一、Response对象   
Response对象是HttpResponse类的一个实例。该类主要是封装来自ASP.NET操作的HTTP响应信息。   
1、Response对象的属性 

|  |  |  |
| --- | --- | --- |
| 属   性 | 说   明 | 属 性 值 |
| BufferOutput | 获取或设置一个值，该值指示是否缓冲输出，并在完成处理整个页之后将其发送 | 如果缓冲了到客户端的输出，则为true；否则为false。默认为true |

续表

|  |  |  |
| --- | --- | --- |
| 属   性 | 说   明 | 属 性 值 |
| Cache | 获取Web页的缓存策略（过期时间、保密性、变化子句） | 包含有关当前响应的缓存策略信息的HttpCachePolicy对象 |
| Charset | 获取或设置输出流的HTTP字符集 | 输出流的HTTP字符集 |
| IsClientConnected | 获取一个值，通过该值指示客户端是否仍连接在服务器上 | 如果客户端当前仍在连接，则为true；否则为false |

实例：使用缓冲区   
由于Response对象的BufferOutput属性默认为True，所以要输出到客户端的数据都暂时存储在缓冲区内，等到所有的事件程序，以及所有的页面对象全部解译完毕后，才将所有在缓冲区中的数据送到客户端的浏览器。下面的例子将演示缓冲区是如何工作的。   
<%   
Response.Write("缓存已清除" + "<Br>");   
%>   
<Script Language="C#" Runat="Server">   
void Page\_Load(Object sender, EventArgs e)   
{   
Response.Write("缓存清除前" + "<Br>");   
Response.Clear();   
}   
</Script>   
上述程序代码实例首先在“Page\_Load”事件中送出“缓存清除前”这一行，此时的数据存在缓冲区中。接着使用Response对象的Clear方法将缓冲区的数据清除，所以刚刚送出的字符串已经被清除。然后IIS开始读取HTML组件的部分，并将结果送至客户端的浏览器。由执行结果只出现“缓存已清除”可知，使用Clear方法之前的数据并没有出现在浏览器上，所以程序开始时是存在缓冲区内的。如果在相同的程序中加入“Response.BufferOutput=false”：   
<%   
Response.Write("清除之后的数据<Br>");   
%>   
<Script Language="C#" Runat="Server">   
void Page\_Load(Object sender, EventArgs e)   
{   
Response.BufferOutput=false;   
Response.Write("清除缓冲区之前的数据" + "<Br>");   
Response.Clear();   
}   
</Script>   
可以发现，执行的结果并没有因为使用Clear方法而将缓冲区的数据清除，这表明数据是直接输出而没有存放在缓冲区内。   
2、Response对象的方法   
Response对象可以输出信息到客户端，包括直接发送信息给浏览器、重定向浏览器到另一个URL或设置cookie的值。下表列举了几个常用的方法。   
Response对象的方法

|  |  |
| --- | --- |
| 方   法 | 说   明 |
| Write | 将指定的字符串或表达式的结果写到当前的HTTP输出 |
| End | 停止页面的执行并得到相应结果 |
| Clear | 用来在不将缓存中的内容输出的前提下，清空当前页的缓存，仅当使用了缓存输出时，才可以利用Clear方法 |
| Flush | 将缓存中的内容立即显示出来。该方法有一点和Clear方法一样，它在脚本前面没有将Buffer属性设置为True时会出错。和End方法不同的是，该方法调用后，该页面可继续执行 |
| Redirect | 使浏览器立即重定向到程序指定的URL |

ASP.NET中引用对象方法的语法是“对象名.方法名”。“方法”就是嵌入到对象定义中的程序代码，它定义对象怎样去处理信息。使用嵌入的方法，对象便知道如何去执行任务，而不用提供额外的指令。以下将通过几个小例子来讲解Response对象的常用方法。   
Response对象的方法

|  |  |
| --- | --- |
| 方   法 | 说   明 |
| Write | 将指定的字符串或表达式的结果写到当前的HTTP输出 |
| End | 停止页面的执行并得到相应结果 |
| Clear | 用来在不将缓存中的内容输出的前提下，清空当前页的缓存，仅当使用了缓存输出时，才可以利用Clear方法 |
| Flush | 将缓存中的内容立即显示出来。该方法有一点和Clear方法一样，它在脚本前面没有将Buffer属性设置为True时会出错。和End方法不同的是，该方法调用后，该页面可继续执行 |
| Redirect | 使浏览器立即重定向到程序指定的URL |

ASP.NET中引用对象方法的语法是“对象名.方法名”。“方法”就是嵌入到对象定义中的程序代码，它定义对象怎样去处理信息。使用嵌入的方法，对象便知道如何去执行任务，而不用提供额外的指令。以下将通过几个小例子来讲解Response对象的常用方法。   
实例：使用Response.write，向客户端发送信息   
for(int i=1;i<=500;i++)   
{   
Response.Write("i= "+i+"<BR>");   
}   
本例使用“write”方法，向屏幕输出500个值。   
实例：使用Response.End方法调试程序   
End方法可以停止当前页面的执行，基于这个原因，可以结合Response.write方法输出当前页面上的某个变量、数组值。   
<form id="Form1" method="post" runat="server">   
输入一个数值：<asp:TextBox id="txtVar" runat="server"></asp:TextBox>   
<asp:Button id="btnSubmit" runat="server" Text="计算该值的平方值" onclick="btnSubmit\_Click"></asp:Button>   
</form>   
<Script Language="C#" Runat="Server">   
void btnSubmit\_Click(Object sender, EventArgs e)   
{   
int N = int.Parse(Request.Form["txtVar"].ToString());   
Response.Write("N=" + N + "<br>");   
Response.Write("该值的平方值是：" + N\*N);   
}   
</Script>   
运行上面的代码，结果如图所示。

使用Response.End调试程序

输入一个值“6”，然后单击“计算该值的平方值”按钮，屏幕将显示如下结果：   
N=6   
该值的平方值是：36   
在代码中加上“Response.End()”，代码如下：   
<Script Language="C#" Runat="Server">   
void btnSubmit\_Click(Object sender, EventArgs e)   
{   
int N = int.Parse(Request.Form["txtVar"].ToString());   
Response.Write("N=" + N + "<br>");   
Response.End();   
Response.Write("该值的平方值是：" + N\*N);   
}   
</Script>   
这时再运行代码，将只会显示：   
N=6   
实验证明，“Response.End()”方法停止了当前页面的执行。这仅仅是一个小例子，读者可以依此类推，在程序中使用End方法进行调试。不过千万要记住调试完代码后，不要忘记把调试用的“Response.End()”删掉。   
实例：使用Redirect方法进行页面重定向   
在网页编程中，经常会遇到在程序执行到某个位置进行页面调转的情况。Response.Redirect方法可以满足这种需求，例如代码：   
Response.Redirect("**错误！超链接引用无效。**);   
执行该代码，页面将跳转到网易163的主页。   
二、 Request对象   
Request对象是HttpRequest类的一个实例。它能够读取客户端在Web请求期间发送的HTTP值。   
**Request对象的属性**   
Request对象的属性如表14-3所示。   
表14-3 Request对象的属性

|  |  |  |
| --- | --- | --- |
| 属   性 | 说   明 | 属 性 值 |
| QueryString | 获取 HTTP 查询字符串变量集合 | NameValueCollection对象 |
| Path | 获取当前请求的虚拟路径 | 当前请求的虚拟路径 |
| UserHostAddress | 获取远程客户端的 IP 主机地址 | 远程客户端的IP地址 |
| Browser | 获取有关正在请求的客户端的浏览器功能的信息 | HttpBrowserCapabilities对象 |

实例14-5：获取QueryString值   
程序中，经常可以使用QueryString来获得从上一个页面传递来的字符串参数。例如，在页面1中创建一个连接，指向页面2，并用QueryString来查询两个变量：   
<a href="Page2.aspx?ID=6&Name=Wang">查看</a>   
在页面2中接收到从页面1中传过来的两个变量：   
<Script Language="C#" Runat="Server">   
void Page\_Load(object sender, System.EventArgs e)   
{   
Response.Write("变量ID的值：" + Request.QueryString["ID"] +"<br>");   
Response.Write("变量Name的值：" + Request.QueryString["Name"]);   
}   
</Script>   
运行上面代码结果如下：   
变量ID的值：6   
变量Name的值：Wang   
上面的例子可以成功地得到QueryString的值。   
用类似方法，可以获取Form，Cookies，SeverVaiables的值。调用方法都是：   
Request.Collectlon[”VariabLe”]   
Collectlon包括QueryString，ForM，Cookies，SeverVaiables四种集合，VariabLe为要查询的关键字。不过，这里的Collectlon是可以省略的，也就是说，Request["Variable"]与Request.Collection["Variable"]这两种写法都是允许的。如果省略了Collection，那么Request对象会依照QueryString，Form，Cookies，SeverVaiables的顺序查找，直至发现Variable所指的关键字并返回其值，如果没有发现其值，方法则返回空值（Null）。   
不过，为了优化程序的执行效率，建议最好还是使用Collection，因为过多地搜索就会降低程序的执行效率。   
**2．Request对象的方法**   
Request对象的方法如表14-4所示。   
表14-4 Request对象的方法

|  |  |
| --- | --- |
| 方   法 | 说   明 |
| BinaryRead | 执行对当前输入流进行指定字节数的二进制读取 |
| MapPath | 为当前请求将请求的 URL 中的虚拟路径映射到服务器上的物理路径 |

实例14-6：获取文件的物理路径   
Request.MapPath("FileName")；   
可以通过这条语句来得到某个文件的实际物理位置，这个方法常常用在需要使用实际路径的地方。   
**三、 Application对象**   
Application对象是HttpApplicationState类的一个实例。   
HttpApplicationState类的单个实例，将在客户端第一次从某个特定的ASP.NET应用程序虚拟目录中请求任何URL资源时创建。对于Web服务器上的每个ASP.NET应用程序，都要创建一个单独的实例。然后通过内部Application对象公开对每个实例的引用。Application对象有如下特点：   
— 数据可以在Application对象内部共享，因此一个Application对象可以覆盖多个用户。   
— 一个Application对象包含事件，可以触发某些Applicatin对象脚本。   
— 个别Application对象可以用Internet Service Manager来设置而获得不同属性。   
— 单独的Application对象可以隔离出来在它们自己的内存中运行，这就是说，如果一个人的Application遭到破坏，就不会影响其他人。   
— 可以停止一个Application对象（将其所有组件从内存中驱除）而不会影响到其他应用程序。   
一个网站可以有不止一个Application对象。典型情况下，可以针对个别任务的一些文件创建个别的Application对象。例如，可以建立一个Application对象来适用于全部公用用户，而再创建另外一个只适用于网络管理员的Application对象。   
Application对象使给定应用程序的所有用户之间共享信息，并且在服务器运行期间持久地保存数据。因为多个用户可以共享一个Application对象，所以必须要有Lock和Unlock方法，以确保多个用户无法同时改变某一属性。Application对象成员的生命周期止于关闭IIS或使用Clear方法清除。   
**1．Application对象的属性**   
Application对象的属性如表14-5所示。   
表14-5 Application对象的属性

|  |  |  |
| --- | --- | --- |
| 属   性 | 说   明 | 属 性 值 |
| AllKeys | 获取HttpApplicationState集合中的访问键 | HttpApplicationState对象名的字符串数组 |
| Count | 获取HttpApplicationState集合中的对象数 | 集合中的Item对象数。默认为0 |

**2．Application对象的方法**   
Application对象的方法如表14-6所示。   
表14-6 Application对象的方法

|  |  |
| --- | --- |
| 方   法 | 说   明 |
| Add | 新增一个新的Application对象变量 |
| Clear | 清除全部的Application对象变量 |
| Get | 使用索引关键字或变数名称得到变量值 |
| GetKey | 使用索引关键字來获取变量名称 |
| Lock | 锁定全部的Application变量 |
| Remove | 使用变量名称删除一个Application对象 |
| RemoveAll | 删除全部的Application对象变量 |
| Set | 使用变量名更新一个Application对象变量的內容 |
| UnLock | 解除锁定的Application变量 |

使用Application对象的语法如下所示：   
Application("变量名")="变量值"   
实例14-7：设置、获取Application对象的内容   
代码如下：   
<script language="C#" runat="server">   
void Page\_Load(object sender, System.EventArgs e)   
{   
Application.Add("App1","Value1");   
Application.Add("App2","Value2");   
Application.Add("App3","Value3");   
  
int N;   
for(N=0;N<Application.Count;N++)   
{   
  Response.Write("变量名："+ Application.GetKey(N));   
  Response.Write("变量值："+ Application.Get(N) +"<br>");   
}   
Application.Clear();   
}   
</script>   
在本例中，首先通过Add方法添加三个Application对象，并赋以初值，接着通过Count属性得到Application对象的数量，然后通过循环操作GetKey方法和Get方法分别得到新增对象的“索引”和“索引”所对应的“值”。   
执行上面代码，得到如下结果：   
变量名：App1变量值：Value1   
变量名：App2变量值：Value2   
变量名：App3变量值：Value3   
实例14-8：Application对象的加锁与解锁   
Lock方法可以阻止其他客户修改存储在Application对象中的变量，以确保在同一时刻仅有一个客户可修改和存取Application变量。如果用户没有明确调用Unlock方法，则服务器将在页面文件结束或超时即可解除对Application对象的锁定。   
Unlock方法可以使其他客户端在使用Lock方法锁住Application对象后，修改存储在该对象中的变量。如果未显式地调用该方法，Web服务器将在页面文件结束或超时后解锁Application对象。   
使用方法如下：   
Application.Lock();   
Application["变量名"]="变量值";   
Application.UnLock();   
**四、 Session对象**   
Session对象是HttpSessionState的一个实例。该类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。   
Session的发明是填补HTTP协议的局限，HTTP协议工作过程是，用户发出请求，服务器端做出响应，这种用户端和服务器端之间的联系都是离散的，非连续的。在HTTP协议中没有什么能够允许服务器端来跟踪用户请求的。在服务器端完成响应用户的请求后，服务器端不能持续与该浏览器保持连接。从网站的观点上看，每一个新的请求都是单独存在的，因此，当用户在多个主页间转换时，就根本无法知道他的身份。   
可以使用Session对象存储特定用户会话所需的信息。这样，当用户在应用程序的Web页之间跳转时，存储在Session对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。   
当用户请求来自应用程序的Web页时，如果该用户还没有会话，则Web服务器将自动创建一个Session对象。当会话过期或被放弃后，服务器将中止该会话。   
当用户第一次请求给定的应用程序中的aspx文件时，ASP.NET将生成一个SessionID。SessionID是由一个复杂算法生成的号码，它唯一标识每个用户会话。在新会话开始时，服务器将Session ID作为一个cookie存储在用户的Web浏览器中。   
在将SessionID cookie存储于用户的浏览器之后，即使用户请求了另一个.aspx文件，或请求了运行在另一个应用程序中的.aspx文件，ASP.NET仍会重用该cookie跟踪会话。与此相似，如果用户故意放弃会话或让会话超时，然后再请求另一个.aspx文件，那么ASP.NET将以同一个cookie开始新的会话。只有当服务器管理员重新启动服务器，或用户重新启动Web浏览器时，此时存储在内存中的SessionID设置才被清除，用户将会获得新的SessionID cookie。   
通过重用SessionID cookie，Web应用程序将发送给用户浏览器的cookie数量降为最低。另外，如果用户决定该Web应用程序不需要会话管理，就可以不让Web应用程序跟踪会话和向用户发送SessionID。   
Session对象最常见的一个用法就是存储用户的首选项。例如，如果用户指明不喜欢查看图形，另外其还经常被用在鉴别客户身份的程序中。要注意的是，会话状态仅在支持cookie的浏览器中保留，如果客户关闭了cookies选项，Session也就不能发挥作用了。   
ASP.NET的Sessions非常好用，能够利用Session对象来对Session全面控制，如果需要在一个用户的Session中存储信息，只需要简单地直接调用Session对象就可以了，下面就是个例子：   
Session("Myname")=Response.form("Username");   
Session("Mycompany")=Response．form("Usercompany");   
应注意的是，Session对象是与特定用户相联系的。针对某一个用户赋值的Session对象是和其他用户的Session对象完全独立的，不会相互影响。换句话说，这里面针对每一个用户保存的信息是每一个用户自己独享的，不会产生共享情况。   
很明显，对于不同的用户，Session对象的Myname变量和Mycompany变量各自是不同的，当每个人在网站的不同主页间浏览时，这种针对个人的变量会一直保留，这样作为身份认证是十分有效的。   
**1．Session对象的属性**   
Session对象的属性如表14-7所示。   
表14-7 Session对象的属性

|  |  |  |
| --- | --- | --- |
| 属   性 | 说   明 | 属 性 值 |
| Count | 获取会话状态集合中Session对象的个数 | Session对象的个数 |
| TimeOut | 获取并设置在会话状态提供程序终止会话之前各请求之间所允许的超时期限 | 超时期限（以分钟为单位） |
| SessionID | 获取用于标识会话的唯一会话ID | 会话ID |

实例14-9：获取Session对象的个数   
Count属性可以帮助统计正在使用的Session对象的个数，语句非常简单，示例如下：   
Response.Write(Session.Count);   
实例14-10：设置Session对象的生存期   
每一个客户端连接服务器后，服务器端都要建立一个独立的Session，并且需要分配额外的资源来管理这个Session，但如果客户端因某些原因，例如，去忙其他的工作，停止了任何操作，但没有关闭浏览器，那么这种情况下，服务器端依然会消耗一定的资源来管理Session，这就造成了对服务器资源的浪费，降低了服务器的效率。所以，可以通过设置Session生存期，以减少这种对服务器资源的浪费。   
要更改Session的有效期限，只要设定TimeOut属性即可；TimeOut属性的默认值是20分钟。   
<Html>   
<Form Runat="Server" ID="Form1">   
<Asp:Button Id="Button1" Text="演示" OnClick="Button1\_Click" Runat="Server" />   
目前时间：<Asp:Label Id="Label1" Runat="Server" />   
<P>   
第一个Session的值：<Asp:Label Id="Label2" Runat="Server" /><Br>   
第二个Session的值：<Asp:Label Id="Label3" Runat="Server" /><Br>   
</Form>   
<Script Language="c#" Runat="Server">   
void Page\_Load(object sender, System.EventArgs e)   
{   
if(!Page.IsPostBack)   
{   
  Session["Session1"]="Value1";   
  Session["Session2"]="Value2";   
  Session.Timeout=1;   
  DateTime now=DateTime.Now;   
  string format="HH:mm:ss";   
  Label1.Text=now.ToString(format);   
  Label2.Text=Session["Session1"].ToString();   
  Label3.Text=Session["Session2"].ToString();   
}   
}   
void Button1\_Click(object sender, System.EventArgs e)   
{   
DateTime now=DateTime.Now;   
string format="HH:mm:ss";   
Label1.Text=now.ToString(format);   
Label2.Text=Session["Session1"].ToString();   
Label3.Text=Session["Session2"].ToString();   
}   
</Script>   
</Html>   
在本例中，通过Timeout属性设置了Session的生存期为1分钟。运行上面代码，显示结果如图14-2所示。

图14-2 设置Session对象的生存期

一分钟后，单击“演示”按钮，页面会出现错误，提示   
Label2.Text=Session["Session1"].ToString()；     
Label3.Text=Session["Session2"].ToString();   
这两句代码错误，为什么会这样呢？原因就在于，Session的生存期限超过了一分钟，已经无法获得Session["Session1"]和Session["Session2"]的值。   
**2．Session对象的方法**   
Session对象的方法如表14-8所示：   
表14-8 Session对象的方法

|  |  |
| --- | --- |
| 方   法 | 说   明 |
| Add | 新增一个Session对象 |
| Clear | 清除会话状态中的所有值 |
| Remove | 删除会话状态集合中的项 |
| RemoveAll | 清除所有会话状态值 |

实例14-11：通过Add方法设置Session对象   
通过Add方法可以设置Session对象的值，语法如下：   
Session.Add("变量名",变量值);   
在具体应用中，可以这样使用：   
int userId = 1;   
string userName = "test";   
string userPwd = "sdgtrertkl";   
Session.Add("userId",userId);   
Session.Add("userName", userName);   
Session.Add("userPwd", userPwd);   
在上面例子中创建了userId，userName，userPwd三个Session来存储用户的登录信息。程序随时都可以通过这三个Session对象来查看用户的连接状态，这是实际的项目中最常见的Session应用。   
需要注意的是，也可以不使用Add方法来设置Session对象，语法如下：   
Session["变量名"]=变量值;   
这样，上面的例子就可以改为：   
Session["userId"] = userId;   
Session["userName"] = userName;   
Session["userPwd"] = userPwd;   
需要指出的是，以上两种语句在作用上是相同的，读者可以根据自己的习惯来使用。   
**五、 Server对象**   
Server对象是HttpServerUtility的一个实例。该对象提供对服务器上的方法和属性的访问。   
**1．Server对象的属性**   
Server对象的属性如表14-9所示：   
表14-9 Server对象的属性

|  |  |  |
| --- | --- | --- |
| 属   性 | 说   明 | 属 性 值 |
| MachineName | 获取服务器的计算机名称 | 本地计算机的名称 |
| ScriptTimeout | 获取和设置请求超时 | 请求的超时设置（以秒计） |

实例14-12：返回服务器计算机名称   
通过Server对象的MachineName属性来获取服务器计算机的名称，示例如下：   
<Script Language="c#" Runat="Server">   
void Page\_Load(object sender, System.EventArgs e)   
{   
String ThisMachine;   
ThisMachine = Server.MachineName;   
Response.Write(ThisMachine);   
}   
</Script>   
实例14-13：设置客户端请求的超时期限   
用法如下：   
Server.ScriptTimeout = 60;   
本例中，将客户端请求超时期限设置为60秒，如果60秒内没有任何操作，服务器将断开与客户端的连接。   
**2．Server对象的方法**   
Server对象的方法如表14-10所示：   
表14-10 Server对象的方法

|  |  |
| --- | --- |
| 方   法 | 说   明 |
| CreateObject | 创建COM 对象的一个服务器实例 |
| CreateObjectFromClsid | 创建COM 对象的服务器实例，该对象由对象的类标识符（CLSID）标识 |
| Execute | 使用另一页执行当前请求 |
| Transfer | 终止当前页的执行，并为当前请求开始执行新页 |
| HtmlDecode | 对已被编码以消除无效HTML字符的字符串进行解码 |
| HtmlEncode | 对要在浏览器中显示的字符串进行编码 |
| 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 |
| UrlDecode | 对字符串进行解码，该字符串为了进行HTTP传输而进行编码并在URL中发送到服务器 |
| 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 |

实例14-14：利用HtmlEncode和HtmlDecode方法对网页内容编码   
当想在网页上显示HTML标签时，若在网页中直接输出则会被浏览器解译为HTML的内容，所以要通过Server对象的HtmlEncode方法将它编码再输出；若要将编码后的结果译码回原本的内容，则使用HtmlDecode方法。下列程序代码范例使用HtmlEncode方法将“<B>HTML内容</B>”编码后输出至浏览器，再利用HtmlDecode方法将把编码后的结果译码还原：   
<html>   
<Script Language="c#" Runat="Server">   
void Page\_Load(object sender, System.EventArgs e)   
{   
String strHtmlContent；   
strHtmlContent=Server.HtmlEncode("<B>HTML 内容</B>");   
Response.Write(strHtmlContent);   
Response.Write("<P>");   
strHtmlContent=Server.HtmlDecode(strHtmlContent)；   
Response.Write(strHtmlContent);   
}   
</Script>   
</html>   
运行上述示例输出结果可以发现，编码后的HTML标注变成了“<B>HTML内容</B>”，这是因为“<B>”变成了“<B>”，“</B>”变成了“</B>”，所以才能在页面中显示HTML标注。   
实例14-15：使用URLEncode方法对URL进行编码   
就像HTMLEncode方法使客户可以将字符串翻译成可接受的HTML格式一样，Server对象的URLEncode方法可以根据URL规则对字符串进行正确编码。当字符串数据以URL的形式传递到服务器时，在字符串中不允许出现空格，也不允许出现特殊字符。为此，如果希望在发送字符串之前进行URL编码，则可以使用Server. URLEncode方法。   
该函数已被重载，语法如下：   
对字符串进行URL编码，并返回已编码的字符串。   
public string UrlEncode(string);   
URL对字符串进行编码，并将结果输出发送到TextWriter输出流。   
public void UrlEncode(string, TextWriter);   
例如下面例子：   
<%Response.Write(Server.URLEncode("**错误！超链接引用无效。**))%>   
产生如下输出：   
http %3A%2F%2Fwww%2Emicrosoft%2Ecom   
利用QueryString在不同主页间传递信息时，如果信息带有空格或特殊字符，那么必须进行Encode操作，因为如果不这样做，很可能使得接受信息的那边接受到一些所不期望的奇怪字符串。注意，不要对QueryString的名称及等号进行Encode操作，只需要将其值进行Encode操作就可以了。   
进行了Encode操作后，效果如下：   
Message=This+Query+String+has+been+URL+ENCODED%2E   
用户并不需要考虑对上面的字符串再进行解码，会自动进行这样的处理。例如，假设有这样的脚本：   
Request.QueryString("message");   
这时，显示结果为：   
This Query String has been URL encoded   
实例14-16：建立虚拟路径与服务器物理目录间映射   
使用MapPath方法可以将指定的相对或虚拟路径映射到服务器上相应的物理目录上。   
语法：   
public string MapPath(string path);   
参数Path表示指定要映射物理目录的相对或虚拟路径。若Path以一个正斜杠（/）或反斜杠（\）开始，则MapPath方法返回路径时将Path视为完整的虚拟路径。若Path不是以斜杠开始，则MapPath方法返回同页面文件中已有的路径相对的路径。这里需要注意的是，MapPath方法不检查返回的路径是否正确或在服务器上是否存在。   
对于下列示例，文件data.txt和包含下列脚本的test.aspx文件都位于目录C:\Inetpub\ wwwroot\aspx下。C:\Inetpub\wwwroot目录被设置为服务器的宿主目录。下列示例使用服务器变量PATH\_INFO映射当前文件的物理路径。以下脚本：   
Server.mappth(Request. ServerVariables("PATH\_INFO"));   
输出：   
c：\inetpub\wwwroot\asp\test.aspx   
由于下列示例中的路径参数不是以斜杠字符开始的，所以它们被相对映射到当前目录，此处是目录C:\Inetpub\wwwroot\aspx。以下脚本：   
Server.mappath("data.txt");   
Server.mappath("aspx/data.txt");   
输出：   
c：\inetpub\wwwroot\aspx\data.txt   
c：\inetpub\wwwroot\aspx\aspx\data.txt   
**六、 Cookie对象**   
什么是Cookie呢？Cookie 是一小段文本信息，伴随着用户请求和页面在Web服务器和浏览器之间传递。用户每次访问站点时，Web应用程序都可以读取Cookie包含的信息。   
Cookie跟Session、Application类似，也是用来保存相关信息，但Cookie和其他对象的最大不同是，Cookie将信息保存在客户端，而Session和Application是保存在服务器端。也就是说，无论何时用户连接到服务器，Web站点都可以访问cookie信息。这样，既方便用户的使用，也方便了网站对用户的管理。   
ASP.NET包含两个内部Cookie集合。通过HttpRequest的Cookies集合访问的集合包含通过Cookie标头从客户端传送到服务器的Cookie。通过HttpResponse的Cookies集合访问的集合包含一些新Cookie，这些Cookie在服务器上创建并以Set-Cookie标头的形式传输到客户端。   
Cookie不是Page类的子类，所以在使用方法上跟Seesion和Application不同。   
使用Cookie的优点：   
— 可配置到期规则。Cookie 可以在浏览器会话结束时到期，或者可以在客户端计算机上无限期存在，这取决于客户端的到期规则。   
— 不需要任何服务器资源。Cookie存储在客户端并在发送后由服务器读取。   
— 简单性。Cookie是一种基于文本的轻量结构，包含简单的键值对。   
— 数据持久性。虽然客户端计算机上Cookie的持续时间取决于客户端上的Cookie过期处理和用户干预，Cookie通常是客户端上持续时间最长的数据保留形式。   
使用Cookie的缺点：   
— 大小受到限制。大多数浏览器对Cookie的大小有4096字节的限制，尽管在当今新的浏览器和客户端设备版本中，支持8192字节的Cookie大小已越发常见。   
— 用户配置为禁用。有些用户禁用了浏览器或客户端设备接收Cookie的能力，因此限制了这一功能。   
— 潜在的安全风险。Cookie可能会被篡改。用户可能会操纵其计算机上的Cookie，这意味着会对安全性造成潜在风险，或者导致依赖于Cookie的应用程序失败。另外，虽然Cookie只能将它们发送到客户端的域访问，历史上黑客已经发现从用户计算机上的其他域访问Cookie的方法。可以手动加密和解密Cookie，但这需要额外的编码，并且因为加密和解密需要耗费一定的时间而影响应用程序的性能。   
**1．Cookie对象的属性**   
Cookie对象的属性如表14-11所示：   
表14-11 Cookie对象的属性

|  |  |  |
| --- | --- | --- |
| 属   性 | 说   明 | 属 性 值 |
| Name | 获取或设置Cookie的名称 | Cookie的名称 |
| Value | 获取或设置Cookie的Value | Cookie的Value |
| Expires | 获取或设置Cookie的过期日期和时间 | 作为DateTime实例的Cookie过期日期和时间 |
| Version | 获取或设置此Cookie符合的HTTP状态维护版本 | 此Cookie符合的HTTP状态维护版本 |

**2．Cookie对象的方法**   
Cookie对象的方法如表11-12所示：   
表14-12 Cookie对象的方法

|  |  |
| --- | --- |
| 方   法 | 说   明 |
| Add | 新增一个Cookie变量 |
| Clear | 清除Cookie集合内的变量 |
| Get | 通过变量名或索引得到Cookie的变量值 |
| GetKey | 以索引值来获取Cookie的变量名称 |
| Remove | 通过Cookie变量名来删除Cookie变量 |

实例14-17：设置Cookie   
下面的示例将创建名为“LastVisit”的新Cookie，将该Cookie的值设置为当前日期和时间，并将其添加到当前Cookie集合中，所有Cookie均通过HTTP输出流在Set-Cookie头中发送到客户端。   
HttpCookie MyCookie = new HttpCookie("LastVisit");   
DateTime now = DateTime.Now;   
MyCookie.Value = now.ToString();   
MyCookie.Expires = now.AddHours(1);   
Response.Cookies.Add(MyCookie);   
运行上面例子，将会在用户机器的Cookies目录下建立如下内容的文本文件：   
mycookie   
LastVisit   
尽管上面的这个例子很简单，但可以从中扩展许多富有创造性的应用程序。   
实例14-18：获取客户端发送的Cookie信息   
下面的示例是依次通过客户端发送的所有Cookie，并将每个Cookie的名称、过期日期、安全参数和值发送到HTTP输出。   
int loop1, loop2;   
HttpCookieCollection MyCookieColl;   
HttpCookie MyCookie;   
MyCookieColl = Request.Cookies;   
//把所有的cookie名放到一个字符数组中   
String[] arr1 = MyCookieColl.AllKeys;   
//用cookie名获取单个cookie对象   
for (loop1 = 0; loop1 < arr1.Length; loop1++)   
{   
MyCookie = MyCookieColl[arr1[loop1]];   
Response.Write("Cookie: " + MyCookie.Name + "<br>");   
Response.Write("Expires: " + MyCookie.Expires + "<br>");   
Response.Write ("Secure:" + MyCookie.Secure + "<br>");   
//将单个cookie的值放入一个对象数组   
String[] arr2 = MyCookie.Values.AllKeys;   
//遍历cookie值集合打印所有值   
for (loop2 = 0; loop2 < arr2.Length; loop2++)   
{   
  Response.Write("Value" + loop2 + ": " + arr2[loop2] + "<br>");   
}   
**七、 Cache对象**   
对于每个应用程序域均创建该类的一个实例，并且只要对应的应用程序域保持活动，该实例便保持有效。有关此类实例的信息通过HttpContext对象的Cache属性或Page对象的Cache属性来提供。   
**1．Cache对象的属性**   
Cache对象的属性如表14-13所示：   
表14-13 Cache对象的属性

|  |  |  |
| --- | --- | --- |
| 属   性 | 说   明 | 属 性 值 |
| Count | 获取存储在缓存中的项数。当监视应用程序性能或使用ASP.NET跟踪功能时，此属性可能非常有用 | 存储在缓存中的项数 |
| Item | 获取或设置指定键处的缓存项 | 表示缓存项的键的String对象 |

**2．Cache对象的方法**   
Cache对象的方法如表14-14所示：   
表14-14 Cache对象的属性

|  |  |
| --- | --- |
| 方   法 | 说   明 |
| Add | 将指定项添加到 Cache 对象，该对象具有依赖项、过期和优先级策略，以及一个委托（可用于在从 Cache 移除插入项时通知应用程序） |
| Get | 从 Cache 对象检索指定项 |
| Remove | 从应用程序的 Cache 对象移除指定项 |
| Insert | 向Cache对象插入项。使用此方法的某一版本改写具有相同key参数的现有Cache项 |

实例14-19：检索为ASP.NET文本框服务器控件缓存的值   
Get方法可以从Cache对象检索指定项，其唯一的参数key表示要检索的缓存项的标识符。该方法返回检索到的缓存项，未找到该键时为空引用。   
下面的示例展示如何检索为ASP.NET文本框服务器控件缓存的值。   
Cache.Get("MyTextBox.Value");   
实例14-20：移除Cache对象   
Remove方法可以从应用程序的Cache对象移除指定项，其唯一的参数key表示要移除的缓存项的String标识符。该方法返回从Cache移除的项。如果未找到键参数中的值，则返回空引用。   
下面的示例创建一个RemoveItemFromCache函数。调用此函数时，它使用Item属性检查缓存中是否包含与Key1键值相关的对象。如果包含，则调用Remove方法来移除该对象。   
public void RemoveItemFromCache(Object sender, EventArgs e)   
{   
if(Cache["Key1"] != null)   
  Cache.Remove("Key1");   
}