

# 上讲回顾

**HTML**

内容

**CSS**

样式

**JavaScript**

行为

# 网页前端开发基础：JS

## ❖ 在ASP.NET中使用弹窗

为按钮添加属性：

```
Button1.Attributes["OnClick"] = "return confirm('are  
you sure?')";
```

确认则执行按钮响应； 否则取消。

# 第3章 ASP.NET技术基础

# 内容

- ❖ **ASP.NET应用程序生命周期**
- ❖ **ASP.NET网页**
- ❖ **Page类的内置对象**
- ❖ **Web应用的配置与配置管理工具**
- ❖ **Web应用的异常处理**

Part 1



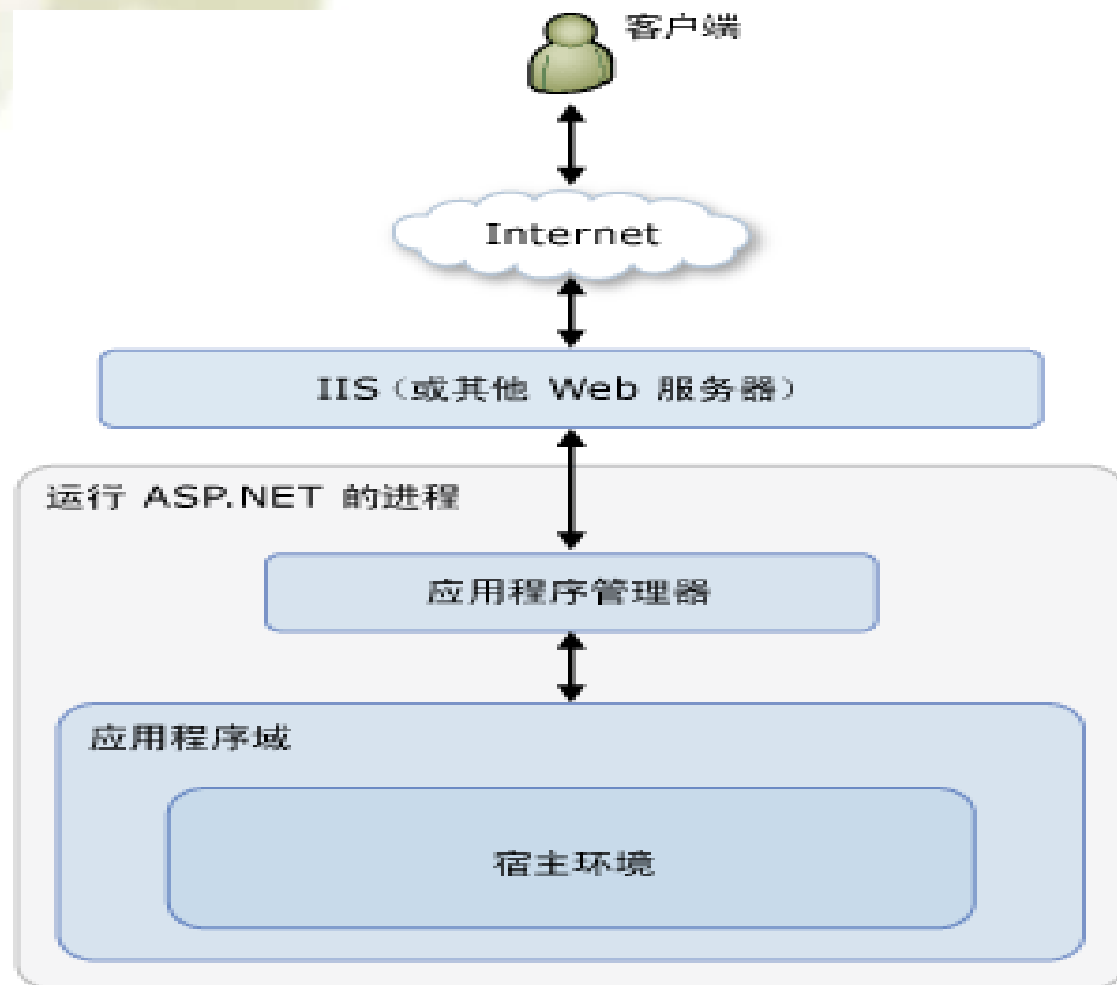
# ASP.NET应用程序生命周期

## 3.1.1 应用程序生命周期概述

- ❖ 阶段1：用户从**Web**服务器请求应用程序资源。
  - **ASP.NET**应用程序的生命周期以**浏览器向Web服务器发送请求为起点**；
  - **Web**服务器接收到请求时，会**对所请求的文件的文件扩展名进行检查，确定应由哪个ISAPI扩展处理该请求**；
  - 然后将该请求传递给合适的**ISAPI**扩展。**ASP.NET**处理已映射到其上的文件扩展名有**.aspx**、**.ascx**、**.ashx**和**.asmx**等。

启动ASP.NET

❖ 阶段2: **ASP.NET**接受对应用程序的第一个请求, 应用程序管理器**创建一个应用程序域**



应用程序域为全局变量提供应用程序隔离, 创建宿主环境。

准备物理环境

## ❖ 阶段3：为每个请求创建ASP.NET核心对象

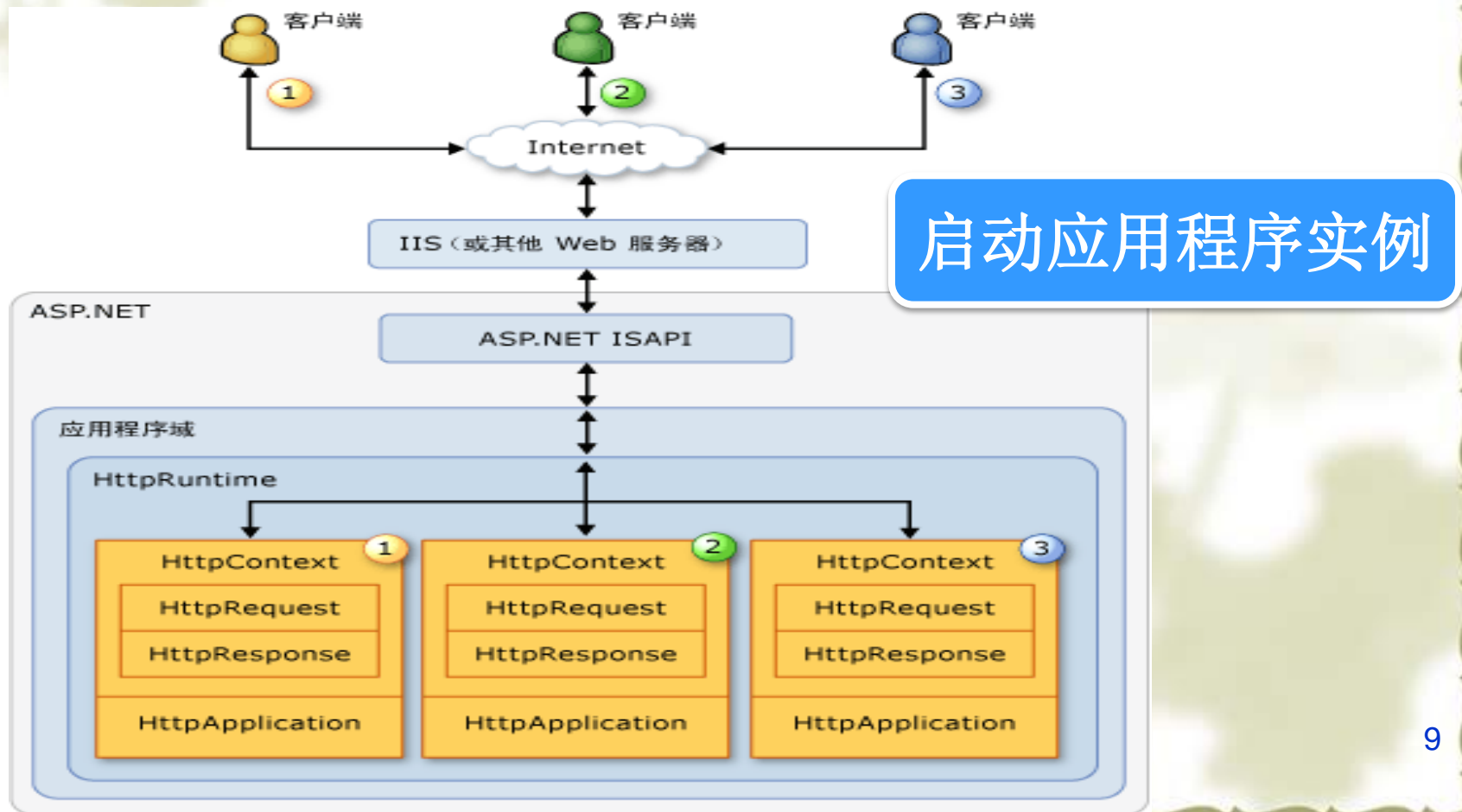
- 创建了应用程序域并实例化了宿主环境之后，ASP.NET将为每个用户**创建并初始化核心对象**（如HttpContent、HttpRequest和HttpResponse）。
- HttpContent类包含特定于当前应用程序请求的对象，如HttpRequest和HttpResponse对象。
- HttpRequest对象包含有关当前请求的信息，如Cookie和浏览器信息。
- HttpResponse对象包含发送到客户端的响应，即所有呈现的输出和Cookie。

准备软件（ASP.NET核心对象）



## ❖ 阶段4：将HttpApplication对象分配给请求

创建HttpApplication类的实例启动应用程序（创建Global.asax类实例、创建配置模块、初始化）



## ❖ 阶段5：由**HttpApplication**管线处理请求

- 在该阶段，将由**HttpApplication**类执行一系列的事件（如 **BeginRequest**、**ValidateRequest**等）；
- 根据所请求资源的文件扩展名，选择**IHttpHandler**类来处理请求。
- 如果该请求是从**Page**类派生的页，则**ASP.NET**会在创建该页的实例前对其进行编译，在装载后用该实例处理这个请求，处理完后通过**HttpResponse**输出，最后释放该实例。

应用程序处理请求

## 3.1.2应用程序生命周期事件

- ❖ 在**ASP.NET**中，当应用程序启动或终止时，都会触发一些事件，使得这些事件可以完成一些特殊的处理工作，例如**错误处理、撰写日志、状态变量初始化**等。
- ❖ **ASP.NET**中，这些事件位于**Global.asax**文件中，开发人员可以在该文件中编写代码响应这些应用程序事件。

## 【例3-1】 演示Global.asax文件的创建及使用。

Application_Start()	在应用程序启动后，当第一个用户请求时触发这个事件，后继的用户请求将不会触发该事件，在该事件中通常用于创建或者缓存一些初始信息便于以后使用
Application_End()	当应用程序关闭时，比如Web服务器重新启动时触发事件，可以在这个事件中插入清除代码
Application_Error()	该事件响应未被处理的错误
Session_Start()	只要有用户请求时，就会触发该事件，该事件对于每个请求的用户都会触发一次，如有100个用户请求，则触发100次
Session_End()	当会话超时或者以编程的方式终止会话时，这个事件被触发

Part 2

# ASP.NET网页



## 3.2 ASP.NET网页

- ❖ **ASP.NET网页语法概述**
- ❖ **ASP.NET网页代码模型**
- ❖ **Page类的属性**
- ❖ **ASP.NET网页的生命周期与Page类的事件**
- ❖ **ASP.NET网页的添加**

## 3.2.1 ASP.NET网页语法概述

❖ **Web窗体文件的扩展名为.aspx**，该类文件的语法结构主要由以下几部分组成：

- 指令
- head
- form（窗体）元素
- Web服务器控件或HTML控件
- 客户端脚本
- 服务端脚本



# 1. 指令

**@Page:** 允许为页面指定多个配置项，包括：

- 页面中代码的服务器编程语言。
- 页面是将服务器代码直接包含在其中（称为单文件页面），还是将代码包含在单独的类文件中（称为代码隐藏页面）。

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



**2. Head:** 在head中的内容不会被显示（除标题外），但它们对于浏览器是非常有用的信息，如脚本和样式表等内容。

**3. form（窗体）元素：** 如果页面包含允许用户与页面交互并提交该页面的控件，则该页面必须包含一个form元素

下面是一个典型的<form>标记：

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

.....

```
</form>
```

## 4. Web服务器控件

在大多数**ASP.NET**页中，都需要添加**允许用户与页面交互的控件**，包含按钮、文本框、列表等。  
下面是**Web服务器控件**使用示例：

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

```
...
```

```
<asp:TextBox ID="TextBox1" runat="server"> </asp:TextBox>
```

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

```
...
```

```
</form>
```

## 5. 将HTML元素作为服务器控件

将普通的HTML元素作为服务器控件使用，可以将**runat="server"属性**和**id属性**添加到页面的任何HTML元素中即可。

下面是HTML元素转换为服务器控件示例：

```
<body runat="server" id="Body">
```

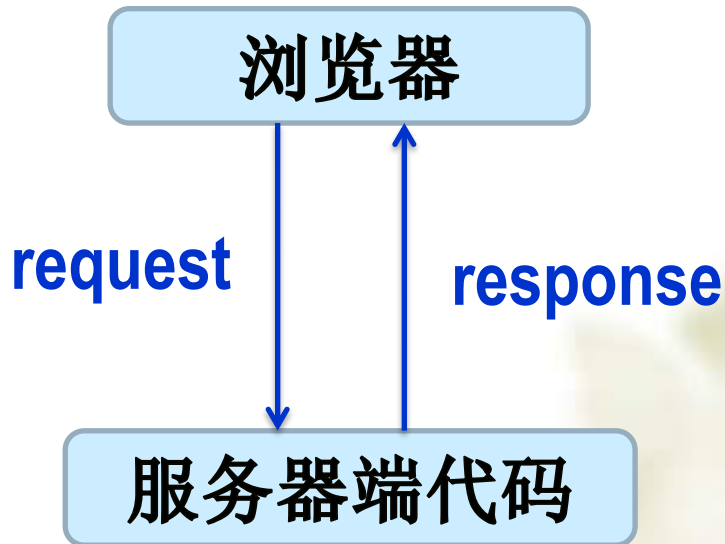
## 6. 客户端代码

客户端代码是在浏览器中执行的，因此执行客户端代码不需要回发Web窗体。客户端代码语言支持JavaScript、VBScript、Jscript和ECMAScript，下面是客户端代码示例：

```
<form id="form1" runat="server">  
    <input type="button" id="btn" value="点击"  
        onclick="show()">  
    <script language="vbscript" type="text/vbscript">  
        sub show()  
            alert("error!")  
        end sub  
    </script>  
</form>
```

## 7. 服务器端代码

- 大多数**ASP.NET**页包含当处理页面时在**服务器上运行的代码**。**ASP.NET**支持多种语言，包括**C#**、**Visual Basic.NET**、**J#**、**Jscript**和其它语言。

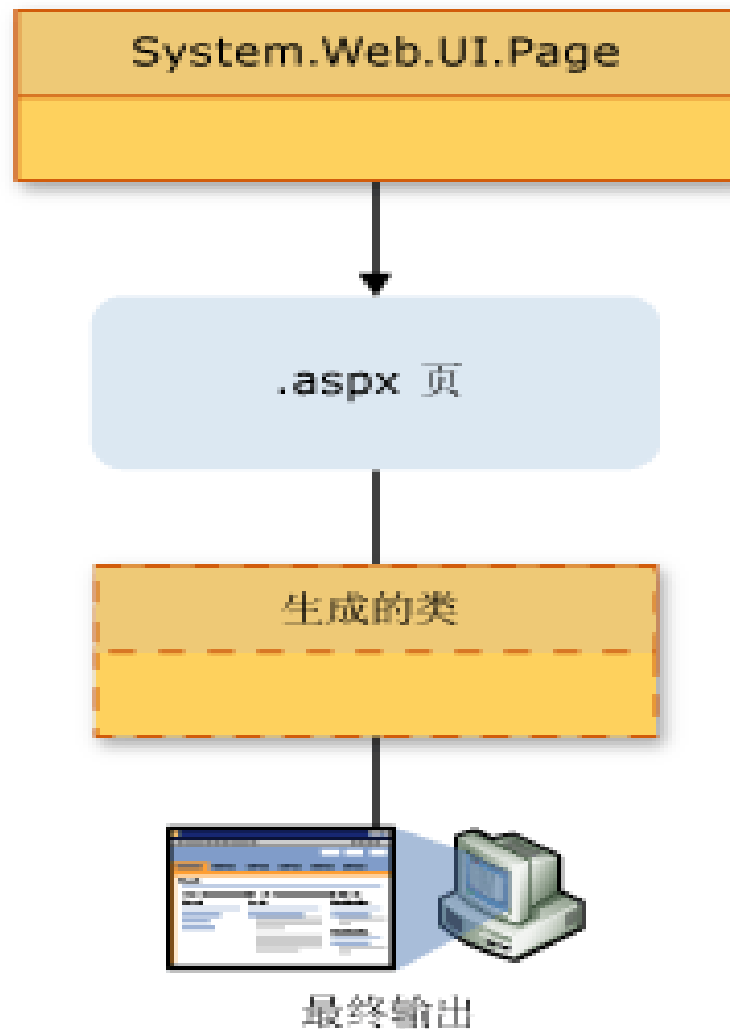


## 3.2.2 ASP.NET网页代码模型

- ❖ **ASP.NET网页由两部分组成：**
  - 可视元素，包括标记、服务器控件和静态文本。
  - 网页的编程逻辑，包括事件处理程序和其它代码。
- ❖ **ASP.NET提供两个用于管理可视元素和代码的模型，即单文件页模型和代码隐藏页模型。**
  - 单文件页模型
  - 代码隐藏页模型

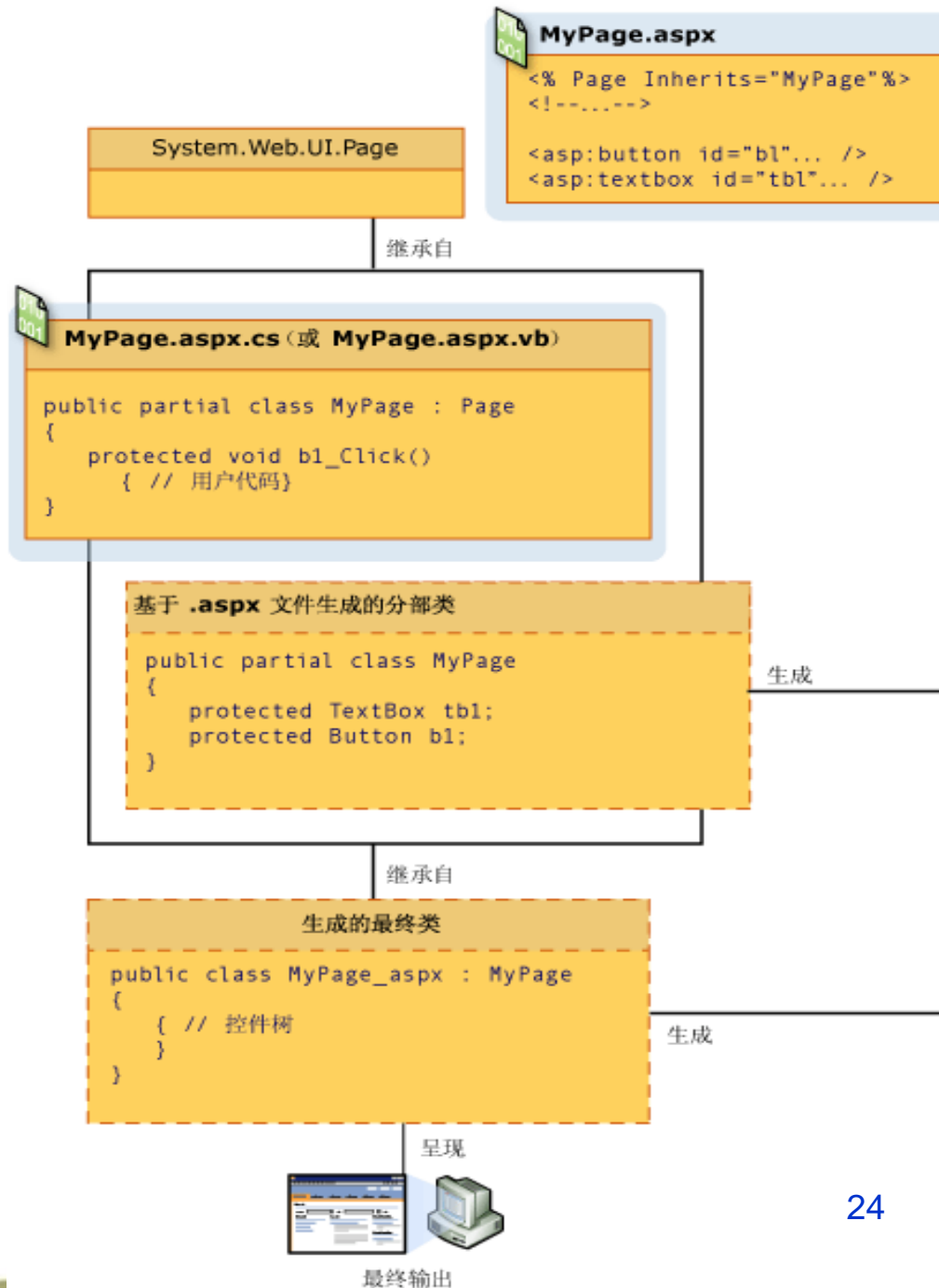
## 1. 单文件页模型

- 在单文件页模型中，页的标记及其编程代码位于同一个物理文件.aspx中。



## 2. 代码隐藏页模型

➤ 通过代码隐藏页模型，可以在一个文件(.aspx)中保存标记，并在另一个文件中(.aspx.cs)保留代码，这就使得页面显示部分和代码逻辑分离。





## 3.2.3 Page类的属性

- ❖ 内置对象
- ❖ **IsPostBack**属性
- ❖ **EnableViewState**属性
- ❖ **IsValid**属性

# 1. 内置对象

对象名	说 明
<b>Request</b>	提供对当前页请求的访问，其中包括请求标题、Cookie、客户端证书、查询字符串等，可以用它来读取浏览器已经发送的内容。
<b>Response</b>	提供对输出流的控制，如可以向浏览器输出信息、Cookie等
<b>Context</b>	提供对整个当前上下文（包括请求对象）的访问，可用于共享页之间的信息
<b>Server</b>	提供用于在页之间传输控件的实用方法，获取有关最新错误的信息，对HTML文本进行编码和解码，获取服务器信息等
<b>Application</b>	提供对所有会话的应用程序范围的方法和事件的访问，还提供对可用于存储信息的应用程序范围的缓存的访问
<b>Session</b>	为当前用户会话提供信息。还提供对可用于存储信息的会话范围的缓存的访问，以及控制如何管理会话的方法。
<b>Trace</b>	提供在HTTP页输出中显示系统和自定义跟踪诊断消息的方法
<b>User</b>	提供对发出页请求的用户身份访问，可以获得该用户的标识及其它信息

## 2. IsPostBack属性

- 该属性表示该页是否为响应客户端回发而加载，或者该页是否被**首次加载和访问**。
- 当IsPostBack为true，表示该请求是页面回发；当**IsPostBack为false**，表示该页是被首次加载和访问。  
如：

```
protected void Page_Load(object sender, EventArgs e){  
    if (!IsPostBack){  
        Response.Write("第一次访问");  
    }  
    else{  
        Response.Write("非第一次访问");  
    }  
}
```

### 3. **EnableViewState**属性

- 该属性表示当前页请求结束时该页是否保持其视图状态以及它包含的任何服务器控件的视图状态。该属性将在第7章详细介绍。

### 4. **IsValid**属性

- 该属性表示页面验证是否成功。在实际应用中，往往会验证页面提交的数据是否符合预期设定的格式要求等，如果所有都符合则**IsValid**值为**true**，反之为**false**。该属性的使用将在第4章的验证控件中详细介绍。

## 3.2.4 ASP.NET网页的生命周期与Page类的事件

### ❖ 常规页生命周期阶段

阶 段	说 明
页请求	页请求发生在页生命周期开始之前。用户请求页时，ASP.NET将确定是否需要分析和编译页（从而开始页的生命周期），或者是否可以在不运行页的情况下发送页的缓存版本以进行响应。
开始	在开始阶段，将设置页属性，如 <code>Request</code> 和 <code>Response</code> 。在此阶段，页还将确定请求是回发请求还是新请求，并设置 <code>IsPostBack</code> 属性。此外，在开始阶段，还将设置页的 <code>UICulture</code> 属性。
页初始化	页初始化期间，可以使用页中的控件，并将设置每个控件的 <code>UniqueID</code> 属性。此外，任何主题都将应用于页。如果当前请求是回发请求，则回发数据尚未加载，并且控件属性值尚未还原为视图状态中的值。

加载	加载期间，如果当前请求是回发请求，则将使用从视图状态和控件状态恢复的信息加载控件属性。
验证	在验证期间，将调用所有验证程序控件的 <code>Validate</code> 方法，此方法将设置各个验证程序控件和页的 <code>IsValid</code> 属性。
回发事件处理	如果请求是回发请求，则将调用所有事件处理程序。
呈现	在呈现之前，会针对该页和所有控件保存视图状态。在呈现阶段中，页会针对每个控件调用 <code>Render</code> 方法，它会提供一个文本编写器，用于将控件的输出写入页的 <code>Response</code> 属性的 <code>OutputStream</code> 中。
卸载	完全呈现页并已将页发送至客户端，准备丢弃该页时，将调用卸载。此时，将卸载页属性（如 <code>Response</code> 和 <code>Request</code> ）并执行清理。



## ❖ 生命周期事件

页 事 件	典 型 使 用
Page_PreInit	检查IsPostBack属性来确定是否是第1次处理该页；创建或重新创建动态控件；动态设置母版页；动态设置Theme属性；读取或设置配置文件属性值。注意：如果请求是回发请求，则控件的值尚未从视图状态还原。如果在此阶段设置控件属性，则其值可能会在下一事件中被覆盖。
Page_Init	在所有控件都已初始化且已应用所有外观设置后引发。使用该事件来读取或初始化控件属性。
Page_Load	读取和设置控件属性；建立数据库连接。

## 控件事件

使用这些事件来处理特定控件事件，如Button控件的Click事件或TextBox控件的TextChanged事件。注意：在回发请求中，如果页包含验证程序控件，在执行控件事件前，一般要检查Page和各个验证控件的IsValid属性，看页面验证是否通过。

## Page\_PreRender

使用该事件对页或其控件的内容进行最后更改。

## Page\_UnLoad

该事件首先针对每个控件发生，继而针对该页发生。在控件中，使用该事件对特定控件执行最后清理，如关闭控件特定数据库连接。对于页自身，使用该事件来执行最后清理工作，如：关闭打开的文件和数据库连接，或完成日志记录或其它请求特定任务。注意：在卸载阶段，页及其控件已被呈现，因此无法对响应流做进一步更改。如果尝试调用方法（如Response.Write方法），则该页将引发异常。



Part 3

# PAGE类的内置对象



## 3.3.1 Response对象

- ❖ **Response**对象主要是将**HTTP**响应数据发送到客户端。
- ❖ 该对象派生自**HttpResponse**类，是**Page**对象的成员，所以在程序中无须做任何的说明即可直接使用。
- ❖ 该对象的主要功能是**输出数据到客户端**。

# Response对象的常用属性

属 性	说 明
<b>BufferOutput</b>	获得或设置一个值，该值指示是否缓冲输出，并在完成处理整个响应之后将其发送
<b>Cache</b>	获得网页的缓存策略（过期时间、保密性等）
<b>Charset</b>	获取或设置输出流的HTTP字符集
<b>Cookies</b>	获得响应Cookie集合
<b>IsClientConnected</b>	获取一个值，通过该值指示客户端是否仍连接在服务器上。
<b>StatusCode</b>	获取或设置返回给客户端的输出的HTTP状态代码
<b>StatusDescription</b>	获取或设置返回给客户端的输出的HTTP状态字符串
<b>SuppressContent</b>	获取或设置一个值，该值指示是否将HTTP内容发送到客户端

# Response对象的常用方法

方 法	说 明
<b>AppendToLog</b>	将自定义日志信息添加到IIS的日志文件中
<b>ClearContent</b>	将缓冲区的内容清除
<b>ClearHeaders</b>	将缓冲区的所有页面标头清除
<b>Close</b>	关闭客户端的联机
<b>End</b>	将目前缓冲区中所有的内容发送到客户端，停止该页的执行，并引发EndRequest事件
<b>Flush</b>	将缓冲区中所有的数据送到客户端
<b>Redirect</b>	将客户端重定向到新的URL
<b>Write</b>	将信息写入HTTP响应输出流
<b>WriteFile</b>	将一个文件直接输出到客户端
<b>BinaryWrite</b>	将一个二进制的字符串写入HTTP输出流

# Response对象应用示例

## 1. 利用Write方法直接向客户端输出信息

例: `Response.Write("<H1>Response对象</H1>");`

## 2. 将文件内容输出到客户端

例: `Response.WriteFile("test1.txt");`

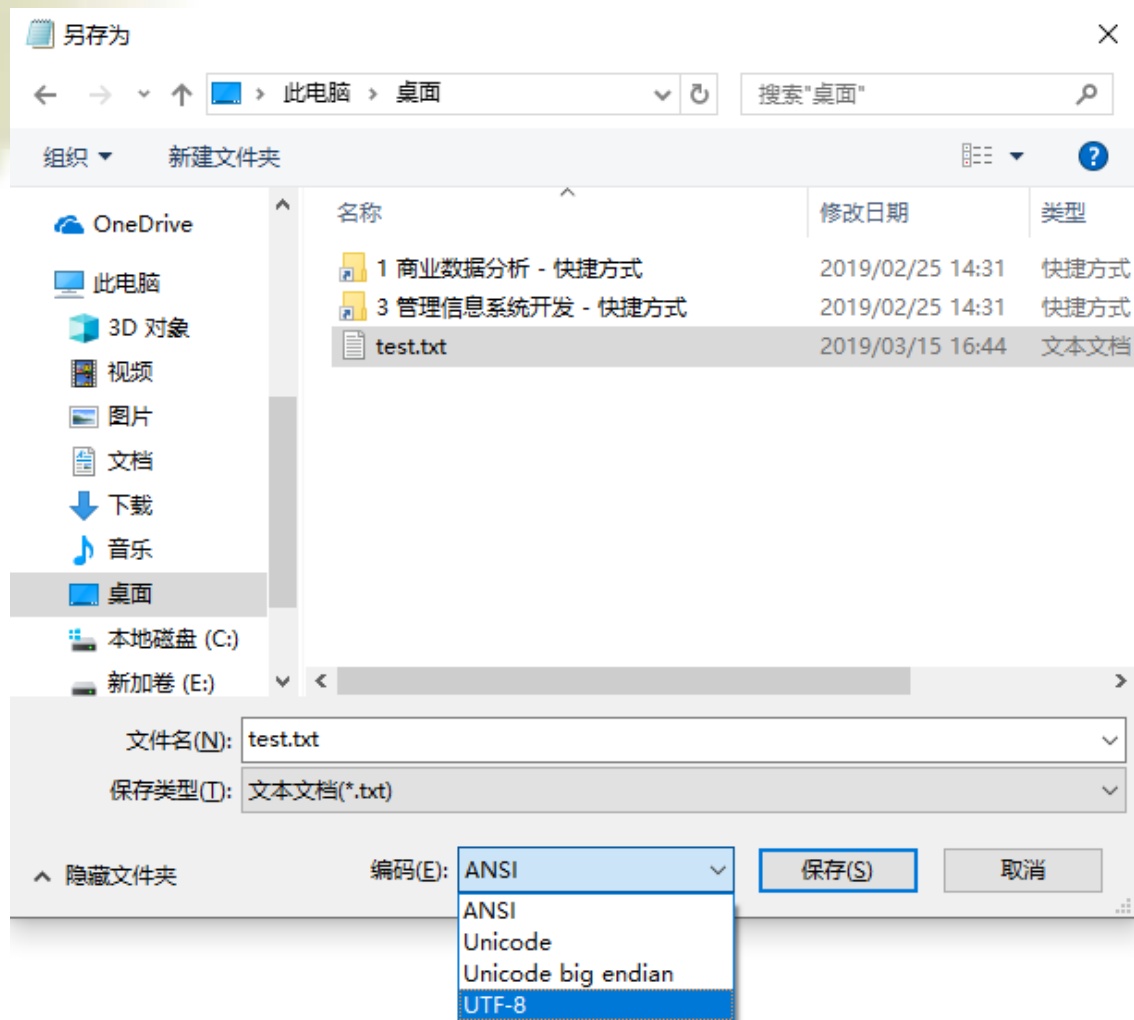
## 3. 实现网页重定向功能

例: `Response.Redirect("Page1.htm");`

## 4. 结束网页的执行

- **Response.End**方法是将当前所有缓冲的输出发送到客户端，停止该页的执行，并引发 **Application\_EndRequest**事件。

# Response.WriteFile("test.txt")中文显示乱码问题



## 3.3.2 Request对象

- ❖ **Request**对象主要提供对当前页请求的访问，其中包括请求标题、**Cookies**、客户端证书、查询字符串等。
- ❖ 该对象派生自**HttpRequest**类，是**Page**类的成员。
- ❖ 它的主要功能是从客户端浏览器取得数据，包括浏览器种类、用户输入表单中的数据、**Cookies**中的数据和客户端认证等。



# Request对象的常用属性

属 性	说 明
<b>ApplicationPath</b>	获取目前正在执行程序的服务器的虚拟根路径
<b>Browser</b>	获取有关正在请求的客户端的浏览器功能的信息
<b>Cookies</b>	获取客户端发送的Cookie集合
<b>FilePath</b>	获取当前请求的虚拟路径
<b>Files</b>	获取客户端上传的文件集合
<b>Form</b>	获取窗体变量集合
<b>Headers</b>	获取HTTP头集合
<b>HttpMethod</b>	获取客户端使用的HTTP数据传输方法
<b>Params</b>	获取QueryString、Form、ServerVariables和Cookies项的组合集合



<b>Path</b>	获取当前请求的虚拟路径
<b>PhysicalApplicationPath</b>	获取当前正在执行的服务器应用程序根目录的物理路径
<b>PhysicalPath</b>	获取当前请求网页在服务器端的物理路径
<b>QueryString</b>	获取附在网址后面的参数信息
<b>ServerVariables</b>	获取Web服务器变量的集合
<b>Url</b>	获取有关目前请求的URL信息
<b>UserAgent</b>	获取客户端浏览器的原始用户代理信息
<b>UserHostAddress</b>	获取远方客户端机器的主机IP地址
<b>UserHostName</b>	获取远方客户端机器的DNS名称
<b>UserLanguages</b>	获取客户端语言首选项的排序字符串数组

**Request**对象的常用方法有以下两个：

- ❖ **MapPath(virtualPath)**: 将参数virtualPath指定的虚拟路径转化为实际路径；
- ❖ **SaveAs(filename,includeHeaders)**: 将HTTP请求保存到磁盘，**filename**是保存的文件路径，**includeHeaders**指定是否保存HTTP标头。

# Request对象应用示例

## 1. 获取文件的路径信息

- Request对象的Url、UserHostAddress、PhysicalApplicationPath、CurrentExecutionFilePath和PhysicalPath属性能够分别获取当前请求的URL、远程客户端的IP主机地址、当前正在执行的服务器应用程序的根目录的物理文件系统路径、当前请求的虚拟路径及获取与请求的URL相对应的物理文件系统路径。

## 示例3-4 Request对象应用举例

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("客户端IP地址: ");
    Response.Write(Request.UserHostAddress + "<br>");
    Response.Write("当前程序根目录的实际路径: ");
    Response.Write(Request.PhysicalApplicationPath + "<br>");
    Response.Write("当前页的虚拟目录文件名称: ");
    Response.Write(Request.CurrentExecutionFilePath + "<br>");
    Response.Write("当前页的实际目录及文件名称: ");
    Response.Write(Request.PhysicalPath + "<br>");
    Response.Write("当前页面的Url: ");
    Response.Write(Request.Url);
}
```

# 实验提示

为什么不是127.0.0.1?

← → ↻ ⓘ localhost:49292/eg3-4.aspx

客户端IP地址: ::1

当前程序根目录的实际路径: F:\workspace\VisualStudio\projects\WebApplication1\WebApplication1\

当前页的虚拟目录文件名称: /eg3-4.aspx

当前页的实际目录及文件名称: F:\workspace\VisualStudio\projects\WebApplication1\WebApplication1\eg3-4.aspx

当前页面的Url: http://localhost:49292/eg3-4.aspx

> 此电脑 > 本地磁盘 (C:) > Windows > System32 > drivers > etc

名称	修改日期	类型
hosts	2019/03/21 14:47	文件

# localhost name resolution is handled within DNS itself.

# 127.0.0.1 localhost

# ::1 localhost

## 2.利用QueryString集合传递参数

例如:

<http://Localhost/MyPage/ShowPage.aspx?Id=2&Name=Zhangsan>

- 在服务器端，可以通过Request对象的QueryString集合来引用这些值，例如，引用上述两个变量的值，可以使用如下方法：

```
Id = Request.QueryString["Id"];
```

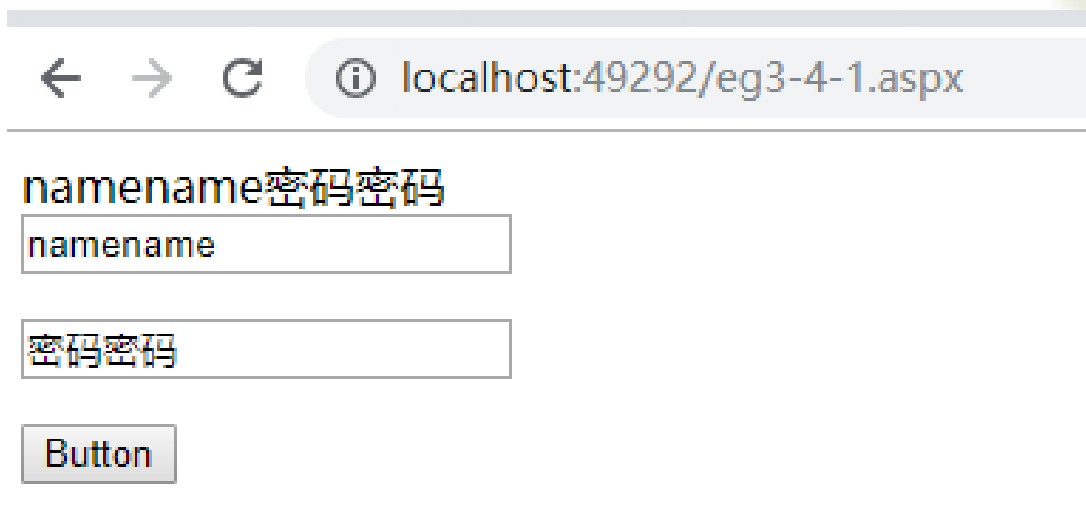
```
Name = Request.QueryString["Name"];
```

结果为Id="2"， Name=" Zhangsan "，接收到的数据类型为字符串型。

### 3. 利用Form集合接受表单数据

- 例如：**Request.Form["TxtName"]**，表示获取表单中名为**TxtName**控件的值。

```
protected void Submit_Click(object sender, EventArgs e)
{
    Response.Write(Request.Form["TxtName"]);
    Response.Write(Request.Form["TxtPwd"]);
}
```



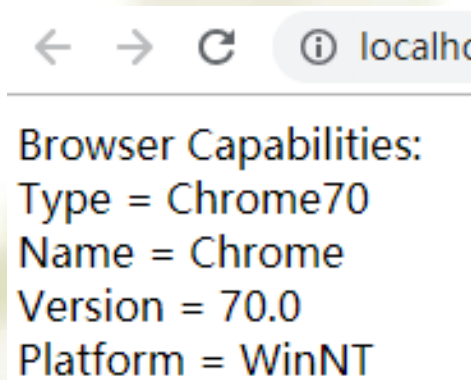
The screenshot shows a web browser window with the address bar displaying "localhost:49292/eg3-4-1.aspx". The page content includes a form with two input fields. The first field is labeled "name" and contains the text "name". The second field is labeled "password" and contains the text "password". Below the input fields is a button labeled "Button".



## 4. 利用Browser对象获取浏览器信息

- Request对象的Browser属性能够返回一个HttpBrowserCapabilities类型的集合对象。
- 该集合对象可以取得目前连接到Web服务器的浏览器的信息。例如可以利用这个对象的一个属性确认访问者所使用的操作系统。

```
protected void Page_Load(object sender, EventArgs e)
{
    HttpBrowserCapabilities bc = Request.Browser;
    Response.Write("Browser Capabilities:<br>");
    Response.Write("Type = " + bc.Type + "<br>");
    Response.Write("Name = " + bc.Browser + "<br>");
    Response.Write("Version = " + bc.Version + "<br>");
    Response.Write("Platform = " + bc.Platform + "<br>");
}
```



Browser Capabilities:  
Type = Chrome70  
Name = Chrome  
Version = 70.0  
Platform = WinNT

### 3.3.3 Server对象

**Server**对象有以下两个属性：

- ❖ **MachineName**: 获取服务器的计算机名称，为只读属性。
- ❖ **ScriptTimeout**: 获取或设置程序执行的最长时间，即程序必须在该段时间内执行完毕，否则将自动终止，时间以秒为单位。

# Server对象的常用方法

方 法	说 明
CreateObject	创建COM对象的一个服务器实例
Execute	执行对另一页的请求，执行完毕后仍继续执行原程序
HtmlDecode	将HTML编码的字符串按HTML语法进行解释
HtmlEncode	对字符串进行编码，使它不会被浏览器按HTML语法进行解释，按字符串原样显示
Transfer	终止当前页的执行，并开始执行新页
UrlDecode	对URL编码的字符串进行解码
UrlEncode	编码字符串，以便通过URL从Web服务器到客户端进行可靠的HTTP传输
UrlPathEncode	对URL字符串的路径部分进行URL编码，并返回已编码的字符串
MapPath	返回与Web服务器上的指定虚拟路径相对应的物理文件路径

# Sever对象应用示例

## 1. 用Execute方法执行对另一页的请求

- 用Execute(URL)执行另一个ASP.NET网页，执行完成后返回原来的网页继续执行。

```
public partial class eg3_7 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("<p>调用Execute方法之前");
        Server.Execute("TestPage.aspx");
        Response.Write("<p>调用Execute方法之后");
    }
}
```

```
public partial class TestPage : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("<p>这是一个测试页</p>");
    }
}
```

localhost:49292/eg3-7.aspx

调用Execute方法之前

这是一个测试页

调用Execute方法之后

# Sever对象应用示例

## 2. 用Transfer方法实现网页重定向

- **Transfer(URL):** 终止当前网页，执行新的网页URL，即实现重定向。

```
public partial class eg3_8 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("<p>调用Transfer方法之前");
        Server.Transfer("TestPage.aspx");
        //Response.Write("<p>调用Transfer方法之后");
    }
}
```

```
public partial class eg3_8 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        //Response.Write("<p>调用Transfer方法之前");
        //Server.Transfer("TestPage.aspx");
        //Response.Write("<p>调用Transfer方法之后");
        Response.Write("<p>调用Redirect方法之前");
        Response.Redirect("TestPage.aspx");
        Response.Write("<p>调用Redirect方法之后");
    }
}
```

localhost:49292/eg3-8.aspx

调用Transfer方法之前

这是一个测试页

localhost:49292/TestPage.aspx

这是一个测试页



# 测一测



- ❖ 一家在线测试中心TestKing公司创建一个ASP.NET应用程序。在用户结束测试后，这个应用程序需要在用户不知道的情况下，提交答案给ProcessTestAnswers.aspx页。这ProcessTestAnswers.aspx页面处理这答案，但不提供任何显示消息给用户。当处理完成时，PassFailStatus.aspx页面显示结果给用户。在PassFailStatus.aspx页面中加（ ）代码，来执行ProcessTestAnswers.aspx页面中的功能。

- A. `Server.Execute("ProcessTestAnswers.aspx")`
- B. `Response.Redirect("ProcessTestAnswers.aspx")`
- C. `Response.WriteFile("ProcessTestAnswers.aspx")`
- D. `Server.Transfer("ProcessTestAnswers.aspx", True)`

### 3. 将虚拟路径转化为实际路径

- **Server.MapPath(Web服务器上的虚拟路径)**返回的是与**Web服务器**上的指定虚拟路径相对应的物理文件路径。

```
public partial class eg3_9 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("网页实际路径为: " + Server.MapPath("eg3-9.aspx") + "</br>");
        Response.Write("根目录为: " + Server.MapPath("~/"));
    }
}
```

← → ↻ ⓘ localhost:49292/eg3-9.aspx

网页实际路径为: F:\workspace\VisualStudio\projects\WebApplication1\WebApplication1\eg3-9.aspx  
根目录为: F:\workspace\VisualStudio\projects\WebApplication1\WebApplication1\



## 3.6 小结

- ❖ **ASP.NET**应用程序生命周期;
- ❖ **ASP.NET**网页;
- ❖ **Page**类的内置对象;
- ❖ **Web**应用的异常处理机制。

# 作业——省平台

## 作业1

题目： 15 总分： 100

### 1.单选题(共15题)

1. CSS样式表不可能实现()功能

6分 一般

- A. 将内容和外观分离
- B. 一个 CSS 文件控制多个网页
- C. 控制图片的精确位置
- D. 兼容所有的浏览器

解析:

2. 下列标记不属于HTML文档的基本结构的是：（）

6分 一般

- A. <html>
- B. <body>
- C. <head>
- D. <form>

解析:



本章结束!