HTTP 内容到客户端。 |

下表提供了一些重要的方法：

| 方法 | 描述 |
| --- | --- |
| AddHeader | 给输出流添加一个 HTTP 标题。提供 AddHeader 是为了 ASP 早期版本的兼容性。 |
| AppendCookie | 基础设施为内部 cookie 集合添加一个 HTTP cookie。 |
| AppendHeader | 给输出流添加一个 HTTP 标题。 |
| AppendToLog | 将自定义日志信息添加到 InterNET 信息服务（IIS）日志文件。 |
| BinaryWrite | 将一串二进制字符写入 HTTP 输出流。 |
| ClearContent | 清除缓冲流中的所有内容输出。 |
| Close | 关闭客户端套接字。 |
| End | 发送所有现有的缓冲输出给客户端，停止页面执行，并且引发 EndRequest 事件。 |
| Equals(Object) | 确定指定对象是否等同于现有对象。 |

| 方法 | 描述 |
| --- | --- |
| Flush | 发送所有现有缓冲输出到客户端。 |
| GetType | 获取现有实例的类型。 |
| Pics | 将一个 HTTP PICS-Label 标题附加到输出流。 |
| Redirect(String) | 将请求重定向到一个新的 URL 并指定新的 URL。 |
| Redirect(String, Boolean) | 将客户端重定向到一个新的 URL。指定新的 URL 并且之指定现有页面是否应该终止。 |
| SetCookie | 在 cookie 集合中更新现存 cookie。 |
| ToString | 返回代表现有对象的一个字符串 |
| TransmitFile(String) | 直接编写指定的文件到一个 HTTP 响应输出流中，不需要在内存中缓冲。 |
| Write(Char) | 编写一个字符到一个 HTTP 响应输出流中。 |
| Write(Object) | 编写一个对象到一个 HTTP 响应流中。 |
| Write(String) | 编写一个字符串到一个 HTTP 响应输出流中。 |
| WriteFile(String) | 直接编写指定文件的内容到一个 HTTP 响应输出流中，作为一个文件块。 |
| WriteFile(String, Boolean) | 直接编写指定文件的内容到一个 HTTP 响应输出流中，作为一个内存块。 |

## 示例

以下简单的例子有一个文本框控件，用户可以输入名称，一个按钮可以发送信息到服务器，还有一个标签控件来显示客户端计算机的 URL。

内容文件：

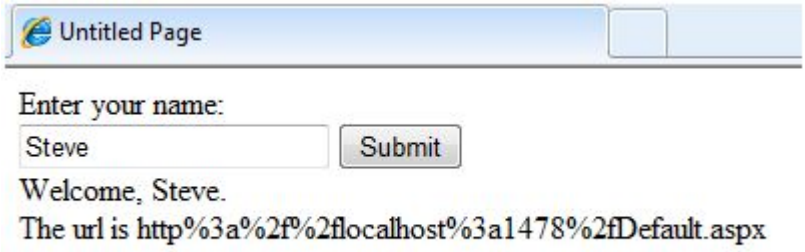
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
  Inherits="server_side._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>Untitled Page</title>
  </head>

  <body>
    <form id="form1" runat="server">
      <div>

        Enter your name:
        <br />
```

```
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Submit" />
        <br />
        <asp:Label ID="Label1" runat="server"/>

      </div>
    </form>
  </body>

</html>
```

Button1_Click 点击后的代码：

```
protected void Button1_Click(object sender, EventArgs e) {

  if (!String.IsNullOrEmpty(TextBox1.Text)) {

    // Access the HttpServerUtility methods through
    // the intrinsic Server object.
    Label1.Text = "Welcome, " + Server.HtmlEncode(TextBox1.Text) + ". <br/> The url is " + Server.UrlEncode(Request.U
  }
}
```

运行页面查看以下结果：




图片 6.1 image

# 7

# 服务器控件

控件是在图形用户界面中的小功能块，其中包括文本框，按钮，复选框，列表框，标签，和许多其它工具。利用这些工具，用户可以输入数据，进行选择并注明自己的喜好。

控件也用于结构性工作，如验证，数据访问，安全保证，创建母版页和数据操作。

ASP.NET 使用五种类型的 Web 控件，它们是：

- HTML 控件
- HTML 服务器控件
- ASP.NET 服务器控件
- ASP.NET Ajax 服务器控件
- 用户控件和自定义控件

ASP.NET 服务器控件是在 ASP.NET 中使用的主要控件。这些控件可被分成以下几类：

- **验证控件** – 用来验证用户输入，并通过运行客户端脚本进行工作。
- **数据源控件** – 提供数据绑定到不同的数据源功能。
- **数据视图控件** – 该控件为各种列表和表格，可以显示从数据源绑定的数据。
- **个性化控件** – 根据用户的喜好，基于用户信息进行页面个性化设置。
- **登陆和安全控件** – 提供用户身份验证。
- **母版页** – 提供整个应用程序一致的布局和界面。
- **导航控件** – 帮助用户导航。例如，菜单，树视图等。
- **丰富功能控件** – 实施特殊功能。例如：AdRotator, FileUpload, 和日历控件。

使用服务器控件的基本语法是：

```
<asp:controlType  ID ="ControlID" runat="server" Property1=value1  [Property2=value2] />
```

此外，Visual Studio还具有以下特点，以帮助产生无差错代码：

- 在设计视图中拖动和丢弃控件。
- 显示及自动完成特性的智能感知功能。
- 直接设置属性值的属性窗口。

## 服务器控件的属性

具有可视化功能的 ASP.NET 服务器控件来源于 WebControl 类，并且继承该类别的所有属性，事件以及方法。

WebControl 类本身以及其他不具有可视化功能的服务器控件都来源于 System.Web.UI.Control 类。例如，PlaceHolder 控件或 XML 控件。

ASP.Net 服务器控件继承了 WebControl 和 System.Web.UI.Control 类的所有属性，事件，以及方法。

下表显示了通用于所有服务器控件的属性：

| 属性 | 描述 |
| --- | --- |
| AccessKey | 同时按下该按键以及 Alt 键以将焦点移至控件。 |
| Attributes | 它是不对应控件属性的任意属性（仅用于视图呈现）的集合。 |
| BackColor | 背景色。 |
| BindingContainer | 包含数据绑定的控件。 |
| BorderColor | 边框颜色。 |
| BorderStyle | 边框样式。 |
| BorderWidth | 边框宽度。 |
| CausesValidation | 引起验证时显示。 |
| ChildControlCreated | 表示服务器控件的子控件是否建立。 |
| ClientID | HTML 标记的控件 ID。 |
| Context | 与服务器控件关联的 HttpContext 对象。 |
| Controls | 控件内全部控件的集合。 |
| ControlStyle | Web 服务器控件的样式。 |
| CssClass | CSS 类。 |
| DataItemContainer | 若命名器执行 IDataItemContainer，则为命名器提供参考。 |
| DataKeysContainer | 若命名器执行 IDataKeysControl，则为命名器提供参考。 |
| DesignMode | 表示控件在设计界面是否被使用。 |
| DisabledCssClass | 当控件禁用时，获取或设置 CSS 类来应用呈现的 HTML 元素。 |
| Enabled | 表示控件是否被禁用。 |
| EnableTheming | 表示主题是否适用于控件。 |
| EnableViewState | 表示是否维持控件的视图状态。 |
| Events | 获取代表控件的事件处理程序的列表。 |
| Font | 字体设定。 |
| Forecolor | 前景颜色。 |
| HasAttributes | 表示控件是否具有属性组。 |
| HasChildViewState | 表示当前服务器控件的子控件是否具有任何已保存的视图状态设置。 |

| 属性 | 描述 |
| --- | --- |
| Height | 高度的像素或百分比。 |
| ID | 控件的标识符。 |
| IsChildControlStateCleared | 表示包含在该控件内部的控件是否具有控件状态。 |
| IsEnabled | 获取表示控件是否被启用的值。 |
| IsTrackingViewState | 表示服务器控件是否会将更改保存到其视图状态。 |
| IsViewStateEnabled | 表示视图状态是否对该控件启用。 |
| LoadViewStateById | 表示控件是否是由 ID 而非索引来参与加载其视图状态。 |
| Page | 包含控件的页面。 |
| Parent | 家长控制功能。 |
| RenderingCompatibility | 指定呈现的 HTML 将与之兼容的 ASP.NET 版本。 |
| Site | 当设计界面显示时容纳当前控件的承载器。 |
| SkinID | 获取或设置适用于控件的皮肤。 |
| Style | 获取将在 Web 服务器控件的外部标签作为样式属性显示的文本属性的集合。 |
| TabIndex | 获取或设置 Web 服务器控件的索引标签。 |
| TagKey | 获取对应该 Web 服务器控件的 HtmlTextWriterTag 值。 |
| TagName | 获取控件标签的名称。 |
| TemplateControl | 包含该控件的模板。 |
| TemplateSourceDirectory | 获取页面的虚拟目录或包含在该控件中的控件。 |
| ToolTip | 获取或设置当鼠标指针停在 Web 服务器控件时显示的文本。 |
| UniqueID | 唯一的标识符。 |
| ViewState | 获取能够穿越同一页面的多重请求后保存和恢复服务器控件视图状态的状态信息词典。 |
| ViewStateIgnoreCase | 表示 StateBag 对象是否不区分大小写。 |
| ViewStateMode | 获取或设置该控件的视图状态。 |
| Visible | 表示服务器控件是否可见。 |
| Width | 获取或设置 Web 服务器控件的宽度。 |

## 服务器控件的方法

服务器控件的方法在以下表格中呈现：

| 方法 | 描述 |
| --- | --- |
| AddAttributesToRender | 添加需要呈现指定 HtmlTextWriterTag 的 HTML 属性和样式。 |
| AddedControl | 在子控件添加到控件对象的控件集合后调用。 |
| AddParsedSubObject | 通报服务器控件一个元素，XML 或 HTML 已被解析，并将该元素添加到服务器控件的控件集合。 |

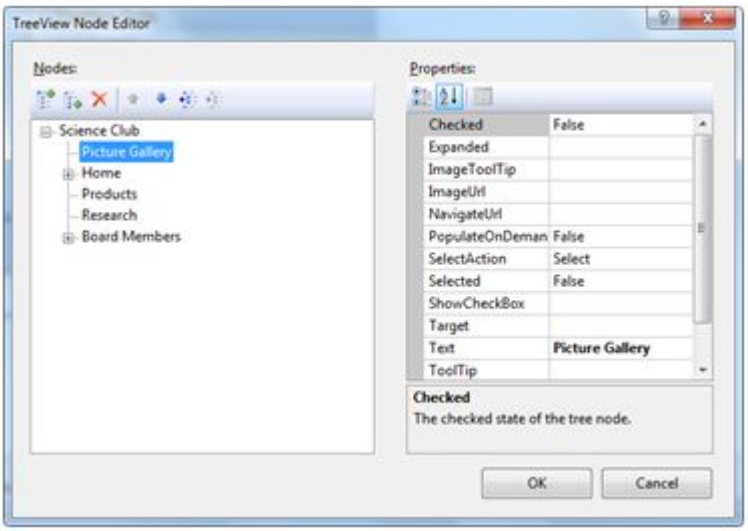
| 方法 | 描述 |
| --- | --- |
| ApplyStyleSheetSkin | 将在页面样式表中定义的样式属性应用到控件中。 |
| ClearCachedClientID | 基础设施。设置缓存的 ClientID 值设置为 null。 |
| ClearChildControlState | 为服务器控件的子控件删除控件状态信息。 |
| ClearChildState | 为所有服务器控件的子控件删除视图状态和控件状态信息。 |
| ClearChildViewState | 为所有服务器控件的子控件删除视图状态信息。 |
| CreateChildControls | 用于创建子控件。 |
| CreateControlCollection | 创建一个用于保存子控件的新控件集合。 |
| CreateControlStyle | 创建一个用于实现所有与样式有关的属性的样式对象。 |
| DataBind | 将数据源绑定到服务器控件及其所有子控件。 |
| DataBind(Boolean) | 将数据源及可以引发 DataBinding 事件的选项绑定到服务器控件及其所有子控件。 |
| DataBindChildren | 将数据源绑定到服务器控件的子控件。 |
| Dispose | 启用一个服务器控件在其从内存中释放出来前去执行最后的清理操作。 |
| EnsureChildControls | 确定服务器控件是否包含子控件。若没有，则创建子控件。 |
| EnsureID | 为没有标识符的控件创建一个标识符。 |
| Equals(Object) | 确定指定对象是否等于当前对象。 |
| Finalize | 允许一个对象去尝试释放资源并在对象被回收站回收前执行其他清理操作。 |
| FindControl(String) | 搜索当前命名容器中具有指定 id 参数的服务器控件。 |
| FindControl(String, Int32) | 搜索当前命名容器中具有指定 id 参数和整数的服务器控件。 |
| Focus | 为控件设置输入焦点。 |
| GetDesignModeState | 获取控件的设计时数据。 |
| GetType | 获取当前实例的类型。 |
| GetUniqueIDRelativeTo | 返回指定控件的唯一 ID 属性的预固定部分。 |
| HasControls | 确定服务器控件是否包含子控件。 |
| HasEvents | 表示事件是否被控件或其他子控件注册。 |
| IsLiteralContent | 确定服务器控件是否仅含有文字内容。 |
| LoadControlState | 恢复控件状态信息。 |
| LoadViewState | 恢复视图状态信息。 |
| MapPathSecure | 检索绝对的或相对的虚拟路径映射到的物理路径。 |
| MemberwiseClone | 创建当前对象的浅复制。 |
| MergeStyle | 复制指定样式的 Web 控件的任意非空白元素，但不覆盖该控件现有的任何样式元素。 |
| OnBubbleEvent | 确定服务器控件的事件是否通过页面的 UI 服务器控件层级。 |
| OnDataBinding | 引发数据绑定事件。 |
| OnInit | 引发 Init 事件。 |
| OnLoad | 引发加载事件。 |
| OnPreRender | 引发 PreRender 事件。 |

| 方法 | 描述 |
| --- | --- |
| OnUnload | 引发卸载事件。 |
| OpenFile | 获取用于读取文件的流。 |
| RemovedControl | 在子控件从控件对象的控件集合中移除后调用。 |
| Render | 显示控件到指定的 HTML 作者。 |
| RenderBeginTag | 显示控件的 HTML 开口标签到指定作者。 |
| RenderChildren | 输出服务器控件子级的内容到提供的 HtmlTextWriter 对象中，从而编写呈现在客户端上的内容。 |
| RenderContents | 显示控件内容到指定作者。 |
| RenderControl(HtmlTextWriter) | 输出服务器控件内容到提供的 HtmlTextWriter 对象并在启用跟踪的情况下保存关于控件的跟踪信息。 |
| RenderEndTag | 显示控件的 HTML 结束标签到指定作者。 |
| ResolveAdapter | 获取负责呈现指定控件的控件适配器。 |
| SaveControlState | 保存自页面回发到服务器后出现的服务器控件的状态改变。 |
| SaveViewState | 保存调用 TrackViewState 方法之后修改的任意状态。 |
| SetDesignModeState | 为控件设置设计时数据。 |
| ToString | 返回代表当前对象的字符串。 |
| TrackViewState | 引发控件跟踪其视图状态的变化，使其可以存储在该对象的视图状态属性中。 |

## 实例

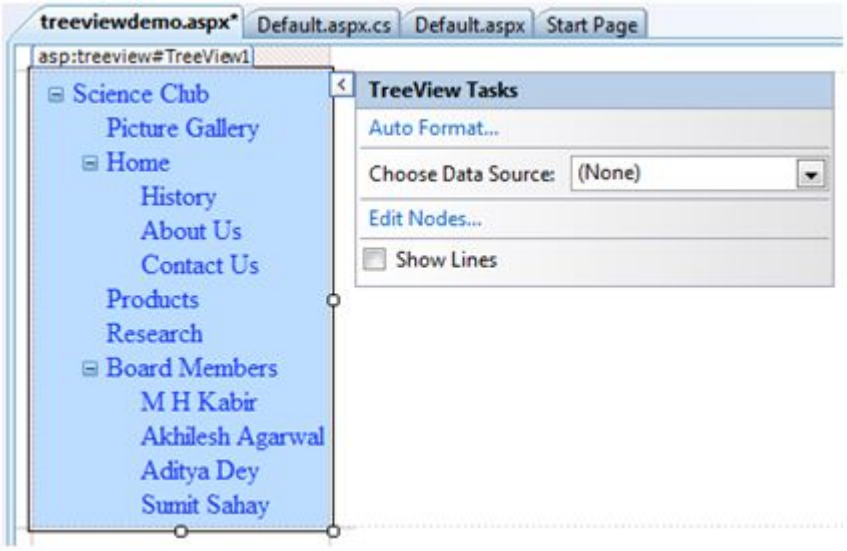
让我们来看一看一个特定的服务器控件 – 树型视图控件。树视图控件属于导航控件。其他导航控件是：菜单控件和 SiteMapPath 控件。

在页面上添加树视图控件。从任务中选择编辑结点...使用树视图结点编辑器编辑每个结点，如下所示：



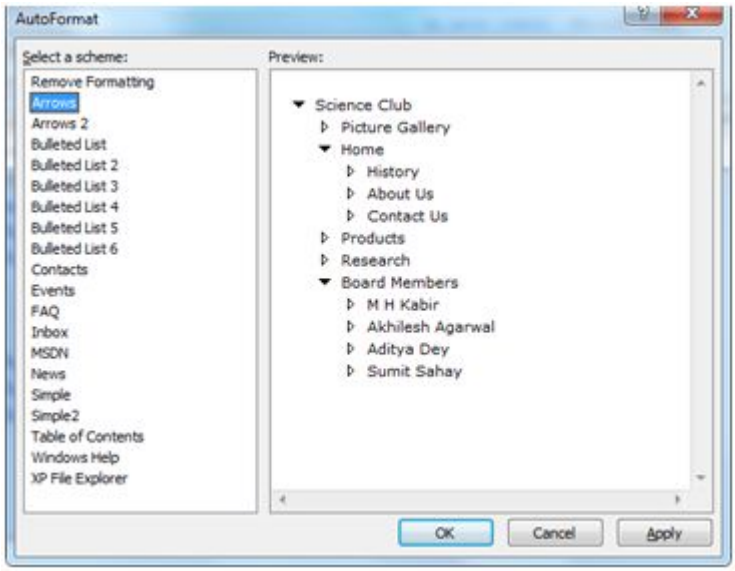

图片 7.1 image

当成功创建结点之后，设计视图下会有如下显示：




图片 7.2 image

AutoFormat... 任务允许您规定树视图的格式，如下所示：




图片 7.3 image

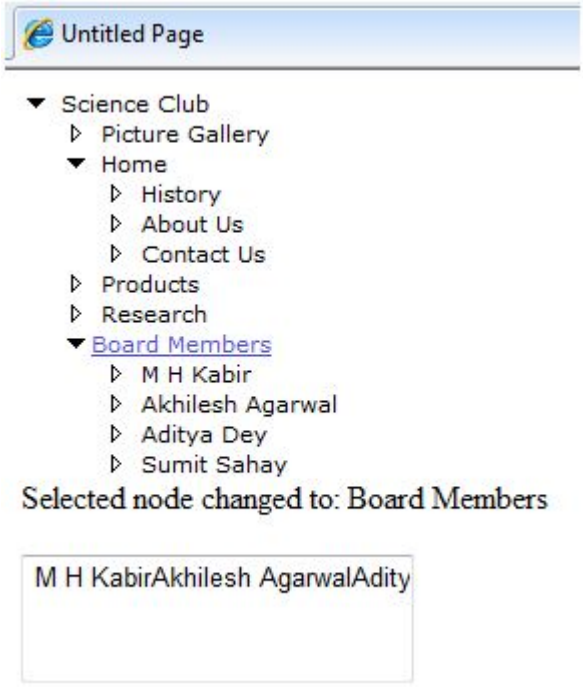
在页面上添加一个标签控件和文本框控件并分别命名为 lblmessage 和 txtmessage。

写几行代码，以确保当一个特定结点被选中时，标签控件显示结点文字且文本框显示所有其下的子结点（如有）。后台文件的代码应如下所示：

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
```

```
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace eventdemo {
  public partial class treeviewdemo : System.Web.UI.Page {

    protected void Page_Load(object sender, EventArgs e) {
      txtmessage.Text = " ";
    }

    protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e) {

      txtmessage.Text = " ";
      lblmessage.Text = "Selected node changed to: " + TreeView1.SelectedNode.Text;
      TreeNodeCollection childnodes = TreeView1.SelectedNode.ChildNodes;

      if(childnodes != null) {
        txtmessage.Text = " ";

        foreach (TreeNode t in childnodes) {
          txtmessage.Text += t.Value;
        }
      }
    }
  }
}
```

执行页面以观看效果，您将可以展开和折叠结点。


Untitled Page
Science Club
Picture Gallery
Home
History
About Us
Contact Us
Products
Research
Board Members
M H Kabir
Akhilesh Agarwal
Aditya Dey
Sumit Sahay
Selected node changed to: Board Members
M H KabirAkhilesh AgarwalAdity

图片 7.4 image

8

# HTML 服务器

HTML 服务器控件主要是保证服务端运行的增强型标准 HTML 控件。HTML 控件不是由服务器处理，而是被发送到浏览器进行显示，比如页面标题标签，链接标签及输入元素。

通过添加 runat = "server" 属性和一个 id 属性，它们可被特定地转化为一个服务器控件，应用于服务器端处理。

例如，HTML 输入控件：

```
<input type="text" size="40">
```

它可以通过添加 runat 和 id 属性被转换成一个服务器控件：

```
<input type="text" id="testtext" size="40" runat="server">
```

## 使用 HTML 服务器控件的优点

尽管 ASP.NET 服务器控件可以完成 HTML 服务器控件执行的每一项工作，HTML 控件在以下情况仍然具有优势：

- 使用静态表达到布局目的。
- 转换一个 HTML 页面到 ASP.NET 下运行。

下面这个表格介绍了 HTML 服务器控件：

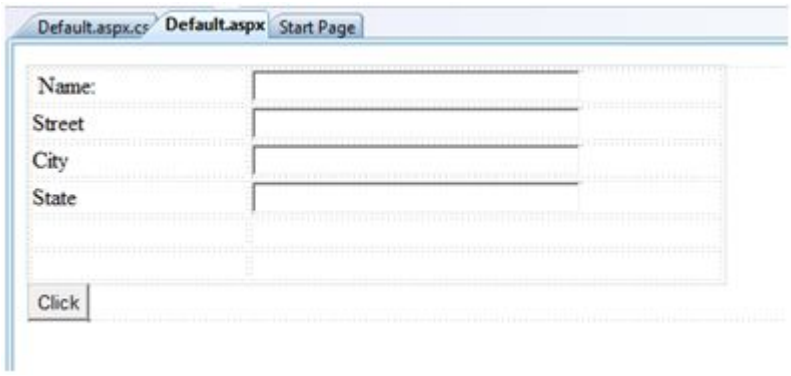
| 控件名称 | HTML 标签 |
| --- | --- |
| HtmlHead | <head>element |
| HtmlInputButton | <input type=button\|submit\|reset> |
| HtmlInputCheckbox | <input type=checkbox> |
| HtmlInputFile | <input type = file> |
| HtmlInputHidden | <input type = hidden> |
| HtmlInputImage | <input type = image> |
| HtmlInputPassword | <input type = password> |
| HtmlInputRadioButton | <input type = radio> |
| HtmlInputReset | <input type = reset> |
| HtmlText | <input type = text\|password> |
| HtmlImage | <img> element |
| HtmlLink | <link> element |
| HtmlAnchor | <a> element |
| HtmlButton | <button> element |
| HtmlButton | <button> element |
| HtmlForm | <form> element |

| | |
|---|---|
| HtmlTable | <table> element |
| HtmlTableCell | <td> and <th> |
| HtmlTableRow | <tr> element |
| HtmlTitle | <title> element |
| HtmlSelect | <select&t; element |
| HtmlGenericControl | 未列出的所有 HTML 控件 |

## 实例

以下实例使用了基本的 HTML 表格进行布局。它使用了用于从用户获得输入诸如姓名，地址，城市，州等的框，还有一个按钮控件，该控件被点击后能够获取该表最后一行中显示的用户数据。

页面在设计视图中应如下所示：




图片 8.1 image

内容页面的代码表明了 HTML 表格元素进行布局的应用。

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>Untitled Page</title>

    <style type="text/css">
      .style1
      {
        width: 156px;
      }
      .style2
      {
```

```
        width: 332px;
    }
  </style>

</head>

<body>
  <form id="form1" runat="server">
    <div>
      <table style="width: 54%;">
        <tr>
          <td class="style1">Name:</td>
          <td class="style2">
            <asp:TextBox ID="txtname" runat="server"  style="width:230px">
            </asp:TextBox>
          </td>
        </tr>

        <tr>
          <td class="style1">Street</td>
          <td class="style2">
            <asp:TextBox ID="txtstreet" runat="server"  style="width:230px">
            </asp:TextBox>
          </td>
        </tr>

        <tr>
          <td class="style1">City</td>
          <td class="style2">
            <asp:TextBox ID="txtcity" runat="server"  style="width:230px">
            </asp:TextBox>
          </td>
        </tr>

        <tr>
          <td class="style1">State</td>
          <td class="style2">
            <asp:TextBox ID="txtstate" runat="server" style="width:230px">
            </asp:TextBox>
          </td>
        </tr>

        <tr>
          <td class="style1"> </td>
          <td class="style2"></td>
```

```
            </tr>

            <tr>
              <td class="style1"></td>
              <td ID="displayrow" runat ="server" class="style2">
              </td>
            </tr>
          </table>

        </div>
        <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Click" />
      </form>
    </body>
</html>
```

按钮控件的后台代码为：

```
protected void Button1_Click(object sender, EventArgs e)
{
  string str = "";
  str += txtname.Text + "<br />";
  str += txtstreet.Text + "<br />";
  str += txtcity.Text + "<br />";
  str += txtstate.Text + "<br />";
  displayrow.InnerHtml = str;
}
```

观察以下陈述：

- 标准 HTML 标签已被使用进行页面布局。
- HTML 表格的最后一行用于数据显示。它需要服务器端进行加工，因此为其添加 ID 属性和 runat 属性。

# 9

# 客户端

ASP.NET 的客户端编码有两方面：

- **客户端脚本**：它在浏览器中运行并且依次加速页面的执行。例如，客户端数据有效性能够捕捉无效数据并相应地提醒用户而不经过在服务器中回发。
- **客户端源代码**：ASP.NET 网页形成了该客户端源代码。例如，ASP.NET 网页的 HTML 源代码包含了若干隐藏区域并能自动注入 Java 描述语言代码，从而保留了信息像视图状态一样，或者进行其他工作保证网页正常运作。

## 客户端脚本

所有 ASP.NET 服务器控件都允许响应通过 Java 语言或者 VBS 语言绘制的编码。有些 ASP.NET 服务器控件端使用客户端脚本进行对用户需求的反应，而并没有回发到服务器。例如，数据有效性控件。

除了这些脚本，按钮控件具有恰当的 OnClientClick 方法，能够在按钮单击时执行客户端脚本。

传统服务器 HTML 控件有以下几个事件能够在脚本发起时执行脚本：

| 事件 | 属性 |
| --- | --- |
| onblur | 当控件失去焦点时触发 |
| onfocus | 当控件获得焦点触发 |
| onclick | 当控件被单击时触发 |
| onchange | 当控件值发生改变时触发 |
| onkeydown | 当用户按下键盘按钮时触发 |
| onkeypress | 当用户按下字母数字的按键时 |
| onkeyup | 当用户释放按键时触发 |
| onmouseover | 当用户移动鼠标指针在控件界面时触发 |
| onserverclick | 当控件界面被单击时，启动 ServerClick 事件控件 |

## 客户端源代码

我们已经在以上内容中讨论过了客户端源代码。ASP.NET 网页通常被编写在两种文件中：

- 内容文件或者审定文件(.aspx)
- 代码后置的文件

内容文件包含 HTML 或者 ASP.NET 控件标签和文字来形成页面结构。代码后置的文件包含了分类定义。在运行时间，内容文件被解析并被传送到一个页面类。

这个页面类以及在编码文件中的类的定义和系统生成的编码共同组成执行编码（集成），这些集成编码加工所有的回发数据，产生响应和发回客户动作。

思考一下这个简单页面：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Click" />
      </div>

      <hr />

      <h3> <asp:Label ID="Msg" runat="server" Text=""> </asp:Label> </h3>
    </form>
  </body>

</html>
```

当这个页面在浏览器中运行时，View Source 选项显示了 HTML 网页并通过 ASP.Net 运行时间发送到浏览器：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head>
    <title>
```

```
      Untitled Page
    </title>
  </head>

  <body>
    <form name="form1" method="post" action="Default.aspx" id="form1">

      <div>
        <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
          value="/wEPDwUKMTU5MTA2ODYwOWRk31NudGDgvhhA7joJum9Qn5RxU2M=" />
      </div>

      <div>
        <input type="hidden" name="__EVENTVALIDATION"  id="__EVENTVALIDATION"
          value="/wEWAwKpjZj0DALs0bLrBgKM54rGBhHsyM61rraxE+KnBTCS8cd1QDJ/"/>
      </div>

      <div>
        <input name="TextBox1" type="text" id="TextBox1" />
        <input type="submit" name="Button1" value="Click" id="Button1" />
      </div>

      <hr />
      <h3><span id="Msg"></span></h3>

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

如果恰当地浏览编码，您就会发现前两个 \ 标签包含了存储的视图状态和有效数据的隐藏域。

10

# 基础控件

这在一章节，我们将讨论在 ASP.NET 中有效的基础控件。

## 按钮控件

ASP.NET 提供了三种不同类型的按钮控件：

- **按钮**：在矩形区域内显示文本。
- **链接按钮**：像超链接一样显示文本。
- **图像按钮**：显示图像。

当用户单击一个按钮时，两个事件被触发：单击和指令。

按钮控件的基础语法：

```
<asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Click" / >
```

按钮控件的通用属性：

| 属性 | 描述 |
| --- | --- |
| Text | 文本显示在按钮上。仅对于按钮和链环按钮的控件。 |
| ImageUrl | 仅对于图像按钮控件。这个图像是为了显示按钮。 |
| AlternateText | 仅对于图像按钮控件。如果浏览器无法显示图像，替换文本会显示。 |
| CausesValidation | 当用户单击按钮时确定是否执行页面验证。默认为真。 |
| CommandName | 当用户单击按钮时传递给命令事件的字符串值。 |
| CommandArgument | 当用户单击按钮时传递给命令事件的字符串值。 |
| PostBackUrl | 当用户单击按钮时出现需要的页面地址。 |

## 文本框和标签

文本框控件是专门接受用户输入而设置。一个文本框控件可以依据文本模式的属性接受一条或多条文本的输入。

标签控件为显示文本提供了一个简单的方法，这种方法能够从执行一个页面到下一个页面。如果想要显示一个不变的文本，那么您可以使用文字文本。

正文控制的基本语法：

```
<asp:TextBox ID="txtstate" runat="server" ></asp:TextBox>
```

文本框和标签的通用属性：

| 属性 | 描述 |
| --- | --- |
| TextMode | 指定文本框类型。单行模式创建标准文本，多行模式创建能够接受多个文本，口令会引发输入待标记的字符。默认为标准文本。 |
| Text | 文本框的文本内容。 |
| MaxLength | 输入文本框中文本字符的最大值。 |
| Wrap | 它确定多行文本框中文本是否自动换行的;默认值是真。 |
| ReadOnly | 确定用户是否可以更改框中的文本;默认为假，即用户可以更改文本。 |
| Columns | 在字符的文本框的宽度。实际宽度是基于用于文本输入的字体来确定。 |
| Rows | 多行文本框的高度。默认值是 0，表示一个单行文本框。 |

大多使用属性的标签控件是 'Text'，它代表在标签上显示的文本。

## 复选框和单选按钮

一个复选框将显示一个选项，用户可以选中或取消。单选按钮呈现一组用户可以只选择一个选项的选项组。

如果要创建一组单选按钮，您可以为每个单选按钮组中的组名属性指定相同的名称。如果一个以上的组需要呈现一个单一的形式，则指定每个组不同的组的名称。

如果您想按照最初显示的形式来选中复选框或单选按钮，可将其选中属性为 true。如果多个单选按钮在一组的属性设置为 true，则只有最后一个被认为是 true。

复选框的基本语法：

```
<asp:CheckBox ID= "chkoption" runat= "Server">
</asp:CheckBox>
```

单选按钮的基本语法：

```
<asp:RadioButton ID= "rdboption" runat= "Server">
</asp: RadioButton>
```

复选框和单选按钮的通用属性：

| 属性 | 描述 |
| --- | --- |
| Text | 在复选框或单选按钮旁边显示的文本。 |
| Checked | 制定是否被选中，默认为未选中。 |
| GroupName | 控件归属组的名称。 |

## 列表控件

ASP.NET 提供以下控件：

- 下拉式列表，
- 列表框，
- 单选按钮列表，
- 复选框列表，
- 项目符号列表。

这些控件让用户可以从一个或多个项目列表中选择。列表框和下拉列表包含一个或多个列表项。这些列表可以通过代码或者由 ListItemCollection 编辑器被加载。

列表框控件的基本语法：

```
<asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"    OnSelectedIndexChanged="ListBox1_SelectedIndexC
</asp:ListBox>
```

下拉列表控件的基本语法：

```
<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"   OnSelectedIndexChanged="DropDownL
</asp:DropDownList>
```

列表框和下拉列表的通用属性：

| 属性 | 描述 |
| --- | --- |
| Items | 代表了控件内项目的 ListItem 对象的集合。此属性回传 ListItemCollection 类型的对象。 |
| Rows | 指定在框中显示的项目数。如果实际的列表中比显示的列表包含更多的行，则滚动条会被添加。 |
| SelectedIndex | 当前所选项目的索引。如果一个以上的项目被选择，则第一个索引选择项目。如果没有选择项目，此属性的值为 -1。 |
| SelectedValue | 当前选定项的值。如果一个以上的项目被选择，则第一项的值被选择。如果没有选中的项，该属性的值是一个空字符串("")。 |
| SelectionMode | 表示一个列表框是否允许单个选择或多个选择。 |

每个列表项对象的通用属性：

| 属性 | 描述 |
| --- | --- |
| Text | 为项目所显示的文本。 |
| Selected | 表示项目是否被选定。 |

| 属性 | 描述 |
| --- | --- |
| Value | 与项目相关的一串字符。 |

需要重点关注的是：

- 如果您要在一个下拉列表或列表框中的项目工作，则需使用该控件的项目属性。此属性返回一个 ListItemCollection 对象，它包含该列表的所有项目。
- 当用户从下拉列表或列表框中选择一个不同的项目时，SelectedIndexChanged 事件被引发。

## ListItemCollection

ListItemCollection 对象是 ListItem 对象的集合。每个 ListItem 对象代表列表中的一个项目。在一个 ListItemCollection 中项目编号从 0 开始。

当一个列表框中的项目被加载过程中使用的字符串是比如：lstcolor.Items.Add ("Blue") 时，那么文字和列表项的值的属性设置是您指定的字符串值。为了以不同的方式设置，你必须创建一个列表项的对象，然后添加该项目到集合。

ListItemCollection 编辑器用于将项目添加到一个下拉列表或列表框。它被用来创建项目的静态列表。若要显示集合编辑器，则从智能标签菜单中选择编辑项目，或者选择控件，然后在属性窗口的项目属性中单击省略号按钮。

ListItemCollection 的通用属性：

| 属性 | 描述 |
| --- | --- |
| Item(integer) | 表示在指定索引处的项目的 ListItem 对象。 |
| Count | 在集合中项目的个数。 |

ListItemCollection 的基本方法：

|方法|描述| |:----|:---| |Add(string)|在集合的末端增加一个新的项目并为项目文本属性分配字符串参数。| |Add(ListItem)|在集合末端添加一个新的项目。| |Insert(integer, string)|在集合中指定索引位置插入项目，并为项目文本属性分配字符串参数。| |Insert(integer, ListItem)|在集合中指定索引中的位置插入项目。| |Remove(string)|移除与文本值相同的字符串的项目。| |Remove(ListItem)|移除指定的项目。| |RemoveAt(integer)|作为整数移除在指定索引中的项目。| |Clear|移除集合中所有项目。| |FindByValue(string)|传回与字符串值相同的项目。 FindByValue(Text)|传回与字符串文本相同的项目。|

## 单选按钮列表和复选框列表

单选按钮列表呈现互相排斥的选项列表。一个复选框列表列呈现独立选项的列表。这些控件包含 ListItem 对象的集合，它们可以通过控件的项目属性被参考。

单选按钮列表的基本语法：

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True"
  OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
</asp:RadioButtonList>
```

复选框列表的基本语法：

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server" AutoPostBack="True"
  OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
</asp:CheckBoxList>
```

复选框和单选按钮列表的通用属性：

| 属性 | 描述 |
| --- | --- |
| RepeatLayout | 该属性指定在提出格式化列表过程中是否使用标签或普通 HTML 流。默认为表格。 |
| RepeatDirection | 它指定了方向，在该方向中控件可以被重复。可用的值是水平和垂直。默认是垂直的。 |
| RepeatColumns | 当重复控件时，它指定了列的数字；默认为 0。 |

## 项目符号列表和编号列表

项目符号列表控件创建项目符号列表或编号列表。这些控件包含 ListItem 对象的集合，它们可以通过控件的项目属性被参考。

项目符号列表的基本语法：

```
<asp:BulletedList ID="BulletedList1" runat="server">
</asp:BulletedList>
```

项目符号列表的通用属性：

| 属性 | 描述 |
| --- | --- |
| BulletStyle | 该属性指定样式和项目编号的外观或者数字。 |
| RepeatDirection | 它指定了方向，在该方向中控件可以被重复。可用的值是水平和垂直。默认是垂直的。 |
| RepeatColumns | 当重复控件时，它指定了列的数字；默认为 0。 |

## 超链接控件

超链接控件就像 `HTML <a>` 元素。

超链接控件的基本语法：

```
<asp:HyperLink ID="HyperLink1" runat="server">
  HyperLink
</asp:HyperLink>
```

它具有以下属性：

| 属性 | 描述 |
| --- | --- |
| ImageUrl | 由控件显示的图像的路径。 |
| NavigateUrl | 目标链接地址。 |
| Text | 作为链接显示的文本。 |
| Target | 加载链接页面的窗口或框架。 |

## 图像控件

若图片无法显示，图像控件则在网页，或者一些替代文本上显示图片。

图像控件的基本语法：

```
<asp:Image ID="Image1" runat="server">
```

它具有以下重要属性：

| 属性 | 描述 |
| --- | --- |
| AlternateText | 图片不存在时显示替代文本。 |
| ImageAlign | 对齐选项控件。 |
| ImageUrl | 由控件显示的图像的路径。 |

# 11

# 指令

ASP.NET 指令是指定可选设置的说明，如注册一个自定义的控制和页面的语言。这些设置介绍了 NET Framework 如何处理单页表单(.aspx)或用户控件(.ascx)网页。

下达指令的基本语法：

```
<%@ directive_name attribute=value [attribute=value] %>
```

在这一部分中，我们将介绍 ASP.NET 指令，同时会在整个教程中应用大多数指令。

## 应用程序指令

应用指令定义特定应用程序的属性。它是在 global.aspx 文件的顶部提供。

应用程序指令的基本语法：

```
<%@ Application Language="C#" %>
```

应用程序指令的属性：

| 属性 | 描述 |
| --- | --- |
| Inherits | 从类的名称中继承。 |
| Description | 应用的文本描述。解析器和编译器忽略这一点。 |
| Language | 应用在代码组中的语言。 |

## 集合指令

集合指令链接着一个网页链接的组件或在分析时的应用程序。这可能会出现在整个应用类型链接 Global.asax 文件中，页面文件中，用于链接到另一个网页的用户控件中或用户控件中。

集合控件的基本语法是：

```
<%@ Assembly Name ="myassembly" %>
```

集合控件的属性是：

| 属性 | 描述 |
| --- | --- |
| Name | 被链接的集合组件的名称。 |
| Src | 源文件被动态链接和编辑的路径。 |

## 控制指令

控制指令是与用户控件一同使用并出现在用户控件(.ascx)文件中。

控制指令的基本语法是:

```
<%@ Control Language="C#"  EnableViewState="false" %>
```

控制指令的属性是:

| 属性 | 描述 |
| --- | --- |
| AutoEventWireup | 允许或禁用事件处理程序的自动关联的布尔值。 |
| ClassName | 控件的文件名。 |
| Debug | 许或禁用编辑调试符号的布尔值。 |
| Description | 控制页面的文字说明，被编译器忽略。 |
| EnableViewState | 页面请求为是否保持视图状态的布尔值。 |
| Explicit | 在 VB 语言下，告知编辑器使用选项显示模式。 |
| Inherits | 控制页面继承的类。 |
| Language | 编码和脚本的语言。 |
| Src | 代码隐藏类的文件名。 |
| Strict | 在 VB 语言下，告知编辑器使用选项标准模式。 |

## 工具指令

工具指令表明网页，母版页或者用户控制页必须执行具有详细说明的.Net 框架界面。

工具指令的基本语法是:

```
<%@ Implements  Interface="interface_name" %>
```

## 导入指令

导入指令导入一个命名空间到用户控制应用程序的页面。如果在 global.asax 文件中指定了 Import 指令，那么会将其应用到整个应用程序。如果它是在用户控制页面的网页中，则会将其应用到该网页或控件中。

导入指令的基本语法是:

```
<%@ namespace="System.Drawing" %>
```

## 主要指令

主要指令指定了一个页面文件作为主页。

样本主页指令的基本语法是：

```
<%@ MasterPage Language="C#"  AutoEventWireup="true"  CodeFile="SiteMater.master.cs" Inherits="SiteMaster"  %>
```

## MasterType 指令

MasterType 指令指定一个类名到页面的主属性，强化其类型。

母版式指令的基本语法是：

```
<%@ MasterType attribute="value"[attribute="value" ...]  %>
```

## 输出缓存指令

输出缓存指令控制网页或用户控件的输出缓存策略。

输出缓存指令的基本语法：

```
<%@ OutputCache Duration="15" VaryByParam="None"  %>
```

## 页面指令

页面指令定义特定的页面分析器和编译器的页面文件的属性。

页面指令的基本语法是：

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

页面指令的属性是：

| 属性 | 描述 |
| --- | --- |
| AutoEventWireup | 允许或禁用正在自动绑定到方法页面事件的布尔值;例如，Page_Load。 |
| Buffer | 允许或禁用 HTTP 响应缓冲的布尔值。 |
| ClassName | 页面的类别名称。 |
| ClientTarget | 服务器控件应呈现的内容的浏览器 |

| 属性 | 描述 |
| --- | --- |
| CodeFile | 代码隐藏文件的名称。 |
| Debug | 允许或禁止使用调试符号编译的布尔值。 |
| Description | 页面的文件说明，由解析器忽略。 |
| EnableSessionState | 启用或禁用页面会话状态为只读。 |
| EnableViewState | 允许或禁止跨页请求视图状态的布尔值。 |
| ErrorPage | 未经处理的页面异常发生的情况下的重定地址。 |
| Inherits | 后台代码或其他类的名称。 |
| Language | 代码的编程语言。 |
| Src | 后台代码类的文件名。 |
| Trace | 启用或禁用跟踪。 |
| TraceMode | 表示跟踪信息的显示方式，并按照时间或者类别排序。 |
| Transaction | 表示交易是否被支持。 |
| ValidateRequest | 表示所有输入数据是否被有效验证为 hardcoded 列表值得布尔值。 |

## 前页型指令

前页型指令为一个页面分配类别，使得该页面类型被强化。

前页型指令的样本的基本语法：

```
<%@ PreviousPageType attribute="value"[attribute="value" ...]  %>
```

## 参考指令

参考指令表明另一个页面或用户控件应编译和链接到当前页面。

参考指令的基本语法是：

```
<%@ Reference Page ="somepage.aspx" %>
```

## 注册指令

注册指令用于注册定制服务器控件和用户控件。

注册指令的基本语法是：

```
<%@ Register Src="~/footer.ascx" TagName="footer" TagPrefix="Tfooter" %>
```

12

# 管理状态

超文本传输协议(HTTP)是一种无状态协议。当客户端从服务器断开连接时，ASP.NET 引擎将丢弃页面对象。这样一来，每个 Web 应用程序能够扩展到同时用于大量请求，但是不会耗尽服务器内存。

然而，需要有一些技术来存储各个请求之间的信息并在需要时取回。这个信息则称为状态，即所有控件的当前值和在当前会话中当前用户使用的变量。

ASP.NET 管理四种状态：

- 视图状态
- 控制状态
- 会话状态
- 应用程序状态

## 视图状态

视图状态是页面及其所有控件的状态。它通过 ASP.NET 框架的反馈保持不变。

当一个页面被发送回客户端，这些页面变化的属性及其控件是确定的，并存储在名为 _VIEWSTATE 的一个隐藏输入字段的值内。当页面被再次回发时，_VIEWSTATE 字段随 HTTP 请求被发送到服务器。

视图状态可以对以下内容启用或者禁用：

- **整个应用程序**：设置 web.config 文件中 部分的 EnableViewState 属性。
- **一个页面**：设置页面指令的 EnableViewState 属性为 <%@ Page Language="C#" EnableViewState="false" %>
- **一个控件**：设置控件 .EnableViewState 属性。

它通过使用视图状态对象，该对象是由被一组视图状态项目定义的 StateBag 类别定义的。该 StateBag 是一种数据结构，包含属性值对并被存储为与对象相关联的字符串。

StateBag 类具有以下属性：

| 属性 | 描述 |
| --- | --- |
| Item(name) | 具有指定名称的视图状态的值，是 StateBag 的默认属性。 |
| Count | 状态集合中的项目名称。 |
| Keys | 集合中所有项目的密钥集合。 |
| Values | 集合中所有项目的值的集合。 |

StateBag 类具有以下方法：

| 方法 | 描述 |
| --- | --- |
| Add(name, value) | 添加一个项目到视图状态集合，更新现有项目。 |
| Clear | 移除集合中所有项目。 |
| Equals(Object) | 确定指定的对象是否等于当前对象。 |
| Finalize | 允许释放资源并执行其他清理操作。 |
| GetEnumerator | 返回存储在 StateBag 对象中重复的 StateItem 对象的密钥/值对的计数器。 |
| GetType | 获取当前实例的类型。 |
| IsItemDirty | 检查存储在 StateBag 对象以确认其是否已被修改。 |
| Remove(name) | 移除制定项目。 |
| SetDirty | 设置 StateBag 对象的状态以及每个由其包含的 StateItem 对象的 Dirty 属性。 |
| SetItemDirty | 为在 StateBag 对象中的指定 StateItem 对象设置 Dirty 属性。 |
| ToString | 返回代表状态包对象的字符串。 |

## 实例

以下实例说明了存储视图状态的字符串的概念。

让我们保持一个计数器，通过点击页面上的一个按钮，该计数器能够在每次页面被调回时递增。标签控件显示计数器的值。

标记文件代码如下所示：

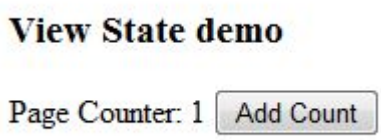
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="statedemo._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <h3>View State demo</h3>

        Page Counter:
```

```
        <asp:Label ID="lblCounter" runat="server" />
        <asp:Button ID="btnIncrement" runat="server" Text="Add Count" onclick="btnIncrement_Click" />
      </div>

    </form>
  </body>

</html>
```

该实例的后台代码文件如下所示：

```
public partial class _Default : System.Web.UI.Page
{
  public int counter
  {
    get
    {
      if (ViewState["pcounter"] != null)
      {
        return ((int)ViewState["pcounter"]);
      }
      else
      {
        return 0;
      }
    }

    set
    {
      ViewState["pcounter"] = value;
    }
  }

  protected void Page_Load(object sender, EventArgs e)
  {
    lblCounter.Text = counter.ToString();
    counter++;
  }
}
```

它将会生成以下结果：




图片 12.1 image

## 控制状态

控制状态不能被直接修改，存取或禁用。

## 会话状态

当用户连接到 ASP.NET 网站，一个新的会话对象将被创建。当会话状态开启时，新的会话状态会为每一个新的请求而创建。这种会话状态对象会成为运行环境中的一部分并可通过页面使用。

会话状态通常用于存储应用程序数据，比如详细目录，供应商清单，客户记录或购物车。它可以存储用户的信息及其偏好信息，并保存用户未决定的路径。

会话由 120 位的 SessionID 识别和跟踪，从客户端传递到服务器并且作为 cookie 或修改的 URL 回传。SessionID 是全球唯一的，随机的。

会话状态对象由 HttpSessionState 类创建，它定义会话状态项集合。

HttpSessionState 类具有以下属性：

| 属性 | 描述 |
| --- | --- |
| SessionID | 唯一的会话标识符。 |
| Item(name) | 具有指定名称的会话状态项的值，是 HttpSessionState 类的默认属性。 |
| Count | 会话状态集合中项的数量。 |
| TimeOut | 获取和设置时间量，几分钟内，在供应商停止会话状态前在请求间被允许。 |

HttpSessionState 类有以下方法：

| 方法 | 描述 |
| --- | --- |
| Add(name, value) | 添加新的项到会话状态集合。 |
| Clear | 移除会话状态集合中所有项。 |
| Remove(name) | 移除会话状态集合中的指定项。 |
| RemoveAll | 移除会话状态集合中所有密钥和值。 |
| RemoveAt | 从会话状态集合中删除指定索引处的项。 |

会话状态对象是一个名 – 值对，它可以从会话状态对象中存储和检索信息。同样地，您可以使用以下代码：

```
void StoreSessionInfo()
{
  String fromuser = TextBox1.Text;
  Session["fromuser"] = fromuser;
}

void RetrieveSessionInfo()
{
  String fromuser = Session["fromuser"];
  Label1.Text = fromuser;
}
```

以上代码只存储在会话词典对象中的字符串，但是，它可以存储所有原始数据类型和由原始数据类型组成的阵列，DataSet, DataTable, HashTable, 和图像对象，以及继承 ISerializable 对象的任意用户定义的类。

**实例**

以下实例说明了存储会话状态的概念。在页面上有两个按钮：输入字符串的文本框按钮和显示从最后一个会话中存储的文本的标签按钮。 标记文件代码如下所示：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">
      <div>
             

        <table style="width: 568px; height: 103px">

          <tr>
            <td style="width: 209px">
```

```
      <asp:Label ID="lblstr" runat="server" Text="Enter a String"  style="width:94px">
      </asp:Label>
    </td>

    <td style="width: 317px">
      <asp:TextBox ID="txtstr" runat="server" style="width:227px">
      </asp:TextBox>
    </td>
  </tr>

  <tr>
    <td style="width: 209px"> </td>
    <td style="width: 317px"> </td>
  </tr>

  <tr>
    <td style="width: 209px">
      <asp:Button ID="btnnrm" runat="server"
        Text="No action button" style="width:128px" />
    </td>

    <td style="width: 317px">
      <asp:Button ID="btnstr" runat="server"
        OnClick="btnstr_Click" Text="Submit the String" />
    </td>
  </tr>

  <tr>
    <td style="width: 209px">  </td>

    <td style="width: 317px">  </td>
  </tr>

  <tr>
    <td style="width: 209px">
      <asp:Label ID="lblsession" runat="server"  style="width:231px"  >
      </asp:Label>
    </td>

    <td style="width: 317px">  </td>
  </tr>

  <tr>
    <td style="width: 209px">
      <asp:Label ID="lblshstr" runat="server">
```

```
            </asp:Label>
          </td>

          <td style="width: 317px">  </td>
        </tr>

      </table>

    </div>
  </form>
 </body>
</html>
```

在设计视图中应有如下显示：


图片 12.2 image

后台代码如下：

```
public partial class _Default : System.Web.UI.Page
{
  String mystr;

  protected void Page_Load(object sender, EventArgs e)
  {
    this.lblshstr.Text = this.mystr;
    this.lblsession.Text = (String)this.Session["str"];
  }

  protected void btnstr_Click(object sender, EventArgs e)
  {
    this.mystr = this.txtstr.Text;
    this.Session["str"] = this.txtstr.Text;
    this.lblshstr.Text = this.mystr;
    this.lblsession.Text = (String)this.Session["str"];
  }
}
```

执行文件并观察其如何运行：


图片 12.3 image

## 应用程序状态

ASP.NET 应用程序是在 Web 服务器上所有网页，代码和单个虚拟目录的其他文件的集合。当信息被存储在应用程序状态，它可以供所有用户使用。

为了提供应用程序状态的使用，ASP.NET 从 HttpApplicationState 类中为每个应用程序创建一个应用程序状态对象，并将该对象存储在服务器内存中。该对象是由类文件 global.asax 表示。

应用程序状态主要被用于存储计数器，其他统计数据及税率，折扣率等所有应用程序数据，并存储用户访问网站的路径。

HttpApplicationState 类具有以下属性：

| 属性 | 描述 |
| --- | --- |
| Item(name) | 具有指定名称的应用程序项的值，是 HttpApplicationState 的默认属性。 |
| Count | 应用程序状态集合中项的数量。 |

HttpApplicationState 类具有以下方法：

| 方法 | 描述 |
| --- | --- |
| Add(name, value) | 添加新的项目到应用程序状态集合。 |
| Clear | 移除应用程序状态集合中的所有项。 |
| Remove(name) | 移除应用程序状态集合中的指定项。 |
| RemoveAll | 移除一个 HttpApplicationState 集合中所有对象。 |
| RemoveAt | 移除从由索引找到的集合中的一个 HttpApplicationState 对象。 |
| Lock() | 锁定应用程序状态集合以便只有当前用户可以访问。 |
| Unlock() | 解锁应用程序状态集合以便所有用户可以访问。 |

应用程序状态的数据通常是由为事件编写的处理程序维护：

- 应用程序开启
- 应用程序结束
- 应用程序错误
- 会话开始
- 会话结束

以下代码片段展示了用于存储应用程序状态信息的基本语法：

```
Void Application_Start(object sender, EventArgs e)
{
  Application["startMessage"] = "The application has started.";
}

Void Application_End(object sender, EventArgs e)
{
  Application["endtMessage"] = "The application has ended.";
}
```

# 13

# 验证器

ASP.NET 的有效性控制是验证用户输入的数据从而确保那些无用的、未经授权的、矛盾的数据不能被存储。

ASP.NET 提供了如下几个方面的验证控制：

- 必要字段验证器（RequiredFieldValidator）
- 范围验证器（RangeValidator）
- 比较验证器（CompareValidator）
- 正则表达式验证器（RegularExpressionValidator）
- 自定义验证器（CustomValidator）
- 验证摘要控件（ValidationSummary）

## BaseValidator 类

有效性验证的类从 BaseValidator 类中继承得到，因此它们继承了它的属性和方法。因此学习这个作为所有有效性控制的基础的基本类的属性和方法对于后续学习将有很大帮助：

| 组成部分 | 描述 |
| --- | --- |
| ControlToValidate | 获取或设置要验证的输入控件。 |
| Display | 说明错误提示如何显示。 |
| EnableClientScript | 说明客户端的是否采取了验证。 |
| Enabled | 开启或者关闭验证器。 |
| ErrorMessage | 说明错误字符串。 |
| Text | 如果验证失败将要显示的文本。 |
| IsValid | 说明控制值是否有效。 |
| SetFocusOnError | 在验证失败时是否将焦点设置到相关的输入控件上。 |
| ValidationGroup | 获取或设置此验证控件所属的验证组的名称。 |
| Validate | 对关联的输入控件执行验证并更新 IsValid 属性。 |

## RequiredFieldValidator 控制

RequiredFieldValidator 控制确保必填字段不为空。它主要和文本框绑定使得用户向文本框输入。

该控制的语法如下：

```
<asp:RequiredFieldValidator ID="rfvcandidate"
  runat="server" ControlToValidate ="ddlcandidate"
  ErrorMessage="Please choose a candidate"
```

```
  InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>
```

## RangeValidator 控件

RangeValidator 控件负责核实输入的值是否在预设的范围之内。

它有三种特定属性：

| 属性 | 描述 |
| --- | --- |
| 类型（Type） | 它定义了数据类型。可用的数据类型包括：Currency, Date, Double, Integer, 和 String |
| 最小值（MinimumValue） | 它指定了范围中的最小值 |
| 最大值（MaximumValue） | 它指定了范围中的最大值 |

这个控件的语法如下：

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
  ErrorMessage="Enter your class (6 – 12)" MaximumValue="12"
  MinimumValue="6" Type="Integer">

</asp:RangeValidator>
```

## CompareValidator 控件

CompareValidator 控件根据输入到另一个输入控件中的值、常量数值或正确的数据类型来验证值。

它有以下的特定属性：

| 属性 | 描述 |
| --- | --- |
| Type | 它定义了数据类型。 |
| ControlToCompare | 它指定了输入控制中需要比较的值。 |
| ValueToCompare | 它指定了输入控制中不变的值。 |
| Operator | 它指定了比较的运算符，可用的值包括：相等、不等、大于等于、小于、小于等于、数据类型检查。 |

这种控件的基本语法如下：

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
  ErrorMessage="CompareValidator">
```

```
</asp:CompareValidator>
```

## RegularExpressionValidator 控件

RegularExpressionValidator 控件允许通过和正则表达式匹配来确定输入的有效性。正则表达式在 Validation Expression 的属性里设置。

下表总结了正则表达式通常所用到的语法结构：

| 转义字符 | 描述 |
| --- | --- |
| \b | 和退格键匹配。 |
| \t | 和 tab 匹配。 |
| \r | 和回车键匹配。 |
| \v | 和垂直制表符匹配。 |
| \f | 和换页符匹配。 |
| \n | 和换行匹配。 |
| \ | 转义符。 |

除了简单的字符匹配，一类字符可以被设置成匹配的，这类字符叫做通配符。

| 通配符 | 描述 |
| --- | --- |
| . | 可以匹配除了 \n 之外的任意字符。 |
| [abcd] | 可以匹配集合中的任意字符。 |
| [^abcd] | 排除集合中的任意字符。 |
| [2-7a-mA-M] | 匹配特定范围内的任意字符。 |
| \w | 匹配任意字母数字字符组和下划线。 |
| \W | 匹配任何非单词字符。 |
| \s | 匹配如空格，制表位，换行等字符。 |
| \S | 匹配任何非空格的字符。 |
| \d | 匹配任何小数字符。 |
| \D | 匹配任何非小数字符。 |

量词可以表明字符出现的特定字数。

| 量词 | 描述 |
| --- | --- |
| * | 零或更多匹配。 |
| + | 一个或更多匹配。 |
| ? | 零或一匹配。 |
| {N} | N 匹配。 |

| 量词 | 描述 |
| --- | --- |
| {N,} | N 或更多匹配。 |
| {N,M} | 在 N 和 M 之间匹配。 |

该控件的基本语法如下：

```
<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"
  ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>
```

## CustomValidator 控件

CustomValidator 控件允许编写客户端和服务器端特定的验证例程来验证值。

客户端验证通过 ClientValidationFunction 来适当的完成。客户端验证例程应该用浏览器能够识别的脚本语言来编写，例如 JavaScript 或者 VBScript。

服务器端的验证例程应该由控件的 ServerValidate 事件处理器来生成。服务器端的验证例程应该用任意的 .Net 语言来编写，例如：C# 或 VB.Net。

这种控件的基本语法如下：

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
  ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>
```

## ValidationSummary 控件

ValidationSummary 控件并不会进行任何验证但是会在页面显示一个所有的错误的总结。这个总结可以显示出所有失败的验证控件的错误信息属性的值。

下面两个相互包含的属性列表列出来错误信息：

- ShowSummary：用特殊格式显示错误信息。
- ShowMessageBox：用单独的窗口显示错误信息。

这个控件的基本语法如下：

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
  DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />
```

## 验证组

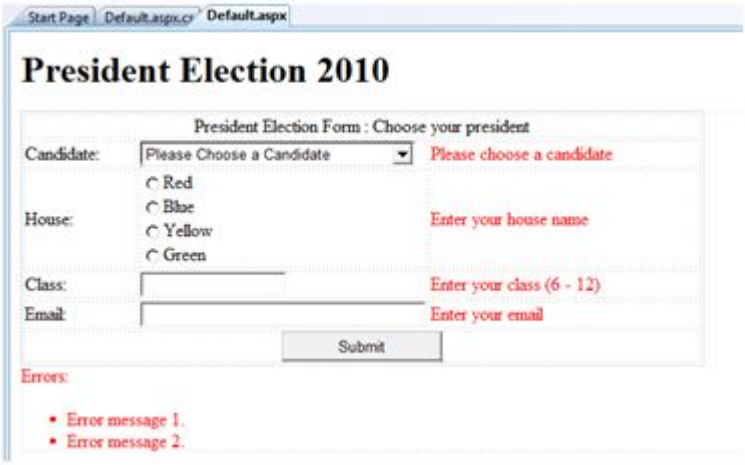
复杂的页面会在不同的层面有不同的信息组。在这种情况下，不同的组就要有不同的验证这种情况可以用验证组来解决。

创建一个验证组，你必须通过设置输入控件和验证控件的 ValidationGroup 属性从而把它们放到相同的逻辑组中。

## 例子

下面这个例子描述了一个将由全校学生填的表格，这个表格分为四部分是用来竞选校长的。在这里，我们将用验证控件来验证用户所输入的。

这是在设计视图下的形式：




这部分内容的代码如下：

```
<form id="form1" runat="server">

  <table style="width: 66%;">

    <tr>
      <td class="style1" colspan="3" align="center">
      <asp:Label ID="lblmsg"
        Text="President Election Form : Choose your president"
        runat="server" />
      </td>
    </tr>
```

```
<tr>
  <td class="style3">
    Candidate:
  </td>

  <td class="style2">
    <asp:DropDownList ID="ddlcandidate" runat="server"  style="width:239px">
      <asp:ListItem>Please Choose a Candidate</asp:ListItem>
      <asp:ListItem>M H Kabir</asp:ListItem>
      <asp:ListItem>Steve Taylor</asp:ListItem>
      <asp:ListItem>John Abraham</asp:ListItem>
      <asp:ListItem>Venus Williams</asp:ListItem>
    </asp:DropDownList>
  </td>

  <td>
    <asp:RequiredFieldValidator ID="rfvcandidate"
      runat="server" ControlToValidate ="ddlcandidate"
      ErrorMessage="Please choose a candidate"
      InitialValue="Please choose a candidate">
    </asp:RequiredFieldValidator>
  </td>
</tr>

<tr>
  <td class="style3">
    House:
  </td>

  <td class="style2">
    <asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
      <asp:ListItem>Red</asp:ListItem>
      <asp:ListItem>Blue</asp:ListItem>
      <asp:ListItem>Yellow</asp:ListItem>
      <asp:ListItem>Green</asp:ListItem>
    </asp:RadioButtonList>
  </td>

  <td>
    <asp:RequiredFieldValidator ID="rfvhouse" runat="server"
      ControlToValidate="rblhouse" ErrorMessage="Enter your house name" >
    </asp:RequiredFieldValidator>
    <br />
  </td>
</tr>
```

```
  <tr>
    <td class="style3">
      Class:
    </td>

    <td class="style2">
      <asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
    </td>

    <td>
      <asp:RangeValidator ID="rvclass"
        runat="server" ControlToValidate="txtclass"
        ErrorMessage="Enter your class (6 – 12)" MaximumValue="12"
        MinimumValue="6" Type="Integer">
      </asp:RangeValidator>
    </td>
  </tr>

  <tr>
    <td class="style3">
      Email:
    </td>

    <td class="style2">
      <asp:TextBox ID="txtemail" runat="server" style="width:250px">
      </asp:TextBox>
    </td>

    <td>
      <asp:RegularExpressionValidator ID="remail" runat="server"
        ControlToValidate="txtemail" ErrorMessage="Enter your email"
        ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
      </asp:RegularExpressionValidator>
    </td>
  </tr>

  <tr>
    <td class="style3" align="center" colspan="3">
      <asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit_Click"
        style="text-align: center" Text="Submit" style="width:140px" />
    </td>
  </tr>
</table>
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
```

```
    DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />
</form>
```

提交按钮的代码如下：

```
protected void btnsubmit_Click(object sender, EventArgs e)
{
  if (Page.IsValid)
  {
    lblmsg.Text = "Thank You";
  }
  else
  {
    lblmsg.Text = "Fill up all the fields";
  }
}
```

# 14

# 数据库存取

ASP.NET 允许存取和使用下列数据源：

- 数据库（例如：Access、 SQL Server、 Oracle、 MySQL ）
- XML 文档
- Business Objects
- Flat files

ASP.NET 隐藏了复杂的数据存取过程并且提供了更为高级的类和对象，通过他们数据可以更容易的存取。这些类隐藏了所有的连接，数据存取，数据检索和数据操纵的复杂的代码。

ADO.NET 技术提供了各种 ASP.NET 控件对象和后台数据之间的桥梁。在本指导中，我们着眼于数据存取并且简单的介绍数据。

## 检索和显示数据

在 ASP.NET 中检索和显示数据需要两种类型的数据控制：

- **数据源控制** – 它管理数据的连接、数据的选择和其他工作，例如数据的分页和缓存等等。
- **数据显示控制** – 这将约束和显示数据并且允许操作数据。

我们将在以后详细探讨数据约束和数据源控制。在本节中，我们将应用 SqlDataSource 控件存取数据。在本章用 GridView 控件显示和操作数据。

我们也会应用 Access 数据库，它包含了市场上有的 .Net 书籍的细节信息。将我们的数据库命名为 ASPDotNetStepByStep.mdb 并且我们将应用名为 DotNetReferences 的数据表。

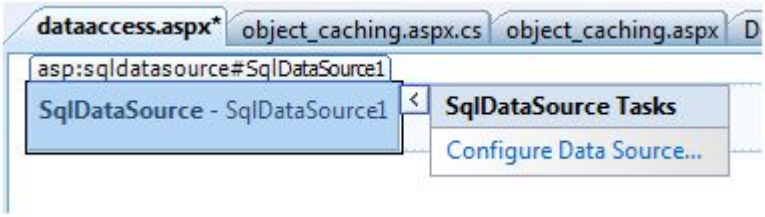
这张表包含了以下栏目：ID、Title、AuthorFirstName、AuthorLastName、Topic 和 Publisher。
下图是这个数据表的截图：

DotNetReferences

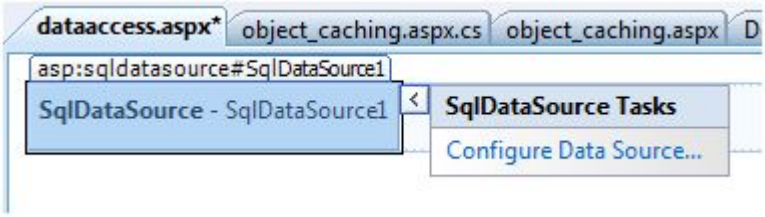
| ID | Title | AuthorLastNam | AuthorFirstNam | Topic |
|---|---|---|---|---|
| 1 | Essentials .NET | Box | Don | Gestalt of .NET |
| 2 | Programming Microsoft Visual C | Shepherd | George | C++ in the .NET World |
| 3 | ASP.NET Step by Step | Shepherd | George | ASP.NET from square one |
| 4 | Programming Microsoft ASP.NET | Esposito | Dino | ASP.NET comprehensive reference |
| 5 | Windows Forms Programming in | Sells | Chris | Windows Uis usin .NET |
| 6 | Applied Microsoft .NET Framewo | Richter | Jeffrey | Comprehensive .NET refrence |
| 7 | .NET Compact Framework Progra | Yao | Paul | How to do .NET on small devices |
| 8 | .NET Framework Essentials | Thai | Thuan | How to do .NET development |
| 9 | Microsoft Visual Basic .NET Progr | MacDonald | Matthew | Digestible Visual Basic examples |
| 10 | Designing Microsoft ASP.NET Ap | Reilly | Douglas | ASP.NET Design topics |
| 11 | The C | Hejlsberg | Anders | Definitive C# Reference |
| 12 | Programming Windows with C | Petzold | Charles | The original Windows programming autho |
| 13 | The CL Infrastructure Annotated | Miller | Jim | Info from someone really close to the CLR |

下面让我们直接按照下面步骤实践：

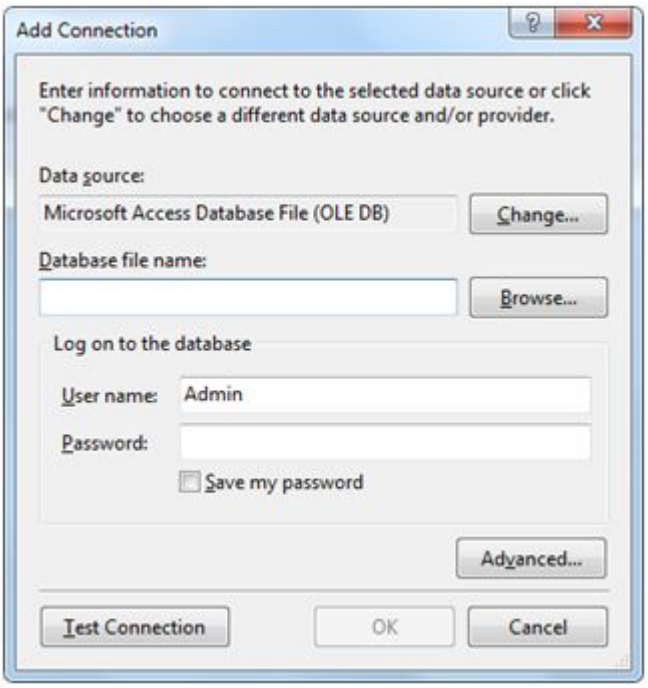
（1）创建一个网站并且在网页表格中添加 SqlDataSourceControl。



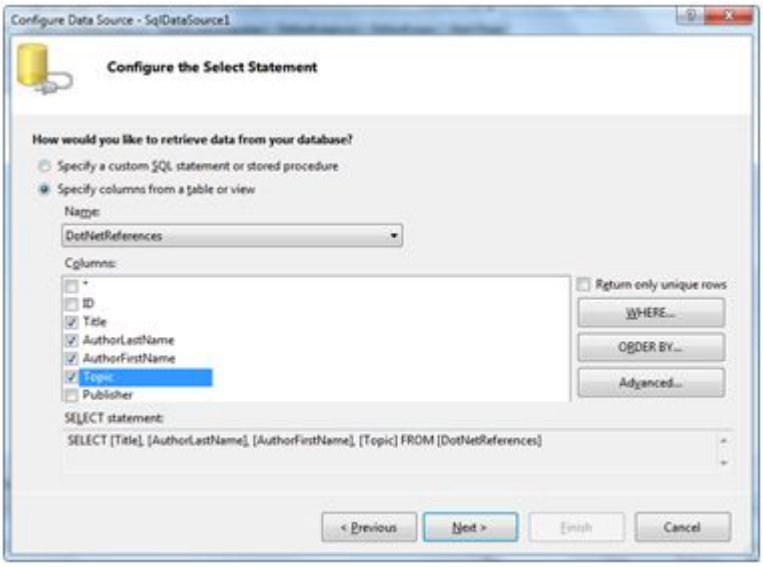

（2）单击 Configure Data Source 选项。




（3）点击 New Connection 按钮建立数据库连接。

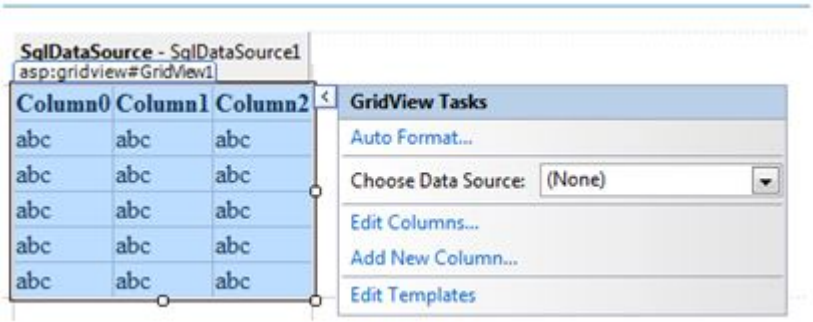



（4）连接一旦建立，你就可以保存他们以备以后应用。下一步，你会被要求设置 select statement：

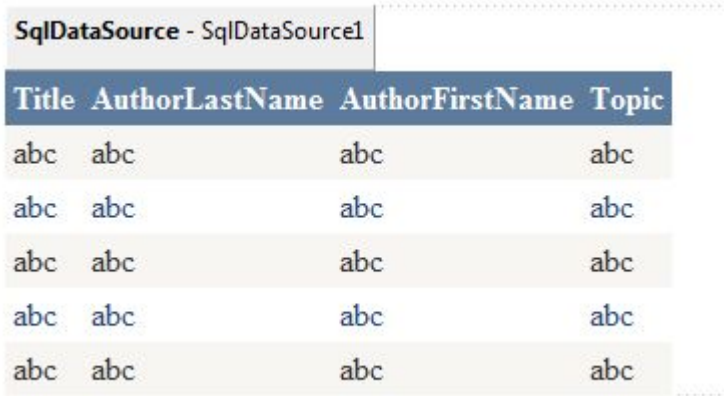



（5）选择好 columns 中的项目后点击 next 按钮完成剩余步骤。观察 WHERE, ORDER BY, 和 Advanced 按钮。这些按钮允许你执行 where 子句，order by 子句并且分别指定 SQL 中的插入，更新和删除命令。这样你就可以对数据进行操作了。

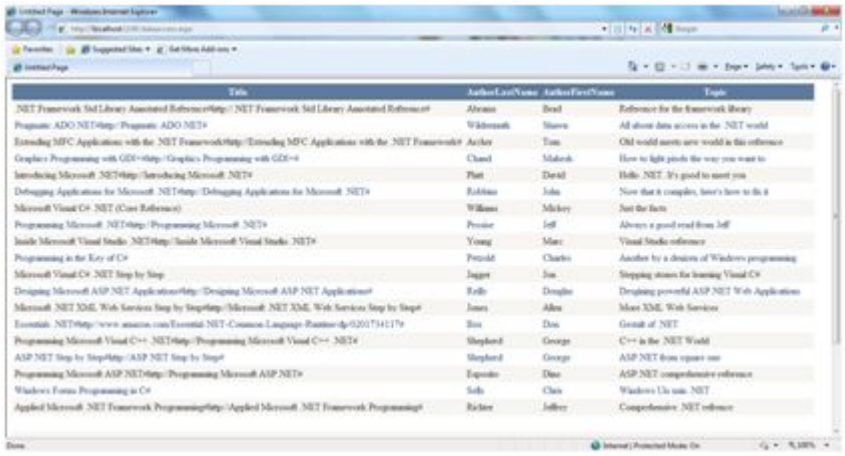
（6）在表中添加 GridView 控件。选择数据源并且用 AutoFormat 选项生成控件。




（7）在这之后设置的 GridView 控件可以显示栏目标题，这个程序就可以执行了。



（8）最后执行该程序。



以上涉及的代码列示如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="dataaccess.aspx.cs"
  Inherits="datacaching.WebForm1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">
      <div>

        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
          ConnectionString= "<%$   ConnectionStrings:ASPDotNetStepByStepConnectionString%>"
          ProviderName= "<%$ ConnectionStrings:
            ASPDotNetStepByStepConnectionString.ProviderName %>"
          SelectCommand="SELECT [Title], [AuthorLastName],
            [AuthorFirstName], [Topic] FROM [DotNetReferences]">
        </asp:SqlDataSource>

        <asp:GridView ID="GridView1" runat="server"
          AutoGenerateColumns="False" CellPadding="4"
          DataSourceID="SqlDataSource1" ForeColor="#333333"
```

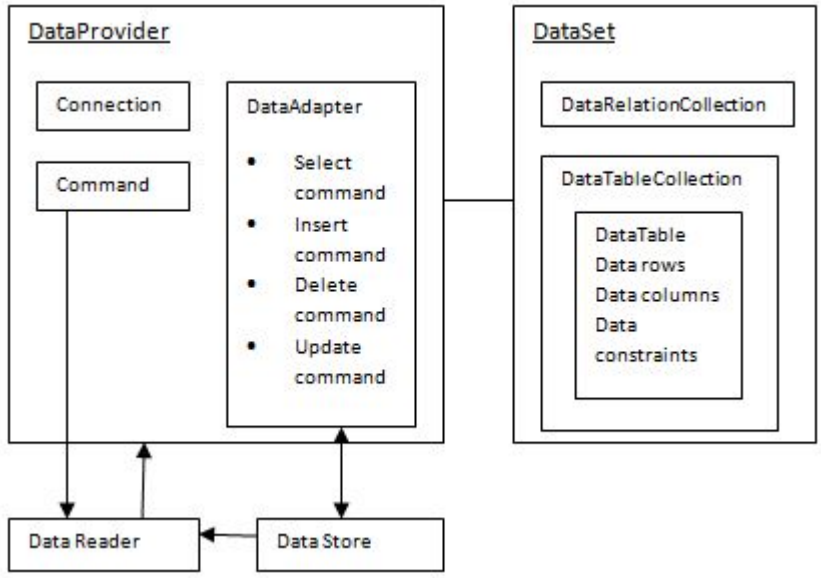
```
          GridLines="None">
          <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />

          <Columns>
            <asp:BoundField DataField="Title" HeaderText="Title"
              SortExpression="Title" />
            <asp:BoundField DataField="AuthorLastName"
              HeaderText="AuthorLastName" SortExpression="AuthorLastName" />
            <asp:BoundField DataField="AuthorFirstName"
              HeaderText="AuthorFirstName" SortExpression="AuthorFirstName" />
            <asp:BoundField DataField="Topic"
              HeaderText="Topic" SortExpression="Topic" />
          </Columns>
          <FooterStyle BackColor="#5D7B9D"
            Font-Bold="True" ForeColor="White" />
          <PagerStyle BackColor="#284775"
            ForeColor="White" HorizontalAlign="Center" />
          <SelectedRowStyle BackColor="#E2DED6"
            Font-Bold="True" ForeColor="#333333" />
          <HeaderStyle BackColor="#5D7B9D" Font-Bold="True"
            ForeColor="White" />
          <EditRowStyle BackColor="#999999" />
          <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
        </asp:GridView>
      </div>
    </form>
  </body>
</html>
```

# 15

# ADO.NET

ADO.NET 提供了前端控件和后端数据库之间的桥梁。ADO.NET 对象封装了与数据显示对象相互作用的所有的数据存取操作和控件。因此便隐藏了数据移动的细节。

下列图表整体展示了 ADO.NET 对象：




## DataSet 类

数据集代表了数据库的一个子集。它和数据库没有持续的连接。为了升级数据库需要进行重新连接。DataSet 包括了 DataTable 和 DataRelation 对象。DataRelation 对象代表了两张表之间的关系。

下表是 DataSet 类的一些重要属性：

| 属性 | 描述 |
| --- | --- |
| CaseSensitive | 说明和数据表进行对比的字符串是否区分大小写。 |
| Container | 为组件获取空间。 |
| DataSetName | 获取或者设置现有数据集合的名称。 |
| DefaultViewManager | 返回数据集合中的数据视图。 |
| DesignMode | 表明组件是否处于设计模式下。 |
| EnforceConstraints | 表明尝试上传文件时是否遵循限制条件。 |
| Events | 获取与本组件相关的事件处理器列表。 |
| ExtendedProperties | 获取与 DataSet 相关的自定义用户的信息的集合。 |
| HasErrors | 表明是否有任何错误。 |
| IsInitialized | 表明 DataSet 是否初始化。 |
| Locale | 获取或者设置用来和表比较字符串的信息。 |
| Namespace | 获取或者设置 DataSet 的命名空间。 |

| 属性 | 描述 |
| --- | --- |
| Prefix | 获取或者设置一个 XML 前缀，它是命名空间的别名。 |
| Relations | 返回 DataRelation 对象的集合。 |
| Tables | 返回 DataTable 对象的集合。 |

下表列出来 DataSet 类的一些重要方法：

| 方法 | 描述 |
| --- | --- |
| AcceptChanges | 接受所有由于装载 DataSet 或者这个方法的更改。 |
| BeginInit | 开始 DataSet 的初始化。该初始化发生在运行时。 |
| Clear | 清除数据。 |
| Clone | 克隆包括所有 DataTable 的结构、关系和限制在内的 DataSet 的结构。但是不克隆数据。 |
| Copy | 复制数据和结构。 |
| CreateDataReader() | 为每个 DataTable 返回带有一个结果集的 DataTableReader，顺序与 Tables 集合中表的显示顺序相同。 |
| CreateDataReader(DataTable[]) | 为每个 DataTable 返回带有一个结果集 DataTableReader。 |
| EndInit | 结束在窗体上使用或由另一个组件使用的 DataSet 的初始化。初始化发生在运行时。 |
| Equals(Object) | 确定指定的对象是否等于当前对象。 |
| Finalize | 释放资源执行其他清除。 |
| GetChanges | 获取 DataSet 的副本，该副本包含自加载以来或自上次调用 AcceptChanges 以来对该数据集进行的所有更改。 |
| GetChanges(DataRowState) | 获取由 DataRowState 筛选的 DataSet 的副本，该副本包含上次加载以来或调用 AcceptChanges 以来对该数据集进行的所有更改。 |
| GetDataSetSchema | 为 DataSet 获取 XmlSchemaSet 副本。 |
| GetObjectData | 用序列化 DataSet 所需的数据填充序列化信息对象。 |
| GetType | 获取当前实例的 Type。 |
| GetXML | 返回存储在 DataSet 中的数据的 XML 表示形式。 |
| GetXMLSchema | 返回存储在 DataSet 中的数据的 XML 表示形式的 XML 架构。 |
| HasChanges() | 获取一个值，该值指示 DataSet 是否有更改，包括新增行、已删除的行或已修改的行。 |
| HasChanges(DataRowState) | 获取一个值，该值指示 DataSet 是否有 DataRowState 被筛选的更改，包括新增行、已删除的行或已修改的行。 |
| IsBinarySerialized | 检查 DataSet 的序列化表示形式的格式。 |
| Load(IDataReader, LoadOption, DataTable[]) | 使用提供的 IDataReader 以数据源的值填充 DataSet，同时使用 DataTable 实例的数组提供架构和命名空间信息。 |
| Load(IDataReader, LoadOption, String[]) | 使用所提供的 IDataReader，并使用字符串数组为 DataSet 中的表提供名称，从而用来自数据源的值填充 DataSet。 |
| Merge() | 将指定的 DataSet、DataTable 或 DataRow 对象的数组合并到当前的 DataSet 或 DataTable 中。这种方法有不同的重载形式。 |

| 方法 | 描述 |
| --- | --- |
| ReadXML() | 将 XML 架构和数据读入 DataSet。这种方法有不同的重载形式。 |
| ReadXMLSchema(0) | 将 XML 架构读入 DataSet。这种方法有不同的重载形式。 |
| RejectChanges | 回滚自创建 DataSet 以来或上次调用 DataSet.AcceptChanges 以来对其进行的所有更改。 |
| WriteXML() | 从 DataSet 写 XML 数据和架构。这种方法有不同的重载形式。 |
| WriteXMLSchema() | 从 DataSet 写 XML 架构。这种方法有不同的重载形式。 |

## DataTable 类

DataTable 类代表了数据库中的表。它有如下的重要属性：大多数属性都是只读属性除了 PrimaryKey 属性：

| 属性 | 描述 |
| --- | --- |
| ChildRelations | 获取此 DataTable 的子关系的集合。 |
| Columns | 获取属于该表的列的集合。 |
| Constraints | 获取由该表维护的约束的集合。 |
| DataSet | 获取此表所属的 DataSet。 |
| DefaultView | 获取可能包括筛选视图或游标位置的表的自定义视图。 |
| ParentRelations | 获取该 DataTable 的父关系的集合。 |
| PrimaryKey | 获取或设置充当数据表主键的列的数组。 |
| Rows | 获取属于该表的行的集合。 |

下表列示出了一些 DataTable 类的重要方法：

| 方法 | 描述 |
| --- | --- |
| AcceptChanges | 提交自加载此 DataSet 或上次调用 AcceptChanges 以来对其进行的所有更改。 |
| Clear | 通过移除所有表中的所有行来清除任何数据的 DataSet。 |
| GetChanges | 获取 DataSet 的副本，该副本包含自上次加载以来或自调用 AcceptChanges 以来对该数据集进行的所有更改。 |
| GetErrors | 获取包含错误的 DataRow 对象的数组。 |
| ImportRows | 将 DataRow 复制到 DataTable 中，保留任何属性设置以及初始值和当前值。 |
| LoadDataRow | 查找和更新特定行。 如果找不到任何匹配行，则使用给定值创建新行。 |
| Merge | 将指定的 DataSet、DataTable 或 DataRow 对象的数组合并到当前的 DataSet 或 DataTable 中。 |
| NewRow | 创建与该表具有相同架构的新 DataRow。 |
| RejectChanges | 回滚自该表加载以来或上次调用 AcceptChanges 以来对该表进行的所有更改。 |
| Reset | 清除所有表并从 DataSet 中删除所有关系、外部约束和表。 子类应重写 Reset，以便将 DataSet 还原到其原始状态。 |

| 方法 | 描述 |
| --- | --- |
| Select | 获取 DataRow 对象的数组。 |

## DataRow 类

DataRow 对象代表了表中的一行，它有如下的重要属性：

| 属性 | 描述 |
| --- | --- |
| HasErrors | 表明是否有错误。 |
| Items | 获取或者设置存储在特定栏目的数据。 |
| ItemArrays | 获取或者设置本行中所有的值。 |
| Table | 返回父表。 |

下表列示了 DataRow 类的重要方法：

| 方法 | 描述 |
| --- | --- |
| AcceptChanges | 应用调用该方法后的所有更改。 |
| BeginEdit | 开始编辑操作。 |
| CancelEdit | 取消编辑操作。 |
| Delete | 删除数据行。 |
| EndEdit | 结束编辑操作。 |
| GetChildRows | 获取本行的子行。 |
| GetParentRow | 获取父行。 |
| GetParentRows | 获取 DataRow 的父行。 |
| RejectChanges | 回滚所有 AcceptChanges 调用后的更改。 |

## DataAdapter 对象

DataAdapter 对象扮演 DataSet 对象与数据库之间的中间者。这有助于 DataSet 从多种数据库或者其他数据源获取数据。

## DataReader 对象

DataReader 对象是 DataSet 和 DataAdapter 结合的备选。这个对象提供了对数据库中的数据记录的定向的存取。这些对象只适合只读存取，例如填充一个列表然后断开连接。

## DbCommand 和 DbConnection 对象

DbConnection 对象代表了数据源的连接。这种连接可以在不同的命令对象间共享。
DbCommand 对象代表了从检索或操纵数据发送到数据库的命令或者或者一个储存的进程。

## 例子

目前为止，我们已经应用了我们电脑中的表和数据库。在本案例中，我们将创建一个表，添加栏目，行和数据，并且用 GridView 控件显示表。

源文件代码如下：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
      </div>

    </form>
  </body>

</html>
```

文件的代码如下：

```
namespace createdatabase
{
  public partial class _Default : System.Web.UI.Page
```

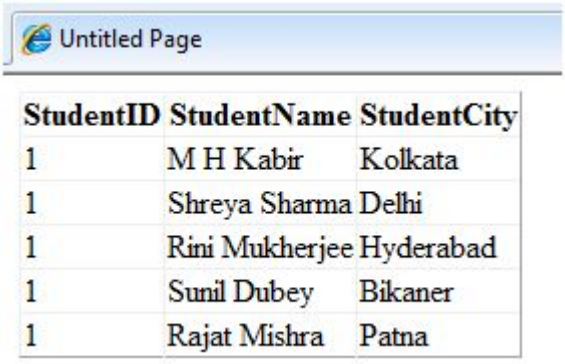
```
{
  protected void Page_Load(object sender, EventArgs e)
  {
    if (!IsPostBack)
    {
      DataSet ds = CreateDataSet();
      GridView1.DataSource = ds.Tables["Student"];
      GridView1.DataBind();
    }
  }

  private DataSet CreateDataSet()
  {
    //creating a DataSet object for tables
    DataSet dataset = new DataSet();

    // creating the student table
    DataTable Students = CreateStudentTable();
    dataset.Tables.Add(Students);
    return dataset;
  }

  private DataTable CreateStudentTable()
  {
    DataTable Students = new DataTable("Student");

    // adding columns
    AddNewColumn(Students, "System.Int32", "StudentID");
    AddNewColumn(Students, "System.String", "StudentName");
    AddNewColumn(Students, "System.String", "StudentCity");

    // adding rows
    AddNewRow(Students, 1, "M H Kabir", "Kolkata");
    AddNewRow(Students, 1, "Shreya Sharma", "Delhi");
    AddNewRow(Students, 1, "Rini Mukherjee", "Hyderabad");
    AddNewRow(Students, 1, "Sunil Dubey", "Bikaner");
    AddNewRow(Students, 1, "Rajat Mishra", "Patna");

    return Students;
  }

  private void AddNewColumn(DataTable table, string columnType, string  columnName)
  {
    DataColumn column = table.Columns.Add(columnName,  Type.GetType(columnType));
  }
```

```
    //adding data into the table
    private void AddNewRow(DataTable table, int id, string name, string city)
    {
      DataRow newrow = table.NewRow();
      newrow["StudentID"] = id;
      newrow["StudentName"] = name;
      newrow["StudentCity"] = city;
      table.Rows.Add(newrow);
    }
  }
}
```

当你执行程序时，观察以下几方面：

- 程序首先创建应一个数据集合然后用 GridView 控件的 DataBind() 方法约束它。
- Createdataset() 方法是用户定义功能，它可以创建一个新的 DataSet 对象并且调用其他的用户定义的 CreateStudentTable() 方法来创建表格然后将他们添加到数据集合的表集合中。
- CreateStudentTable() 方法调用用户定义的 AddNewColumn() 和 AddNewRow() 方法来创建表的栏目和行同时向行中添加数据。

  当页面得到执行，它返回的表的行如下图所示：



# 文件上传

ASP.NET 包含两个控件可以使用户向网页服务器上传文件。一旦服务器接受了上传的文件数据，那么应用程序就可以进行保存，进行检查或者忽略它。接下来的控件允许文件上传：

- HtmlInputFile – HTML 服务器控件
- FileUpload – ASP.NET 网页控件

两种控件都允许文件上传，但是 FileUpload 控件自动设置编码格式，然而 HtmlInputFile 控件并不会这样。

本指导中，我们将应用 FileUpload 控件。这个控件允许用户预览选择将要上传的文件，它提供了一个预览按钮和一个可以输入文件名的文本框。

一旦用户在文本框中输入文件名或者预览文件，FileUpload 控件的 SaveAs 方法就会将文件保存到硬盘。

FileUpload 的基本语法如下：

```
<asp:FileUpload ID= "Uploader" runat = "server" />
```

FileUpload 类是从 WebControl 类中得出的，并且它继承了它的所有元素，FileUpload 类具有以下这些只读属性：

| 属性 | 描述 |
| --- | --- |
| FileBytes | 返回一组将要上传文件的字节码 |
| FileContent | 返回将要被上传的的文件的流对象 |
| FileName | 返回将以上传的文件名称 |
| HasFile | 判断控件是否有文件需要上传 |
| PostedFile | 返回一个关于已上传文件的参考 |

发布的文件以 HttpPostedFile 形式的对象进行封装，这个对象可以通过 FileUpload 类的 PostedFile 属性被存取。

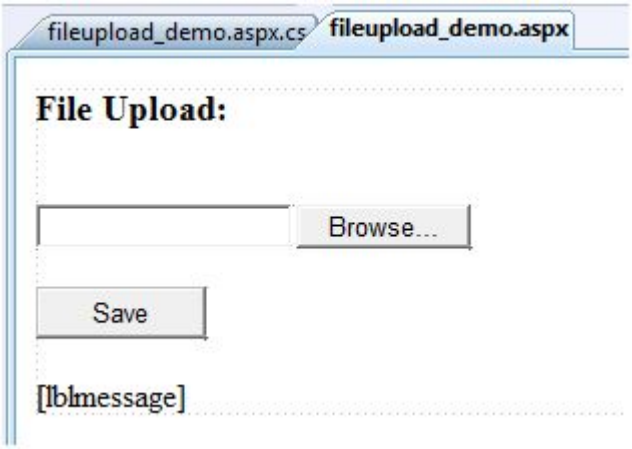
HttpPostedFile 类具有以下常用的属性：

| 属性 | 描述 |
| --- | --- |
| ContentLength | 返回已上传的文件的字节大小 |
| ContentT ype | 返回已上传的文件的 MIME 类型 |
| FileName | 返回文件全名 |
| InputStream | 返回将要被上传的的文件的流对象 |

## 例子

下面的例子说明了 FileUpload 控件以及它的属性。这个表格有一个 FileUpload 控件以及一个保存按钮和一个真实文件名称、类型、长度的标签控件。

在设计模式下，表格如下图所示：




相关文件代码列示如下：

```
<body>
  <form id="form1" runat="server">

    <div>
      <h3> File Upload:</h3>
      <br />
      <asp:FileUpload ID="FileUpload1" runat="server" />
      <br /><br />
      <asp:Button ID="btnsave" runat="server" onclick="btnsave_Click"  Text="Save" style="width:85px" />
      <br /><br />
      <asp:Label ID="lblmessage" runat="server" />
    </div>

  </form>
</body>
```

保存按钮的代码列示如下：

```
protected void btnsave_Click(object sender, EventArgs e)
{
  StringBuilder sb = new StringBuilder();
```

```
  if (FileUpload1.HasFile)
  {
    try
    {
      sb.AppendFormat(" Uploading file: {0}", FileUpload1.FileName);

      //saving the file
      FileUpload1.SaveAs("<c:\\SaveDirectory>" + FileUpload1.FileName);

      //Showing the file information
      sb.AppendFormat("<br/> Save As: {0}",  FileUpload1.PostedFile.FileName);
      sb.AppendFormat("<br/> File type: {0}",    FileUpload1.PostedFile.ContentType);
      sb.AppendFormat("<br/> File length: {0}",  FileUpload1.PostedFile.ContentLength);
      sb.AppendFormat("<br/> File name: {0}",  FileUpload1.PostedFile.FileName);

    }catch (Exception ex)
    {
      sb.Append("<br/> Error <br/>");
      sb.AppendFormat("Unable to save file <br/> {0}", ex.Message);
    }
  }
  else
  {
    lblmessage.Text = sb.ToString();
  }
}
```

注意以下问题：

- StringBuilder 类是由 System.IO 命名空间产生，所以应该包括它。
- try 和 catch 区域是用来捕捉错误、显示错误信息的。

17

# 广告轮转器

广告轮转控制器从一个列表里随机选择在外部 XML 定时文件中指定的横幅图像。这个外部 XML 定时文件被叫做广告文件。

广告轮转控件允许你指定一个广告文件和窗口的类型，链接应该分别遵循 AdvertisementFile 和 Target 的属性。

添加 AdRotator 的基本语法如下：

```
<asp:AdRotator  runat = "server" AdvertisementFile = "adfile.xml"  Target =  "_blank" />
```

在了解 AdRotator 控件以及它的属性之前，让我们先来看看广告文件的构成。

## 广告文件

广告文件是一种 XML 文件，它包括了广告所要被显示的信息。

可扩展标记语言（XML）是一种 W3C 的标准文本文档标记语言。它是一个基于文本的标记语言，它使您可以通过使用有意义的标签来让数据存储在结构化格式中。术语 'extensible' 意味着可以扩展功能，通过给应用程序定义有意义的标签来描述文档。

XML 本身不是一种语言，如 HTML ，而是一组用于创建新的标记语言的规则。它是一个元标记语言。它允许开发人员创建自定义标记集作特殊用途。它构建，存储并传输的信息。

下面是 XML 文件的一个例子：

```
<BOOK>
  <NAME> Learn XML </NAME>
  <AUTHOR> Samuel Peterson </AUTHOR>
  <PUBLISHER> NSS Publications </PUBLISHER>
  <PRICE> $30.00</PRICE>
</BOOK>
```

像所有的 XML 文件，该广告文件需要被具有良好定义并标记的结构化文本文件来描绘数据。这里也有一些在广告文件中常用的标准 XML 元素：

| 元素 | 描述 |
| --- | --- |
| Advertisements | 包围广告文件。 |
| Ad | 界定独立的广告。 |
| ImageUrl | 将要显示的图像的路径。 |
| NavigateUrl | 当用户点击该广告时出现的链接。 |
| AlternateText | 如果图像不能被显示，则会显示文本。 |
| Keyword | 关键字用来识别一组广告，用于过滤。 |

| 元素 | 描述 |
| --- | --- |
| Impressions | 该数字显示广告出现的频率。 |
| Height | 显示图像的高度。 |
| Width | 显示图像的宽度。 |

除了这些标签，带有一般属性的习惯性的标签也可以被包含进去。下面的代码演示了一个广告文件，ads.xml：

```
<Advertisements>
  <Ad>
    <ImageUrl>rose1.jpg</ImageUrl>
    <NavigateUrl>http://www.1800flowers.com</NavigateUrl>
    <AlternateText>
      Order flowers, roses, gifts and more
    </AlternateText>
    <Impressions>20</Impressions>
    <Keyword>flowers</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose2.jpg</ImageUrl>
    <NavigateUrl>http://www.babybouquets.com.au</NavigateUrl>
    <AlternateText>Order roses and flowers</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose3.jpg</ImageUrl>
    <NavigateUrl>http://www.flowers2moscow.com</NavigateUrl>
    <AlternateText>Send flowers to Russia</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>russia</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose4.jpg</ImageUrl>
    <NavigateUrl>http://www.ediblеblooms.com</NavigateUrl>
    <AlternateText>Edible Blooms</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>
</Advertisements>
```

## AdRotator 类的属性和事件

AdRotator 类是从 WebControl 类中派生的并且继承其属性。除了这些属性，AdRotator 类还具有以下属性：

| 属性 | 描述 |
| --- | --- |
| AdvertisementFile | 广告文件的路径。 |
| AlternateTextFeild | 提供替代文本的域的元素名称。默认值是 Alternate Text。 |
| DataMember | 当不使用广告文件时，要绑定的数据的特定列表的名称。 |
| DataSource | 控制检索数据。 |
| DataSourceID | 检索数据的控制 ID。 |
| Font | 指定与广告横幅控件相关联的字体属性。 |
| ImageUrlField | 提供 URL 图像的域的名称。默认值是 ImageUrl。 |
| KeywordFilter | 只显示基于关键字的广告。 |
| NavigateUrlField | 提供要导航到的 URL 的域的元素名称。默认值是 NavigateUrl。 |
| Target | 显示链接的网页的内容的浏览器窗口或框架。 |
| UniqueID | 获得 AdRotator 控件的唯一的、以分层形式限定的标识符。 |

以下是的 AdRotator 类的非常重要的事件：

| 事件 | 描述 |
| --- | --- |
| AdCreated | 每次往返服务器创建控件后，但是在页面渲染之前被触发。 |
| DataBinding | 当服务器控件绑定到数据源时触发。 |
| DataBound | 在服务器控件绑定到数据源之后发生。 |
| Disposed | 当服务器控件从内存释放，在服务器控件生命周期的最后一个阶段请求 ASP.NET 页时触发。 |
| Init | 当服务器控制被初始化时触发，其生命周期中的第一个步骤出现。 |
| Load | 当服务器控件加载到 Page 对象中时触发。 |
| PreRender | 加载 Control 对象之后，但在此之前呈现触发。 |
| Unload | 当服务器控件从内存中卸载时触发。 |

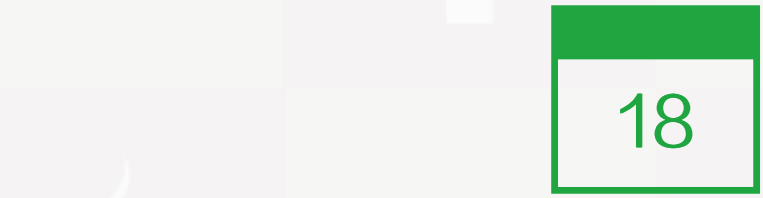
## 使用 AdRotator 控件

创建一个新的网页，并在其上放置一个 AdRotator 控件。

```
<form id="form1" runat="server">
  <div>
    <asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile  ="~/ads.xml" onadcreated="AdRotator1_AdCreate
  </div>
</form>
```

该 ads.xml 文件和图像文件应该位于网站的根目录。

试着执行上述应用程序，并观察到页面每次重载时，广告都被改变。



# 18
# 日历

日历控件是一个功能丰富的网络控件，它提供了以下功能：

- 一次显示一个月
- 选择一天，一个星期或一个月
- 选择某范围之内的一天
- 在月份和月份之间移动
- 格式化的控制天数的显示

日历控件的基本的语法为：

```
<asp:Calender ID = "Calendar1" runat = "server">
</asp:Calender>
```

## 日历控件的属性和事件

日历控件有很多的属性和事件，使用它们你可以自定义操作并且控制显示。下表提供了日历控件的一些重要的属性：

| 属性 | 描述 |
| --- | --- |
| Caption | 获取或设置日历控件的标题。 |
| CaptionAlign | 获取或设置标题的排列。 |
| CellPadding | 获取或设置数据和单元格边界之间的空间。 |
| CellSpacing | 获取或设置单元格之间的空间。 |
| DayHeaderStyle | 获得样式属性来显示一星期中的一天。 |
| DayNameFormat | 获取或设置星期中的日期。 |
| DayStyle | 获取样式属性来显示月份中的日期。 |
| FirstDayOfWeek | 获取或设置星期中的日期并显示在第一行。 |
| NextMonthText | 获取或设置下个月的导航文本，默认值是 >。 |
| NextPrevFormat | 获取或设置下个月或上个月的导航控件。 |
| OtherMonthDayStyle | 获取没有显示在月份中的日期的样式属性。 |
| PrevMonthText | 获取或设置上个月的导航文本，默认值是 <。 |
| SelectedDate | 获取或设置选中的日期。 |
| SelectedDates | 获取一个 DateTime 对象的集合代表所选日期。 |
| SelectedDayStyle | 获取选中日期的样式属性。 |
| SelectionMode | 获取或设置选择模式来指定用户是否可以选择一天，一周或是一个月。 |
| SelectMonthText | 获取或设置在选择器列中的选择月份元素的文本。 |

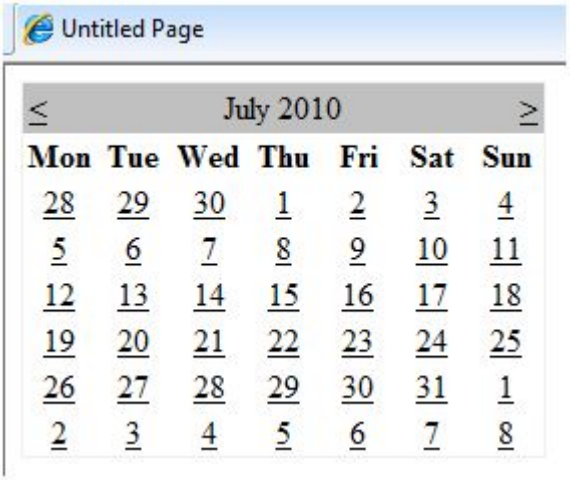
| 属性 | 描述 |
| --- | --- |
| SelectorStyle | 获取星期或月的选择器列的样式属性。 |
| SelectWeekText | 获取或设置选择器列中的星期选择元素的文本显示。 |
| ShowDayHeader | 获取或设置值，该值指示星期中日期的标题是否被显示。 |
| ShowGridLines | 获取或设置值显示网格线是否会被显示。 |
| ShowNextPrevMonth | 获取或设置一个值，该值指示下一个月和上一个月的导航元素是否在标题部分显示。 |
| ShowTitle | 获取或设置一个值，该值指示标题部分是否被显示。 |
| TitleFormat | 获取或设置标题的格式。 |
| Titlestyle | 获取日期控件的标题的样式属性。 |
| TodayDayStyle | 获取今天日期的样式属性。 |
| TodaysDate | 获取或设置今天的日期的值。 |
| UseAccessibleHeader | 获取或设置一个值，该值显示是否呈现表格标题 <th> HTML 元素给日期标头而不是表格数据 <td> HTML 元素。 |
| VisibleDate | 获取或设置指定月的日期并显示。 |
| WeekendDayStyle | 获取或设置周末日期的样式属性。 |

日期控件有以下三个最重要的事件来允许开发者编写日期控件。它们是：

| 事件 | 描述 |
| --- | --- |
| SelectionChanged | 当一天，一周或一个月被选中时，它会被触发。 |
| DayRender | 日历控件的每一个数据单元呈现时，它会被触发。 |
| VisibleMonthChanged | 用户更改月份时，它会被触发。 |

## 使用日历控件

使用一个没有任何代码的初始的日历控件给网站提供一个有效的日历，以显示一年中的月份和日期。它也含有下个月和上个月的导航。



图片 18.1 image

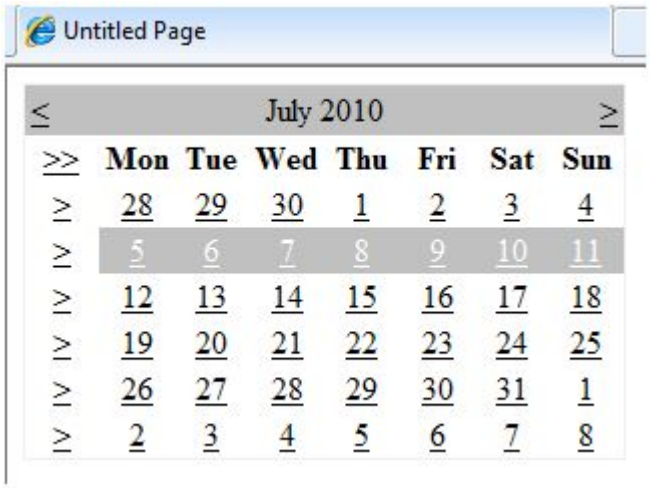
日历控件允许用户选择一天,一周,或一整个月。这是通过使用 SelectionMode 属性来实现的。这个属性有以下值:

| 属性 | 描述 |
| --- | --- |
| Day | 选择一天。 |
| DayWeek | 选择一天或一整个星期。 |
| DayWeekMonth | 选择一天一星期或一整个月。 |
| None | 什么也不能被选择。 |

选择日期的语法:

```
<asp:Calender ID = "Calendar1" runat = "server" SelectionMode="DayWeekMonth">
</asp:Calender>
```

当选择模式选择为 DayWeekMonth 时，会出现一个用符号 > 标识的额外的列来选择星期，并且 >> 符号出现在天名的左边来选择月份。




图片 18.2 image

## 例子

下面的例子演示了选择一个日期并且显示在一个标签内:

内容文件代码如下:

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <h3> Your Birthday:</h3>
        <asp:Calendar ID="Calendar1" runat="server  SelectionMode="DayWeekMonth" onselectionchanged="Calendar1_
        </asp:Calendar>
      </div>

      <p>Todays date is:
        <asp:Label ID="lblday" runat="server"></asp:Label>
      </p>

      <p>Your Birday is:
        <asp:Label ID="lblbday" runat="server"></asp:Label>
      </p>

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

事件处理程序的事件 SelectionChanged:

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
  lblday.Text = Calendar1.TodaysDate.ToShortDateString();
  lblbday.Text = Calendar1.SelectedDate.ToShortDateString();
}
```

运行该文件时,它将生成以下输出:

**Your Birthday:**

| < | December 2010 | | | | | | > |
|---|---|---|---|---|---|---|---|
| >> | **Mon** | **Tue** | **Wed** | **Thu** | **Fri** | **Sat** | **Sun** |
| > | 29 | 30 | 1 | 2 | 3 | 4 | 5 |
| > | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| > | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| > | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| > | 27 | 28 | 29 | 30 | 31 | 1 | 2 |
| > | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Todays date is: 11-07-2010

Your Birday is: 16-12-2010

图片 18.3 image

# 19 多视图

MultiView 和 View 控件允许你将一个页面的内容分成不同的组，一次只显示一组。每个视图控件管理一个组的内容，并且所有视图控件包括在 MultiView 控件中。

多视图控件一次只负责显示一个视图。视图显示称为活动视图。

MultiView 控件的语法是：

```
<asp:MultView ID= "MultiView1" runat= "server">
</asp:MultiView>
```

View 控制的语法是：

```
<asp:View ID= "View1" runat= "server">
</asp:View>
```

然而，该控件不能自行存在。如果您尝试单独使用它会出现错误。它总是和一个多视点控制器一起使用：

```
<asp:MultView ID= "MultiView1" runat= "server">
  <asp:View ID= "View1" runat= "server"> </asp:View>
</asp:MultiView>
```

## View 和 MultiView 控件的属性

视图和多视图控件都来源于 Control 类。并继承其所有属性、方法和事件。视图控件的最重要属性是可视 Boolean 属性，它设置了一个视图的可见性。

多视图控件具有以下重要特性：

| 属性 | 描述 |
| --- | --- |
| Views | 集多视图在内的视图控件。 |
| ActiveViewIndex | 从零开始的索引，它表示该活动视图。如果没有视图处于活动状态，那么索引值为 -1。 |

与 MultiView 控件的导航相关的按钮控制 CommandName 属性都与 MultiView 控件的一些相关字段关联。

例如，如果一个按钮控制的 CommandName 值作为与多视图的导航相关，单击按钮时它会自动导航到下一个视图中。

下表显示了上述属性的默认命令名：

| 元素 | 描述 |
| --- | --- |
| NextViewCommandName | 下一视图 |
| PreviousViewCommandName | 上一视图 |
| SwitchViewByIDCommandName | SwitchViewByID |

| 元素 | 描述 |
| --- | --- |
| SwitchViewByIndexCommandName | SwitchViewByIndex |

多视点控制的重要方法是：

| 方法 | 描述 |
| --- | --- |
| SetActiveview | 设置活动视图 |
| GetActiveview | 检索活动视图 |

每一个视图改变时，页面被回传到服务器，同时一些事件被引发。一些重要的事件是：

| 事件 | 描述 |
| --- | --- |
| ActiveViewChanged | 当一个视图发生改变时触发 |
| Activate | 通过活跃视图触发 |
| Deactivate | 通过不活跃视图触发 |

除了上面提到的属性、方法和事件，多视图控件继承了控制和对象类的成员。

## 例子

示例页面有三个视图。每个视图的导航视图有两个按钮。

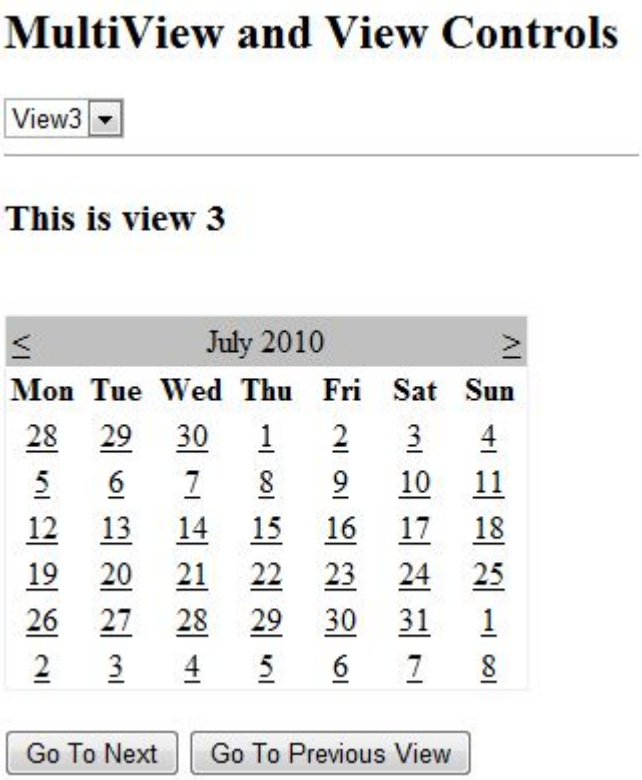
内容文件的代码如下：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <h2>MultiView and View Controls</h2>

        <asp:DropDownList ID="DropDownList1" runat="server" onselectedindexchanged="DropDownList1_SelectedIndex
        </asp:DropDownList>
```

```
        <hr />

        <asp:MultiView ID="MultiView1" runat="server" ActiveViewIndex="2"  onactiveviewchanged="MultiView1_ActiveVie
          <asp:View ID="View1" runat="server">
            <h3>This is view 1</h3>
            <br />
            <asp:Button CommandName="NextView" ID="btnnext1" runat="server" Text = "Go To Next" />
            <asp:Button CommandArgument="View3" CommandName="SwitchViewByID" ID="btnlast" runat="server" Text
          </asp:View>

          <asp:View ID="View2" runat="server">
            <h3>This is view 2</h3>
            <asp:Button CommandName="NextView" ID="btnnext2" runat="server" Text = "Go To Next" />
            <asp:Button CommandName="PrevView" ID="btnprevious2" runat="server" Text = "Go To Previous View" />
          </asp:View>

          <asp:View ID="View3" runat="server">
            <h3> This is view 3</h3>
            <br />
            <asp:Calendar ID="Calender1" runat="server"></asp:Calendar>
            <br />
            <asp:Button  CommandArgument="0" CommandName="SwitchViewByIndex" ID="btnfirst"   runat="server" Tex
            <asp:Button CommandName="PrevView" ID="btnprevious" runat="server" Text = "Go To Previous View" />
          </asp:View>

        </asp:MultiView>
      </div>

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

注意以下事项：

MultiView.ActiveViewIndex 确定了哪些视图将要显示。这是页面上呈现的唯一视图。没有视图显示时 ActiveViewIndex 的默认值是 −1。 由于范例中 ActiveViewIndex 被定义为 2，所以被执行时它显示的是第三个视图。


MultiView and View Controls
View3
This is view 3
≤ July 2010 ≥
Mon Tue Wed Thu Fri Sat Sun
28 29 30 1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31 1
2 3 4 5 6 7 8
Go To Next
Go To Previous View

图片 19.1 image

# 20

# Panel 控件

Panel 控件可以作为一个页面上的其他控件的容器。它控制其包含的控件的外观和可视度。它还允许生成控件编程。

面板控件的基本语法如下：

```
<asp:Panel ID= "Panel1"  runat = "server">
</asp:Panel>
```

面板控件从 WebControl 类派生。因此，它同样地继承了所有的属性、方法和事件。它不具有任何自己的方法或事件。然而，它有自己的以下属性：

| 属性 | 描述 |
| --- | --- |
| BackImageUrl | 面板背景图像的地址。 |
| DefaultButton | 获取或设置包含在 Panel 控件的默认按钮的标识符。 |
| Direction | 面板中的文本方向。 |
| GroupingText | 允许文本作为一个字段分组。 |
| HorizontalAlign | 水平对齐面板中的内容。 |
| ScrollBars | 指定面板内滚动条的可见性和位置。 |
| Wrap | 允许文本换行。 |

## 使用面板控件

让我们从一个具体的高度和宽度、边框样式简单的滚动面板开始。滚动条属性设置为两个滚动条，因此两个滚动条同时被呈现。

源文件具有如下的面板标签代码：

```
<asp:Panel ID="Panel1" runat="server" BorderColor="#990000" BorderStyle="Solid"
  Borderstyle="width:1px" Height="116px" ScrollBars="Both" style="width:278px">

  This is a scrollable panel.
  <br />
  <br />

  <asp:Button ID="btnpanel" runat="server" Text="Button" style="width:82px" />
</asp:Panel>
```

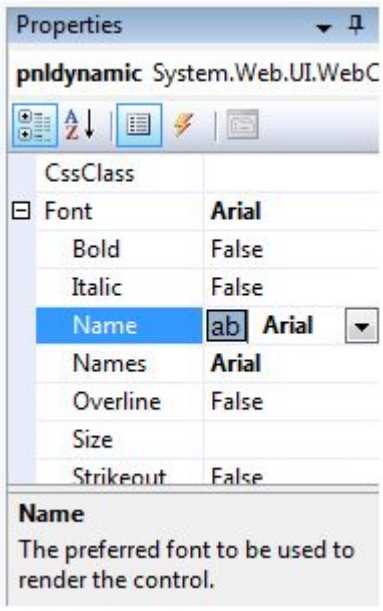
面板呈现如下：


图片 20.1 image

## 例子

下面的例子演示了动态内容生成。用户提供要在面板上产生的标签控件和文本框的数目。控件以编程方式生成。

用属性窗口更改面板属性。当您在设计视图中选择一个控件时，该属性窗口中将显示特定控件的属性，并允许您更改，而无需键入。




图片 20.2 image

示例的源文件如下：

```
<form id="form1" runat="server">
  <div>
    <asp:Panel ID="pnldynamic" runat="server" BorderColor="#990000"
      BorderStyle="Solid" Borderstyle="width:1px" Height="150px"  ScrollBars="Auto" style="width:60%" BackColor="#CCC

      This panel shows dynamic control generation:
      <br />
      <br />
```

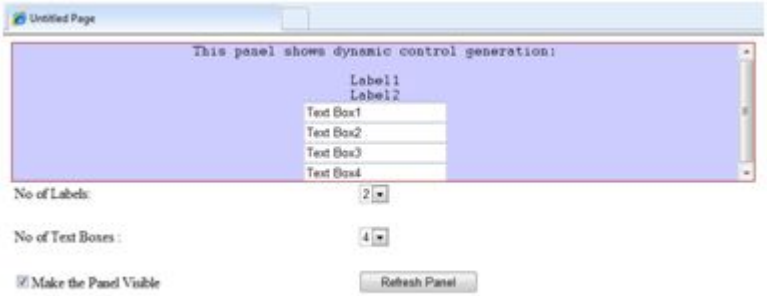
```
  </asp:Panel>
</div>

<table style="width: 51%;">
  <tr>
    <td class="style2">No of Labels:</td>
    <td class="style1">
      <asp:DropDownList ID="ddllabels" runat="server">
        <asp:ListItem>0</asp:ListItem>
        <asp:ListItem>1</asp:ListItem>
        <asp:ListItem>2</asp:ListItem>
        <asp:ListItem>3</asp:ListItem>
        <asp:ListItem>4</asp:ListItem>
      </asp:DropDownList>
    </td>
  </tr>

  <tr>
    <td class="style2"> </td>
    <td class="style1"> </td>
  </tr>

  <tr>
    <td class="style2">No of Text Boxes :</td>
    <td class="style1">
      <asp:DropDownList ID="ddltextbox" runat="server">
        <asp:ListItem>0</asp:ListItem>
        <asp:ListItem Value="1"></asp:ListItem>
        <asp:ListItem>2</asp:ListItem>
        <asp:ListItem>3</asp:ListItem>
        <asp:ListItem Value="4"></asp:ListItem>
      </asp:DropDownList>
    </td>
  </tr>

  <tr>
    <td class="style2"> </td>
    <td class="style1"> </td>
  </tr>

  <tr>
    <td class="style2">
      <asp:CheckBox ID="chkvisible" runat="server"
        Text="Make the Panel Visible" />
    </td>
```

```
      <td class="style1">
        <asp:Button ID="btnrefresh" runat="server" Text="Refresh Panel"
          style="width:129px" />
      </td>
    </tr>
  </table>
</form>
```

在 Page_Load 事件背后的负责动态生成的控件的源代码为：

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    //make the panel visible
    pnldynamic.Visible = chkvisible.Checked;

    //generating the lable controls:
    int n = Int32.Parse(ddllabels.SelectedItem.Value);
    for (int i = 1; i <= n; i++)
    {
      Label lbl = new Label();
      lbl.Text = "Label" + (i).ToString();
      pnldynamic.Controls.Add(lbl);
      pnldynamic.Controls.Add(new LiteralControl("<br />"));
    }

    //generating the text box controls:

    int m = Int32.Parse(ddltextbox.SelectedItem.Value);
    for (int i = 1; i <= m; i++)
    {
      TextBox txt = new TextBox();
      txt.Text = "Text Box" + (i).ToString();
      pnldynamic.Controls.Add(txt);
      pnldynamic.Controls.Add(new LiteralControl("<br />"));
    }
  }
}
```

当被执行时，面板呈现为：


Untitled Page
This panel shows dynamic control generation:
Label1
Label2
Text Box1
Text Box2
Text Box3
Text Box4
No of Labels:
2
No of Text Boxes :
4
Make the Panel Visible
Refresh Panel

图片 20.3 image

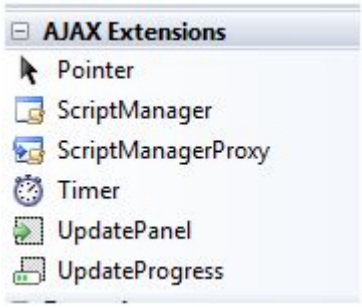
21

# Ajax 控制

AJAX 代表 Asynchronous JavaScript and XML。这是一项跨平台的能加速响应时间的技术。AJAX 服务器控件将脚本添加到页面，它由浏览器执行并处理。

然而像其他 ASP.NET 服务器控件一样，这些 AJAX 服务器控件也能拥有与它们相联系的方法和事件句柄，它们都在服务器端处理。

在 Visual Studio IDE 里的 control 工具箱含有一组叫作 'AJAX' 的控制组件。




图片 21.1 1

## ScriptManager 控件

ScriptManager 控件是最重要的控件并且必须出现在页面上以让其他控件工作。

它有基本语法：

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

如果你创建一个 'Ajax Enabled site' 或者从 'Add Item' 对话框添加一个 'AJAX Web Form'，网页将自动形成，并包含 script manager 控件。ScriptManager 控件为所有的服务器端的控件照顾客户机端的脚本。

## UpdatePanel 控件

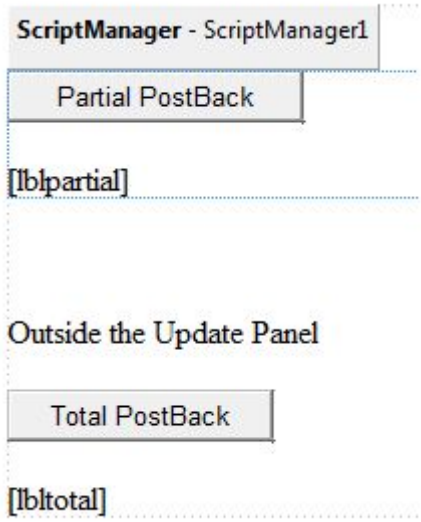
UpdatePanel 控件是一个容器控件并且源自 Control 类。它作为它里面的子控件的容器而运作并且不拥有它自己的接口。当它其中的一个控件触发提交回来，UpdatePanel 干预异步启动并更新部分页面。

例如，如果一个 button 控件在 update panel 内，并且它被点击了，只有 update panel 内的控件将被影响，页面其他部分的控件将不会被影响。这被叫做部分提交返回或者异步提交返回。

例子

在你的应用程序中添加一个 AJAX 网页表单。它包含默认的 script manager 控件。插入一个 update panel。将一个 button 控件和一个 label 标签放置在 update panel 控件内。将另一个 button 和 label 集放置在 panel 外。

设计视图如下所示：




图片 21.2 2

资源文件如下所示：

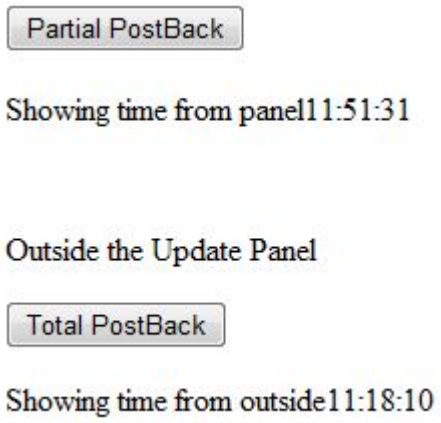
```
<form id="form1" runat="server">
  <div>
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
  </div>

  <asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
      <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click" Text="Partial PostBack"/>
      <br />
      <br />
      <asp:Label ID="lblpartial" runat="server"></asp:Label>
    </ContentTemplate>
  </asp:UpdatePanel>

  <p> </p>
  <p>Outside the Update Panel</p>
  <p>
```

```
    <asp:Button ID="btntotal" runat="server" onclick="btntotal_Click" Text="Total PostBack" />
  </p>

  <asp:Label ID="lbltotal" runat="server"></asp:Label>
</form>
```

button 控件对时间处理程序都拥有相同的代码：

```
string time = DateTime.Now.ToLongTimeString();
lblpartial.Text = "Showing time from panel" + time;
lbltotal.Text = "Showing time from outside" + time;
```

观察当页面被执行时，如果总的提交返回按钮被点击了，它将更新标签中都更新时间，但是如果部分提交返回按钮被点击，它仅仅更新在 update panel 内的标签。




图片 21.3 3

UpdatePanel Control 的属性

| 属性 | 描述 |
| --- | --- |
| ChildrenAsTriggers | 这个属性表示返回是否来自于子控件，这将引起 update panel 的刷新。 |
| ContentTemplate | 它是内容模板并且定义了当它出现时什么出现在 update panel 内。 |
| ContentTemplateContainer | 检索动态创建的 template container 对象并被用来以编程方式添加子控件。 |
| IsInPartialRendering | 指出 panel 是否被更新作为部分提交返回的一部分。 |
| RenderMode | 展示 render 模式。可用的模式是 Block 和 Inline。 |
| UpdateMode | 通过确定一些条件来获得或设置 rendering 模式。 |
| Triggers | 定义 collection trigger 对象，每一个对应于一个引发 panel 自动更新的事件。 |

UpdatePanel Control 的方法

以下表格展示了 update panel 控件的方法：

| 方法 | 描述 |
| --- | --- |
| CreateContentTemplateContainer | 创建了一个 Control 对象来作为定义 UpdatePanel 控件内容的子控件的容器。 |
| CreateControlCollection | 返回所有包含在 UpdatePanel 控件内的控件集合 |
| Initialize | 如果部分页面绘制被运行的话，初始化 UpdatePanel 控件触发器集合。 |
| Update | 引起 UpdatePanel 控件内容的更新。 |

update panel 的行为依赖于 UpdateMode 属性和 ChildrenAsTriggers 属性的值。

| 方法 | 描述 | 影响 |
| --- | --- | --- |
| Always | False | 不合法的参数。 |
| Always | True | 如果整个页面更新或者一个它上面的一个子控件返回，UpdatePanel 更新。 |
| Conditional | False | 如果整个页面更新或者它外部的一个触发的控件开始一次更新，UpdatePanel 更新。 |
| Conditional | True | 如果整个页面更新或者一个它上面的一个子控件返回或者一个它外部的触发控件开始一次更新，UpdatePanel 更新。 |

## UpdateProgress 控件

当一个或者更多的 update panel 控件被更新时，UpdateProgress 控件提供了浏览器的一种反馈。例如，当一个用户登录或者当执行一些面向数据库的工作时等待服务器响应。

它提供了如 "Loading page..." 的视觉确认，表示工作在处理中。

UpdateProgress 控件的语法是：

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" DynamicLayout="true" AssociatedUpdatePanelID="UpdateP

  <ProgressTemplate>
    Loading...
  </ProgressTemplate>

</asp:UpdateProgress>
```

以上的片段展示了一个简单的带有 ProgressTemplate 标签的信息。但是，它可以是一张图片或者其他相关的控件。UpdateProgress 控件显示每一个异步的返回，除非它使用 AssociatedUpdatePanelID 属性，被指定为单独的 update panel。

### UpdateProgress 控件的属性

以下的表格展示了 update progress 控件的属性。

| 属性 | 描述 |
| --- | --- |
| AssociatedUpdatePanelID | 获得并用这个控件所联系的控件设置 update panel 的 ID。 |
| Attributes | 获得并设置 UpdateProgress 控件的 cascading style sheet(CSS)属性。 |
| DisplayAfter | 在处理模板被展示后获得并以毫秒设置时间。默认是 500。 |
| DynamicLayout | 指示进程模板是否被动态展示。 |
| ProgressTemplate | 指示模板在一个比 DisplayAfter 时间花了更多时间的异步提交返回的过程中展示。 |

### UpdateProgress 控件的方法

以下的表格展示了 update progress 控件的方法：

| 方法 | 描述 |
| --- | --- |
| GetScriptDescriptors | 返回一个 UpdateProgress 控件的客户端功能所需要的组件，行为和客户端控件的列表。 |
| GetScriptReferences | 返回一个客户端脚本依赖 UpdateProgress 控件的列表。 |

## Timer 控件

timer 控件被用来自动初始化提交返回。这可以用两种方式完成：

(1)设置 UpdatePanel 控件的 Triggers 属性。

```
<Triggers>
  <asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />
</Triggers>
```

(2)直接在 UpdatePanel 内部放置一个 timer 控件来作为一个子控件的触发器。一个单独的 timer 能作为许多 UpdatePanel 的触发器。

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Always">

  <ContentTemplate>
    <asp:Timer ID="Timer1" runat="server" Interval="1000">
      </asp:Timer>

    <asp:Label ID="Label1" runat="server" Height="101px" style="width:304px" >
      </asp:Label>
  </ContentTemplate>

</asp:UpdatePanel>
```

# 22

# 数据源

一个 data sourse 控件与数据绑定的控件相互作用，并隐藏了复杂的数据的联编过程。这些是提供数据给 data bound 控件的工具，并且支持如插入，删除和更新操作的执行。

每一个 data sourse 控件包裹了一个特殊的数据提供者相关的数据库，XML 文件，或者是自定义类，并且帮助：

- 管理连接
- 选择数据
- 管理像分页，缓存等的展示方面
- 操控数据

有许多可在 ASP.NET 中获得的 data sourse 控件，为从 SQL 服务器，ODBC 或者 OLE DB 服务器，从 XML 文件，和从业务对象中获得数据。

基于数据类型，这些控件能被分为两个种类：

- 分层的 data sourse 控件
- 基于表格的 data sourse 控件

用于分层数据的 data sourse 控件是：

- XMLDataSource – 它允许用或者不用模式信息绑定 XML 文件和字符串。
- SiteMapDataSource – 它允许绑定一个提供站点地图信息的提供者。

用作表格数据的 data source 控件是：

| Data source 控件 | 描述 |
| --- | --- |
| SqlDataSource | 它表示到返回 SQL 数据的 ADO.NET data provider 的连接，包括通过 OLEDB 和 QDBC 可获得的 data sources。 |
| ObjectDataSource | 它允许绑定一个返回数据的自定义的 .Net business 对象 |
| LinqdataSource | 它允许绑定 Linq-t0-SQL 查询的结果。(仅由 ASP.NET 3.5 支持) |
| AccessDataSource | 它表示到 Microsoft Access 数据库的连接。 |

## Data Source 视图

Data source 视图是 DataSourceView 类的对象，它代表一个自定义的为不同数据操作如排序，过滤等而设计的数据视图。

DataSourceView 类作为所有 data source 视图类的基本类而使用，它定义了 data source 控件的性能。

以下表格提供了 DataSourceView 类的属性：

| 属性 | 描述 |
| --- | --- |
| CanDelete | 表示是否允许删除潜在的 data source。 |
| CanInsert | 表示是否允许插入潜在的 data source。 |
| CanPage | 表示是否允许给潜在的 data source 分页。 |
| CanRetrieveTotalRowCount | 表示总的行信息能否获得。 |
| CanSort | 表示数据是否能排序。 |
| CanUpdate | 表示是否允许在潜在的 data source 上更新。 |
| Events | 获得 data source 视图代表的事件句柄的列表。 |
| Name | 视图的名字。 |

以下的表格提供了 DataSourceView 类的方法：

| 方法 | 描述 |
| --- | --- |
| CanExecute | 确定指定的命令是否能执行。 |
| ExecuteCommand | 执行指定的命令。 |
| ExecuteDelete | 在 DataSourceView 对象所表示的数据列表上执行一个删除操作。 |
| ExecuteInsert | 在 DataSourceView 对象所表示的数据列表上执行一个插入操作。 |
| ExecuteSelect | 从潜在的数据存储中获取数据列表。 |
| ExecuteUpdate | 在 DataSourceView 对象所表示的数据列表上执行一个更新操作。 |
| Delete | 在和视图所联系的数据上执行一个删除操作。 |
| Insert | 在和视图所联系的数据上执行一个插入操作。 |
| Select | 返回被查询的数据。 |
| Update | 在和视图所联系的数据上执行一个更新操作。 |
| OnDataSourceViewChanged | 提出 DataSourceViewChanged 事件。 |
| RaiseUnsupportedCapabilitiesError | 由 RaiseUnsupportedCapabilitiesError 方法调用来将 ExecuteSelect 操作所需要的能力和视图所支持的能力相比较。 |

## SqlDataSource 控件

SqlDataSource 控件代表到相关数据库比如 SQL Server 或者 Oracle数据库，或者通过 OLEDB 或 Open Database Connectivity(ODBC) 的可存取数据的连接。数据连接通过两个重要的属性 ConnectionString 和 ProviderName 完成。

以下的代码片段提供了控件的基本语法：

```
<asp:SqlDataSource runat="server" ID="MySqlSource"
  ProviderName='<%$ ConnectionStrings:LocalNWind.ProviderName  %>'
  ConnectionString='<%$ ConnectionStrings:LocalNWind %>'
  SelectionCommand= "SELECT * FROM EMPLOYEES" />

<asp:GridView ID="GridView1" runat="server" DataSourceID="MySqlSource" />
```

在潜在的数据上配置不同的数据操作依赖于 data source 控件的不同属性(属性集)。

以下的表格提供了相关的 SqlDataSource 控件的属性集，它提供了控件的编程接口：

| 属性组 | 描述 |
| --- | --- |
| DeleteCommand, DeleteParameters, DeleteCommandType | 获取或设置 SQL 语句，参数和在潜在数据中删除行的类型。 |
| FilterExpression, FilterParameters | 获取并设置数据过滤字符串和参数。 |
| InsertCommand, InsertParameters, InsertCommandType | 获取或设置 SQL 语句，参数和在潜在数据中插入行的类型。 |
| SelectCommand, SelectParameters, SelectCommandType | 获取或设置 SQL 语句，参数和在潜在数据中检索行的类型。 |
| SortParameterName | 获取或设置一个输入参数的名字，它将被命令存储的过程用来给数据排序。 |
| UpdateCommand, UpdateParameters, UpdateCommandType | 获取或设置 SQL 语句，参数和在潜在数据中更新行的类型。 |

以下的代码片段展示了能被用来做数据操作的 data source 控件：

```
<asp:SqlDataSource runat="server" ID= "MySqlSource"
  ProviderName='<%$ ConnectionStrings:LocalNWind.ProviderName  %>'
  ConnectionString=' <%$ ConnectionStrings:LocalNWind %>'
  SelectCommand= "SELECT * FROM EMPLOYEES"
  UpdateCommand= "UPDATE EMPLOYEES SET LASTNAME=@lame"
```

```
  DeleteCommand= "DELETE FROM EMPLOYEES WHERE EMPLOYEEID=@eid"
  FilterExpression= "EMPLOYEEID > 10">
  .....
  .....
</asp:SqlDataSource>
```

## ObjectDataSource 控件

ObjectDataSource 控件使 user-defined 类能让它们方法的输出和 data bound 控件相连接。这个类的编程接口几乎和 SqlDataSource 控件相同。

以下是绑定客户对象的两个重要方面：

- 可绑定的类应该拥有一个默认的构造函数，它应该是无状态的，并且拥有能够映射到选择，更新，插入，和删除语意的方法。
- 对象必须一次更新一个项目，批处理操作是不支持的。

让我们直接到一个例子中来使用这个控件。student 类是被用来和一个 data source 对象一起使用的类。这个类有三个属性：a student id，name，和 city。它有一个默认的构造函数和一个检索数据的 GetStudents 方法。

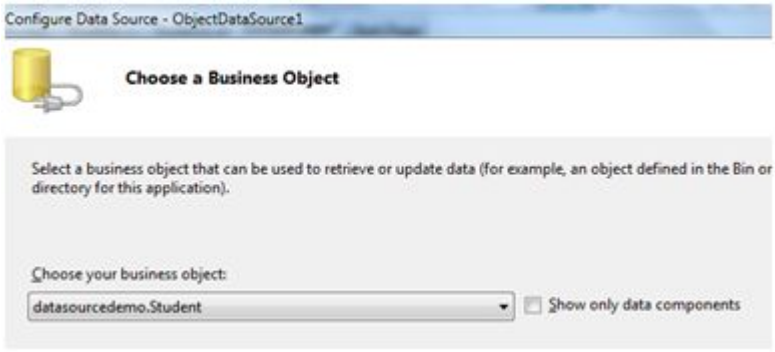
student 类：

```
public class Student
{
  public int StudentID { get; set; }
  public string Name { get; set; }
  public string City { get; set; }

  public Student()
  { }

  public DataSet GetStudents()
  {
    DataSet ds = new DataSet();
    DataTable dt = new DataTable("Students");

    dt.Columns.Add("StudentID", typeof(System.Int32));
    dt.Columns.Add("StudentName", typeof(System.String));
    dt.Columns.Add("StudentCity", typeof(System.String));
    dt.Rows.Add(new object[] { 1, "M. H. Kabir", "Calcutta" });
    dt.Rows.Add(new object[] { 2, "Ayan J. Sarkar", "Calcutta" });
    ds.Tables.Add(dt);
```

```
    return ds;
  }
}
```

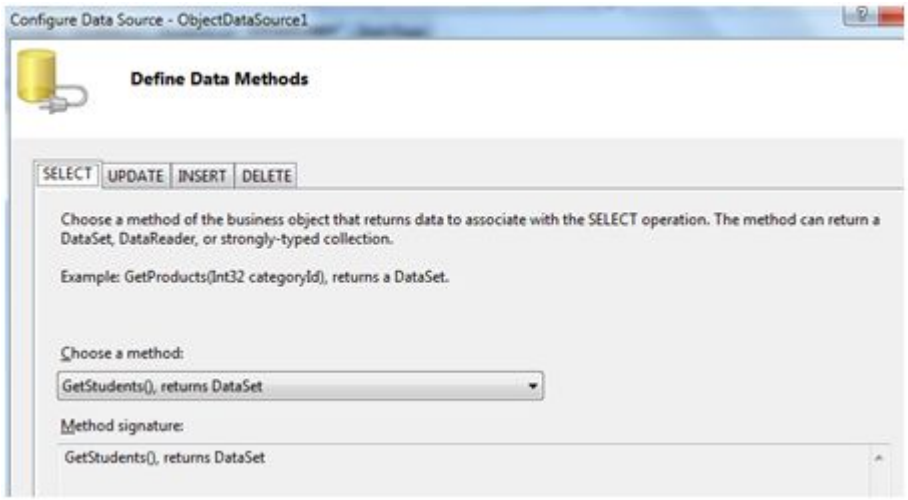
采取以下的步骤来将对线绑定到一个 data source 对象和检索数据：

- 创建一个新的网页。
- 通过右击 Solution Explorer 的项目来给它添加一个类(Student.cs)，添加一个类模板，将上面的代码放在里面。
- 建立方法使得应用程序可以使用类的引用。
- 在网页表单中放置一个 data source 控件对象。
- 通过选择对象来配置 data source。



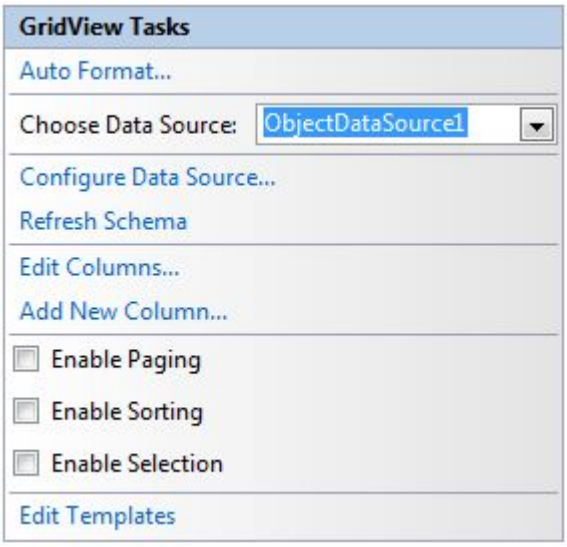

图片 22.1 1

- 给不同的数据操作选择数据方法。在这个例子中，仅有一个方法。



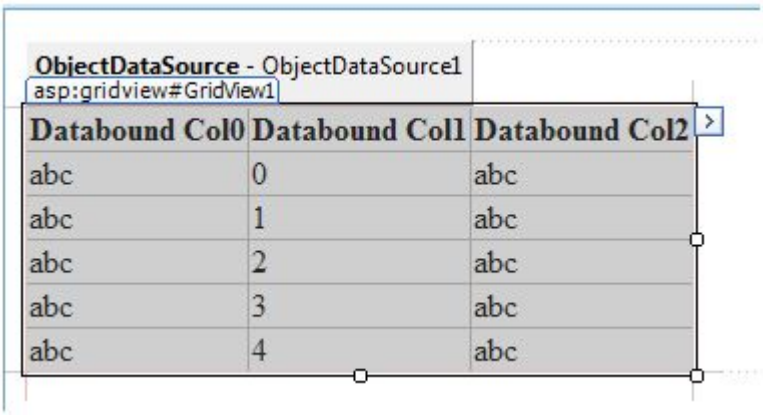

图片 22.2 2

- 在页面上放置一个 data bound 控件比如 grid view 并且选择 data source 对象作为潜在的 data source。



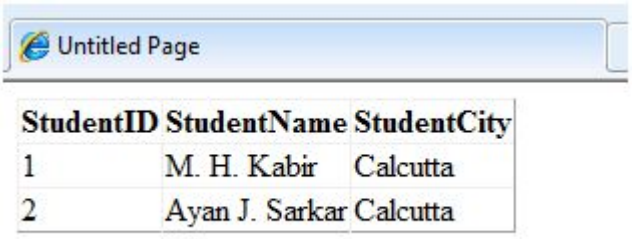

图片 22.3 3

- 在这个阶段，设计视图应该像以下这样：




图片 22.4 4

- 运行项目，它检索了 students 类中的硬编码的元祖。




图片 22.5 5

## AccessDataSource 控件

AccessDataSource 控件代表了到 Access 数据库的连接。它基于 SqlDataSource 控件并提供了更简单的编程接口。以下的代码片段提供了 data source 的基本语法：

```
<asp:AccessDataSource ID="AccessDataSource1 runat="server"
  DataFile="~/App_Data/ASPDotNetStepByStep.mdb" SelectCommand="SELECT * FROM [DotNetReferences]">
</asp:AccessDataSource>
```

AccessDataSource 控件打开了只读模式的数据库。但是，它也能被用来执行插入，更新或者删除操作。这以使用 ADO.NET 命令和参数集合来完成。

更新对于 ASP.NET 应用程序内的 Access 数据库来说是有问题的，这是因为 Access 数据库是一个纯文本并且默认的 ASP.NET 应用程序账户可能有写数据库文件的权限。

23

# ASP.NET – 数据绑定

每一个 ASP.NET 网页表单控件从它的父控件类继承了 DataBind 方法，它给予了它继承的能力来绑定数据到它属性中的至少一个属性。这就是所谓的**简单数据绑定**或者**内部数据绑定**。

简单数据绑定包括将任何实现 IEnumerable 接口的集合(项目集合)，或者 DataSet 和 DataTable 类附加到控件的 DataSource 属性。

另一方面，一些控件可以通过 DataSource 控件绑定记录，列表，或者数据列到它们的结构中。这些控件源自 BaseDataBoundControl 类。这被叫做**描述性数据绑定**。

data source 控件帮助 data-bound 控件实现了比如排序，分页和编辑数据集合的功能。

BaseDataBoundControl 是一个抽象类，它通过两个抽象类继承：

- DataBoundControl
- HierarchicalDataBoundControl

抽象类 DataBoundControl 也由两个抽象类继承：

- ListControl
- CompositeDataBoundControl

能够简单绑定数据的控件源自 ListControl 抽象类并且这些控件是：

- BulletedList
- CheckBoxList
- DropDownList
- ListBox
- RadioButtonList

能够描述性数据绑定的控件(一个更复杂的数据绑定)源自抽象类 CompositeDataBoundControl。这是控件是：

- DetailsView
- FormView
- GridView
- RecordList

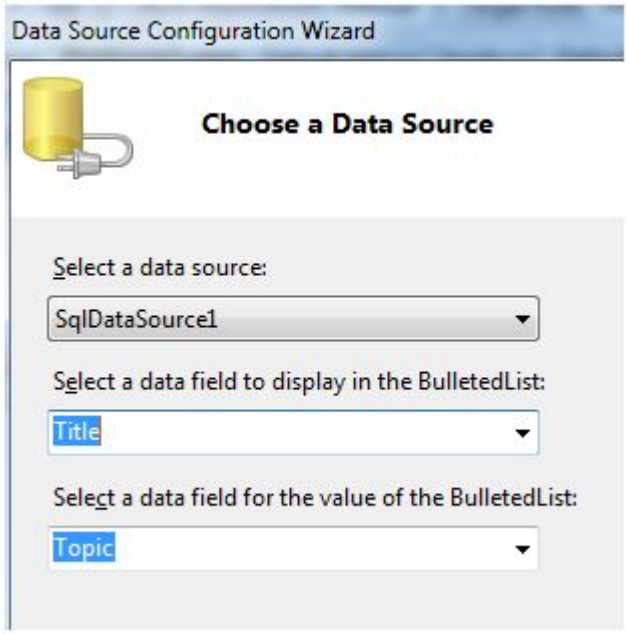
## 简单数据绑定

简单数据绑定包括只读选择列表。这些控件能绑定一个数组列或者数据库的字段。选择列表从数据库中或 data source 中取两个值；一个值用过列表表示而另一个被认为是相应显示的值。

让我们使用一个小例子来理解这个概念。用一个项目符号列表和一个 SqlDataSource 控件来创建一个网页。配置 data source 控件来从你的数据库中(我们在之前的章节中使用相同的 DotNetReferences 表)检索两个值。

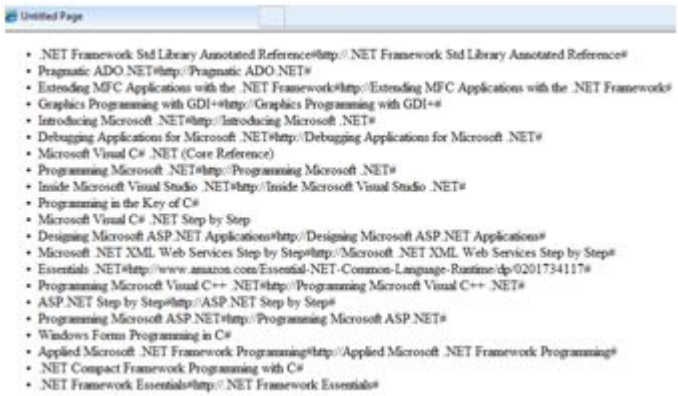
为包含的项目符号列表控件选择一个 data source：

- 选择 data source 控件
- 选择一个字段来展示，它被叫做数据字段
- 选择值的字段




图片 23.1 1

当应用程序执行的时候，检查整个标题列绑定到项目符号列表并被展示。




图片 23.2 2

## 描述性数据绑定

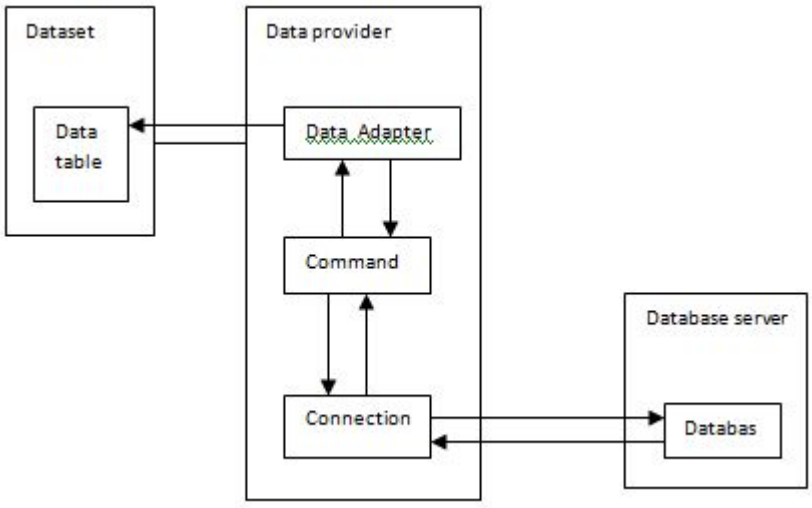
我们已经在之前的指南中使用 GridView 控件来使用描述性数据绑定。其他复合的能够以表格的方式展示并操作数据的 data bound 控件是 DetailsView, FormView 和 RecordList 控件。

在下一个指南中，我们将研究解决数据库，i.e，ADO.NET 的 技术。

但是，数据绑定包括以下对象：

- 存储从数据库检索数据的数据集。
- 数据提供者，它通过使用一个连接的命令从数据库中检索数据。
- 发出存储在 command 对象中的选择语句的数据适配器；它也能通过发出 Insert，Delete，和 Updata 语句来更新数据库中的数据。

data bonding 对象间的关系：




图片 23.3 3

例子

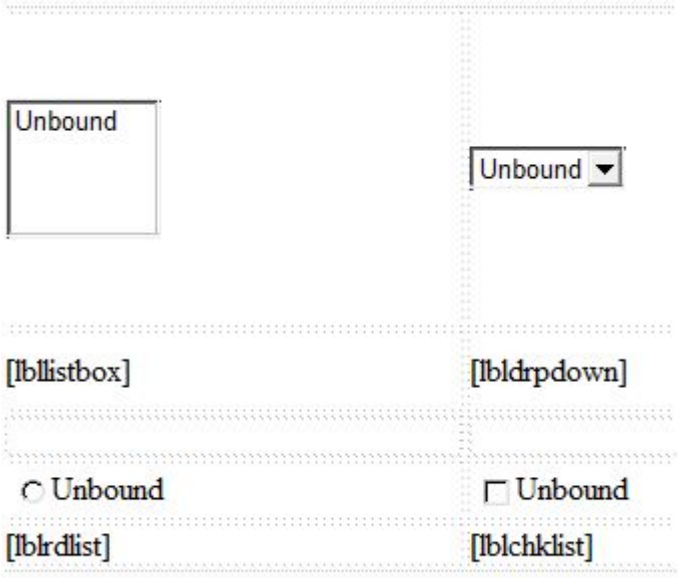
让我们采取以下的步骤：

**步骤(1):**创建一个新的网页。通过右击在 Solution Explorer 上的 solution 名字和从 'Add Item' 对话框中选择项目 'Class' 来添加一个名为 booklist 的类。将它命名为 booklist.cs。

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace databinding
{
  public class booklist
  {
    protected String bookname;
    protected String authorname;
    public booklist(String bname, String aname)
    {
      this.bookname = bname;
      this.authorname = aname;

    }

    public String Book
    {
      get
      {
        return this.bookname;
      }
      set
      {
        this.bookname = value;
      }
```

```
    }

    public String Author
    {
      get
      {
        return this.authorname;
      }
      set
      {
        this.authorname = value;
      }
    }
  }
}
```

步骤(2):在页面上添加四个列表控件，一个 list box 控件，一个 radio button 控件，一个 check box 控件和一个 drop down list 和四个与这些列表控件一起的四个表单。在设计视图中页面应该看起来像这样：




图片 23.4 4

源文件应该看起来像下面这样：

```
<form id="form1" runat="server">
  <div>

    <table style="width: 559px">
      <tr>
        <td style="width: 228px; height: 157px;">
          <asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"
            OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
```

```
      </asp:ListBox>
    </td>

    <td style="height: 157px">
      <asp:DropDownList ID="DropDownList1" runat="server"
        AutoPostBack="True" OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
      </asp:DropDownList>
    </td>
  </tr>

  <tr>
    <td style="width: 228px; height: 40px;">
      <asp:Label ID="lbllistbox" runat="server"></asp:Label>
    </td>

    <td style="height: 40px">
      <asp:Label ID="lbldrpdown" runat="server">
      </asp:Label>
    </td>
  </tr>

  <tr>
    <td style="width: 228px; height: 21px">
    </td>

    <td style="height: 21px">
    </td>
  </tr>

  <tr>
    <td style="width: 228px; height: 21px">
      <asp:RadioButtonList ID="RadioButtonList1" runat="server"
        AutoPostBack="True"  OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
      </asp:RadioButtonList>
    </td>

    <td style="height: 21px">
      <asp:CheckBoxList ID="CheckBoxList1" runat="server"
        AutoPostBack="True" OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
      </asp:CheckBoxList>
    </td>
  </tr>

  <tr>
    <td style="width: 228px; height: 21px">
```

```
          <asp:Label ID="lblrdlist" runat="server">
          </asp:Label>
        </td>

        <td style="height: 21px">
          <asp:Label ID="lblchklist" runat="server">
          </asp:Label>
        </td>
      </tr>
    </table>

  </div>
</form>
```

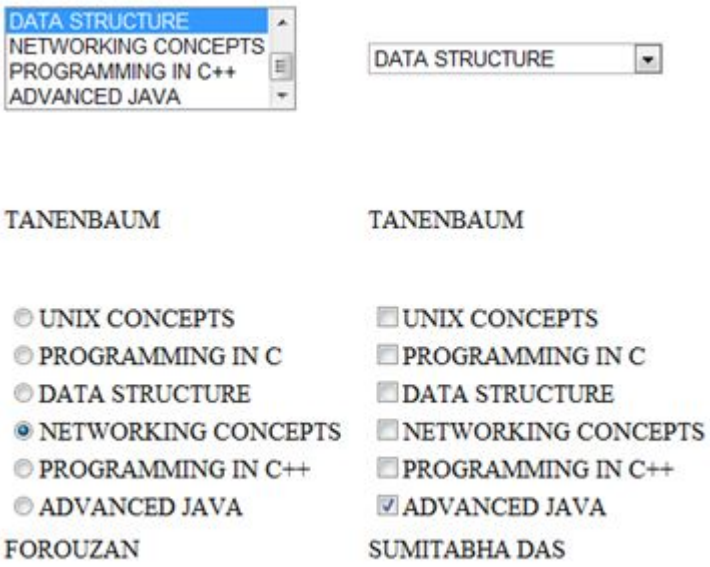
步骤(3):最后，在应用程序的例行程序后写下面的代码：

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    IList bklist = createbooklist();

    if (!this.IsPostBack)
    {
      this.ListBox1.DataSource = bklist;
      this.ListBox1.DataTextField = "Book";
      this.ListBox1.DataValueField = "Author";

      this.DropDownList1.DataSource = bklist;
      this.DropDownList1.DataTextField = "Book";
      this.DropDownList1.DataValueField = "Author";

      this.RadioButtonList1.DataSource = bklist;
      this.RadioButtonList1.DataTextField = "Book";
      this.RadioButtonList1.DataValueField = "Author";

      this.CheckBoxList1.DataSource = bklist;
      this.CheckBoxList1.DataTextField = "Book";
      this.CheckBoxList1.DataValueField = "Author";

      this.DataBind();
    }
  }

  protected IList createbooklist()
  {
```

```
    ArrayList allbooks = new ArrayList();
    booklist bl;

    bl = new booklist("UNIX CONCEPTS", "SUMITABHA DAS");
    allbooks.Add(bl);

    bl = new booklist("PROGRAMMING IN C", "RICHI KERNIGHAN");
    allbooks.Add(bl);

    bl = new booklist("DATA STRUCTURE", "TANENBAUM");
    allbooks.Add(bl);

    bl = new booklist("NETWORKING CONCEPTS", "FOROUZAN");
    allbooks.Add(bl);

    bl = new booklist("PROGRAMMING IN C++", "B. STROUSTROUP");
    allbooks.Add(bl);

    bl = new booklist("ADVANCED JAVA", "SUMITABHA DAS");
    allbooks.Add(bl);

    return allbooks;
  }

  protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
  {
    this.lbllistbox.Text = this.ListBox1.SelectedValue;
  }

  protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
  {
    this.lbldrpdown.Text = this.DropDownList1.SelectedValue;
  }

  protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)
  {
    this.lblrdlist.Text = this.RadioButtonList1.SelectedValue;
  }

  protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
  {
    this.lblchklist.Text = this.CheckBoxList1.SelectedValue;
  }
}
```

观察以下：

- booklist 类有两个属性：bookname 和 authorname。
- createbooklist 方法是一个用户定义的可以创建名为 allboods 的 booklist 类的数组的方法。
- Page_Load 事件句柄确保了 books 的列表被创建。该列表是 IList 型的，它实现了 IEnumerable 接口并能和列表控件绑定。Page load 时间句柄用控件绑定了 IList 对象'bklist'。bookname 属性被展示并且 auth orname　属性被视为这个值。
- 当页面运行时，如果用户选择了一本书，则它的名字被选择并且通过 list 控件被显示出来，而相应的标签显示作者的名字，它是 list 控件所选择的相应的值。




图片 23.5 5

# 24

# 自定义控件

ASP.NET 允许用户创建控件。这些用户定义的控件被分类为：

- 用户控件
- 自定义控件

## 用户控件

用户控件行为像微型 ASP.NET 页面或者网页表单，它能被许多其他页面使用。这些都是源自 System.Web.UI.UserControl 类。这些控件有下列特性：

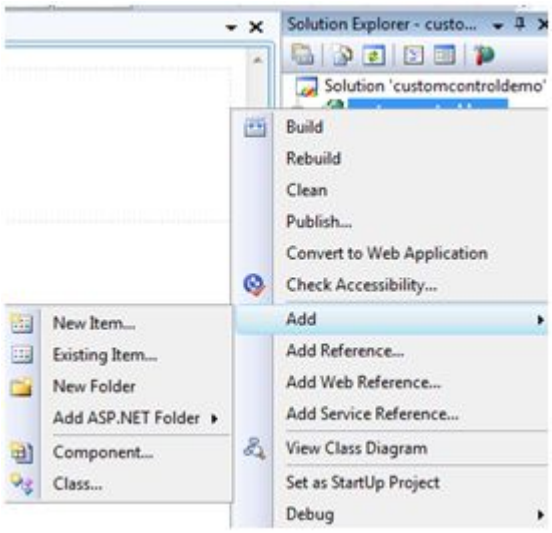
- 它们有 .ascx 扩展。
- 它们可能不会含有任何 , 或者

<

form> 标签。 – 它们有一个 Control 指令而不是一个 Page 指令。

为了理解这个概念，让我们创建一个简单的用户控件，它将作为 web 页面的页脚使用。为了创建和使用用户控件，采取以下步骤：

- 创建一个新的 web 应用程序。
- 在 Solution Explorer 上右击项目文件夹并且选择 ADD New Item。




图片 24.11

- 从 Add New Item 对话框中选择 Web User Control 并且把它命名为 footer.ascx。最初，footer.ascx 仅含有一个 Control 指令。

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="footer.ascx.cs"
  Inherits="customcontroldemo.footer" %>
```

- 给文件添加下列代码：

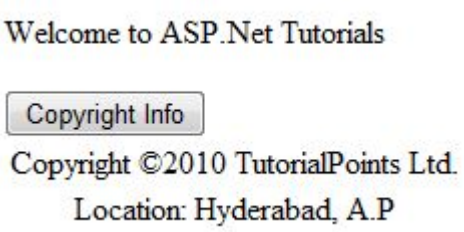
```
<table>
  <tr>
    <td align="center"> Copyright ©2010 TutorialPoints Ltd.</td>
  </tr>

  <tr>
    <td align="center"> Location: Hyderabad, A.P </td>
  </tr>
</table>
```

为给你的 web 网页添加用户控件，你必须添加 Register 指令和一个页面用户控件的实例。以下的代码展示了说明：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="customcontroldemo._Defa

<%@ Register Src="~/footer.ascx" TagName="footer" TagPrefix="Tfooter" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>

    <form id="form1" runat="server">
      <div>

        <asp:Label ID="Label1" runat="server" Text="Welcome to ASP.Net Tutorials "></asp:Label>
        <br />  <br />
        <asp:Button ID="Button1" runat="server" onclick="Button1_Click"  Text="Copyright Info" />

      </div>
      <Tfooter:footer ID="footer1" runat="server" />
    </form>
```

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

当执行后，页面显示了页脚而且这个控件能在所有你的网站的页面中被使用。




图片 24.2 2

观察以下：

(1)Register 指令为控件指定了一个标签名称和标签前缀。

```
<%@ Register Src="~/footer.ascx" TagName="footer" TagPrefix="Tfooter" %>
```

(2)下列的标签名称和前缀应该在页面上添加用户控件时被使用:

```
<Tfooter:footer ID="footer1" runat="server" />
```

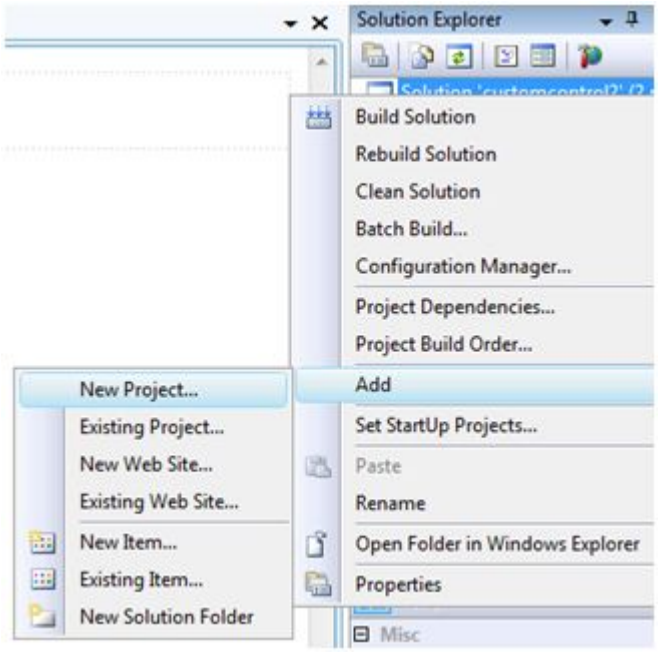
## 自定义控件

自定义控件被部署为单独的集合。它们被编译成动态链接库(DLL)并且作为任何其他的 ASP.NET 服务控件来使用。它们能被以下方法中的任何一个来创建:

- 通过从一个存在的控件中获得一个自定义控件。
- 通过联合两个或者更多的存在的控件来组成一个新的自定义控件。
- 通过从基本的控件类中获得。

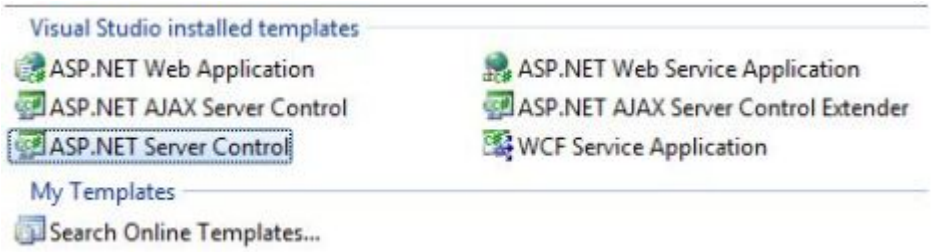
为了理解这个概念，让我们创建一个自定义类，它将简单地在浏览器上呈现一条短信。为了创建控件，采取以下步骤：

创建一个新的网站。在 Solution Explorer 中树的顶端右击 solution(不是项目)。




图片 24.3 3

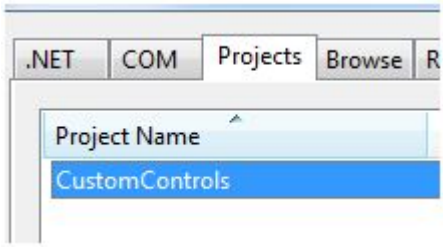
在 New Project 对话框中，从项目模板中选择 ASP.NET Server Control。




图片 24.4 4

上面的步骤添加了一个新的项目并且给 solution 创建了一个完整的自定义控件，叫做 ServerControl1。在这个例子中，让我命名 CustomControls 项目。为了使用这个控件，它必须在页面上注册之前作为引用添加到网页中。为了添加引用到已存在的项目中，右击项目(不是 solution)，并且点击 Add Reference。

从 Add Reference 对话框中的 Projects 标签选择 CustomControl 项目。Solution Explorer 能显示引用。




图片 24.5 5

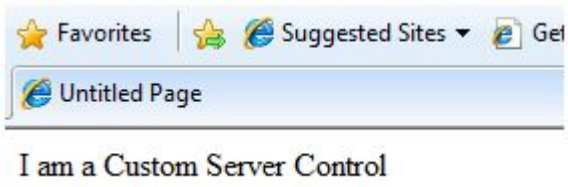
为了在页面上使用控件，在 @Page 指令下添加 Register 指令。

```
<%@ Register Assembly="CustomControls"  Namespace="CustomControls"  TagPrefix="ccs" %>
```

而且，你可以使用控件，和任何其他控件类似。

```
<form id="form1" runat="server">
  <div>
    <ccs:ServerControl1 runat="server" Text = "I am a Custom Server Control" />
  </div>
</form>
```

当执行后，控件的 Text 属性被展示在浏览器上，如下所示:




图片 24.6 6

## 使用自定义类

在之前的例子中，自定义类的 Text 属性值被设置了。当控件被创建时，ASP.NET 默认添加了这个属性。以下控件的文件后的代码揭示了这个。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;

using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CustomControls
{
  [DefaultProperty("Text")]
  [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1 >")]

  public class ServerControl1 : WebControl
  {
    [Bindable(true)]
    [Category("Appearance")]
    [DefaultValue("")]
```

```
    [Localizable(true)]

    public string Text
    {
      get
      {
        String s = (String)ViewState["Text"];
        return ((s == null) ? "[" + this.ID + "]" : s);
      }

      set
      {
        ViewState["Text"] = value;
      }
    }

    protected override void RenderContents(HtmlTextWriter output)
    {
      output.Write(Text);
    }
  }
}
```

上述的代码自动生成给一个自定义控件。事件和方法能被添加到 custom control 类中。

**例子**

让我们扩展之前的名为 ServerControl1 的自定义控件。让我们给予它一个名为 checkpalindrome 的方法，它将给它权限来检查 palindrome。

Palindrome 是当颠倒时仍拼写相同的文字/字面值。例如，Malayalam，madam,saras 等。

扩展自定义控件的代码，它应该看起来如下所示：

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;

using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
```

```
namespace CustomControls
{
  [DefaultProperty("Text")]
  [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1  >")]

  public class ServerControl1 : WebControl
  {
    [Bindable(true)]
    [Category("Appearance")]
    [DefaultValue("")]
    [Localizable(true)]

    public string Text
    {
      get
      {
        String s = (String)ViewState["Text"];
        return ((s == null) ? "[" + this.ID + "]" : s);
      }

      set
      {
        ViewState["Text"] = value;
      }
    }

    protected override void RenderContents(HtmlTextWriter output)
    {
      if (this.checkpanlindrome())
      {
        output.Write("This is a palindrome: <br />");
        output.Write("<FONT size=5 color=Blue>");
        output.Write("<B>");
        output.Write(Text);
        output.Write("</B>");
        output.Write("</FONT>");
      }
      else
      {
        output.Write("This is not a palindrome: <br />");
        output.Write("<FONT size=5 color=red>");
        output.Write("<B>");
        output.Write(Text);
        output.Write("</B>");
        output.Write("</FONT>");
```

```
      }
    }

    protected bool checkpanlindrome()
    {
      if (this.Text != null)
      {
        String str = this.Text;
        String strtoupper = Text.ToUpper();
        char[] rev = strtoupper.ToCharArray();
        Array.Reverse(rev);
        String strrev = new String(rev);

        if (strtoupper == strrev)
        {
          return true;
        }
        else
        {
          return false;
        }
      }
      else
      {
        return false;
      }
    }
  }
}
```

当你改变空间的代码时，你必须通过点击 Build --> Build Solution 来构建方法，这样改变才能反映在你的项目中。给页面添加一个 text box 和一个 button 控件，这样用户才能提供一段 text。当 button 被点击时，它就被用来检查 palindrome。

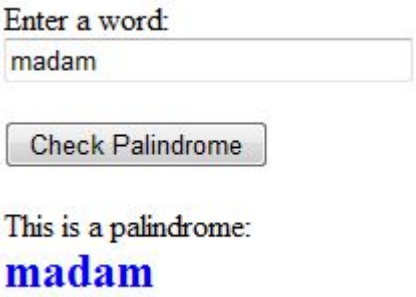
```
<form id="form1" runat="server">
  <div>
    Enter a word:
    <br />
    <asp:TextBox ID="TextBox1" runat="server" style="width:198px"> </asp:TextBox>

    <br /> <br />

    <asp:Button ID="Button1" runat="server onclick="Button1_Click" Text="Check Palindrome" style="width:132px" />

    <br /> <br />
```

```
    <ccs:ServerControl1 ID="ServerControl11" runat="server" Text = "" />
  </div>
</form>
```

button 的 Click 事件句柄简单地将 text box 中的 text 复制到自定义控件的 text 属性中。

```
protected void Button1_Click(object sender, EventArgs e)
{
  this.ServerControl11.Text = this.TextBox1.Text;
}
```

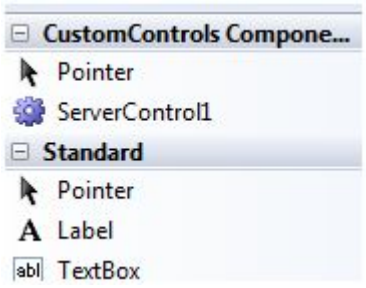
当被执行后，控件成功地检测到了 palindromes。




图片 24.7 7

观察以下:

(1) 当你给自定义控件添加一个引用时，它被添加到 toolbox 并且你可以像其他控件一样从 toolbox 中直接使用它。




图片 24.8 8

(2)custom control 类的 RenderContents 方法被覆写了，你可以添加你自己的方法和事件。

(3)RenderContents 方法采用了 HtmlTextWriter 型的参数，它将对在浏览器上展示负责。

# 25 个性化

网站是为用户的重复访问而设计的。个性化允许一个网站记住用户标识和其他信息细节，并且它给每个用户提供了一个个人的环境。

ASP.NET 为满足特性客户的品味和喜好而个性化一个网站提供服务。

## 理解特征文件

ASP.NET 个性化服务基于用户的特征文件。用户特征文件定义了该网站需要用户的信息。例如，名字，年龄，地址，出生日期和手机号码。

这个信息被定义在应用程序的 web.config 文件中并且 ASP.NET 运行时间阅读并使用它。这个工作由个性化提供者所完成。

用户数据所含有的用户特征文件被存储在默认的 ASP.NET 创建的数据库中。你可以创建你自己的数据库来存储特征文件。特征文件数据定义被存储在配置文件 web.config 中。

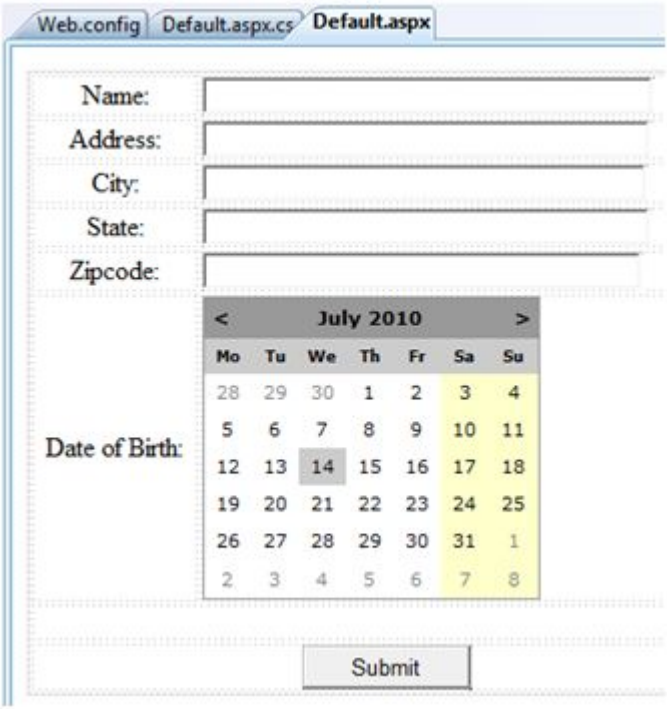
### 例子

让我们创建一个样本网站，那里我们想要我们的应用程序记住用户细节，像名字，地址，出生日期等。在 web.config 文件中用 <system.web> 元素添加特征文件细节。

```
<configuration>
<system.web>

<profile>
  <properties>
    <add name="Name" type ="String"/>
    <add name="Birthday" type ="System.DateTime"/>

    <group name="Address">
      <add name="Street"/>
      <add name="City"/>
      <add name="State"/>
      <add name="Zipcode"/>
    </group>

  </properties>
</profile>

</system.web>
</configuration>
```

当特征文件在 web.config 文件中被定义时，特征文件可以通过在当前的 HttpContext 中找到的 Profile 属性使用并且通过页面获得。

添加 text box 来获取在特征文件中定义的用户输入，添加一个 button 来提交数据：




图片 25.11

更新 Page_load 来展示特征文件信息：

```
using System;
using System.Data;
using System.Configuration;

using System.Web;
using System.Web.Security;

using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    if (!this.IsPostBack)
    {
```

```
        ProfileCommon pc=this.Profile.GetProfile(Profile.UserName);

        if (pc != null)
        {
          this.txtname.Text = pc.Name;
          this.txtaddr.Text = pc.Address.Street;
          this.txtcity.Text = pc.Address.City;
          this.txtstate.Text = pc.Address.State;
          this.txtzip.Text = pc.Address.Zipcode;
          this.Calendar1.SelectedDate = pc.Birthday;
        }
      }
    }
}
```

为提交按钮写以下的句柄，将用户数据存入特征文件中：

```
protected void btnsubmit_Click(object sender, EventArgs e)
{
  ProfileCommon pc=this.Profile.GetProfile(Profile.UserName);

  if (pc != null)
  {
    pc.Name = this.txtname.Text;
    pc.Address.Street = this.txtaddr.Text;
    pc.Address.City = this.txtcity.Text;
    pc.Address.State = this.txtstate.Text;
    pc.Address.Zipcode = this.txtzip.Text;
    pc.Birthday = this.Calendar1.SelectedDate;

    pc.Save();
  }
}
```

当页面第一次执行时，用户需要输入信息。但是，下一次用户的细节将被自动加载。

## 元素的属性

除了我们已经使用过的名字和类型属性，元素还有其它属性。以下的表格展示了这些属性中的一些：

| 属性 | 描述 |
| --- | --- |
| name | 属性的名字。 |
| type | 类型默认是 string 但是它允许任何完全的类名称作为数据类型。 |
| serializeAS | 当序列化这个值时使用的格式。 |

| 属性 | 描述 |
| --- | --- |
| readOnly | 只读的特征文件值不能被改变，这个属性默认是 false。 |
| defaultValue | 一个默认的值，如果特征文件不存在或者没有信息的话它被使用。 |
| allowAnonymous | 一个指示这个属性是否能和匿名文件使用的布尔值。 |
| Provider | 应该被用来管理这个属性的特征文件提供者。 |

## 匿名个性化

匿名个性化允许用户在标识它们自己之前个性化网站。例如，Amazon.com 允许用户在登录前在购物车中添加物品。为了启用此功能，web.config 文件可以被配置成以下：

```
<anonymousIdentification enabled ="true" cookieName=".ASPXANONYMOUSUSER"
  cookieTimeout="120000" cookiePath="/" cookieRequiresSSL="false"
  cookieSlidingExpiration="true" cookieprotection="Encryption"
  coolieless="UseDeviceProfile"/>
```

# 26

# 异常处理

在 ASP.NET 中异常处理有三个方面：

- Tracing – 在页面级或者应用程序级追踪程序执行。
- Error handling – 在页面级或者应用程序级解决标准错误或者自定义错误。
- Debugging – 在程序中前进，设置断点来分析代码。

在这一章中，我们将讨论 tracing 和 handling。并且在这一章中，我们将涉及 debugging。

为了理解概念，创建以下的样本应用程序。它有一个 label 控件，一个 dropdown 列表和一个链接。dropdown 列表加载了一个名言的 array 列表并且被选择的引用将显示在下面的标签中。它也拥有一个超链接，它指向一个不存在的链接。

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Tracing, debugging and error handling
    </title>
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <asp:Label ID="lblheading" runat="server" Text="Tracing, Debuggin  and Error Handling">
        </asp:Label>

        <br /> <br />

        <asp:DropDownList ID="ddlquotes" runat="server" AutoPostBack="True"  onselectedindexchanged="ddlquotes_S
        </asp:DropDownList>

        <br /> <br />

        <asp:Label ID="lblquotes" runat="server">
        </asp:Label>

        <br /> <br />

        <asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="mylink.htm">Link to:</asp:HyperLink>
```

```
      </div>

    </form>
  </body>

</html>
```

文件后的代码：

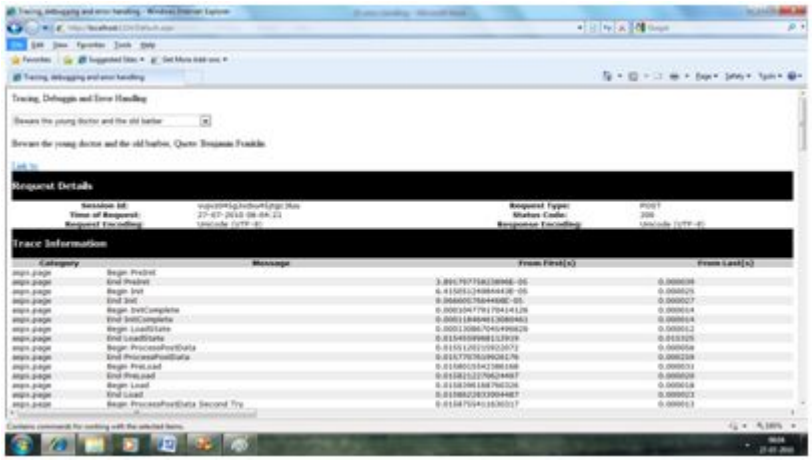
```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    if (!IsPostBack)
    {
      string[,] quotes =
      {
        {"Imagination is more important than Knowledge.", "Albert Einsten"},
        {"Assume a virtue, if you have it not" "Shakespeare"},
        {"A man cannot be comfortable without his own approval", "Mark Twain"},
        {"Beware the young doctor and the old barber", "Benjamin Franklin"},
        {"Whatever begun in anger ends in shame", "Benjamin Franklin"}
      };

      for (int i=0; i<quotes.GetLength(0); i++)
        ddlquotes.Items.Add(new ListItem(quotes[i,0], quotes[i,1]));
    }
  }

  protected void ddlquotes_SelectedIndexChanged(object sender, EventArgs e)
  {
    if (ddlquotes.SelectedIndex != -1)
    {
      lblquotes.Text = String.Format("{0}, Quote: {1}", ddlquotes.SelectedItem.Text, ddlquotes.SelectedValue);
    }
  }
}
```

## Tracing

为了允许页面级别的追踪，你需要修改 Page 指令并且如下添加一个 Trace 属性：

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

现在当你执行文件时，你将得到追踪信息：



图片 26.1 1

它在首部提供了以下的信息：

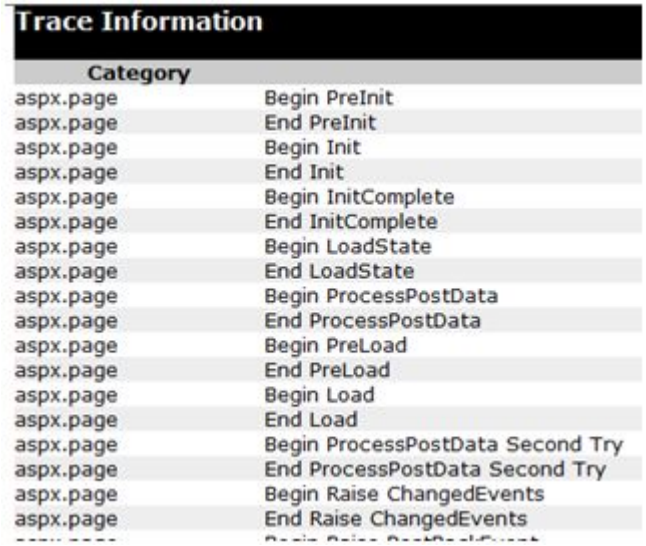
- Session ID
- Status Code
- Time of Request
- Type of Request
- Request and Response Encoding

每次页面被需要时，从服务器发出的 status 代码显示名字和错误时间，如果有的话。以下的表格显示普通的 HTTP status 代码：

| 数字 | 描述 |
| --- | --- |
| **通知(100 – 199)** | |
| 100 | 继续 |
| 101 | 转换协议 |
| **成功(200 – 299)** | |
| 200 | OK |
| 204 | 无内容 |
| **重定向(300 – 399)** | |
| 301 | 永久移动 |
| 305 | 使用代理 |
| 307 | 暂时重定向 |
| **来自客户端的错误(400 – 499)** | |
| 400 | 错误请求 |
| 402 | 支付需求 |

| 数字 | 描述 |
| --- | --- |
| 404 | 未找到 |
| 408 | 请求超时 |
| 417 | 期望失败 |
| 来自服务器的错误(500 – 599) | |
| 500 | 内部服务器错误 |
| 503 | 服务不可用 |
| 505 | HTTP 版本不支持 |

在顶级信息下，有一个 Trace 日志，它提供了页面生命周期的细节。它提供了页面被初始化后的以秒为单位的运行时间。

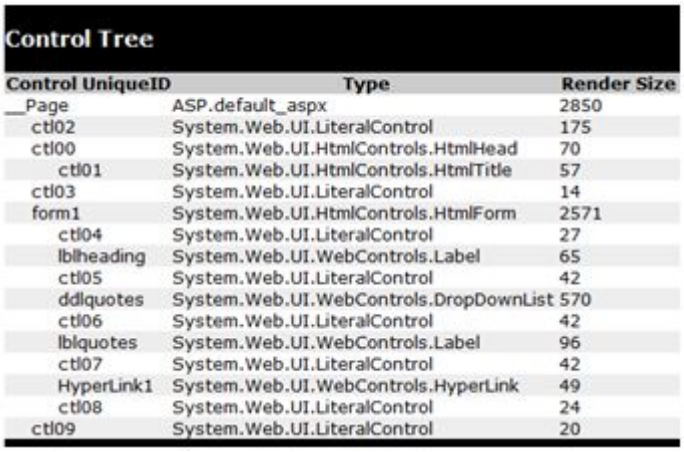


Trace Information

| Category | |
| --- | --- |
| aspx.page | Begin PreInit |
| aspx.page | End PreInit |
| aspx.page | Begin Init |
| aspx.page | End Init |
| aspx.page | Begin InitComplete |
| aspx.page | End InitComplete |
| aspx.page | Begin LoadState |
| aspx.page | End LoadState |
| aspx.page | Begin ProcessPostData |
| aspx.page | End ProcessPostData |
| aspx.page | Begin PreLoad |
| aspx.page | End PreLoad |
| aspx.page | Begin Load |
| aspx.page | End Load |
| aspx.page | Begin ProcessPostData Second Try |
| aspx.page | End ProcessPostData Second Try |
| aspx.page | Begin Raise ChangedEvents |
| aspx.page | End Raise ChangedEvents |

图片 26.2 2

下一个部分是控件树，它以分层的形式列举了页面上所有的控件：



Control Tree

| Control UniqueID | Type | Render Size |
| --- | --- | --- |
| __Page | ASP.default_aspx | 2850 |
| ctl02 | System.Web.UI.LiteralControl | 175 |
| ctl00 | System.Web.UI.HtmlControls.HtmlHead | 70 |
| ctl01 | System.Web.UI.HtmlControls.HtmlTitle | 57 |
| ctl03 | System.Web.UI.LiteralControl | 14 |
| form1 | System.Web.UI.HtmlControls.HtmlForm | 2571 |
| ctl04 | System.Web.UI.LiteralControl | 27 |
| lblheading | System.Web.UI.WebControls.Label | 65 |
| ctl05 | System.Web.UI.LiteralControl | 42 |
| ddlquotes | System.Web.UI.WebControls.DropDownList | 570 |
| ctl06 | System.Web.UI.LiteralControl | 42 |
| lblquotes | System.Web.UI.WebControls.Label | 96 |
| ctl07 | System.Web.UI.LiteralControl | 42 |
| HyperLink1 | System.Web.UI.WebControls.HyperLink | 49 |
| ctl08 | System.Web.UI.LiteralControl | 24 |
| ctl09 | System.Web.UI.LiteralControl | 20 |

图片 26.3 3

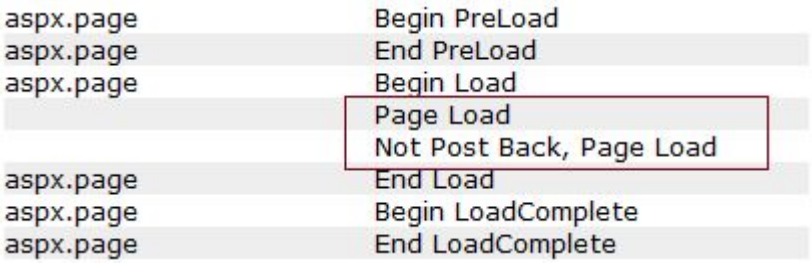
Session 和 Application 中的最后声明了跟随了所有服务器变量的 summaries,cookies 和 headers 集合。

Trace 对象允许你给 trace 输出添加自定义信息。它有两个方法来完成：Write 方法和 Warn 方法。

改变 Page_Load 事件句柄在检测 Write 方法：

```
protected void Page_Load(object sender, EventArgs e)
{
  Trace.Write("Page Load");

  if (!IsPostBack)
  {
    Trace.Write("Not Post Back, Page Load");
    string[,] quotes =
    ....................
  }
}
```

运行来观察影响：




图片 26.4 4

为了检测 Warn 方法，让我们在被选择的 index changed 事件句柄中强制输入一些错误的代码：

```
try
{
  int a = 0;
  int b = 9 / a;
}catch (Exception e)
{
  Trace.Warn("UserAction", "processing 9/a", e);
}
```

Try-Catch 是一个 C# 编程结构。try 块持有任何可以或不可以产生错误的代码，catch 块捕获了错误。当程序运行时，它在 trace 日志中发送警告。


图片 26.5 5

应用程序层次的追踪应用到网站中的所有的页面。它通过将以下代码放入 web.config 文件被实现：

```
<system.web>
  <trace enabled="true" />
</system.web>
```

## 错误解决

尽管 ASP.NET 能检测所有的运行时错误，仍然有一些微小的错误仍在那儿。通过追踪观察错误是为开发者准备的，而不是用户。

因此，为了拦截这样情况的发生，你可以在应用程序的 web.config 中添加错误解决设置。它是应用程序范围的错误解决。例如，你可以在 web.config 文件中添加以下的代码：

```
<configuration>
  <system.web>

    <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">
      <error statusCode="403" redirect="NoAccess.htm"    />
      <error statusCode="404" redirect="FileNotFound.htm" />
    </customErrors>

  </system.web>
<configuration>
```

部分有可能的属性：

- Mode：它允许或者不允许自定义错误页面。它有三个可能的值：
  - On:展示自定义页面。
  - Off：展示 ASP.NET 错误页面(黄色页面)
  - remoteOnly：它展示了自定义错误到客户端，展示本地的 ASP.NET 错误。
- defaultRedirect:它含有页面的 URL 来展示以备不能解决的错误。

为了给不同错误类型放置不同的自定义错误页面，子标签被使用，那里不同的错误页面基于错误的 status 代码被指定。

为了实现页面级别的错误解决，Page 指令能被修改为：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
  Inherits="errorhandling._Default" Trace ="true" ErrorPage="PageError.htm" %>
```

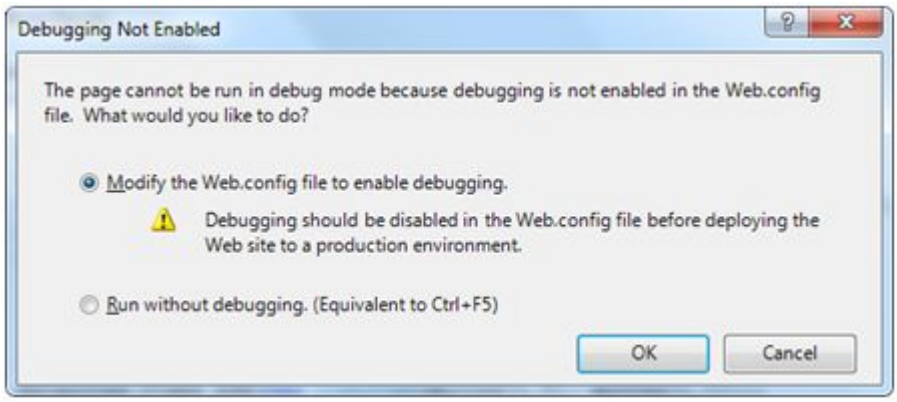
因为 ASP.NET Debugging 是它内部一个重要的主题，因此我们将在下一章单独地讨论它。

# 27

# 调试

Debugging 可以让开发人员一步一步的看到代码是怎样工作的，变量的值是如何变化的和对象是怎样被创建又是怎样被销毁的等等。

当一个网页第一次被运行时，Visual Studio 会弹出一个提示框来询问 Debugging 是否需要被启用：




图片 27.1 debugging_info

当 debugging 被启用时，下面几行代码将在 web.config 文件中出现：

```
<system.web>
  <compilation debug="true">
    <assemblies>
    ..............
    </assemblies>
  </compilation>
</system.web>
```

Debugging 工具栏会提供所有 debugging 所需的工具：


图片 27.2 debugging_toolbar.jpg

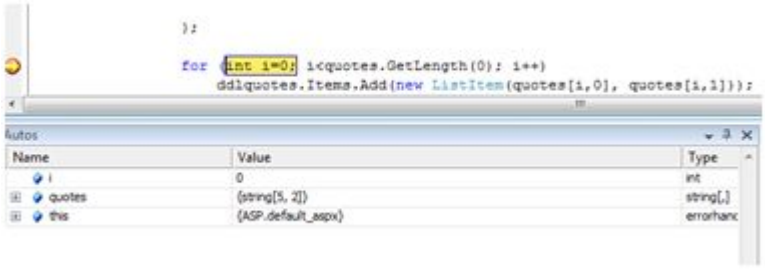
## 断点

断点规定程序在运行时在运行完指定的代码行之后立即停止运行，这样可以测试代码并且完成各种各样的 debugging 工作，例如，观察变量值的变化，单步调试代码，函数方法的跳入跳出等。

在代码上单击右键选择插入一个间断点来设置断点。然后在左边会出现一个红点并且该行代码被高亮显示，效果如图所示：


图片 27.3 breakpoint_highlighted.jpg

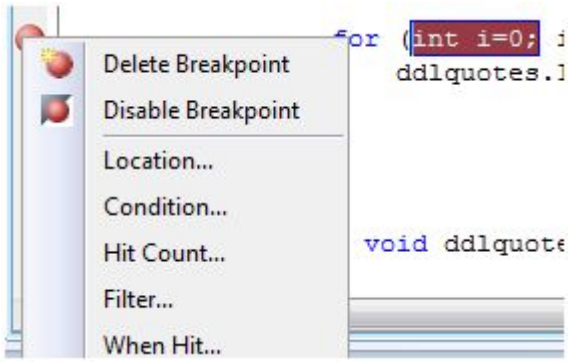
之后你运行这段代码，将会观察到断点的行为。




图片 27.4 breakpoint_highlighted2.jpg

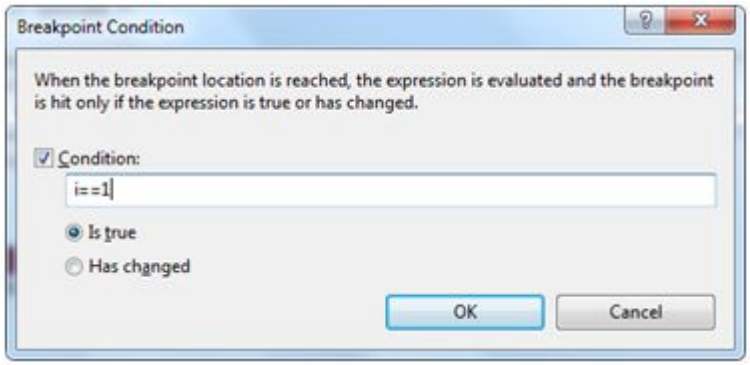
在这个阶段，你可以单步调试代码，观察运行的流程和变量值、属性、对象等。

如果你需要修改断点属性，你可以在断点标志上单击右键，在“属性”菜单中找到：



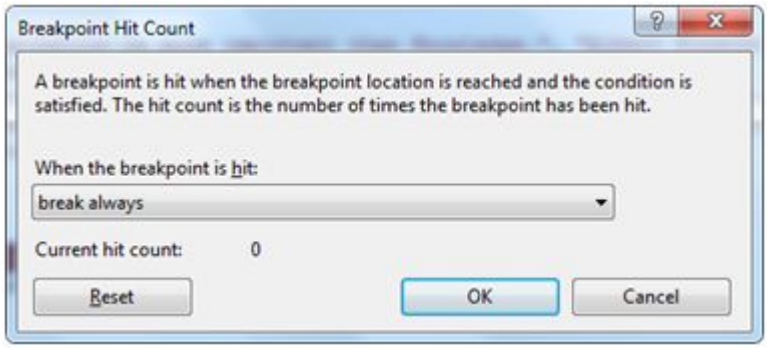

图片 27.5 breakpoint_dropdown.jpg

location 对话框显示文件所在位置，以及所选中的代码所在行数和字符数。condition 菜单允许你输入一个有效的表达式来估算程序是否运行到了断点:



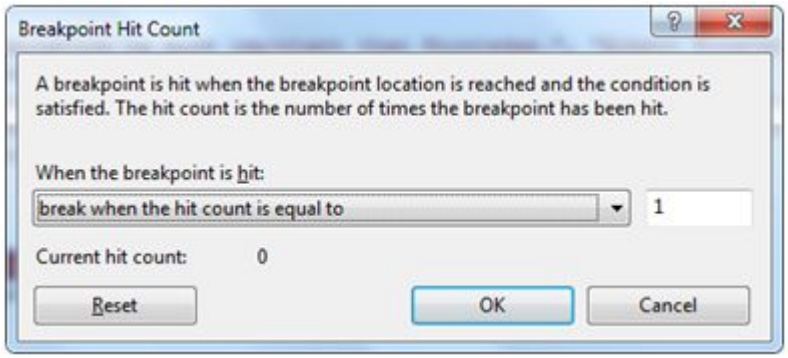

图片 27.6 breakpoint_condition.jpg

Hit Count 菜单显示一个对话框来显示断点被运行的次数。



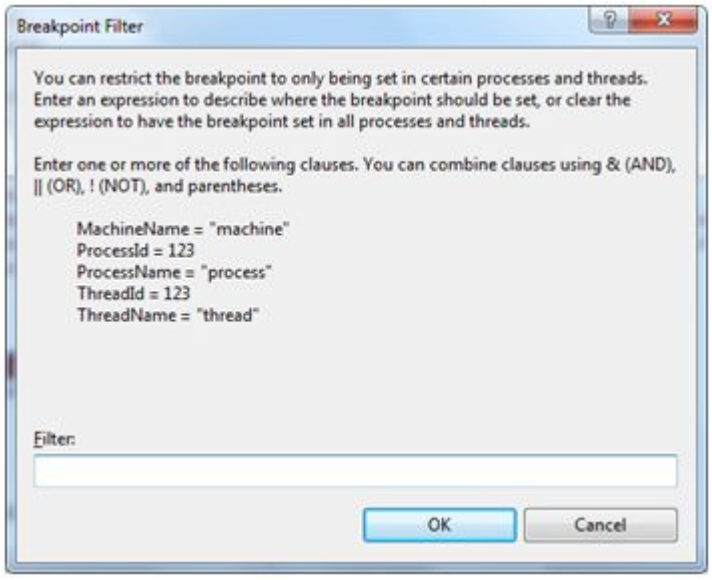

图片 27.7 breakpoint_asp.net.jpg

点击下拉菜单中的任何一个选项会打开一个用来输入命中次数的编辑框。这在分析循环结构的代码时非常有用。



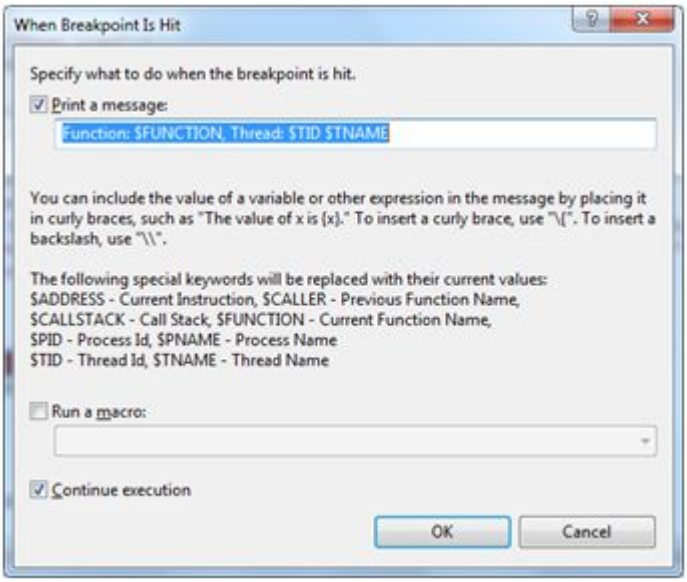

图片 27.8 breakpoint_asp.net2.jpg

Filter 菜单允许设置一个对特定机制、过程、线程或是任何组合的过滤使断点对它们生效。




图片 27.9 breakpoint_filters.jpg

When Hit 菜单允许你来指定当断点命中时的动作。




图片 27.10 breakpoint_asp.net3.jpg

## Debug 窗口

Visual Studio 提供下面的 debug 窗口，其中每一个都显示一些程序信息。下表列出了一些窗口：

| 窗口 | 描述 |
| --- | --- |
| 直接 | 显示变量和表达式。 |
| 自动 | 显示当前所有变量以及之前的状态。 |
| 本地 | 显示当前上下文的所有变量。 |
| 观察 | 显示多达四个不同集合的变量。 |
| 调用栈 | 显示调用栈中的所有方法。 |
| 线程 | 显示并控制线程。 |

# 28

# 语言集成查询

大多数应用都是以数据为中心的，然而大多数的数据仓库是关系型数据库。这些年，设计者和开发者设计了基于对象模型的应用程序。

对象来负责连接访问数据的组件——称为数据访问层( DAL )。这里我们需要考虑三点：

- 一个应用程序所需要的所有数据可以不存储在一个资源中。这个资源可以是关系型数据库、业务对象、XML文件或者一个WEB服务器。
- 访问内存中的对象要比访问数据库、XML文件中的数据更简单，更廉价。
- 被访问到的数据不是直接使用的，而是被转存、排序、分组、修改等。

因此如果存在只是用几行代码就能实现轻易整合各种各样的数据——可以整合来自不同源的数据，并且能够执行基本的数据操作的工具，那将非常有用。

语言集成查询( LINQ )就是上述那样的一种工具。 LINQ 是 .NET Framework 3.5 的一个扩展集并且它的管理语言使查询更类似于是一种对象。它定义了一种通用的语法和程序模型，使我们可以使用一种惯用的语法完成查找不同类型的数据。

相关操作像查找、工程、链接、分组、分区、集合操作等可以在 LINQ 中使用，并且在 .NET Framework 3.5 中的 C# 和 VB 编译器支持 LINQ 的语法，这就使得它可以通过配置数据来存储，而不需要求助于 ADO.NET。

举个例子，在 Northwind 数据库中查询 Constomers 这张表，使用 C# 中的 LINQ ，代码应该是这样：

```
var data = from c in dataContext.Customers
where c.Country == "Spain"
select c;
```

其中：

- from关键字逻辑上依次通过每个集合。
- 包含关键字where的表达式会比较集合中的每个对象。
- select声明会选择被比较出的对象加入到列表中并返回。
- 关键字var用于变量声明。因为返回对象的准确类型不明确，它表明信息需要被动态的推测。

LINQ 查询语句可以应用在任何继承于 IEnumerable 的有数据支撑的类，这里 T 可以是任何一个数据类型，例如 List< Book >。

让我们来看一个示例理解一下概念。示例中使用了如下类:Book.cs

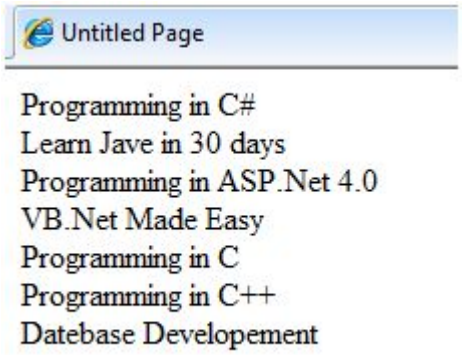
```
public class Books
{
   public string ID {get; set;}
```

```
    public string Title { get; set; }
    public decimal Price { get; set; }
    public DateTime DateOfRelease { get; set; }

    public static List<Books> GetBooks()
    {
        List<Books> list = new List<Books>();
        list.Add(new Books { ID = "001",
            Title = "Programming in C#",
            Price = 634.76m,
            DateOfRelease = Convert.ToDateTime("2010-02-05") });

        list.Add(new Books { ID = "002",
            Title = "Learn Jave in 30 days",
            Price = 250.76m,
            DateOfRelease = Convert.ToDateTime("2011-08-15") });

        list.Add(new Books { ID = "003",
            Title = "Programming in ASP.Net 4.0",
            Price = 700.00m,
            DateOfRelease = Convert.ToDateTime("2011-02-05") });

        list.Add(new Books { ID = "004",
            Title = "VB.Net Made Easy",
            Price = 500.99m,
            DateOfRelease = Convert.ToDateTime("2011-12-31") });

        list.Add(new Books { ID = "005",
            Title = "Programming in C",
            Price = 314.76m,
            DateOfRelease = Convert.ToDateTime("2010-02-05") });

        list.Add(new Books { ID = "006",
            Title = "Programming in C++",
            Price = 456.76m,
            DateOfRelease = Convert.ToDateTime("2010-02-05") });

        list.Add(new Books { ID = "007",
            Title = "Datebase Developement",
            Price = 1000.76m,
            DateOfRelease = Convert.ToDateTime("2010-02-05") });

        return list;
    }
}
```

在 web 网页中使用这个类要有简单的标签控制，来显示书的标题。Page_Load 方法创建了一个书的列表并且通过使用 LINQ 查询返回标题:

```
public partial class simplequery : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      List<Books> books = Books.GetBooks();
      var booktitles = from b in books select b.Title;

      foreach (var title in booktitles)
         lblbooks.Text += String.Format("{0} <br />", title);
   }
}
```

当网页被运行，标签显示查询结果:




图片 28.1 linq_result.jpg

上面的 LINQ 表达式:

```
var booktitles =
from b in books
select b.Title;
```

等价于下述 SQL 语句:

```
SELECT Title from Books
```

## LINQ 运算符

除了到目前为止使用过的运算符之外，还有很多其他运算符来执行查询子句。我们来看一些运算符和子句。

### join 子句

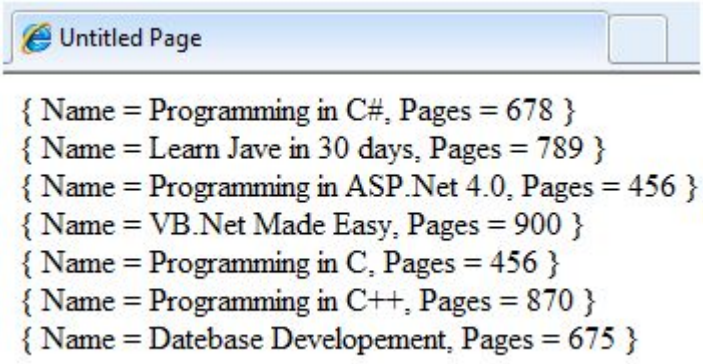
SQL 中的‘join clause’用来连接两个数据表并显示在两个数据表中都出现的列中的数据集合。LINQ 也可以支持这种功能。为了检测这一点，在之前的工程里增加另一个类名为 Saledetails.cs:

```
public class Salesdetails
{
   public int sales { get; set; }
   public int pages { get; set; }
   public string ID {get; set;}

   public static IEnumerable<Salesdetails> getsalesdetails()
   {
      Salesdetails[] sd =
      {
         new Salesdetails { ID = "001", pages=678, sales = 110000},
         new Salesdetails { ID = "002", pages=789, sales = 60000},
         new Salesdetails { ID = "003", pages=456, sales = 40000},
         new Salesdetails { ID = "004", pages=900, sales = 80000},
         new Salesdetails { ID = "005", pages=456, sales = 90000},
         new Salesdetails { ID = "006", pages=870, sales = 50000},
         new Salesdetails { ID = "007", pages=675, sales = 40000},
      };

      return sd.OfType<Salesdetails>();
   }
}
```

在 Page_Load 函数中添加代码来用 join 子句处理在两张表里完成查询:

```
protected void Page_Load(object sender, EventArgs e)
{
   IEnumerable<Books> books = Books.GetBooks();
   IEnumerable<Salesdetails> sales = Salesdetails.getsalesdetails();

   var booktitles = from b in books join s in sales on b.ID equals s.ID
      select new { Name = b.Title, Pages = s.pages };

   foreach (var title in booktitles)
      lblbooks.Text += String.Format("{0} <br />", title);
}
```

结果页显示如下:



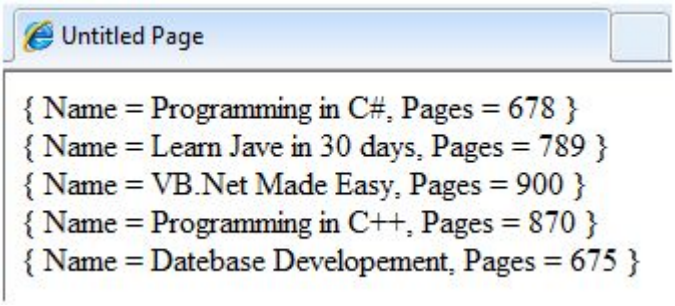

图片 28.2 linq_result2.jpg

### where 子句

where 子句允许在查询中添加筛选条件。例如，如果你想获得页数多于 500 的书目，可以改变 Page_Load 方法中的句柄成下述样子:

```
var booktitles = from b in books join s in sales on b.ID equals s.ID
   where s.pages > 500 select new { Name = b.Title, Pages = s.pages };
```

查询语句只返回那些页数大于 500 的列:



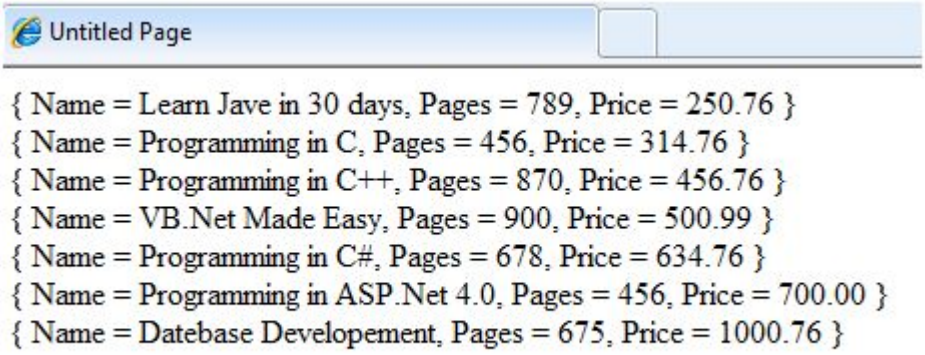

图片 28.3 linq_result3.jpg

### 正序倒序排序子句

这些子句允许将查询结果进行排序。为了查询出标题、页数和书的价格，并且按照价格排序，在 Page_Load 方法中的句柄里写如下代码:

```
var booktitles = from b in books join s in sales on b.ID equals s.ID
   orderby b.Price select new { Name = b.Title,  Pages = s.pages, Price = b.Price};
```

返回的元组是:




图片 28.4 linq_result4.jpg

### Let 子句

let 子句允许定义一个变量并且将数据计算的一个值赋给它。举个例子，计从上述两个销售值中计算总销售值，你需要这样计算:

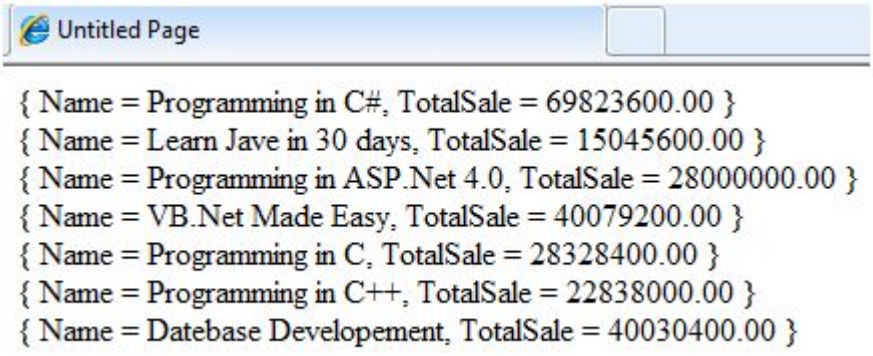
```
TotalSale = Price of the Book * Sales
```

为了完成这个算式，加入下面这个代码片段在 Page_Load 方法的句柄里:

let 子句允许定义一个变量并且将数据计算的一个值赋给它。举个例子，计从上述两个销售值中计算总销售值，你需要这样计算:

```
var booktitles = from b in book join s in sales on b.ID equals s.ID
  let totalprofit = (b.Price * s.sales)
  select new { Name = b.Title, TotalSale = totalprofit};
```

查询结果如下图所示:




图片 28.5 linq_result5.jpg

# 29

# 安全性

实现网站的安全性关系到如下几方面：

- **身份认证**:即确认用户身份和真实性的过程。ASP.NET 中提供了四种类型的认证：
  - Windows 认证
  - 表单认证
  - 身份验证
  - 自定义认证
- **授权**：即定义并为特定用户分配特定角色的过程。
- **机密性**：包括对客户端浏览器和网络服务器的加密。
- **完整性**：保持数据完整性。例如，实现数字签名。

## 基于表单的认证

一般来讲，基于表单的认证包括编辑网络配置文件以及具有验证码的注册页面。
网络配置文件可由如下代码编写：

```
<configuration>

<system.web>
   <authentication mode="Forms">
      <forms loginUrl ="login.aspx"/>
   </authentication>

   <authorization>
      <deny users="?"/>
   </authorization>
</system.web>
...
...
</configuration>
```

上面的代码段中提及的 login.aspx 页面可能会包含如下代码，包含验证用的用户名和密码在文件之后很难编码进去。

```
protected bool authenticate(String uname, String pass)
{
   if(uname == "Tom")
   {
      if(pass == "tom123")
```

```
            return true;
    }

    if(uname == "Dick")
    {
        if(pass == "dick123")
            return true;
    }

    if(uname == "Harry")
    {
        if(pass == "har123")
            return true;
    }

    return false;
}

public void OnLogin(Object src, EventArgs e)
{
    if (authenticate(txtuser.Text, txtpwd.Text))
    {
        FormsAuthentication.RedirectFromLoginPage(txtuser.Text, chkrem.Checked);
    }
    else
    {
        Response.Write("Invalid user name or password");
    }
}
```

注意到 FormsAuthentication 类是用于认证过程的。

然而，不用写任何代码 Visual Studio 就能够通过网站管理工具轻松地无缝实现用户创建、身份认证和授权。这种工具能够实现用户和角色的创建。

除此之外，ASP.NET 有现成的登录控制系列，可以为你控制执行所有的工作。

## 基于表单的安全性的实现

为了建立基于表单的认证，你需要做到如下几点：

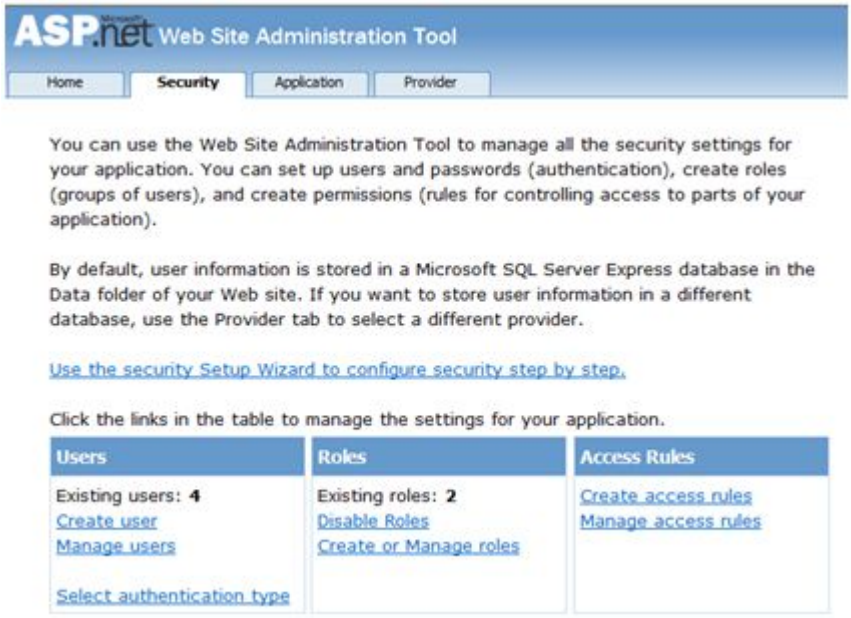
- 支持认证过程的用户数据库
- 一个使用数据库的网站

- 用户账户
- 角色
- 用户活动和群体活动的限制
- 一个显示用户状态及其他信息的用户页面
- 允许用户登录、找回密码、修改密码的登录界面。

为了创建一个用户，需要采取以下步骤：

**第一步：**选择网站 -> 配置 ASP.NET 以打开网络应用管理工具。

**第二步：**点击安全选项。

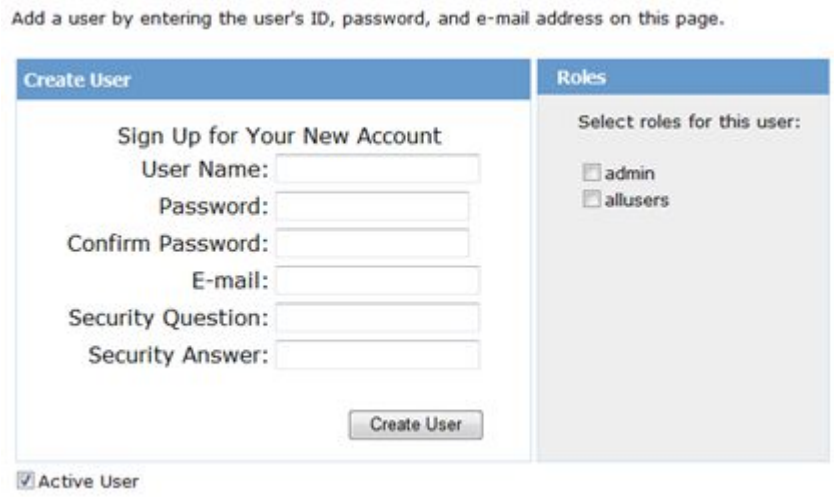



图片 29.1 security_tab.jpg

**第三步：**选择 'Forms based authentication' 选项，以将认证类型设定为 'From the Internet'。


图片 29.2 authentication_type.jpg

**第四步**：点击 'Create Users'。如果你已经创建了角色，你正好可以在这一步把角色分配给该用户。



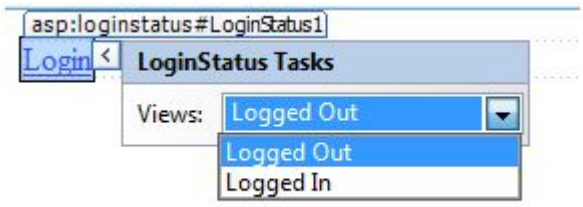

**图片 29.3** create_users_link.jpg

**第五步**：创建一个网站，并添加如下页面：

- 欢迎页面
- 登录页面
- 注册页面
- 找回密码页面
- 修改密码页面

**第六步**：在欢迎页面的登录部分设置一个登录状态控件。包含两个标准框：LoggedIn 和 LoggedOut。

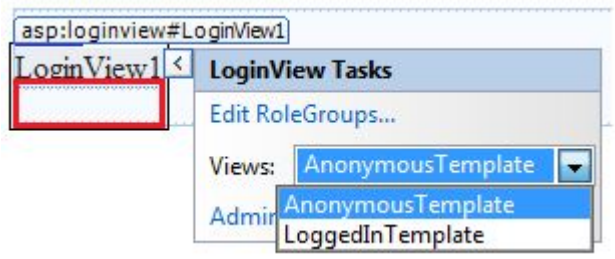
LoggedIn 有查看已经登录用户的选项，LoggedOut 内有查看已经退出用户的选项。你可以在属性窗口里改变登录和退出的文本属性。




**图片 29.4** login_status_control.jpg

**第七步**：在 LoginStatus 控件的下面设置一个 LoginView 控件。你可以在此设置一些能反应用户是否已经登录的其他文本或其他控件（如超链接、按钮等）。

这个控件有两个标准框： Anonymous 框和 LoggedIn 框。选择每种视图，并为用户编写一些文本，以作为选择标准框时要显示的内容。文本应该被放在如下图中标红的区域。




图片 29.5 login_view_control.jpg

**第八步：**由开发者创建应用用户。你也许想要允许游客也能够创建一个用户账户。要实现这个，你可以在 LoginView 控件下添加一个可以转到注册页面的链接。

**第九步：**在注册页面设置一个 CreateUserWizard 控件。设置这个控件的 ContinueDestinationPageUrl 属性，以保证能够转到欢迎页面。


图片 29.6 createuserwizard_control.jpg

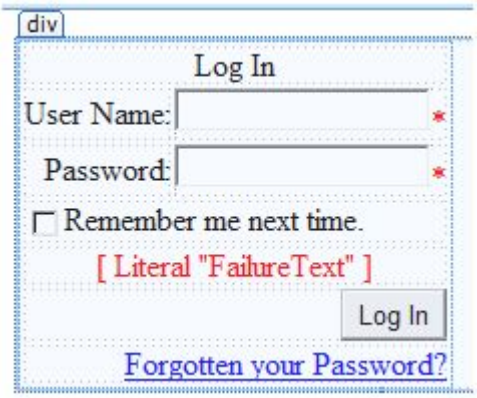
**第十步：**创建登录页面。在这个页面上设置一个 Login 控件。 LoginStatus 控件会自动地连接到登录页面。在网络配置文件里做如下改动可以改变这种默认设置。

例如，如果你把你的登录页面命名为 signup.aspx ，可以在网络配置文件的 部分添加如下几行代码。

```
<configuration>
   <system.web>
      <authentication mode="Forms">
         <forms loginUrl ="signup.aspx" defaultUrl = â€œWelcome.aspxâ€? />
      </authentication>
   </system.web>
</configuration>
```

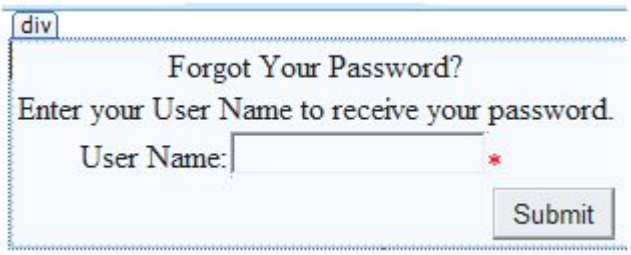
**第十一步：**用户经常会忘记密码。PasswordRecovery 控件帮助用户重新获得登录这个账户。选择登录控件。打开它的小标签，并选择 'Convert to Template'。

通过自定义这个控件的用户界面，在登录按钮下方放置一个超链接控件，这个控件应该是能够链接到找回密码页面的。



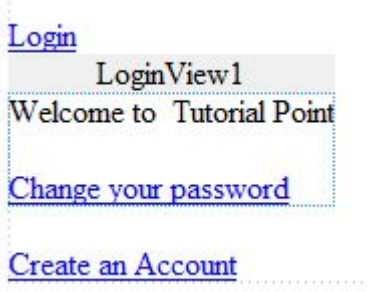

**图片 29.7** passwordrecovery_control.jpg

**第十二步：**在找回密码页面设置一个 PasswordRecovery 控件。这个控件需要邮件服务器把密码发送给用户。



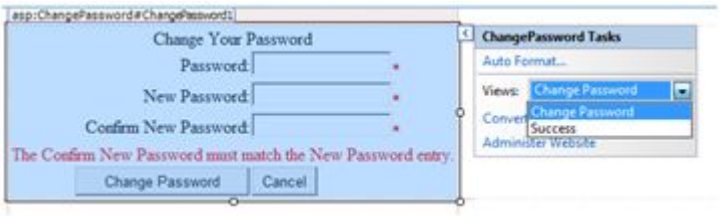

**图片 29.8** passwordrecovery_control2.jpg

**第十三步：**在欢迎页面的 LoginView 控件的 LoggedIn 框内设置一个转到修改密码页面的链接。




**图片 29.9** changepassword_control.jpg

**第十四步：**在修改密码页面设置一个 ChangePassword 控件，这个控件有两种视图：



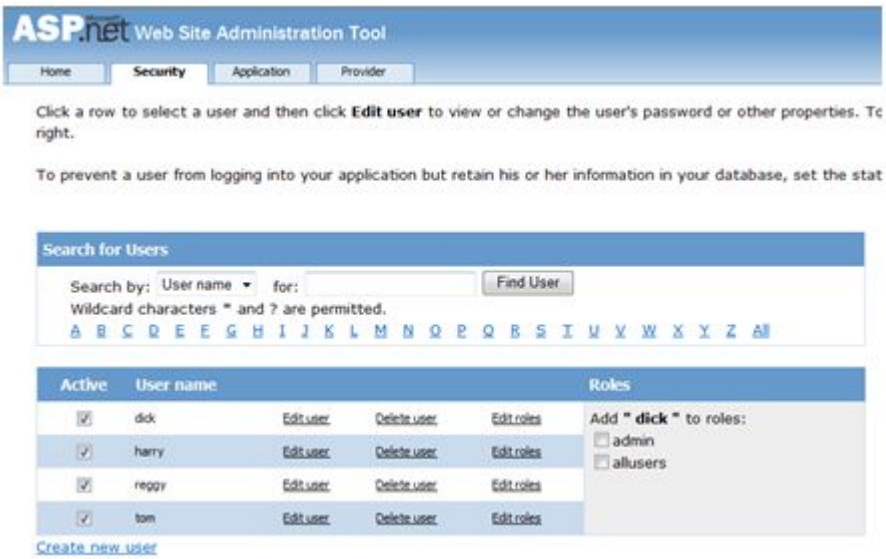

图片 29.10 changepassword_control2.jpg

现在运行这个应用，观察不同的安全操作。

可以回到网络应用管理工具，点击安全选项，来创建角色。点击 'Create Roles' 为这个应用来创建一些角色。


图片 29.11 web_application_administration.jpg

点击 'Manage Users'，可以给用户分配角色。




图片 29.12 manage_users.jpg

## IIS 认证：SSL

安全套接层（SSL）是用来确保安全连接的协议。通过使用 SSL，浏览器会把送到服务器的所有数据加密，并解密来自服务器的所有数据。与此同时，服务器也会对俩字浏览器的所有数据进行加解密。

安全连接的 URL 使用的是 HTTPS 协议而不是 HTTP 协议。一个很小的加锁也会被使用了安全连接的浏览器显示出来。当浏览器使用 SSL 主动地与服务器进行交流时，服务器会发送一个安全证书以对服务器本身进行认证。

要想使用 SSL ，你需要从一个可以信任的认证机构（CA）购买一个数字安全证书，并在网络服务器上安装这个证书。以下是一些可以信任的，有较好名誉认证机构:

- www.verisign.com
- www.geotrust.com
- www.thawte.com

SSL 是建立在所有主要的浏览器和服务器上的。要启用 SSL，你需要安装数字证书。不同数字证书的强度不同，是根据加密过程中产生的密钥长度而有所区别。密钥越长，证书就越安全，连接也就越安全。

| 强度 | 描述 |
| --- | --- |
| 40 比特 | 支持大多数浏览器但是很容易破解。 |
| 56 比特 | 比 40 比特的更健壮。 |
| 128 比特 | 很难破解，但并不是所有的浏览器都支持。 |

30

# 数据缓存

## 什么是缓存?

缓存是一种将经常使用的数据/信息存储在内存中的技术,这样,下次需要相同的数据/信息时,可以直接从内存检索,而不是再从应用程序中生成。

缓存在用于提高 ASP 性能方面是非常重要的，因为 ASP 的页面和控件是都动态生成的。这对于交互相关的数据是极其重要的,因为响应时间是很宝贵的。

在需要快速访问的媒体，如计算机的随机存取存储器，缓存放置了被频繁使用的数据。ASP 的运行时间包含一个叫做缓存的 CLR 对象的键值对。它位于应用程序内,并且通过 HttpContext 和 System.Web.UI.Page 可用。

在某些方面，缓存和存储状态对象有相似之处。然而，状态对象的存储信息是确定的，比如，你可以计算存储在状态对象的数据，但是缓存的数据是不确定的。

在下列情况里，数据是不可用的:

- 如果它的生命周期已结束，
- 如果该应用释放了它的内存，
- 如果由于某些原因缓存没有被替换。

您可以使用一个索引器在缓存中访问项目,并且有可能控制缓存中对象的生命周期和设置缓存的对象及其物理资源之间的联系。

## ASP.NET 中的缓存

ASP提供如下几种不同类型的缓存:

- **输出缓存**：输出缓存可以存储最后显现的网页的副本，或者是发送到客户机的部分页面。下次客户机请求该页面时，这个页面的缓存副本就会被发送给客户机，而不是重新生成这个页面，这样一来就节省了时间。
- **数据缓存**：数据缓存是指从数据源缓存数据。只要缓存没被替换，那么再请求该数据时就会从缓存中获取。当缓存被替换的时候，会从数据源中获取新数据，缓存也会被再次充满。
- **对象缓存**：对象缓存是缓存页面的对象，比如数据绑定控件等。缓存的数据放在服务器的内存。
- **类缓存**：网页或 Web 服务是第一次运行时在组装编译成页类。然后组装会在服务器缓存。当下次请求该页面或者服务,就会使用缓存的装配。当改变源代码时,CLR 重新编译程序集。
- **配置缓存**：应用程序配置信息存储在一个配置文件。配置缓存存储在服务器内存配置信息。

在本教程中,我们将考虑输出缓存,数据缓存和对象缓存。

## 输出缓存

呈现一个页面可能涉及一些复杂的过程,如,数据库访问，呈现复杂的控件等。输出缓存允许通过在内存中缓存数据，而绕过往返服务器。甚至可以缓存整个页面。

OutputCache 指令负责输出缓存。它启用输出缓存,并对其行为提供一定程度的控制。

OutputCache 指令的语法:

```
<%@ OutputCache Duration="15" VaryByParam="None" %>
```

把这个指令放在页面指令下。这告诉环境需要缓存页面,持续 15 秒。以下页面加载事件处理程序将帮助确认页面是否已被缓存完毕。

```
protected void Page_Load(object sender, EventArgs e)
{
   Thread.Sleep(10000);
   Response.Write("This page was generated and cache at:" +
   DateTime.Now.ToString());
}
```

Thread.Sleep()方法使特定时间内的进程停止。在这个例子中，线程停止了 10 秒钟，因此，当页面第一次下载时，需要花费 10 秒钟的时间。然而，下次刷新页面的时候，就不会花费时间了，因为这个页面已经从缓存中获取了，不要再下载。

当帮助控制输出缓存的行为 OutputCache 指令有以下特性:

| 属性 | 值 | 描述 |
| --- | --- | --- |
| DiskCacheable | true/false | 描述输出是否可以写入带有缓存的磁盘。 |
| NoStore | true/false | 描述 "no store" 缓存头部是否被发送。 |
| CacheProfile | 字符串名 | 存储在 web.config 中的缓存配置文件名字。 |
| VaryByParam | None<br>*<br>参数名 | GET 请求中使用分号分隔的字符串值或者是 POST 请求中的变量值。 |
| VaryByHeader | *<br>头文件名 | 可能是由客户端提交的用分号分隔的指定头的字符串。 |
| VaryByCustom | 浏览器<br>自定义字符串 | 通知 ASP.NET 通过浏览器名字版本或者客户端字符串改变输出缓存。 |
| Location | 任何<br>客户端 | 任何:页面可能缓存在任何位置<br>客户端:缓存内容包含在浏览器中 |

| 属性 | 值 | 描述 |
| --- | --- | --- |
| | 下载流<br>服务器<br>None | 下载流:缓存内容保存在下载流和服务器中<br>服务器:缓存仅保存在服务器之中<br>None:不允许缓存。 |
| Duration | 数字 | 被缓存页面或者操作的秒数。 |

让我们为前面的示例添加一个文本框和一个按钮，并添加这个按钮的事件处理程序。

```
protected void btnmagic_Click(object sender, EventArgs e)
{
   Response.Write("<br><br>");
   Response.Write("<h2> Hello, " + this.txtname.Text + "</h2>");
}
```

改变 OutputCache 指令:

```
<%@ OutputCache Duration="60" VaryByParam="txtname" %>
```

程序执行的时候，ASP 在文本框中依据名字缓存页面。

## 数据缓存

数据缓存的主要方面是数据源控件缓存。我们已经讨论了数据源控件代表一个数据源中的数据,如数据库或 XML 文件。这些控件从抽象类 DataSourceControl 中派生，并有以下继承属性以实现缓存:

- 缓存期 — 为缓存数据的数据源计时。
- 缓存期满策略 — 定义了当数据在缓存中过期时，缓存的行为。
- 缓存值依赖 — 定义了一个控件值，这个控件可以在数据期满时自动将其移出缓存。
- 启用缓存 — 可以确认是否缓存了数据。

### 实例

为了演示数据缓存,我们创建一个新的网站,在上面添加一个新的网络表单。在数据库中添加一个连接数据访问教程的 SqlDataSource 控件。

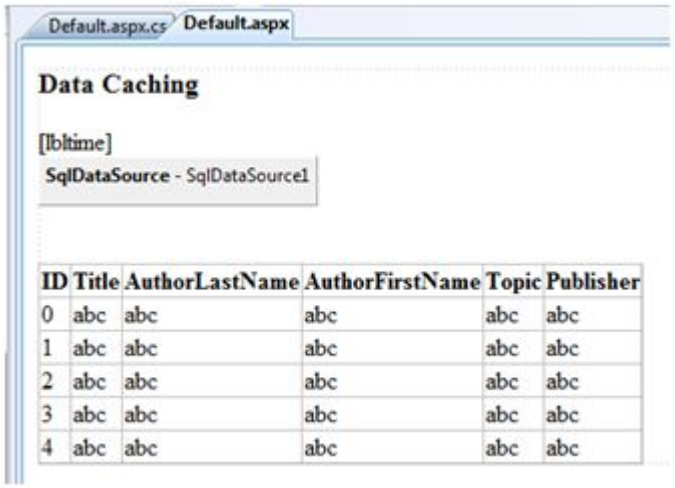
在这个实例中，我们给页面添加一个标签，这个标签可以显示页面的回应时间。

```
<asp:Label ID="lbltime" runat="server"></asp:Label>
```

除了这个标签，整个页面和数据访问教程是一样的。为这个页面添加一个事件处理器，来下载时间。

```
protected void Page_Load(object sender, EventArgs e)
{
   lbltime.Text = String.Format("Page posted at: {0}", DateTime.Now.ToLongTimeString());
}
```

设计的页面应该是如下这个样子的：




**图片 30.1** data_caching.jpg

当你第一次执行页面时，并没有发生什么不同。标签显示,每次刷新页面,页面会重新加载,而且在标签上会显示时间的变化。

接下来,把数据源控件的 EnableCaching 的属性设置为“真”,将 Cacheduration 属性设置为 “60”。它将实现缓存,缓存将每隔 60 秒到期。

每一次刷新，时间戳都会变化。但如果你在 60 秒之内改变表中的数据,在缓存到期之前将不会显示。

```
<asp:SqlDataSource ID = "SqlDataSource1" runat = "server"
   ConnectionString = "<%$ ConnectionStrings: ASPDotNetStepByStepConnectionString %>"
    ProviderName = "<%$ ConnectionStrings: ASPDotNetStepByStepConnectionString.ProviderName %>"
    SelectCommand = "SELECT * FROM [DotNetReferences]"
    EnableCaching = "true" CacheDuration = "60">
</asp:SqlDataSource>
```

## 对象缓存

对象缓存比其他缓存技术提供了更大的灵活性。你可以利用对象缓存在缓存中放置任何对象。对象也可以是任意类型的 — 数据类型，网络控件，类，数据设置对象等等。仅仅需要给这些项目分配一个值名，它们就可以被添加到缓存中，就像下面展示的这样：

```
Cache["key"] = item;
```

为了在缓存中插入对象， ASP 提供了 Insert() 方法。这种方法有四种重载版本。我们来看一下：

| 重载 | 描述 |
| --- | --- |
| Cache.Insert((key, value); | 以键值对的方式插入缓存，优先权和生命周期为默认 。 |
| Cache.Insert(key, value, dependencies); | 以键值对的方式插入缓存，优先权和生命周期为默认，和链接到其他文件或内容的缓存依赖，这样缓存修改就不再还有限的了。 |
| Cache.Insert(key, value, dependencies, absoluteExpiration, slidingExpiration); | 指出上述配置的有效期。 |
| Cache.Insert(key, value, dependencies, absoluteExpiration, slidingExpiration, priority, onRemoveCallback); | 与配置一起也允许设置缓存内容的优先权并委派，指出一种方法来调用当一个对象移除时。 |

动态生命周期使用于移除一个不作用于任何一个指定的时间跨度的缓存项。下面代码段用来保存一个具有 10 分钟滑动生命周期的无依赖的缓存项:

```
Cache.Insert("my_item", obj, null, DateTime.MaxValue, TimeSpan.FromMinutes(10));
```

**实例**

仅仅使用一个按钮和一个标签创建一个页面。在页面加载事件中写入如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
   if (this.IsPostBack)
   {
      lblinfo.Text += "Page Posted Back.<br/>";
   }
   else
   {
      lblinfo.Text += "page Created.<br/>";
   }

   if (Cache["testitem"] == null)
   {
      lblinfo.Text += "Creating test item.<br/>";
      DateTime testItem = DateTime.Now;
      lblinfo.Text += "Storing test item in cache ";
      lblinfo.Text += "for 30 seconds.<br/>";
      Cache.Insert("testitem", testItem, null,
      DateTime.Now.AddSeconds(30), TimeSpan.Zero);
   }
   else
   {
```

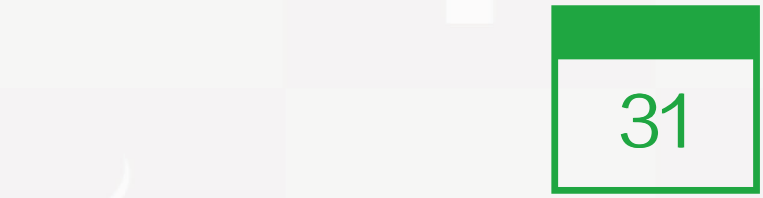
```
        lblinfo.Text += "Retrieving test item.<br/>";
        DateTime testItem = (DateTime)Cache["testitem"];
        lblinfo.Text += "Test item is: " + testItem.ToString();
        lblinfo.Text += "<br/>";
    }

    lblinfo.Text += "<br/>";
}
```

当页面第一次加载时，会显示：

```
Page Created.
Creating test item.
Storing test item in cache for 30 seconds.
```

如果你在 30 秒钟内再次点击按钮，虽然页面被删除了，但是标签控件会从缓存中得到信息，如下所示：

```
Page Posted Back.
Retrieving test item.
Test item is: 14-07-2010 01:25:04
```



# 31

# Web 服务

Web 服务是一个基于网络的功能，可被 web 应用通过 web 网络协议获取。web 服务开发主要包含以下三方面：

- 创建 web 服务
- 创建代理服务器
- 使用 web 服务

## 创建 web 服务

一个 web 服务就是一个 web 应用，基本形式为一个类包含可以被其他应用调用的多个方法，它也采用隐藏代码结构例如 ASP.NET 网页，但它不存在用户接口。

为了更好地理解这个概念让我们创建一个提供股票价格信息的 web 服务。该服务的客户端可以通过股票的标签查询相关的名字和价格。为了简化这个例子，我们设置股票价格为固定值，保存在一个二维列表中。这个 web 服务包含三个方法：

- 一个默认的 HelloWorld 方法
- 一个 GetName 方法
- 一个 GetPrice 方法

采取以下步骤创建该服务：

**步骤 (1)** : 在 Visual Studio 中选择 File -> New -> Web Site，然后选择 ASP.NET Web Service。

**步骤 (2)** : 一个名为 Service.asmx 的 web 服务文件和它的代码被隐藏，Service.cs 会在这个工程的 App_Code 路径下被创建。

**步骤 (3)** : 将文件名修改为 StockService.asmx 和 StockService.cs。

**步骤 (4)** : .asmx 文件简化了一个 WebService 指令如下：

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/StockService.cs"
Class="StockService" %>
```

**步骤 (5)** : 打开 StockService.cs 文件，在该文件里生成的代码是 Hello World 服务的基础代码。默认的 web 服务代码如下:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
```

```
using System.Linq;

using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

using System.Xml.Linq;

namespace StockService
{
  // <summary>
  // Summary description for Service1
  // <summary>

  [WebService(Namespace = "http://tempuri.org/")]
  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
  [ToolboxItem(false)]

  // To allow this Web Service to be called from script,
  // using ASP.NET AJAX, uncomment the following line.
  // [System.Web.Script.Services.ScriptService]

  public class Service1 : System.Web.Services.WebService
  {
    [WebMethod]

    public string HelloWorld()
    {
      return "Hello World";
    }
  }
}
```

步骤 (6) : 修改文件内的代码增加一个存储了各股票标签，名称和价格的字符串的二维指针，并编写获取股票信息的两个 web 方法如下；

``` using System; using System.Linq;

```
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

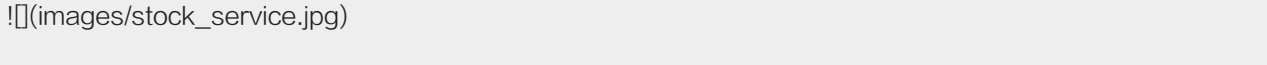
using System.Xml.Linq;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
```
```

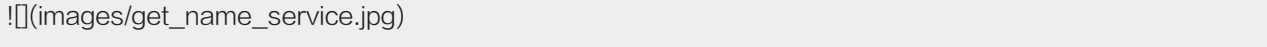
```
// To allow this Web Service to be called from script,
// using ASP.NET AJAX, uncomment the following line.
// [System.Web.Script.Services.ScriptService]

public class StockService : System.Web.Services.WebService
{
   public StockService () {
  //Uncomment the following if using designed components
  //InitializeComponent();
   }

   string[,] stocks =
   {
     {"RELIND", "Reliance Industries", "1060.15"},
     {"ICICI", "ICICI Bank", "911.55"},
     {"JSW", "JSW Steel", "1201.25"},
     {"WIPRO", "Wipro Limited", "1194.65"},
     {"SATYAM", "Satyam Computers", "91.10"}
   };

  [WebMethod]
   public string HelloWorld() {
     return "Hello World";
   }

  [WebMethod]
   public double GetPrice(string symbol)
   {
     //it takes the symbol as parameter and returns price
     for (int i = 0; i < stocks.GetLength(0); i++)
     {
       if (String.Compare(symbol, stocks[i, 0], true) == 0)
       return Convert.ToDouble(stocks[i, 2]);
     }

     return 0;
   }

   [WebMethod]
   public string GetName(string symbol)
   {
     // It takes the symbol as parameter and
     // returns name of the stock
     for (int i = 0; i < stocks.GetLength(0); i++)
```

```
    {
      if (String.Compare(symbol, stocks[i, 0], true) == 0)
      return stocks[i, 1];
    }

    return "Stock Not Found";
  }
}
```

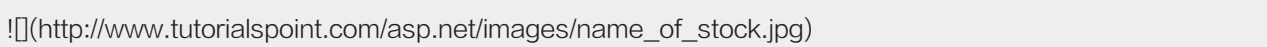
**步骤 (7)** : 运行 web 服务应用给出了一个 web 服务测试页面，我们可以在该页面测试服务方法。



**步骤 (8)** : 点击一个方法名字，确认它是否在正确运行。



**步骤 (9)** : 为检测 GetName 方法，提供已经被定义的股票标签中的一个，正确的话会返回相关股票的名称。



### 使用 Web 服务

为使用该 web 服务，我们在相同的解决方案(Solution)下创建一个网站，只需在解决方案管理器上右击该解决方案名字即可，web 服务

web 应用的文件内容如下：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>

    <form id="form1" runat="server">
      <div>

        <h3>Using the Stock Service</h3>
```

```
        <br /> <br />

        <asp:Label ID="lblmessage" runat="server"></asp:Label>

        <br /> <br />

        <asp:Button ID="btnpostback" runat="server" onclick="Button1_Click" Text="Post Back" style="width:132px" />

        <asp:Button ID="btnservice" runat="server" onclick="btnservice_Click"  Text="Get Stock" style="width:99px" />

      </div>
    </form>

  </body>
</html>
```

web 应用的代码如下：

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

//this is the proxy
using localhost;

namespace wsclient
{
  public partial class _Default : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      if (!IsPostBack)
      {
```

```
            lblmessage.Text = "First Loading Time: " +  DateTime.Now.ToLongTimeString
        }
        else
        {
            lblmessage.Text = "PostBack at: " + DateTime.Now.ToLongTimeString();
        }
    }

    protected void btnservice_Click(object sender, EventArgs e)
    {
        StockService proxy = new StockService();
        lblmessage.Text = String.Format("Current SATYAM Price:{0}",
        proxy.GetPrice("SATYAM").ToString());
    }
 }
}
```

### 创建代理服务器

代理服务器指的是一个 web 服务代码的代替者。在使用 web 服务之前，我们必须创建一个代理服务器。这个代理服务器是由客户端应

该代理服务器将调用，并用适当的格式将调用像发送 SOAP 请求一样发送到服务器。SOAP 支持简单对象访问协议（Simple Object A

当此服务器响应并返回一个 SOAP 包给客户端时，代理服务器将一切呈现给客户端应用程序。

使用 btnservice_click 调用 Web 服务之前，Web 应用应该被添加到应用程序。这将透明地创建一个代理类，可由 btnservice_click

```
protected void btnservice_Click(object sender, EventArgs e)
{
  StockService proxy = new StockService();
  lblmessage.Text = String.Format("Current SATYAM Price: {0}",
  proxy.GetPrice("SATYAM").ToString());
}
```

采取以下步骤创建代理：

**步骤 (1)** : 在解决方案管理器（SolutionExplorer）的 web 应用入口处右击选择 ‘Add Web Reference’。

!□(images/add_web_reference.jpg)

**步骤 (2)** : 选择 ‘Web Services in this solution’，会返回我们编写的股票服务引用。

!□(images/select_web_services.jpg)

**步骤 (3)** : 点击该服务打开测试页面，创建代理时默认为 ‘localhost’，当然你也可以进行重命名。点击 ‘Add Reference’ 来

!(images/stock_service2.jpg)

在代码中加入以下语句使之包含该代理：

using localhost; ```

# 32

# 多线程

一个线程被定义为一个程序的执行路径。每个线程都定义了一个独特的流量控制。如果你的应用程序涉及到复杂的和耗时的操作，如数据库访问或一些激烈的 I/O 操作，那么往往设置不同的执行路径或线程，每个线程执行一个特定的工作是非常有益的。

线程是轻量级的进程。使用线程的一个常见的例子是现代操作系统并行编程的的实现。使用线程节省了 CPU 周期的损失，提高了应用效率。

到目前为止我们编译好的程序在一个线程作为一个单一的过程运行，即是应用程序的运行实例。然而，这样的应用程序只可以在某一时刻执行一个工作。让它在同一时间执行多个任务，可以把它分成更小的线程。

在 .Net，线程是通过 ‘System.Threading’ 的命名空间处理的。创造的 system.threading.thread 类型的变量允许你创建一个新线程开始工作。它允许你在一个单独的线程创建和访问独立的线程。

## 创建线程

一个线程是由一个线程对象创建的，并给出了它的构造函数的开启线程的参考。

```
ThreadStart childthreat = new ThreadStart(childthreadcall);
```

## 线程生命周期

一个线程的生命周期开始于 system.threading.thread 类的一个对象被创建，结束于线程被终止或执行完成。

以下是在一个线程的生命周期的各种状态：

- **待开启状态：**该线程的实例被创建但启动方法还未被调用的情况。
- **就绪状态：** 当线程准备好执行并且在等待 CPU 周期的情况。
- **不可运行状态：**线程不能被运行的情况，有以下几种可能：
    - 当前睡眠的方法被调用
    - 等待的方法被调用
    - 被 I/O 操作阻塞
- **死亡状态：**线程执行完毕或已中止的情况。

## 线程优先权

Thread 类中的优先级属性主要是相对于其他线程指定一个线程的优先级。 .NET 运行时选择具有最高优先级的就绪线程。 优先权可分为：

- 高于正常
- 低于正常
- 最高
- 最低
- 正常

一旦一个线程被创建，系统就会使用 Thread 类的优先级设置系统设定好它的优先级。

```
NewThread.Priority = ThreadPriority.Highest;
```

## 线程的属性和方法

线程类具有以下重要特性：

| 属性 | 描述 |
| --- | --- |
| CurrentContext | 获取当前正在执行的线程的内容。 |
| CurrentCulture | 获取或设置当前线程的环境。 |
| CurrentPrinciple | 获取或设置当前进程关于基于角色的安全机制的原则。 |
| CurrentThread | 获取当前正在运行的线程。 |
| CurrentUICulture | 获取或设置当前运行的进程的资源管理器用于查找特定资源的当前环境。 |
| ExecutionContext | 获取包含有关当前线程的上下文信息的 ExecutionContext 对象。 |
| IsAlive | 获取一个值，指示当前线程的执行状态。 |
| IsBackground | 后台获取或设置一个值指示线程是否是后台线程。 |
| IsThreadPoolThread | 获取一个值，指示线程是否属于托管线程池。 |
| ManagedThreadId | 获取托管线程的当前唯一标识符。 |
| Name | 获取或设置线程的名称。 |
| Priority | 获取或设置一个值，表示一个线程的调度优先级。 |
| ThreadState | 获取一个值，包含当前线程的状态。 |

线程类具有以下重要方法：

| 方法 | 描述 |
| --- | --- |
| Abort | 调用一个 ThreadAbortException 开始终止线程的过程，调用此方法通常会终止线程。 |
| AllocateDataSlot | 向所有线程分配一个未命名的数据槽。为了获得更好的性能，使用标有 ThreadStaticAttribute 属性的域。 |
| AllocateNamedDataSlot | 向所有线程上分配已命名的数据槽。为了获得更好的性能，使用的是标有 ThreadStaticAttribute 属性的域。 |
| BeginCriticalRegion | 通知宿主执行即将进入代码区域，那里线程中止或未处理的异常的影响可能危及其他任务的应用领域。 |
| BeginThreadAffinity | 通知主机托管代码将要执行，取决于当前的物理操作系统线程的标识说明。 |
| EndCriticalRegion | 通知宿主执行即将进入代码区域，那里线程中止或未处理的异常仅影响当前任务。 |
| EndThreadAffinity | 通知宿主托管代码执行完成，取决于当前的物理操作系统线程的标识说明。 |
| FreeNamedDataSlot | 为进程中的所有线程消除名称与槽之间的关联，为了获得更好的性能，使用的是标有 ThreadStaticAttribute 属性的域。 |
| GetData | 在当前线程的当前域从当前线程指定的插槽检索值。为了获得更好的性能，使用的是标有 ThreadStaticAttribute 属性的域。 |
| GetDomain | 返回当前域中当前正在执行的线程。 |
| GetDomainID | 返回唯一的应用程序域标识符。 |
| GetNamedDataSlot | 查找已命名的数据槽。为了获得更好的性能，使用的是标有 ThreadStaticAttribute 属性的域。 |
| Interrupt | 中断一个在 WaitSleepJoin 线程状态的线程。 |
| Join | 阻塞调用线程，直到某个线程终止，同时继续执行标准的 COM 和 SendMessage 。该方法具有不同的重载形式。 |
| MemoryBarrier | 同步内存访问如下：处理当前线程的加速器不能以存储器访问调用 MemoryBarrier 后先调用内存访问执行这种方式对指令重新排序。 |
| ResetAbort | 取消当前线程的中止请求。 |
| SetData | 设置数据在指定的时隙上当前运行的线程，该线程的当前域。为了获得更好的性能，应用领域有 ThreadStaticAttribute 属性的域。 |
| Start | 启动一个线程。 |
| Sleep | 使线程暂停一个时间段。 |
| SpinWait | 使线程等待的参数定义的迭代次数。 |
| VolatileRead() | 读取字段的值。最新的值是由计算机的任何处理器写入，不论处理器或处理器缓存的状态数。该方法具有不同的重载形式。 |
| VolatileWrite() | 立即向字段写入一个值，这样的值是对计算机中的所有处理器可见。该方法具有不同的重载形式。 |
| Yield | 使调用线程执行可以在当前的处理器运行的另一个线程，操作系统选用转向的县城 |

例子

下面的例子阐明了对线程类的使用。该页面有一个控制标签显示子线程传来的消息。从主程序传来的消息直接使用 response.write(50) 的方法显示出来，因此它们出现在页面的顶部。

源文件如下：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-trans

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">
      <div>
        <h3>Thread Example</h3>
      </div>

      <asp:Label ID="lblmessage" runat="server" Text="Label">
      </asp:Label>
    </form>
  </body>

</html>
```

后台代码如下：

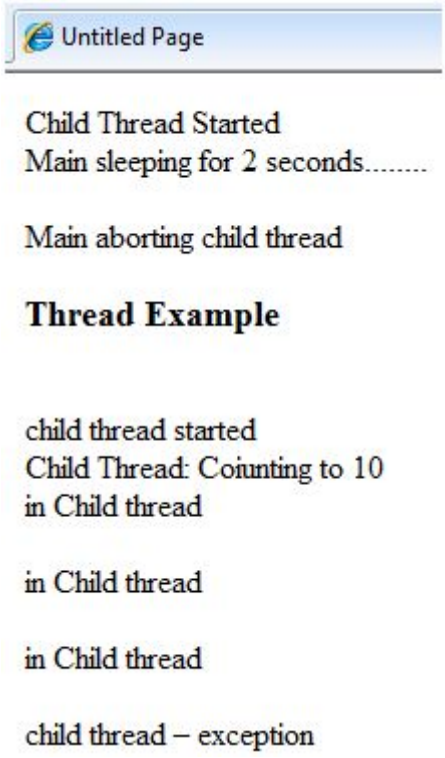
```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
```

```
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;
using System.Threading;

namespace threaddemo
{
  public partial class _Default : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      ThreadStart childthreat = new ThreadStart(childthreadcall);
      Response.Write("Child Thread Started <br/>");
      Thread child = new Thread(childthreat);

      child.Start();

      Response.Write("Main sleeping  for 2 seconds.....<br/>");
      Thread.Sleep(2000);
      Response.Write("<br/>Main aborting child thread<br/>");

      child.Abort();
    }

    public void childthreadcall()
    {
      try{
        lblmessage.Text = "<br />Child thread started <br/>";
        lblmessage.Text += "Child Thread: Coiunting to 10";

        for( int i =0; i<10; i++)
        {
          Thread.Sleep(500);
          lblmessage.Text += "<br/> in Child thread </br>";
        }

        lblmessage.Text += "<br/> child thread finished";

      }catch(ThreadAbortException e){

      lblmessage.Text += "<br /> child thread – exception";

    }finally{
      lblmessage.Text += "<br /> child thread – unable to catch the  exception";
```

```
    }
   }
  }
 }
```

## 观察以下：

- 当页面被加载时，一个新的线程会以 childthreadcall() 为参考开始启动。主线程的活动会直接显示在网页。
- 第二个线程运行并将消息发送到控制标签。
- 在子线程执行时主线程休眠 2000 毫秒。
- 子线程持续运行直到它被主线程中止，然后它会抛出 ThreadAbortException 异常并被终止。
- 控制返回到主线程。
- 当执行时程序会发送以下信息：



# 33

# 配置

一个 ASP.NET 应用程序的行为是由以下两个配置文件中的不同设置决定的：

- machine.config
- web.config

machine.config 文件包含所有支持设置项的默认和设置机器的具体值。机器的设置是由系统管理员，且应用程序通常不能访问这个文件。

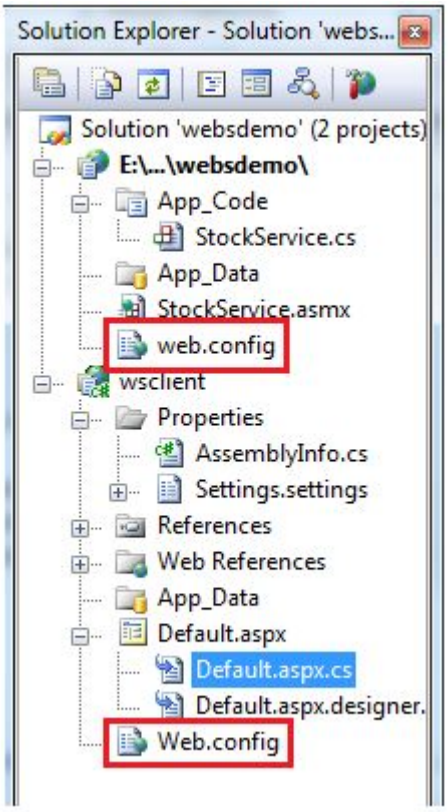
然而，一个应用程序，可以通过在它的根文件夹中创建 web.config 文件覆盖默认值。web.config 文件是 machine.config 文件的一个子集。

如果应用程序包含子目录，那么它可以为每个文件夹定义一个 web.config 文件。每个配置文件的范围是用一个分层的自上而下的方式确定。

任何 web.config 文件都可以在本地扩展，限制，或重写任何设置在上层的定义。

Visual Studio 会为每个项目生成默认的 web.config 文件。应用程序可以在没有 web.config 文件的情况下执行，然而，我们不能调试一个没有 web.config 文件的应用程序。

下图显示的是用于 web 服务教程中的解决方案资源管理器为样本的例子：




在这种应用中，存在两个 web.config 文件分别对应于调用 web 服务的 web 服务和 web 站点。

web.config 文件中的配置元素是作为根节点的。此元素中的信息分为两个主要领域：配置节处理程序声明区域，和配置节设置区域。

下面的代码片段显示了一个配置文件的基本语法：

```
<configuration>

  <!-- Configuration section-handler declaration area. -->
    <configSections>
      <section name="section1" type="section1Handler" />
      <section name="section2" type="section2Handler" />
    </configSections>
  <!-- Configuration section settings area. -->

  <section1>
    <s1Setting1 attribute1="attr1" />
  </section1>

  <section2>
    <s2Setting1 attribute1="attr1" />
  </section2>

  <system.web>
    <authentication mode="Windows" />
  </system.web>

</configuration>
```

## Configuration Section Handler 声明

配置节处理程序声明是包含在 的标签中的，每个配置处理程序指定配置节的名称，并包含在提供了一些配置数据的文件中。它具有以下基本语法：

```
<configSections>
  <section />
  <sectionGroup />
  <remove />
  <clear/>
</configSections>
```

它具有以下元素：

- Clear – 所有涉及继承的节和节组的引用。

- Remove – 删除一个继承引用的部分和部分组。
- Section – 定义了配置节处理程序和配置元素之间的关联。
- Section group – 它定义了一个配置节处理程序与配置节之间的关联。

## 应用程序设置

应用程序设置允许存储只读访问的应用程序的名称–数值对。例如，你可以定义一个自定义应用程序设置如下：

```
<configuration>
  <appSettings>
    <add key="Application Name" value="MyApplication" />
  </appSettings>
</configuration>
```

例如，你还可存储一本书的 ISBN 号和名字数据对：

```
<configuration>
  <appSettings>
    <add key="appISBN" value="0-273-68726-3" />
    <add key="appBook" value="Corporate Finance" />
  </appSettings>
</configuration>
```

## 连接字符串

连接字符串展示的是可用于网站的数据库连接字符串。例如：

```
<connectionStrings>
  <add name="ASPDotNetStepByStepConnectionString"
    connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=E:\\projects\datacaching\ /
    datacaching\App_Data\ASPDotNetStepByStep.mdb"
    providerName="System.Data.OleDb" />

  <add name="booksConnectionString"
    connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=C:\ \databinding\App_Data\books.mdb"
    providerName="System.Data.OleDb" />
</connectionStrings>
```

## 系统的网络元素

system.web 元素为 ASP.NET 配置节指定了根元素，并且包含了配置 ASP.NET Web 应用程序和控制应用程序运转的配置元素。

它控制大多数比较常见的需要调整的配置元素。该元素的基本语法如下：

```
<system.web>
  <anonymousIdentification>
  <authentication>
  <authorization>
  <browserCaps>
  <caching>
  <clientTarget>
  <compilation>
  <customErrors>
  <deployment>
  <deviceFilters>
  <globalization>
  <healthMonitoring>
  <hostingEnvironment>
  <httpCookies>
  <httpHandlers>
  <httpModules>
  <httpRuntime>
  <identity>
  <machineKey>
  <membership>
  <mobileControls>
  <pages>
  <processModel>
  <profile>
  <roleManager>
  <securityPolicy>
  <sessionPageState>
  <sessionState>
  <siteMap>
  <trace>
  <trust>
  <urlMappings>
  <webControls>
  <webParts>
  <webServices>
```

```
    <xhtmlConformance>
  </system.web>
```

下表提供了一些常用的 system.web 元素的子元素的简要描述：

### AnonymousIdentification

这是在需要用户身份确认时对未被认证的用户进行识别的。

### Authentication

它是配置授权支持的，基本的语法是：

```
  <authorization>
    <allow .../>
    <deny .../>
  </authorization>
```

### Caching

它配置缓存设置，基本的语法是：

```
  <caching>
    <cache>...</cache>
    <outputCache>...</outputCache>
    <outputCacheSettings>...</outputCacheSettings>
    <sqlCacheDependency>...</sqlCacheDependency>
  </caching>
```

### CustomErrors

它定义了自定义错误消息，基本的语法是：

```
  <customErrors defaultRedirect="url" mode="On|Off|RemoteOnly">
    <error. . ./>
  </customErrors>
```

### Deployment

它定义了用于部署的配置设置。基本语法如下：

```
<deployment retail="true|false" />
```

### HostingEnvironment

它为托管环境定义了配置设置。基本语法如下：

```
<hostingEnvironment idleTimeout="HH:MM:SS" shadowCopyBinAssemblies="true|false"
  shutdownTimeout="number" urlMetadataSlidingExpiration="HH:MM:SS" />
```

### Identity

它用于配置对应用程序的认证机制，基本语法如下:

```
<identity impersonate="true|false" userName="domain\username"
  password="<secure password>"/>
```

### MachineKey

它用于配置用于加密和解密数据的表单验证 Cookie 的密钥。

它还允许配置验证密钥对视图状态数据和 Forms 身份验证票证执行消息认证检查。基本的语法是：

```
<machineKey validationKey="AutoGenerate,IsolateApps" [String]
  decryptionKey="AutoGenerate,IsolateApps" [String]
  validation="HMACSHA256" [SHA1 | MD5 | 3DES | AES | HMACSHA256 |
  HMACSHA384 | HMACSHA512 | alg:algorithm_name]
  decryption="Auto" [Auto | DES | 3DES | AES | alg:algorithm_name]
/>
```

### Membership

它用于配置管理和认证用户参数。基本的语法是：

```
<membership defaultProvider="provider name"
  userIsOnlineTimeWindow="number of minutes" hashAlgorithmType="SHA1">
  <providers>...</providers>
</membership>
```

### Pages

它提供了网页的具体配置。基本的语法是：

```
<pages asyncTimeout="number" autoEventWireup="[True|False]"
    buffer="[True|False]" clientIDMode="[AutoID|Predictable|Static]"
    compilationMode="[Always|Auto|Never]"
    controlRenderingCompatibilityVersion="[3.5|4.0]"
    enableEventValidation="[True|False]"
    enableSessionState="[True|False|ReadOnly]"
    enableViewState="[True|False]"
    enableViewStateMac="[True|False]"
    maintainScrollPositionOnPostBack="[True|False]"
    masterPageFile="file path"
    maxPageStateFieldLength="number"
    pageBaseType="typename, assembly"
    pageParserFilterType="string"
    smartNavigation="[True|False]"
    styleSheetTheme="string"
    theme="string"
    userControlBaseType="typename"
    validateRequest="[True|False]"
    viewStateEncryptionMode="[Always|Auto|Never]" >

  <controls>...</controls>
  <namespaces>...</namespaces>
  <tagMapping>...</tagMapping>
  <ignoreDeviceFilters>...</ignoreDeviceFilters>
</pages>
```

### Profile

它用于配置用户配置文件参数。基本的语法是：

```
<profile enabled="true|false" inherits="fully qualified type reference"
  automaticSaveEnabled="true|false" defaultProvider="provider name">

  <properties>...</properties>
  <providers>...</providers>

</profile>
```

### RoleManager

为用户角色配置设置信息。基本的语法是：

```
<roleManager cacheRolesInCookie="true|false" cookieName="name"
  cookiePath="/" cookieProtection="All|Encryption|Validation|None"
```

```
    cookieRequireSSL="true|false " cookieSlidingExpiration="true|false "
    cookieTimeout="number of minutes" createPersistentCookie="true|false"
    defaultProvider="provider name" domain="cookie domain">
    enabled="true|false"
    maxCachedResults="maximum number of role names cached"

    <providers>...</providers>
  </roleManager>
```

### SecurityPolicy

用于配置安全策略。基本的语法是：

```
  <securityPolicy>
    <trustLevel />
  </securityPolicy>
```

### UrlMappings

它定义了用于隐藏原始URL的映射，并提供更具用户友好性的的 URL 。基本的语法是：

```
  <urlMappings enabled="true|false">
    <add.../>
    <clear />
    <remove.../>
  </urlMappings>
```

### WebControls

它提供了对客户端脚本共享位置的名称。基本的语法是：

```
  <webControls clientScriptsLocation="String" />
```

### WebServices

用于配置 Web 服务。

# 34

# 部署

目前存在两类 ASP.NET 部署：

- Local deployment（**本地部署**）：在这种情况下，整个应用程序都包含在一个虚拟目录下，所有的内容和程序集都包含在其中，可被应用程序使用。
- Global deployment（**全局部署**）：在这种情况下，组件可以被每一个在应用服务器上运行的应用程序所使用。

然而在部署中我们可以利用多种不同的技术，以下我们将讨论最常见和最简单的部署方式：

- XCOPY 部署
- 复制一个网站
- 创建一个设置项目

## XCOPY 部署

XCOPY 部署是说将所有文件递归拷贝到目标计算机上的目标文件夹。你可以使用以下任何的常用技术：

- FTP 传输
- 使用提供对远程站点复制的服务器管理工具
- MSI 安装程序应用

XCOPY 部署简单地拷贝程序文件到生产服务器，并在其中设置一个虚拟目录。你需要使用互联网信息微软管理控制台（MMC 管理单元）去设置虚拟目录。

## 复制一个网站

在 Visual Studio 中复制网站是可用的选项。它可在 Website -> Copy Web Site 菜单选项中实现。此菜单项允许复制当前网站到另一个本地或远程位置。它是一种集成的 FTP 工具。

使用以下几种选项连接到你的目的地来选择所需的复制模式：

- 覆盖
- 从源到目标文件
- 同步源和目标项目

然后通过物理方式复制文件。这里不像 XCOPY 部署，这个过程是从 Visual Studio 环境中进行部署的。然而，以下两个以上部署的方法都有以下问题：

- 你传递了你的源代码。
- 没有预编译和错误检查的文件。
- 初始页面加载会很慢。

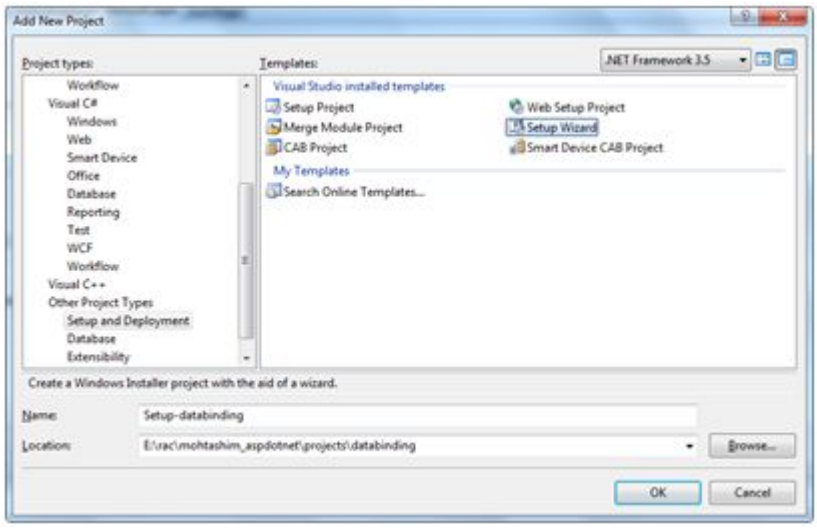
## 创建一个设置项目

在这种方法中，你使用了 Windows Installer 并且打包好 Web 应用程序使它可以部署在生产服务器。Visual Studio 允许你创建部署包。那么让我们测试一个我们现有的项目，数据绑定方案。

打开项目，采取以下步骤：

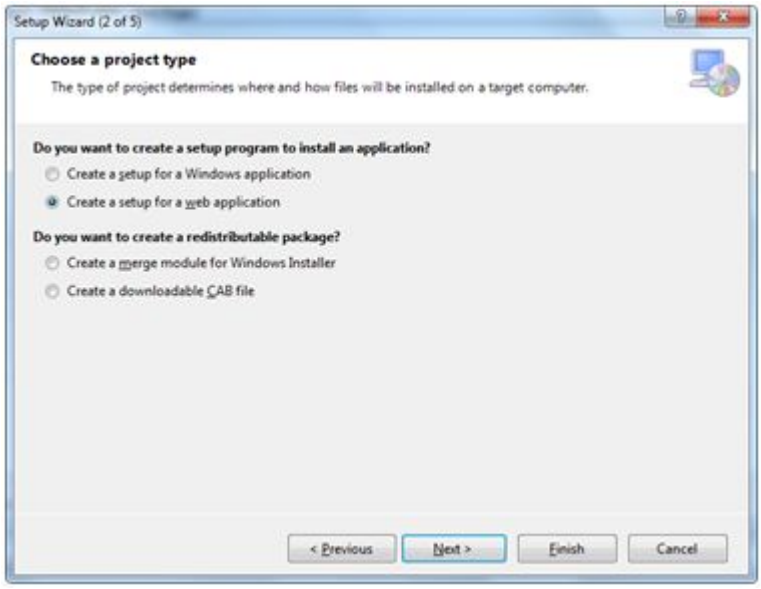
**步骤（1）：** Select File -> Add -> New Project，使用显示在解决方案资源管理器的网站根目录。

**步骤（2）：** 在 Other Project Types 下选择 Setup and Deployment，然后选择 Setup Wizard。

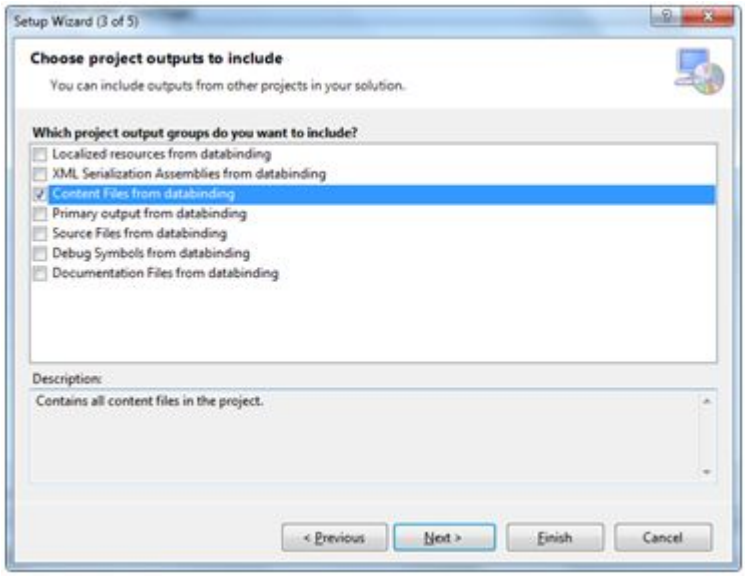



**步骤（3）：** 选择默认位置，确保将在根目录下自己的文件夹站点下建立项目。点 Okay 我们就得到了向导的第一个屏幕。

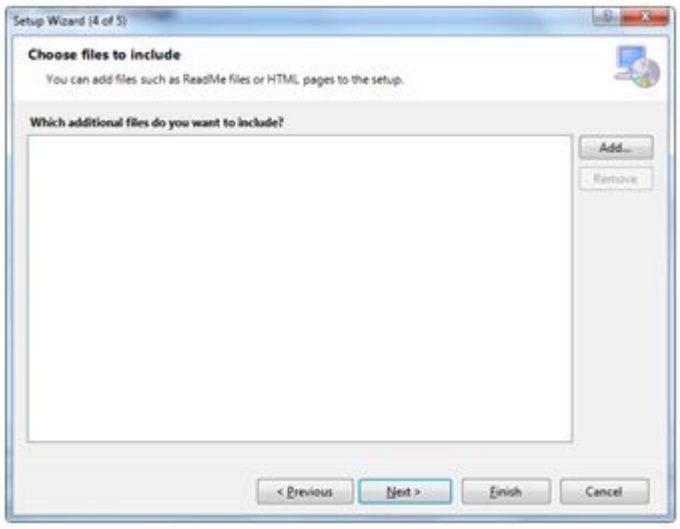

步骤（4）：选择项目类型。选择 ‘Create a setup for a web application’。



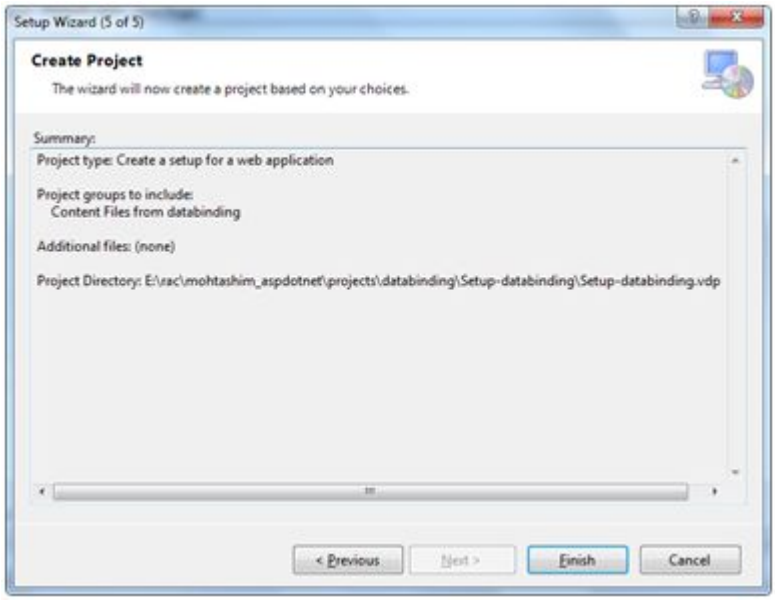

步骤（5）：下一步，第三屏要从解决方案中的所有项目选择项目输出。选中复选框旁边的 ‘Content Files from...’



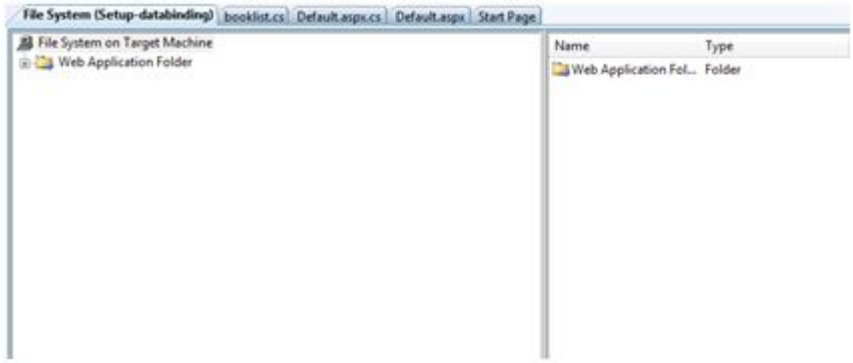

步骤（6）：第四屏幕允许包括例如自述等其他文件。然而，在我们的案例中并没有这样的文件。单击结束。



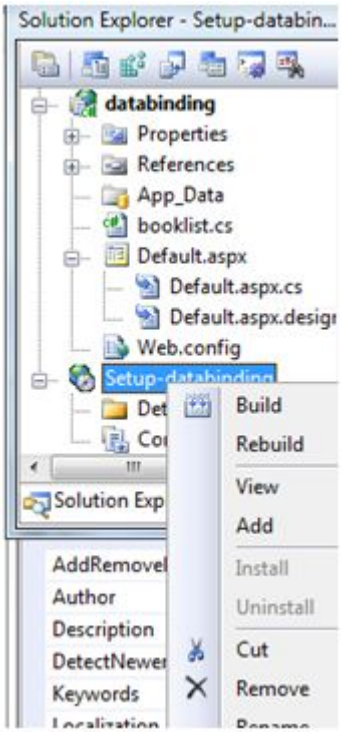

步骤（7）：最后的屏幕显示设置项目总结。




步骤（8）：设置项目添加到解决方案资源管理器，主设计窗口中会显示一个文件系统编辑器。




步骤（9）：下一步是创建安装项目。在 Solution Explorer 中右键单击项目名称，选择 Build。




步骤（10）：当建立完成后，你在输出窗口中会得到以下信息：

```
Packaging file 'Web.config'...
Packaging file 'Default.aspx'...
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

两个文件在生成过程被创建：

- Setup.exe
- Setup-databinding.msi

你需要将这些文件复制到服务器。在本地机器上双击安装文件来安装 MSI 文件的内容。