第 4 章　ASP.NET 的网页代码模型及生命周期

从本章开始，就进入了 ASP.NET 应用程序开发的世界。在了解了 C#的结构，以及面向对象的概念后，就可以从面向对象的思想开发 ASP.NET 应用程序。在 ASP.NET 中，能够使用面向对象的思想和软件开发中的一些思想，例如封装、派生、继承以及高级的设计模式等。本章首先介绍 ASP.NET 中最重要的概念---网页代码模型。

4.1　ASP.NET 的网页代码模型

在 ASP.NET 应用程序开发中，微软提供了大量的控件，这些控件能够方便用户的开发以及维护。这些控件具有很强的扩展能力，在开发过程中无需自己手动编写。不仅如此，用户还能够创建自定义控件进行应用程序开发以扩展现有的服务器控件的功能。

4.1.1　创建 ASP.NET 网站

在 ASP.NET 中，可以创建 ASP.NET 网站和 ASP.NET 应用程序，ASP.NET 网站的网页元素包含可视元素和页面逻辑元素，并不包含 designer.cs 文件。而 ASP.NET 应用程序包含 designer.cs 文件。创建 ASP.NET 网站，首先需要创建网站，单击【文件】按钮，在下拉菜单中选择【新建网站】选项，单击后会弹出对话框用于 ASP.NET 网站的创建，如图 4-1 所示。

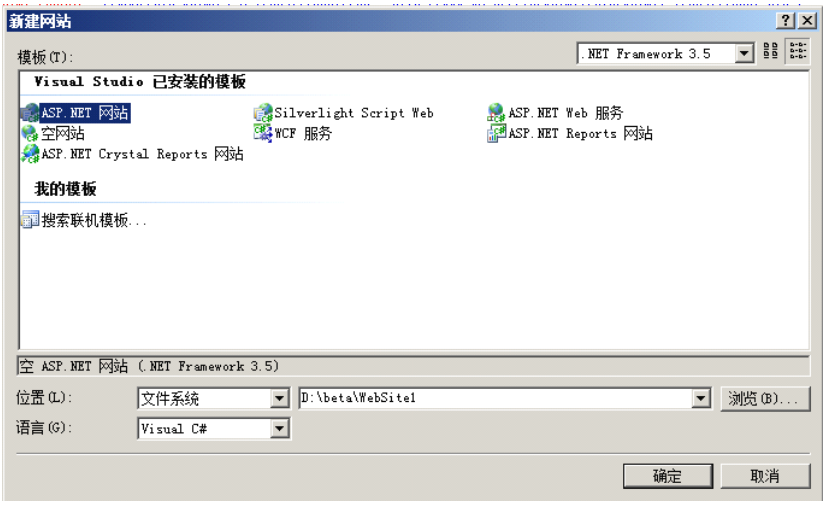

图 4-1　新建 ASP.NET 网站

在【位置】选项中，旁边的【下拉菜单】可以按照开发的需求来写，一般选择文件系统，地址为本机的本地地址。语言为.NET 网站中使用的语言，如果选择 Visual C#，则默认的开发语言为 C#，否则为 Visual Basic。创建了 ASP.NET 网站后，系统会自动创建一个代码隐藏页模型页面 Default.aspx。ASP.NET 网页一般由三部分组成，这三个部分如下所示。

- 可视元素：包括 HTML，标记，服务器空间。
- 页面逻辑元素：包括事件处理程序和代码。
- designer.cs 页文件：用来为页面的控件做初始化工作，一般只有 ASP.NET 应用程序（Web Application）才有。

ASP.NET 页面中包含两种代码模型，一种是单文件页模型，另一种是代码隐藏页模型。这两个模型的功能完全一样，都支持控件的拖拽，以及智能的代码生成。

4.1.2 单文件页模型

单文件页模型中的所有代码，包括控件代码、事物处理代码以及 HTML 代码全都包含在.aspx 文件中。编程代码在 script 标签，并使用 runat="server"属性标记。创建一个单文件页模型，在【文件】按钮中选择【新建文件】选项，在弹出对话框中选择【Web 窗体】或在右击当前项目，在下拉菜单中选择【添加新建项】选项即可创建一个.aspx 页面，如图 4-2 所示。

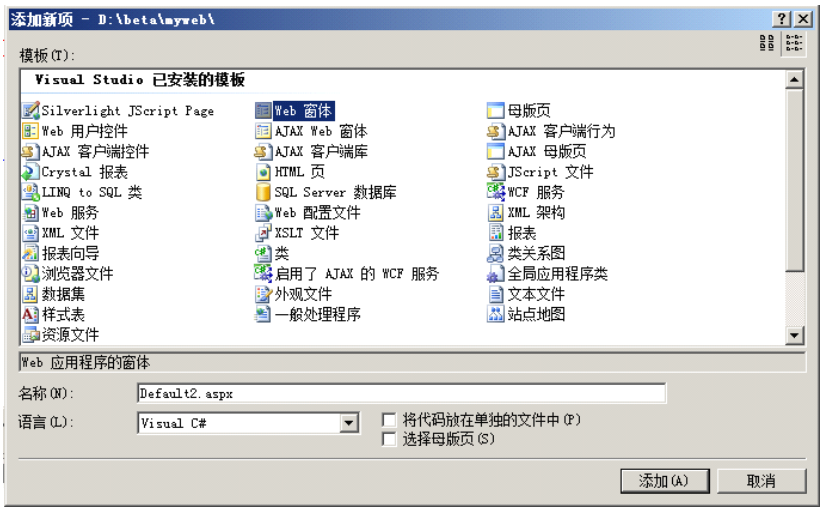

图 4-2 创建单文件页模型

在创建时，去掉【将代码放在单独的文件中】复选框的选择即可创建单文件页模型的 ASP.NET 文件。创建后文件会自动创建相应的 HTML 代码以便页面的初始化，示例代码如下所示。

```
<%@ Page Language="C#" %>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>无标题页</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    </div>
    </form>
</body>
```

```
</html>
```

编译并运行，即可看到一个空白的页面被运行了。ASP.NET 单文件页模型在创建并生成时，开发人员编写的类将编译成程序集，并将该程序集加载到应用程序域，并对该页的类进行实例化后输出到浏览器。可以说，.aspx 页面的代码也即将会生成一个类，并包含内部逻辑。在浏览器浏览该页面时，.aspx 页面的类实例化并输出到浏览器，反馈给浏览者。ASP.NET 单文件页模型运行示例图如图 4-3 所示。

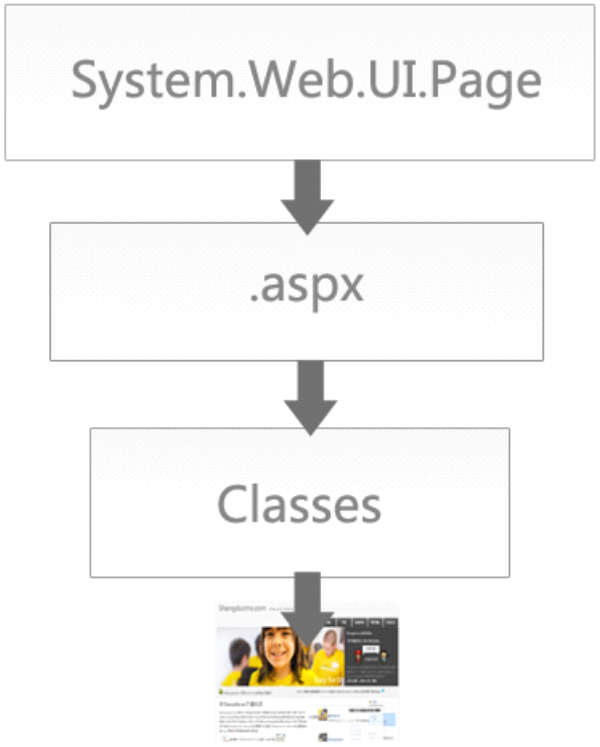

图 4-3　单文件页模型

4.1.3　代码隐藏页模型

代码隐藏页模型与单文件页模型不同的是，代码隐藏页模型将事物处理代码都存放在 cs 文件中，当 ASP.NET 网页运行的时候，ASP.NET 类生成时会先处理 cs 文件中的代码，再处理.aspx 页面中的代码。这种过程被成为代码分离。

代码分离有一种好处，就是在.aspx 页面中，开发人员可以将页面直接作为样式来设计，即美工人员也可以设计.aspx 页面，而.cs 文件由程序员来完成事务处理。同时，将 ASP.NET 中的页面样式代码和逻辑处理代码分离能够让维护变得简单，同时代码看上去也非常的优雅。在.aspx 页面中，代码隐藏页模型的.aspx 页面代码基本上和单文件页模型的代码相同，不同的是在 script 标记中的单文件页模型的代码默认被放在了同名的.cs 文件中，.aspx 文件示例代码如下所示。

```
<%@ 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-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>无标题页</title>
```

```
</head>
<body>
    <form id="form1" runat="server">
    <div>
    </div>
    </form>
</body>
</html>
```

从上述代码中可以看出，在头部声明的时候，单文件页模型只包含 Language="C#"，而代码隐藏页模型包含了 CodeFile="Default.aspx.cs"，说明被分离出去处理事物的代码被定义在 Default.aspx.cs 中，示例代码如下所示。

```
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;                          //使用 HtmlControls
using System.Web.UI.WebControls;                           //使用 WebControls
using System.Web.UI.WebControls.WebParts;                  //使用 WebParts
public partial class _Default : System.Web.UI.Page         //继承自 System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
}
```

上述代码为 Default.apx.cs 页面代码。从上述代码可以看出，其格式与类库、编写类的格式相同，这也说明了.aspx 页面允许使用面向对象的特性，如多态、继承等。但是 ASP.NET 代码隐藏页模型的运行过程比单文件页模型要复杂，运行示例图如图 4-4 所示。

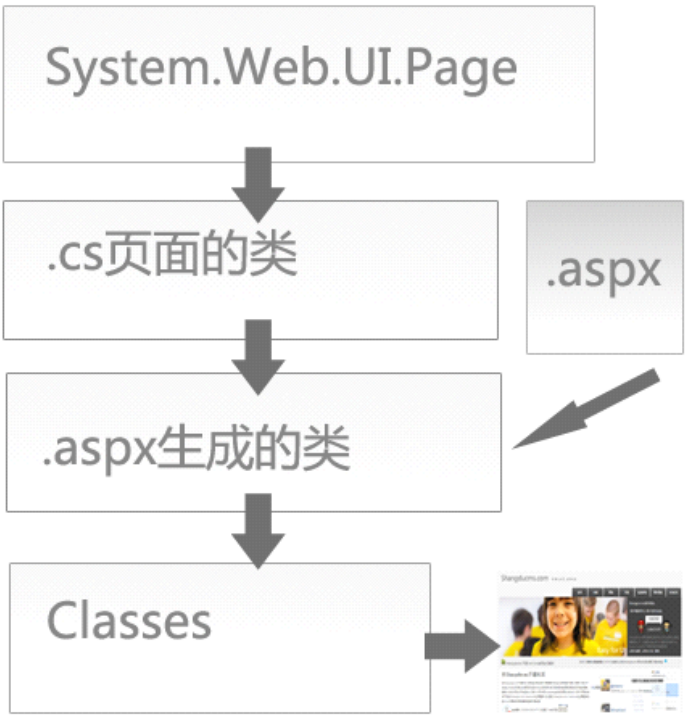

图 4-4　代码隐藏页模型

上述描述了代码隐藏类模型的页面生成模型。当页面被呈现之前，ASP.NET 应用程序会解释并编译相应的 cs 文件中的代码，与此同时，ASP.NET 应用程序还会将.aspx 页面进行编译并生成.aspx 页面对应的类。生成.aspx 页面对应的类后会将该类与 cs 文件中的类进行协调生成新的类，该类会通过 IIS 在用户浏览页面时呈现在用户的浏览器中。

4.1.4　创建 ASP.NET Web Application

ASP.NET 网站有一种好处，就是在编译后，编译器将整个网站编译成一个 DLL（动态链接库），在更新的时候，只需要更新编译后的 DLL（动态链接库）文件即可。但是 ASP.NET 网站却有一个缺点，编译速度慢，并且类的检查不彻底。

相比之下，ASP.NET Web Application 不仅加快了速度，只生成一个程序集，而且可以拆分成多个项目进行管理。创建 Application，首先需要新建项目用于开发 Web Application，单击菜单栏上的【文件】按钮，在下拉菜单中选择【新建项目】选项，在弹出窗口中选择【ASP.NET 应用程序】选项，如图 4-5 所示。

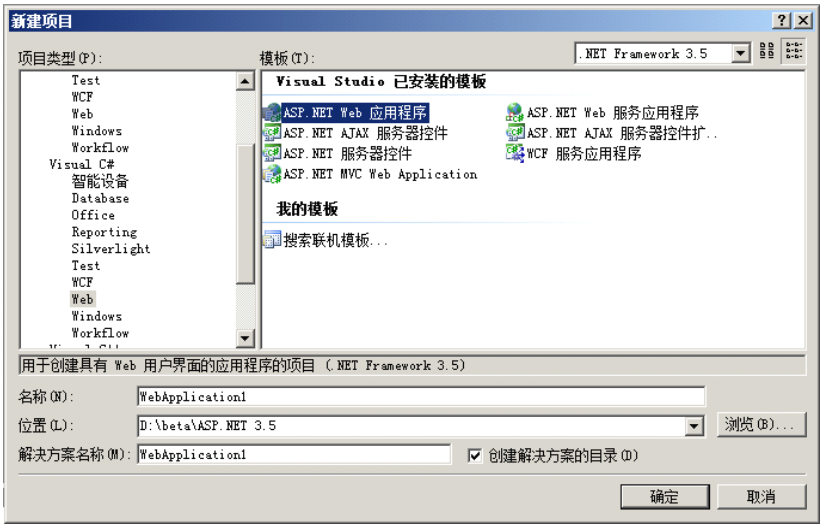

图 4-5　创建 ASP.NET 应用程序

在创建了 ASP.NET 应用程序后，系统同样会默认创建一个 Default.aspx 页面，不同的是，多出了一个 Default.aspx.designer.cs，用来初始化页面控件，一般不需要修改。

4.1.5　ASP.NET 网站和 ASP.NET 应用程序的区别

在 ASP.NET 中，可以创建 ASP.NET 网站和 ASP.NET 应用程序，但是 ASP.NET 网站和 ASP.NET 应用程序开发过程和编译过程是有区别的。ASP.NET 应用程序主要有以下特点：

- ❑ 可以将 ASP.NET 应用程序拆分成多个项目以方便开发，管理和维护。
- ❑ 可以从项目中和源代码管理中排除一个文件或项目。
- ❑ 支持 VSTS 的 Team Build 方便每日构建。
- ❑ 可以对编译前后的名称，程序集等进行自定义。

- 对 App_GlobalResources 的 Resource 强类支持。

ASP.NET WebSite 编程模型具有以下特点：

- 动态编译该页面，而不用编译整个站点。
- 当一部分页面出现错误不会影响到其他的页面或功能。
- 不需要项目文件，可以把一个目录当作一个 Web 应用来处理。

总体来说，ASP.NET 网站适用于较小的网站开发，因为其动态编译的特点，无需整站编译。而 ASP.NET 应用程序适应大型的网站开发、维护等。

4.2 代码隐藏页模型的解释过程

在 ASP.NET 的代码隐藏页模型中，一个完整的.aspx 页面包含两个页面，分别是以.aspx 和.cs 文件为后缀的文件，这两个文件在形成了整个 Web 窗体。在编译的过程中都被编译成由项目生成的动态链接库（.DLL），同时，.aspx 页面同样也会编译。但是与.cs 页面编译过程不同的是，当浏览者第一次浏览到.aspx 页面时，ASP.NET 自动生成该页的.NET 类文件，并将其编译成另一个.DLL 文件。

当浏览者再一次浏览该页面的时候，生成的.DLL 就会在服务器上运行，并响应用户在该页面上的请求或响应，ASP.NET 应用程序的解释过程图如 4-6 所示。

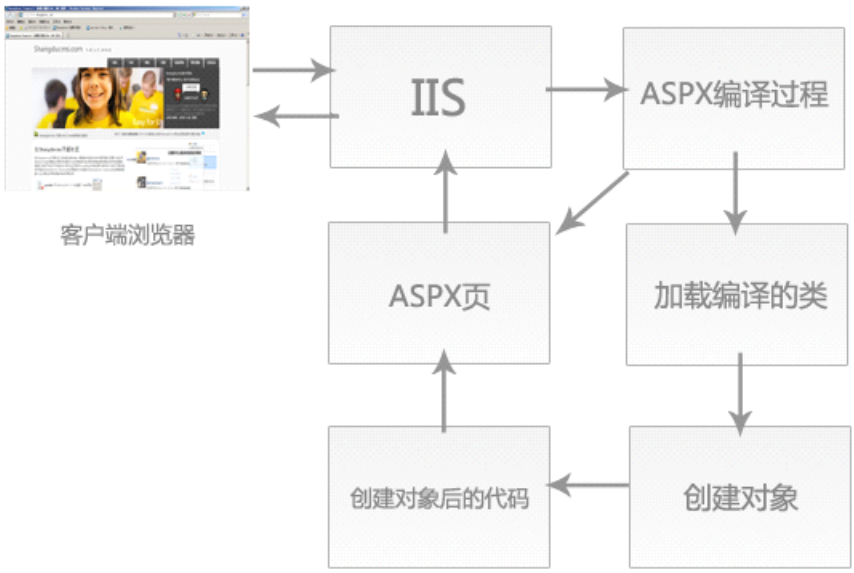

图 4-6 代码隐藏页模型页面的执行过程

在客户端浏览器访问该页面时，浏览器会给 IIS 发送请求消息，IIS 则会开始执行 ASP.NET 编译过程，如果不存在编译过后的 DLL 文件，则加载编译的类并创建对象。当创建对象完成，生成创建对象后的代码并生成一个 ASPX 页面代码，该页面代码反馈给 IIS，IIS 再反馈成 HTML 页面的形式给客户端。

4.3　代码隐藏页模型的事件驱动处理

在传统的 ASP 开发中，ASP 的事件都是按照网页的顺序来处理的，一般情况下，ASP 页面的事件都是从上到下处理事件，可以说 ASP 的开发是一个线性的处理模型。在用户的浏览操作中，每一次用户的操作都会导致页面重新被发送到服务器。因此，重复的操作必然导致客户端和服务器的往返过程，服务器必须重新创建页面，当创建页面后，服务器再按照原来的从上到下的顺序进行事件处理。

在 ASP.NET 中，通过使用模拟事件驱动模型的行为代替了 ASP 的线性处理模型。ASP.NET 页框架模型隐式的为用户建立了事件和事件处理程序的关联。ASP.NET 让用户可以为从浏览器传递的事件在服务器代码中设置相应的处理程序。假设某个用户正在浏览网站并与页面之间产生了某种交互，用户的操作就会引发事件，事件通过 HTTP 被传输到服务器。在服务器中，ASP.NET 框架解释信息，并触发事件与之对应的处理程序。该程序可以是.aspx 页面中的处理程序，也可以是开发者自定义的类库，或者 COM 组件等。事件驱动处理如图 4-7 所示。

图 4-7　页面框架的事件驱动处理模型

上图则说明了当一个浏览者通过浏览器触发 ASPX 页面时，浏览器、服务器和服务器返回页的过程 。

4.4　ASP.NET 客户端状态

Web 开发不像软件开发，Web 应用实际上是没有状态的，这就说明 Web 应用程序不自动指示序列中的请求是否来自相同的浏览器或客户端，也无法判断浏览器是否一直在浏览一个页面或者一个站点，也无法判断用户执行了哪个操作并统计用户的喜好。

4.4.1 视图状态

从上面的章节中可以知道，当服务器每次的往返过程，都将销毁页面并重新创建新的页面。如果一个页面中的信息超出了页面的生命周期，那么这个页面中的相关信息就不存在了。如果注销了页面的信息，那么用户的一些信息可能就不存在了。

在 ASP.NET 中，网页包含视图状态来保存用户的信息，视图状态在页面发回到自身时，跨页过程存储和用户自己的页面的特定值，视图状态的优点如下所示。

- 不需要任何服务器资源。
- 在默认情况下，对控件启用状态的数据进行维护，不会被破坏。
- 视图状态的值经过哈希运算和压缩保护，安全性更高。

视图状态同样有一些缺点，缺点如下所示。

- 视图状态会影响性能，如果页面存储较大较多的值，则性能会有较大的影响。
- 在手机，移动终端上，可能无法保存视图状态中使用的值。
- 视图状态虽然安全性较高，但是还是有风险，如果直接查看页面代码，可以看到相应代码。

4.4.2 控件状态

ASP.NET 中还提供了控件状态属性作为在服务器往返过程中存储自定义控件中的数据的方法。在页面控件中，如果有多个自定义控件使用多个不同的控件来显示不同的数据结构，为了让这些页面控件能够在在页面上协调的工作，则需要使用控件状态来保护控件，同时，控件状态是不能被关闭的。同样，控件状态也有它的优点，优点如下所示。

- 与视图状态相同的是，不需要任何服务器资源。
- 控件状态是不能被关闭的，提供了控件管理的更加可靠的方法。
- 控件状态具有通用性。

4.4.3 隐藏域

在 ASP 中，通常使用隐藏域保存页面的信息。在 ASP.NET 中，同样具有隐藏域来保存页面的信息，作为维护页面状态的一种形式，但是隐藏域的安全性并不高，最好不要在隐藏域保存过多的信息。隐藏域具有以下优点。

- 不需要任何服务器资源。
- 支持广泛，任何客户端都支持隐藏域。
- 实现简单，隐藏域属于 HTML 控件，无需像服务器控件那样有需要编程知识。

而隐藏域具有一些不足，如下所示。

- 具有较高的安全隐患。
- 存储结构简单。
- 同样，如果存储了较多的较大的值，则会导致性能问题。
- 如果隐藏域过多，则在某些客户端中被禁止。
- 隐藏域将数据存储在服务器上，而不存储在客户端。

注意：如果开发中，页面的隐藏域过多，这些隐藏域被存储在服务器。当客户端浏览页面的时候，会有一些防火墙扫描页面，以保证操作系统的安全，如果页面的隐藏域过多，那么这些防火墙可能会禁止页面的某些功能。

4.4.4 Cookie

Cookie 在客户端用户保存网站的少量的用户信息，服务器可以通过编程的方法获取用户信息，Cookie 信息和页面请求通常一起发送到服务器，服务器对客户端传递过来的 Cookie 信息做处理。通常 Cookie 保存用户的登录状态、用户名等基本信息等等，在后面的章节会详细介绍使用 ASP.NET 操作 Cookies。

4.4.5 客户端状态维护

虽然使用某些客户端状态并不使用服务器资源，但是这些状态都具有潜在的安全隐患，如 Cookie。非法用户可以使用 Cookie 欺骗来攻击网站进行用户信息的获取，不过使用客户端状态能够使用客户端的资源从而提高服务器性能。使用客户端状态，虽然有安全隐患，但是具有良好的编程能力，以及基本的安全知识，能够较好的解决安全问题，同时也能够提高服务器性能。下面小结了一些客户端状态的优缺点。

- 视图状态：推荐当存储少量挥发到自身的页面的信息时使用。
- 控件状态：不需要任何服务器资源，控件状态是不能被关闭的，提供了控件管理的更加可靠和更通用的方法。
- 隐藏域：实现简单，但是在应用程序中会造成一些安全隐患。
- Cookie：实现简单，同样也能够简单的获取用户的信息，但是 Cookie 有大小的限制，不适宜存储大量的代码。

4.5 ASP.NET 页面生命周期

ASP.NET 页面运行时，也同类的对象一样，有自己的生命周期。ASP.NET 页面运行时，ASP.NET 页面将经历一个生命周期，在生命周期内，该页面将执行一系列的步骤，包括控件的初始化，控件的实例化，还原状态和维护状态等，以及通过 IIS 反馈给用户呈现成 HTML。

ASP.NET 页面生命周期是 ASP.NET 中非常重要的概念，了解 ASP.NET 页面的生命周期，就能够在合适的生命周期内编写代码，执行事务。同样，熟练掌握 ASP.NET 页面的生命周期，可以开发高效的自定义控件。ASP.NET 生命周期通常情况下需要经历几个阶段，这几个阶段如下所示。

- 页请求：页请求发生在页生命周期开始之前。当用户请求一个页面，ASP.NET 将确定是否需要分析或者编译该页面，或者是否可以在不运行页的情况下直接请求缓存响应客户端。
- 开始：发生了请求后，页面就进入了开始阶段。在该阶段，页面将确定请求是发回请求还是新的客户端请求，并设置 IsPostBack 属性。
- 初始化：在页面开始后，进入了初始化阶段。初始化期间，页面可以使用服务器控件，并为每个服务器控件进行初始化。
- 加载：页面加载控件。
- 验证：调用所有的验证程序控件的 Vailidate 方法，来设置各个验证程序控件和页的属性。

- 回发事件：如果是回发请求，则调用所有事件处理的程序。
- 呈现：在呈现期间，视图状态被保存并呈现到页。
- 卸载：完全呈现页面后，将页面发送到客户端并准备丢弃时，将调用卸载。

4.6 ASP.NET 生命周期中的事件

在页面周期的每个阶段，页面将引发可运行用户代码进行处理事件。对于控件产生的事件，通过声明的方式执行代码，并将事件处理程序绑定到事件。不仅如此，事件还支持自动事件连接，最常用的就是 Page_Load 事件了，除了 Page_Load 事件以外，还有 Page_Init 等其他事件，本节将会介绍此类事件。

4.6.1 页面加载事件（Page_PreInit）

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

```
protected void Page_PreInit(object sender, EventArgs e)                    //Page_PreInit 事件
{
    Label1.Text = “OK”;                                                    //标签赋值
}
```

在上述代码中，当触发了 Page_PreInit 事件时，就会执行该事件的代码，上述代码将 Lable1 的初始文本值设置为“OK”。Page_PreInit 事件能够让用户在页面处理中，能够让服务器加载时只执行一次而当网页被返回给客户端时不被执行。在 Page_PreInit 中可以使用 IsPostBack 来实现，当网页第一次加载时 IsPostBack 属性为 false，当页面再次被加载时，IsPostBack 属性将会被设置为 true。IsPostBack 属性的使用能够影响到应用程序的性能。

4.6.2 页面加载事件（Page_Init）

Page_Init 事件与 Page_PreInit 事件基本相同，区别在于 Page_Init 并不能保证完全加载各个控件。虽然在 Page_Init 事件中，依旧可以访问页面中的各个空间，但是当页面回送时，Page_Init 依然执行所有的代码并且不能通过 IsPostBack 来执行某些代码，示例代码如下所示。

```
protected void Page_Init(object sender, EventArgs e)                      //Page_Init 事件
{
    if (!IsPostBack)                                                       //判断是否第一次加载
    {
        Label1.Text = “OK”;                                                //将成功信息赋值给标签
    }
    else
    {
        Label1.Text = “IsPostBack”;                                        //将回传的值赋值给标签
    }
}
```

4.6.3 页面载入事件（Page_Load）

大多数初学者会认为 Page_Load 事件是当页面第一次访问触发的事件，其实不然，在 ASP.NET 页生命周期内，Page_Load 远远不是第一次触发的事件，通常情况下，ASP.NET 事件顺序如下所示。

- 1. Page_Init()。
- 2. Load ViewState。
- 3. Load Postback data。
- 4. Page_Load()。
- 5. Handle control events。
- 6. Page_PreRender()。
- 7. Page_Render()。
- 8. Unload event。
- 9. Dispose method called。

Page_Load 事件是在网页加载的时候一定会被执行的事件。在 Page_Load 事件中，一般都需要使用 IsPostBack 来判断用户是否进行了操作，因为 IsPostBack 指示该页是否正为响应客户端回发而加载，或者它是否正被首次加载和访问，示例代码如下所示。

```
protected void Page_Load(object sender, EventArgs e)                    //Page_Load 事件
{
    if (!IsPostBack)
    {
        Label1.Text = “OK”;                                              //第一次执行的代码块
    }
    else
    {
        Label1.Text = “IsPostBack”;                                      //如果用户提交表单等
    }
}
```

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

4.6.4 页面卸载事件（Page_Unload）

在页面被执行完毕后，可以通过 Page_Unload 事件用来执行页面卸载时的清除工作，当页面被卸载时，执行此事件。以下情况会触发 Page_Unload 事件。

- 页面被关闭。
- 数据库连接被关闭。
- 对象被关闭。
- 完成日志记录或者其他的程序请求。

4.6.5 页面指令

页面指令用来通知编译器在编译页面时做出的特殊处理。当编译器处理 ASP.NET 应用程序时，可以通过这些特殊指令要求编译器做特殊处理，例如缓存、使用命名空间等。当需要执行页面指令时，通

常的做法是将页面指令包括在文件的头部，示例代码如下所示。

```
<%@ Page
Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="MyWeb._Default" %>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

上述代码中，就使用了@Page 页面指令来定义 ASP.NET 页面分析器和编译器使用的特定页的属性。当代码隐藏页模型的页面被创建时，系统会自动增加@Page 页面指令。

ASP.NET 页面支持多个页面指令，常用的页面指令如下所示。

- @ Page：定义 ASP.NET 页分析器和编译器使用的页特定（.aspx 文件）属性，可以编写为<%@ Page attribute="value" [attribute="value"…]%>。
- @ Control：定义 ASP.NET 页分析器和编译器使用的用户控件（.ascx 文件）特定的属性。该指令只能为用户控件配置。可以编写为<%@ Control attribute="value" [attribute="value"…]%>。
- @ Import：将命名空间显示导入到页中，使所导入的命名空间的所有类和接口可用户该页。导入的命名空间可以是.NET Framework 类库或用户定义的命名空间的一部分。可以编写为<%@ Import namespace="value" %>。
- @ Implements：提示当前页或用户控件实现制定的.NET Framework 接口。可以编写为<%@ Implements interface="ValidInterfaceName" %>。
- @ Reference：以声明的方式指示，应该根据在其中声明此指令的页对另一个用户控件或页源文件进行动态编译和链接。可以编写为<%@ Reference page | control="pathtofile" %>。
- @ Output Cache：以声明的方式空间 ASP.NET 页或页中包含的用户控件的输出缓存策略。可以编写为 <%@ Output Cache Duration="#ofseconds" Location="Any | Client | Downstream | Server | None" Shared="True |False" VaryByControl="controlname" VaryByCustom="browser | customstring" VaryByHeader="headers" VaryByParam="parametername" %>
- @ Assembly：在编译过程中将程序集链接到当前页，以使程序集的所有类和接口都可用在该页上。可以编写为<%@ Assembly Name="assemblyname" %>或<%@ Assembly Src="pathname" %>的方式。
- @ Register：将别名与命名空间以及类名关联起来，以便在自定义服务器控件语法中使用简明的表示法。可以编写为 <%@ Register tagprefix=" tagprefix" Namespace="namepace" Assembly="assembly" %> 或 <%@ Register tagprefix=" tagprefix" Tagname="tagname" Src="pathname" %>的方式。

4.7 ASP.NET 网站文件类型

在 ASP.NET 中包含诸多的文件类型，这些类型的文件由 ASP.NET 支持和管理，而除了这些文件以外，其他的文件都由 IIS 托管。使用 VS2008 能够创建大部分可以使用 ASP.NET 托管运行的程序。同时，使用应用程序映射可以将文件类型映射到应用程序。当需要伪静态时，很可能需要将.html 后缀托管到 IIS 中的应用扩展，因为默认情况下 ASP.NET 不会处理 HTML 的操作。

技巧：现在的网站构架中，生成静态是一种降低网站压力的一种很好的解决方案。在某些情况下，服务器可能需要伪静态支持，就是将.aspx 页面后缀显式成.html 后缀，让搜索引擎能够更好的搜录。

1．ASP.NET 管理的文件类型

ASP.NET 管理的文件类型能够在 ASP.NET 应用程序中被 ASP.NET 应用程序的不同模块进行访问

和调用，这些文件可能是用户能够直接访问的，也有可能是用户无法直接访问的。ASP.NET 管理的文件类型如表 4-1 所示。

表 4-1　ASP.NET管理的文件类型

文件类型	保存位置	描述
.asax	根目录。	Global.asax 文件。包含 HttpApplication 对象的派生代码，用于重新展示 Application 对象。
.ascx	根目录或子目录。	可重用的自定义 Web 控件。
.ashx	根目录或子目录。	处理器文件。包含实现 IHttpHandler 接口的代码，用于处理输入请求。
.asmx	根目录或子目录。	XML Web Services 文件。包含由 SOAP 提供给其他 Web 应用的类对象和功能。
.aspx	根目录或子目录。	ASP.NET Web 窗体。包含 Web 控件和其他业务逻辑。
.axd	根目录。	跟踪视图文件。通常是 Trace.axd。
.browser	App_Browsers 目录。	浏览器定义文件。用于识别客户端浏览器的可用特征。
.cd	根目录或子目录。	类图文件。
.compile	Bin 目录。	定位于适当汇编集中的预编译文件。可执行文件（.aspx，.ascx，.master，theme）预编译后放在 Bin 目录。
.config	根目录或子目录。	Web.config 配置文件。包含用于配置 ASP.NET 若干特征的 XML 元素集。
.cs，.jsl，vb	App_Code 目录。有些是 ASP.NET 的代码分离文件，位于与 Web 页面相同的目录。	运行时被编译的类对象源代码。类对象可以是 HTTP 模块，HTTP 处理器，或 ASP.NET 页面的代码分离文件。
.csproj，vbproj，vjsproj	Visual Studio 工程目录。	Visual Studio 客户工程文件。
.disco，.vsdisco	App_WebReferences 目录。	XML Web Services Discovery 文件。用于定位可用 Web Services。
.dsdgm，dsprototype	根目录或子目录。	分布式服务图表（DSD）文件。可添加到 Visual Studio 方案中，为反向引擎提供消耗 Web Services 时的交互性图表。
.dll	Bin 目录。	已编译类库文件。作为替代，可将类对象源代码保存到 App_Code 目录。
.licx，.webinfo	根目录或子目录。	许可协议文件。许可协议有助于保护控件开发者的知识产权，并对控件用户的使用权进行验证。
.master	根目录或子目录。	模板文件定义 Web 页面的统一布局，并在其他页面中得到引用。
.mdb，.ldb	App_Data 目录。	Access 数据库文件。
.mdf	App_Data 目录。	SQLServer 数据库文件。
.msgx，.svc	根目录或子目录。	Indigo Messaging Framework（MFx）服务文件。
.rem	根目录或子目录。	远程处理器文件。
.resources	App_GlobalResources 或 App_LocalResources 目录。	资源文件。包含图像，本地化文本，或其他数据的资源引用串。
.resx	App_GlobalResources 或 App_LocalResources 目录。	资源文件。包含图像，本地化文本，或其他数据的资源引用串。
.sdm，.sdmDocument	根目录或子目录。	系统定义模型（SDM）文件。
.sitemap	根目录。	网站地图文件。包含网站的结构。ASP.NET 通过默认的网站地图提供者，简化导航控件对网站地图文件的使用。
.skin	App_Themes 目录。	皮肤定义文件。用于确定显示格式。
.sln	Visual Web Developer 工程目录。	Visual Web Developer 工程的项目文件。
.soap	根目录或子目录。	SOAP 扩展文件。

注意：ASP.NET 管理的文件类型映射到 IIS 的 Aspnet_isapi.dll。

2．IIS 管理的文件类型

在 ASP.NET 应用程序中，有些动态的文件如 asp 文件就不被 ASP.NET 应用程序框架管理，这些文件由 IIS 进行管理，由 IIS 管理的文件类型如表 4-2 所示。

表 4-2　IIS管理的文件类型

文件类型	保存位置	描述
.asa	根目录。	Global.asa 文件。包含 ASP 会话对象或应用程序对象生命周期中的各种事件处理。
.asp	根目录或子目录.	ASP Web 页面。包含 @ 指令和使用 ASP 内建对象的脚本代码。
.cdx	App_Data 目录.	Visual FoxPro 的混合索引文件。
.cer	根目录或子目录。	证明文件。用于对网站的授权。
.idc	根目录或子目录。	Internet Database Connector（IDC）文件。被映射到 httpodbc.dll。 注意：由于无法为数据库连接提供足够的安全性，IDC 将不再被继续使用。IIS 6.0 是最后一个支持 IDC 的版本。
.shtm，.shtml，.stm	根目录或子目录。	包含文件。被映射到 ssinc.dll。

注意：IIS 管理的文件类型被映射到 IIS 的 asp.dll

3．静态文件类型

IIS 仅提供已注册 MIME 类型的静态文件服务，注册信息保存在 Mime Map IIS 元数据库中。如果某种文件类型已经映射到指定应用程序，在不需要作为静态文件的情况之下，无需再在 MIME 类型列表中进行包含。默认的静态文件类型如表 4-3 所示。

表 4-3　静态文件类型

文件类型	保存位置	描述
.css	根目录或子目录，以及 App_Themes 目录。	样式表文件。用于确定 HTML 元素的显示格式。
.htm，.html	根目录或子目录。	静态网页文件。由 HTML 代码编写。

注意：虽然 ASP.NET 的代码页面也能够手动添加到 MIME 类型列表中，但是这样操作浏览者就能够看到页面源代码，从而暴露 ASP.NET 页面源代码，相对于服务器而言是非常不安全的。

4.8　小结

本章介绍了 ASP.NET 页面生命周期，以及 ASP.NET 页面的几种模型。ASP.NET 页面生命周期是 ASP.NET 中非常重要的概念，熟练掌握 ASP.NET 生命周期能对 ASP.NET 开发，自定义控件开发起到促进作用。本章还介绍了：

- 代码隐藏页模型的解释过程。
- 代码隐藏页模型的事件驱动处理。
- ASP.NET 网页的客户端状态。
- ASP.NET 页面生命周期。
- ASP.NET 生命周期中的事件。
- ASP.NET 网站文件类型。

上面的章节都分开的讲解了 ASP.NET 运行中的一些基本机制，在了解了这些基本运行机制后，就能够在.NET 框架下做 ASP.NET 开发了。虽然这些都是基本概念，但是在今后的开发中，会起到非常重要的作用。