**[ASP.NET基础之HttpContext学习](http://www.cnblogs.com/wujy/p/3264475.html)**

**http://www.cnblogs.com/wujy/p/3264475.html**

上一个随笔主要针对HttpModeul[[ASP.NET基础之HttpModule学习](http://www.cnblogs.com/wujy/p/3261141.html)]内容进行一个简单的学习；本章内容我将对ASP.NET另一个主要的知识点HttpContext进行学习；首先我们简单了解一些理论的知识，然后结合一些实例演示我们开发时经常要用到的一些功能，其中有些部分是从网上整理所得，希望对您的阅读有所帮助；

**一：HttpContext理论知识：**

1：HttpContext类它对Request、Respose、Server等等都进行了封装,并保证在整个请求周期内都可以随时随地的调用；为继承 IHttpModule 和 IHttpHandler 接口的类提供了对当前 HTTP 请求的 HttpContext 对象的引用。该对象提供对请求的内部Request、Response 和 Server 属性的访问。HttpContext的命名空间：System.Web（在 system.web.dll 中）;除了对几个对象进行封装外它还有个HttpContext.Item，通过它你可以在HttpContext的生存周期内提前存储一些临时的数据，方便随时使用。

2：生存周期：从客户端用户点击并产生了一个向服务器发送请求开始---服务器处理完请求并生成返回到客户端为止。针对每个不同用户的请求，服务器都会创建一个新的HttpContext实例直到请求结束,服务器销毁这个实例。

3：当我们创建一个一般处理程序Handler.ashx时，我们可以在文件中看到这一句  public void ProcessRequest (HttpContext context)；

4:可以通过HttpContext.Current获得当前的上下文httpContext的内容；这样可以在多处方便获取我们想要的数据；

**二：HttpContext.Item的运用**

在文中第一点我们有提到HttpContext.Item可以通过它来存储一些临时的数据；我们将通过一个实例来演示它的实现，以及在此过程中应该注意什么；还针对ASP.NET常见的两种跳转进行比较（分别为：Server.Transfer和Response.Redirect）；

1:首先我们新建一个页面对HttpContext.Item它进行写入数据(此处不一样的是当写完值后不同的页面跳转方式):

[复制代码](javascript:void(0);)

protected void Btn\_Transfer\_Click(object sender, EventArgs e)

{

List<String> list = new List<string>();

list.Add("踏浪帅");

list.Add("www.cnblogs.com/wujy");

HttpContext.Current.Items["TransferName"] = list;

Server.Transfer("Index.aspx");

}

protected void Btn\_Redirect\_Click(object sender, EventArgs e)

{

List<String> list = new List<string>();

list.Add("踏浪帅");

list.Add("www.cnblogs.com/wujy");

HttpContext.Current.Items["RedirectName"] = list;

Response.Redirect("Index.aspx");

}

[复制代码](javascript:void(0);)

接着我们再新建一个Index.aspx页面接收此页面的值：

[复制代码](javascript:void(0);)

public partial class Index : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

if(!IsPostBack)

{

List<string> list = HttpContext.Current.Items["TransferName"] as List<string>;

if (list != null)

{

foreach (string item in list)

{

Response.Write(string.Format("Transfer输出的值为：{0} <br/>", item));

}

}

List<string> ResList = HttpContext.Current.Items["RedirectName"] as List<string>;

if (ResList != null)

{

foreach (string item in ResList)

{

Response.Write(string.Format("Redirect输出的值为：{0}", item));

}

}

else

{

Response.Write("Response.Redirect传过来为空!");

}

}

}

}

[复制代码](javascript:void(0);)

现在我们看一下运行的效果：

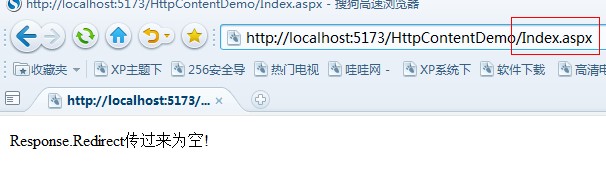
|  |  |
| --- | --- |
| Server.Transfer方式 | Response.Redirect方式 |
| http://images.cnitblog.com/blog/80291/201308/17135254-fee6e0025b1743b096735bfb443b929e.jpg | http://images.cnitblog.com/blog/80291/201308/17135309-e4b4729095f14357b99bb1fbcfd5b9ae.jpg |

从上面运行的结果我们不然发现使用Server.Transfer方式我们在接收页面成功获得我们写入的数据；而Response.Redirect方式却导致数据的丢失；出现这个问题的原因是Server.Transfer是在服务器直接操作的的和调用的地方属于同一次http请求，此处如果使用Response.Redirect是从客户端重新发起了一次Http请求；而正如我们前面所说的HttpContext的生存周期只能在一个http请求才不会被服务器销毁这个实例；

针对上面Server.Transfer是在服务器直接操作和调用的我们其实可以看一下我们跳转时URL地址的变化就知道了，URL它是没有变化的；

|  |  |
| --- | --- |
| Server.Transfer |  |
| http://images.cnitblog.com/blog/80291/201308/17140448-0f9732e25e76431597f1122433b562cd.jpg | http://images.cnitblog.com/blog/80291/201308/17140505-5ae2380ef0684778ba3bf0e7fd4b18ca.jpg |

而Response.Redirect跳转后的URL地址是发生变化（它是从客户端再去做起一次请求访问到index.aspx页面）：



注意：一般的情况下尽可能用Server.Transfer方法（前提是服务器是IIS)，Server.Transfer方法更快速，而且因为只在服务器上执行，所以可以和任何浏览器兼容。

Response.Redirect其实上是当服务器碰到这条语句时发送一条指令（包含新的地址）给浏览器，然后让浏览器去发送http请求，请求Response.Redirect后面的那个新的http地址，流程如下:浏览器文件请求->服务器执行->遇到response.redirect语句->服务器发送Server.Transfer后面的地址给客户机端的浏览器->浏览器请求执行新的地址（服务器返回的Response.Redirect后面的地址）这就是一个小小的Response.Redirect的全部过程，  
Server.Transfer语句当接受地址后是直接转向后面的地址，流程如下:浏览器文件请求->服务器执行->遇到Server.Transfer语句->服务器转向新的文件。

2:截获Http请求并作特殊处理

主要是用在Application\_BeginRequest方法里。Application\_BeginRequest方法在global.asax.cs里定义。你可以在Http请求刚刚开始的时候，截获他，做一些特殊的处理。这里你不能用Session来做，因为此时Session对象还没有被建立。 利用Context.Item集合，你可以在Http请求的整个生命期，通过HttpModules, HttpHandlers, Webforms, and Application 事件。

**三：HttpContet对象之Request**

1：Request理论知识：

ASP.NET Request 封装了客户端请求信息，是从客户端得到数据(从浏览器获取数据);常用的三种取得数据的方法是：Request.Form、Request.QueryString、Request其第三种是前两种的一个缩写，可以取代前两种情况。而前两种主要对应的Form提交时的两种不同的提交方法：分别是Post方法和Get方法。

2：新建一个页面用来显示Requesst一些常见属性信息的内容；

[复制代码](javascript:void(0);)

public partial class RequestPage : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

if (!IsPostBack)

{

StringBuilder strBuider = new StringBuilder();

strBuider.Append("客户端主机的IP地址：" + HttpContext.Current.Request.UserHostAddress + "</br>");

strBuider.Append("客户端浏览器版本:" + Request.UserAgent + "</br>");

strBuider.Append("当前由哪个页面URL跳转过来:" + Request.UrlReferrer+ "</br>");

strBuider.Append("当前要求的URL:" + Request.Url + "</br>");

strBuider.Append("当前要求的URL绝对地址:" + Request.Url.AbsolutePath + "</br>");

strBuider.Append("当前要求的URL绝对URI:" + Request.Url.AbsoluteUri + "</br>");

strBuider.Append("当前要求的URL主机名跟端口:" + Request.Url.Authority + "</br>");

strBuider.Append("当前要求的URL实例主机的一部分:" + Request.Url.Host + "</br>");

strBuider.Append("当前要求的URL端口:" + Request.Url.Port + "</br>");

strBuider.Append("当前要求的URL的内容第一个段:" + Request.Url.Segments[0] + "</br>");

strBuider.Append("当前要求的URL的内容第二个段:" + Request.Url.Segments[1] + "</br>");

strBuider.Append("当前要求的URL的内容第三个段:" + Request.Url.Segments[2] + "</br>");

strBuider.Append("浏览器地址栏后的参数"+Request.QueryString + "</br>");

strBuider.Append("当前网页在服务器端的实际路径:" + Request.PhysicalPath + "</br>");

strBuider.Append("当前文件的物理地址:" + Request.PhysicalApplicationPath + "</br>");

strBuider.Append("当前网页的相对地址:" + Request.Path + "<br/>");

strBuider.Append("当前页面的URL:" + Request.RawUrl + "<br/>");

strBuider.Append("客户端上传的文件（个数）:" + Request.Files.Count + "<br/>");

strBuider.Append("当前执行网页的相对地址:" + Request.FilePath + "<br/>");

strBuider.Append("客户端浏览器的信息:" + Request.Browser + "<br/>");

strBuider.Append("当前运行程序的服务器端虚拟目录:" + Request.ApplicationPath + "<br/>");

strBuider.Append("客户端浏览器的字符设置:" + Request.ContentEncoding + "<br/>");

Response.Write(strBuider.ToString());

}

}

}

[复制代码](javascript:void(0);)

运行结果显示（获得客户端浏览器以及一些相关的信息）：



3：针对表单的提交方式如上面所说的有两种方式(POST以及GET);接下来我们则通过几段代码来演示有关表单提交的两种方式以及一些应该注意的事项；

（1）表单Get方式提交：

 相信在我们平常的开发过程中使用Get方式进行提交的表单是最经常碰到的，它也是最简单的一种方式，主要注意在使用Get提交时URL对数据的大小是有限止；还经常碰到的情况是对URL传参时一些特殊符号的处理问题；接下来我们将通过小实例来演示两种对URL传参处理特殊符号方式；

首先是使用ASP.NET代码的Server对象对要传参进行一个编码的处理（Server.UrlEncode(string s)）；

string UserName = "踏浪帅￥%—\*@";

Server.Transfer("GetPageResult.aspx?UserName="+Server.UrlEncode(UserName)+"&UserID=001");

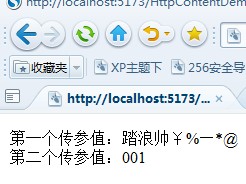
接上时再对收到的参数进行一个解码（Get方式使用Request.QueryString[Name]和Request[Name]进行接收参数;解码Server.UrlDecode(string s)）：

string UserName = Request.QueryString["UserName"];

Response.Write("第一个传参值：" + Server.UrlDecode(UserName) + "</br>");

string UserID = Request["UserID"];

Response.Write("第二个传参值：" + UserID + "</br>");

查后运行结果：  


另外一种方式是使用JS进行传参，此得使用到的JS中的escape();而它接收解码跟上面使用ASP.NET代码是一样；

[复制代码](javascript:void(0);)

<head runat="server">

<title></title>

<script type="text/javascript" src="jquery-1.4.2.min.js"></script>

<script language="javascript" type="text/javascript">

$(function () {

$("#Btn\_Get").bind('click',GetResult);

})

function GetResult() {

var UserName = "踏浪帅￥%—\*@";

location.href = "GetPageResult.aspx?UserID=002&UserName="+escape(UserName);

}

</script>

</head>

<body>

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

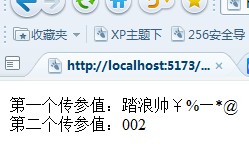
<input id="Btn\_Get" type="button" value="HTML中的JS传值" />

</form>

</body>

</html>

[复制代码](javascript:void(0);)

运行结果：  


（2）Post方式提交：

Post方式的表单提交比Get方式更加安全，它不会把我们要传递的参数显示在URL里，还有它可以针对一些要传递内容比较大(比如文件上传等);先通过一段代码来演示此方式的提交；后面我们再顺便介绍Post方式提交时常遇到的一个错误；

首先同样新建一个页面（我们把表单通过Post提交到PostPageResult.aspx页面，method就是用来设置我们使用Post方式）：

[复制代码](javascript:void(0);)

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" action="PostPageResult.aspx" method="post">

<div>

Cnblogs账号：<input type="text" size="20" name="Name" value="踏浪帅"/>

<br />

<input type="submit" value="提交" />

</div>

</form>

</body>

</html>

[复制代码](javascript:void(0);)

接收页在的代码（Post可以采用Request.Form[Name]和Request[Name]进行接收参数，若是提交过来有上传文件可以使用 Request.Files进行获得跟处理）：

Response.Write("Post方式采用Request.Form[Name]：" + Request.Form["name"] + "</br>");

Response.Write("Post方式采用Request[Name]：" + Request["name"]);

在使用Post经常会无意中碰到一个错误：  


解决方式(当runat="server"的表单通过修改action提交数据到其它页面时，也会引发这个问题;把表单中的runat="server"去掉);



网上还有其它解决的办法：

a:添加enableEventValidation="false" enableViewStateMac="false"或在webconfig中添加<pages enableEventValidation="false" enableViewStateMac="false" />

b:不使用跨页面提交，提交到本页后在page.load中redirect.

c:使用 <asp:Button runat="server" PostBackUrl="~/Register/DoRegister.aspx" Text="提交" />

4:平常通过Request获得客户端的Cookies值；

HttpCookie cookie = HttpContext.Current.Request.Cookies["HttpDemo"];

**四：HttpContet对象之Response**

1:Response理论知识：

Response代表了服务器响应对象。每次客户端发出一个请求的时候，服务器就会用一个响应对象来处理这个请求，处理完这个请求之后，服务器就会销毁这个相应对象，以便继续接受其它客服端请求。

2：Response主要用来一些输出的运用；只要简单记住其几个重要的属性以及方法就差不多了；

[复制代码](javascript:void(0);)

protected void Button1\_Click(object sender, EventArgs e)

{

Response.Redirect("Default.aspx");

}

protected void Button2\_Click(object sender, EventArgs e)

{

string FileContent = Server.MapPath(".") + "\\TextFile.txt";

Response.WriteFile(FileContent);

}

protected void Button3\_Click(object sender, EventArgs e)

{

Response.Write("这是第一句");

Response.End(); //停止运行，不再执行任何语句

Response.Write("这是第二句");

Response.Clear(); //清空缓冲区中的所有内容输出

Response.Close();//关闭当前服务器到客户端的连接

}

[复制代码](javascript:void(0);)

 Response还有个我们经常使用到的地方是针对下载文件时输出时的内容：

[复制代码](javascript:void(0);)

public static void ResponseStream(Stream stream, string fileName)

{

if (stream.Length > 0)

{

try

{

long fileSize = stream.Length;

System.Web.HttpContext.Current.Response.ContentType = "application/octet-stream"; //控制送出的文件类型

System.Web.HttpContext.Current.Response.AppendHeader("Content-Disposition", " attachment;filename=" + HttpUtility.UrlEncode(fileName, System.Text.Encoding.UTF8));//向响应输出流增加HTTP头信息

System.Web.HttpContext.Current.Response.AddHeader("Content-Length", fileSize.ToString());

byte[] fileBuffer = new byte[fileSize];

stream.Read(fileBuffer, 0, (int)fileSize);

stream.Close();

System.Web.HttpContext.Current.Response.BinaryWrite(fileBuffer);

System.Web.HttpContext.Current.Response.End();

}

catch (Exception ex)

{

throw;

}

}

}

[复制代码](javascript:void(0);)

针对Response.ContentType是表示送出的文件类型（下在我们列出一些比较常见的类型，其它类型可以到网络上进行搜索）：

[复制代码](javascript:void(0);)

'doc' => 'application/msword',

'bin' => 'application/octet-stream',

'exe' => 'application/octet-stream',

'dll' => 'application/octet-stream',

'pdf' => 'application/pdf',

'xls' => 'application/vnd.ms-excel',

'ppt' => 'application/vnd.ms-powerpoint',

'zip' => 'application/zip',

'mp3' => 'audio/mpeg',

'wav' => 'audio/x-wav',

'pdb' => 'chemical/x-pdb',

'bmp' => 'image/bmp',

'gif' => 'image/gif',

'ief' => 'image/ief',

'jpeg' => 'image/jpeg',

'jpg' => 'image/jpeg',

'jpe' => 'image/jpeg',

'png' => 'image/png',

[复制代码](javascript:void(0);)

另外当我们在写入Cookies时通过Response的Set-Cookie向客户端设置的Cookie;

HttpCookie ck = new HttpCookie("HttpDemo");

ck.Values.Add("UserID", "踏浪帅");

Response.SetCookie(ck);

Response.Redirect("Default.aspx");

**五：HttpContet对象之Sever**

1:Server理论知识：

Server对象是用于获取服务器的相关信息的对象

2：对于Server对象我们也是简单的了解一些它的属性内容；

[复制代码](javascript:void(0);)

StringBuilder strBuilder = new StringBuilder();

strBuilder.Append("获取计算机名:"+HttpContext.Current.Server.MachineName+"</br>");

strBuilder.Append("获取指定相对路径在服务器上的物理路径:" + Server.MapPath(".") + "</br>");

strBuilder.Append("在服务器里实现跳转;不用再返回客户端；Server.Transfer(string Path)</br>");

strBuilder.Append("先执行路径所代表的URL，然后执行完之后再执行本页:Server.Execute(string Path)</br>");

strBuilder.Append("对特殊字符串的处理：Server.HtmlDecode(string s)与Server.HtmlEncode(string s)</br>");

strBuilder.Append("对URL路径字符串进行编解码：Server.UrlDecode(string s)与Server.UrlEncode(string s)</br>");

Response.Write(strBuilder.ToString());

[复制代码](javascript:void(0);)

如果，您认为阅读这篇博客让您有些收获，不妨点击一下右下角的[【**推荐**】](javascript:void(0);)按钮。  因为，我的写作热情也离不开您的肯定支持。

感谢您的阅读，[[源代码下载](http://download.csdn.net/detail/u011539854/5964451)]

标签: [ASP.NET基础](http://www.cnblogs.com/wujy/tag/ASP.NET%E5%9F%BA%E7%A1%80/)