状态保持

# HTTP无状态

通俗：一个浏览者发出的请求都是由实现了IHttpHandler接口的对象进行响应，由于下次访问不一定还是上次那个对象进行响应，上次响应完毕对象可能已经被销毁了，写的类变量值早就不存在了，因此不要将状态信息保存到类变量中。

例子：

|  |
| --- |
| private int i;  public void Count()  {  Response.ContentType = "text/plain";  Response.Write(i++);  } |

多次刷新我们发现，变量根本不会记忆上次的值。

## 无状态的根本原因

浏览器和服务器使用Socket通信，服务器将请求结果返回给浏览器后，会关闭当前Socket连接。而且服务器会在处理页面完毕后销毁页面对象。

## 应用层面的原因

浏览器和服务器之间通信都遵守HTTP协议。

Http协议是无状态的，不会记得上次和网页“发生了什么”。服务器不记得上次给了浏览器什么。

## 对网站造成的影响

如果用户录入了一些信息，当跳转到下一个页面时，数据丢失，再也不能获得那些数据。

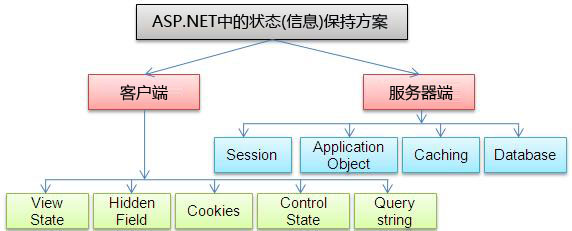
如果要知道上一次的状态信息，我们就得把这个状态信息记录在某个地方：

a.服务器端

b.浏览器端

c. 表单元素中—如：隐藏域<input type=“hidden”/>(Http报文)

## g状态保持方案



### 隐藏域

例如，想要实现div中数字的自增，因为div不会把数据提交到服务器，所以需要把div中数字保存到一个某个地方。这里我们使用隐藏域。当提交时把隐藏域中的信息提交到服务器，服务器处理后，把数据同时写入div与隐藏域中。

#### 前端代码

|  |
| --- |
| <body>  <form id="form1" runat="server">  <div>  <%= this.selfAdd%>  </div>  <input type="hidden" name="hidAdd" value="<%=this.selfAdd %>" " />  <input type="submit" name="name" value="提交" />  </form>  </body> |

#### 后置代码

|  |
| --- |
| public int selfAdd { get; set; }  protected void Page\_Load(object sender, EventArgs e)  {  if (!IsPostBack)  {  selfAdd = 0;  }  else  {  int num = int.Parse(Request.Form["hidAdd"]);  num++;  selfAdd = num;  }  } |

### ViewState

两种使用方式： 作用域---页面级

1.用户数据保存方式：

ViewState["myKey"]="MyData";

读取数据方式：

string myData;

if(ViewState["myKey"]!=null)

{ myData=(string)ViewState["myKey"]; }

注意：ViewState不能存储所有的数据类型，仅支持可序列化对象。

2.非单值服务器控件的状态也自动保存在ViewState中

例如：下拉框。（文本框是单值控件，所以不会保存在ViewState中）

使用ViewState的前提：

页面上必须有一个服务器端窗体标记(<form runat=“server”>)

服务器在接收到用户请求一个页面后，会自动在请求报文中找看是否包含\_\_VIEWSTATE的隐藏域，如果有，则将中间的值解码后添加到页面的ViewState属性中。

服务器在输出的时候，也会自动的将ViewState中的值添加到表单里名叫\_\_VIEWSTATE的隐藏域中

VIEWSTATE适用于同一个页面在不关闭的情况下多次与服务器交互，用来同一个页面的传值。

#### ViewState的实质

ViewState的实质是隐藏域。

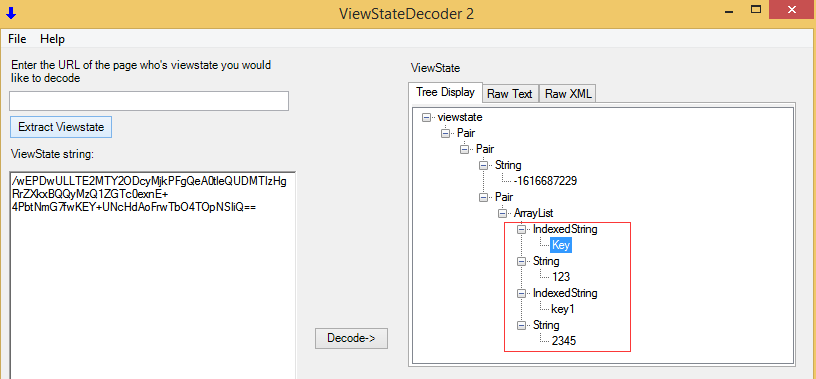
例如，下面的代码保存了2个值

|  |
| --- |
| ViewState["Key"] = "123";  ViewState["key1"] = "2345"; |

执行时，在浏览器中查看源代码：

|  |
| --- |
|  |

会发现多了一个隐藏域。反编译其value：



可以发现，Viewstate中定义的键值对在value中保存着。

#### 案例：文本框数据自增



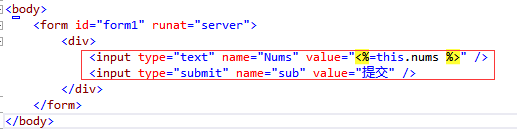
功能：点击提交按钮，文本框中的数字自增1.

思路：

1. 如果是首次加载则文本框值初始化为0。同时把改值保存到viewstate中。
2. 点击提交。从viewstate中取得上次保存的值。自增1。然后把新值保存到viewstate中。

#### 实现

前台页面中添加一个文本框



后置代码中

|  |
| --- |
| public partial class ViewStateDemo : System.Web.UI.Page  {  public int nums;  protected void Page\_Load(object sender, EventArgs e)  {  if (!IsPostBack)  {  nums = 0;  ViewState["Nums"] = nums;//数据保存到viewstate  }  else  {  nums = (int)ViewState["Nums"];//从viewstate读取数据。  nums++;  ViewState["Nums"] = nums;  }  }  } |

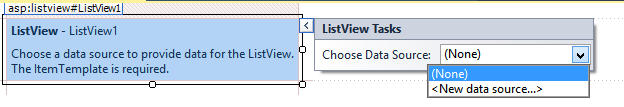
点击提交按钮，自增

#### 禁用viewState

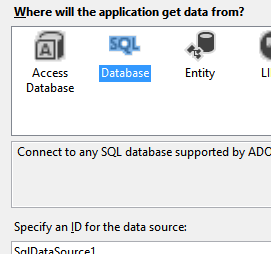
Viewstate会在客户端和服务器端来回传递，效率差，影响性能。特别是在用服务器空控件时，如果仅仅是展示数据，不需要用viewstate来保存状态，但是服务器控件会在viewState中插入大量数据。

例如，我们添加一个listview并绑定数据，最后查看以下viewState。

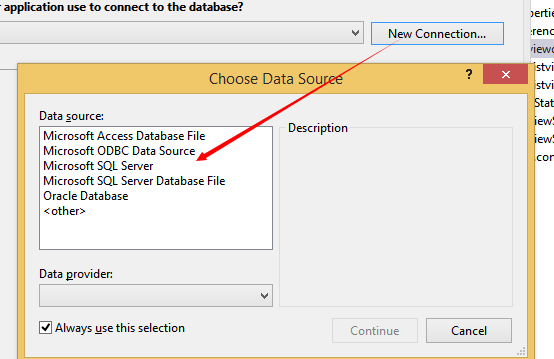
1. 添加listview并选择 new data soruce



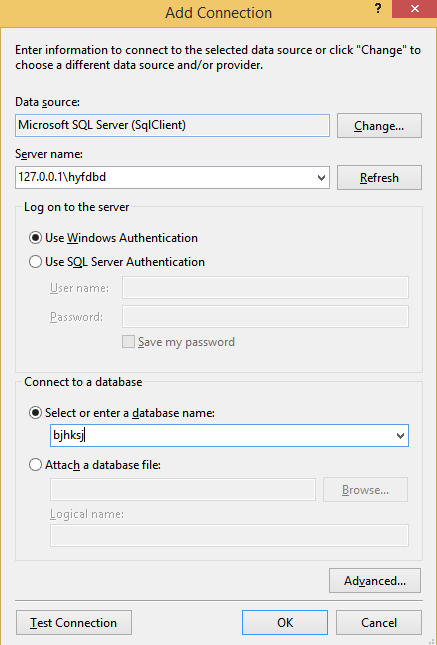
1. 选择数据库



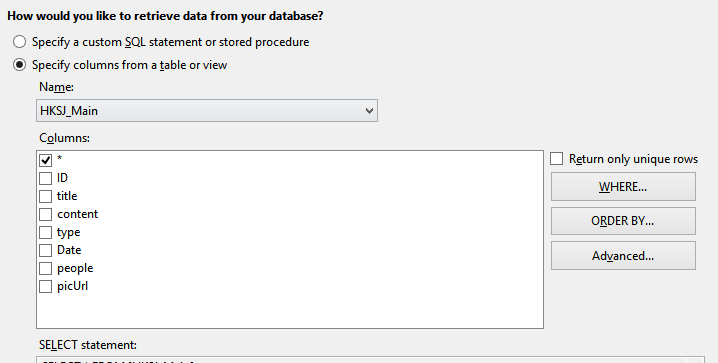
1. 选择数据源



1. 创建连接



1. 选择数据表



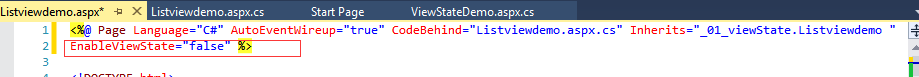
1. 绑定成功，在浏览器中查看源代码。发现viewState的值数据量非常大。



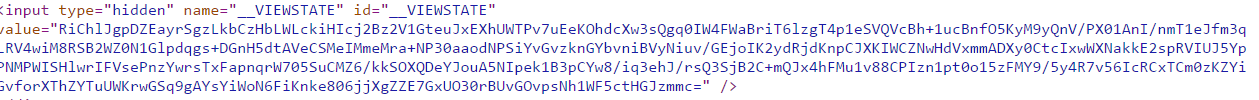
1. 我们可以禁用viewState来增强性能。
2. 禁用viewstate的手段有两种：一种是禁止整个页面的viewState，另外是禁止单个控件的viewState.

##### 禁用整个网页的viewState

在aspx的Page指令区加上EnableViewState="false" 。内网系统、互联网的后台可以尽情的用ViewState。



再次在浏览器查看源代码，可以发现ViewState大量减少。



禁用整个网页的viewstate后，别的控件即使想用也用不成viewstate了。有需要，可以仅禁止不需要viewstate的控件。

##### 禁用单个控件的viewstate

在控件的属性中设置EnableViewState为false。



# Cookie

## 基础知识

### 什么是Cookies

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。Cookie 包含每次用户访问站点时 Web 应用程序都可以读取的信息。

例如，如果在用户请求站点中的页面时应用程序发送给该用户的不仅仅是一个页面，还有一个包含日期和时间的 Cookie，用户的浏览器在获得页面的同时还获得了该 Cookie，并将它存储在用户硬盘上的某个文件夹中。

以后，如果该用户再次请求您站点中的页面，当该用户输入 URL 时，浏览器便会在本地硬盘上查找与该 URL 关联的 Cookie。如果该 Cookie 存在，浏览器便将该 Cookie 与页请求一起发送到您的站点。然后，应用程序便可以确定该用户上次访问站点的日期和时间。您可以使用这些信息向用户显示一条消息，也可以检查到期日期。

Cookie 与网站关联，而不是与特定的页面关联。因此，无论用户请求站点中的哪一个页面，浏览器和服务器都将交换 Cookie 信息。用户访问不同站点时，各个站点都可能会向用户的浏览器发送一个 Cookie；浏览器会分别存储所有 Cookie。

Cookie 帮助网站存储有关访问者的信息。一般来说，Cookie 是一种保持 Web 应用程序连续性（即执行状态管理）的方法。除短暂的实际交换信息的时间外，浏览器和 Web 服务器间都是断开连接的。对于用户向 Web 服务器发出的每个请求，Web 服务器都会单独处理。但是在很多情况下，Web 服务器在用户请求页时识别出用户会十分有用。例如，购物站点上的 Web 服务器跟踪每位购物者，这样站点就可以管理购物车和其他的用户特定信息。因此，Cookie 可以作为一种名片，提供相关的标识信息帮助应用程序确定如何继续执行。

使用 Cookie 能够达到多种目的，所有这些目的都是为了帮助网站记住用户。例如，一个实施民意测验的站点可以简单地将 Cookie 作为一个 Boolean 值，用它来指示用户的浏览器是否已参与了投票，这样用户便无法进行第二次投票。要求用户登录的站点则可以通过 Cookie 来记录用户已经登录，这样用户就不必每次都输入凭据。

### Cookies如何存储

Cookies保存在用户的本地机器上，不同的浏览器存储在不同的文件夹中，并且按照域名分别保存。即网站之间的Cookies不会彼此覆盖。

IE浏览器的用户可以通过在本地的文档中找到Cookies的txt文件， 不同操作系统的位置不同，windows server 2003/xp都保存在：

C:\Documents and Settings\Administrator\Cookies 文件夹下。

其中名称txt按照域名保存，比如localhost域下的cookies为：

administrator@localhost[1].txt 或者 administrator@localhost[2].txt

其中后面的[1]和[2]是随着每次保存交替变化的。

### Cookies如何传递

Cookies的信息是在Web服务器和浏览器之间传递的。保存在Http请求中。

（1）请求页面

在请求一个页面的Http头中，会将属于此页面的本地Cookies信息加在Http头中，注意下面加粗的部分：

|  |
| --- |
| GET /Cookies/Test.aspx HTTP/1.1  Host: localhost:1335  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; zh-CN; rv:1.9.1.1) Gecko/20090715 Firefox/3.5.1 GTB5 (.NET CLR 3.5.30729)  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,zh;q=0.5  Accept-Encoding: gzip,deflate  Accept-Charset: GB2312,utf-8;q=0.7,\*;q=0.7  Keep-Alive: 300  Connection: keep-alive  Cookie: My.Common.TestCookieInfo=Pkid=999&TestValue=aaabbbcccdddeee |

（2）页面响应

如果页面要求写入Cookies信息，则返回的Http如下，注意加粗的部分：

|  |
| --- |
| HTTP/1.x 200 OK  Server: ASP.NET Development Server/9.0.0.0  Date: Thu, 06 Aug 2009 03:40:59 GMT  X-AspNet-Version: 2.0.50727  Set-Cookie: My.Common.TestCookieInfo=Pkid=999&TestValue=aaabbbcccdddeee; expires=Fri, 07-Aug-2009 03:40:59 GMT; path=/  Cache-Control: private  Content-Type: text/html; charset=utf-8  Content-Length: 558  Connection: Close |

### Cookies如何查看

（1）查看Cookies的txt文件

IE用户可以直接查看Cookies的txt文件。

比如：C:\Documents and Settings\Administrator\Cookies\administrator@localhost[1].txt

（2）使用插件

FF下使用Web Developer插件可以很方便的查看、删除和修改Cookies：

## Cookies高级知识

### Cookie 的限制

大多数浏览器支持最大为 4096 字节的 Cookie。

浏览器还限制站点可以在用户计算机上存储的 Cookie 的数量。大多数浏览器只允许每个站点存储 20 个 Cookie；注意这里的20个是指主键值，也就是20条Cookies记录，但是每个Cookies记录还可以包含若干子键，下面会详细解释。如果试图存储更多 Cookie，则最旧的 Cookie 便会被丢弃。有些浏览器还会对它们将接受的来自所有站点的 Cookie 总数作出绝对限制，通常为 300 个。

### Cookies的存储格式

Cookies可以包含一个主键， 主键再包含子键。比如asp.net中获取Cookies的格式是：

Request.Cookies[key][subkey].ToString();

其中的key就是主键，subkey就是主键关联的子键。

（1）本地磁盘存储格式：

My.Common.TestCookieInfo

Pkid=999&TestValue=aaabbbcccdddeee

localhost/

1536

3059603968

30021392

2348960464

30021191

\*

其中的Pkid=999&TestValue=aaabbbcccdddeee 是Cookies的值，由于使用了subkey=subvalue的格式， 所以此Cookies是包含子键的。

（2）Javascript中的Cookie格式

在Javascript中给的Cookie是一个字符串，通过document.cookies获取。字符格式如下：

My.Common.SubKey=Pkid=999&TestValue=aaabbbcccdddeee; SingleKey=SingleKeyValue

上面的字符串包含了两个Cookies，一个是不包含子键的SingleKey， 一个是包含pkid和TextValue两个子键的My.Common.SubKey，两个Cookie通过“;”分割。

（3）Asp.Net中的Cookies格式

和所有的服务器端语言一样，Asp.Net中使用集合类保存Cookies集合：

public sealed class HttpCookieCollection : NameObjectCollectionBase

{...}

通过HttpResquest和HttpResponse对象的Cookies属性，可以获取和写入当前页面的Cookies。

### Cookies的内容编码格式

Cookies的值中可以保存除了“;”以外的标点符号。但是不能保存汉字。保存汉字会出现乱码。

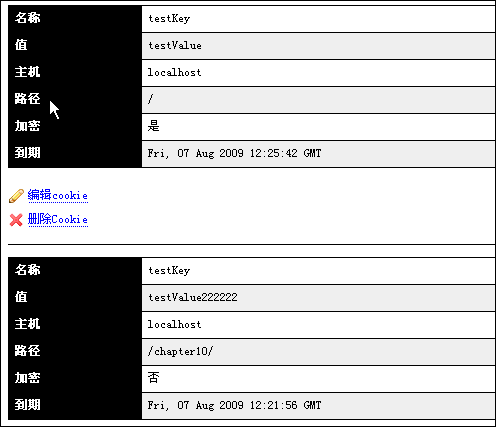
所以对于Cookies中的内容要进行统一的编码和解码。为了在浏览器端和服务器端都能够进行解码和编码， 所以要统一使用UTF编码格式。

主要是因为javascript中只能使用UTF编码格式。

### Cookies的Path属性

Cookies的Path属性表示当前的Cookies可以作用在网站的那个路径下。

比如下面的两个同名的Cookies：



允许存在两个同名但是Path不同的Cookies。

无论是服务器端还是客户端，在获取时优先获取本页路径下面的Cookies。

也就是说如果在、/chapter10/路径下面的页面， 获取testKey这个Cookies的值，则只能获取到testValue222222这个值。

### Cookies的过期时间

如果保存Cookies时未设置过期时间， 则Cookies的过期时间为“当前浏览器进程有效”，即和Session一样关闭浏览器后则消失。在asp.net中还可以通过设置HttpCookie对象的过期时间为DateTime.MinValue来指定此Cookies为跟随浏览器生效。（这句话来之不易啊，在脑袋等人的帮助下才查到的。）

如果设置了过期时间并且大于当前时间，则会保存Cookies值。

如果设置了过期时间但是小于等于当前时间，则清除Cookies值。

### Cookies与Session

有时我们会忽略Cookies与Session的关系。但是两者是密不可分的。

Session的唯一标示：SessionID是通常保存在Cookies中的（也可以保存在URL中）。对于Asp.Net而言，SessionID保存在键值为“ASP.NET\_SessionId”的Cookies中，如图：



因为Cookies的存储数量是有限制的，所以我们的系统在保存Cookies的时候一定要注意防止冲掉这一个关键的Cookies。在下文介绍的最佳实践-以强对象方式保存Cookies的代码中特意对这个Cookies做了处理。

注意，在客户端使用javascript脚本无法获取“ASP.NET\_SessionId”的Cookies， 因为此Cookies在服务器端设置了HttpOnly属性为true。

ASP.Net中HttpCookie对象的HttpOnly 属性 指定一个Cookie 是否可通过客户端脚本访问。不能通过客户端脚本访问为 true；否则为 false。默认值为 false。此属性并不能完全阻止客户端在本地获取cookies，但是可以增加通过脚本直接获取的难度。

Microsoft Internet Explorer 版本 6 Service Pack 1 和更高版本支持 Cookie 属性 HttpOnly。

### Cookies加密

在设置Cookies的属性时，有一个选项Secure用来控制Cookie的加密特性。

如果通过 SSL 连接 (HTTPS) 传输 Cookie，则为 true；否则为 false。默认为 false。

如果我们保存一个Cookies并设置加密，那么在非HTTPS的页面中，无论是使用javascript还是服务器端都无法获得此Cookies。但是在本地依然可以看到此Cookies的存在。

### Cookies与Ajax

如果Ajax请求访问一个服务器页面，此服务器页面是可以向用户浏览器写入Cookies和Session的。

## Cookies最佳实践

在了解了Cookies的相关知识后，下面提出最佳的事件方法。其中包括客户端和服务器端两部分。

### Asp.Net 中保存Cookies

通常，我们使用Request和Response对象来直接操作Cookies：

写入Cookies：

|  |
| --- |
| Response.Cookies["k1"].Value = "k1Value";  Response.Cookies["k2"]["k2-1"] = "k2-1Value";  Response.Cookies.Add(new HttpCookie("k3", "k3Value")); |

读取Cookies：

|  |
| --- |
| Request["k1"] ;  Request.Cookies["k1"].Value ;  Request.Cookies["k2"]["k2-1"];  Request.Cookies.Get(0).Value; |

注意Request["k1"]这个大家熟悉的获取get和post参数的方法，同时还能够获取Cookies的值！

另外上面语句中的有些是必须通过Value属性访问的，有些则不需要。

### 以对象方式保存Cookies

下面提供一个可以以对象方式整体保存Cookies的工具类。并且只占用一条Cookies，所有的属性都存在子键上。

•源代码：

|  |
| --- |
| /// <summary>  /// Cookies基类。将需要保存Cookies的数据类此类派生，可以将强类型对象在Cookies中的保存和读取。  /// </summary>  /// <remarks>  /// 2009.8.6 ziqiu.zhang created  /// </remarks>  /// <example>  /// 假设MyCookiesInfo是从 从Cookies中获取对象：  /// <code>  /// CookieInfo item = new CookieInfo(); //new以后已经从Cookies中构造了对象。  /// </code>  /// 将对象保存在Cookies中：  /// <code>  /// CookieInfo item = new CookieInfo();  /// item.value = "test value";  /// item.SetCookies("1"); //Cookies有效期为1天  /// </code>  /// </example>  [System.Serializable]  public class CookieInfo  {  #region ==================== Constructed Method ====================  /// <summary>  /// 构造函数  /// </summary>  public CookieInfo()  {  }  #endregion  #region ==================== Public Method ====================  /// <summary>  /// 得到当前Cookies的过期时间  /// </summary>  /// <returns>过期时间</returns>  public DateTime GetExpiresTime()  {  string cookieName = GetType().ToString();  if (HttpContext.Current.Request.Cookies[cookieName] != null)  {  return HttpContext.Current.Request.Cookies[cookieName].Expires;  }  return DateTime.MinValue;  }  /// <summary>  /// 保存Cookies，过期时间为浏览器关闭则失效。  /// </summary>  /// <param name="expiresTime">Cookies过期事件</param>  /// <returns>是否保存成功</returns>  public bool Save()  {  return this.Save(DateTime.MinValue);  }  /// <summary>  /// 保存Cookies，需要指定过期时间。  /// </summary>  /// <param name="expiresTime">Cookies过期事件</param>  /// <returns>是否保存成功</returns>  public bool Save(DateTime expiresTime)  {  string CookieName = GetType().ToString();  HttpCookie SessionCookie = null;  //对 SessionId 进行备份.  if (HttpContext.Current.Request.Cookies["ASP.NET\_SessionId"] != null)  {  string SesssionId = HttpContext.Current.Request.Cookies["ASP.NET\_SessionId"].Value.ToString();  SessionCookie = new HttpCookie("ASP.NET\_SessionId");  SessionCookie.Value = SesssionId;  }  //设定cookie 过期时间.  DateTime dtExpiry = expiresTime;  HttpContext.Current.Response.Cookies[CookieName].Expires = dtExpiry;  //设定cookie 域名.  string domain = string.Empty;  if (HttpContext.Current.Request.Params["HTTP\_HOST"] != null)  {  //domain = "www.elong.com";  domain = HttpContext.Current.Request.Params["HTTP\_HOST"].ToString();  }  //如果是www.elong.com或多级域名,需要转化为elong.com  if (domain.IndexOf(".") > -1)  {  string[] temp = domain.Split('.');  if (temp.Length >= 3)  {  domain = temp[temp.Length - 2].Trim() + "." + temp[temp.Length - 1].Trim();  }  HttpContext.Current.Response.Cookies[CookieName].Domain = domain;  }  //把类的属性, 写入Cookie.  PropertyInfo[] Propertys = GetType().GetProperties();  foreach (PropertyInfo pi in Propertys)  {  object oj = pi.GetValue(this, null);  Type type = pi.PropertyType;  string valueStr = string.Empty;  if (oj != null && oj.ToString() != string.Empty)  {  if (type == Type.GetType("System.DateTime"))  {  valueStr = ((DateTime)oj).ToString("yyyy/MM/dd HH:mm:ss", System.Globalization.DateTimeFormatInfo.InvariantInfo);  }  else  {  valueStr = oj.ToString();  }  HttpContext.Current.Response.Cookies[CookieName][pi.Name] = HttpUtility.UrlEncode(valueStr);  }  }  //如果cookie总数超过20 个, 重写ASP.NET\_SessionId, 以防Session 丢失.  if (HttpContext.Current.Request.Cookies.Count > 20 && SessionCookie != null)  {  if (SessionCookie.Value != string.Empty)  {  HttpContext.Current.Response.Cookies.Remove("ASP.NET\_SessionId");  HttpContext.Current.Response.Cookies.Add(SessionCookie);  }  }  return true;  }  /// <summary>  /// 找回Cookie值  /// </summary>  public void Load()  {  string cookieValue = string.Empty;  string CookieName = GetType().ToString();  //通过遍历属性, 从cookie 中找回值, 回写到属性.  PropertyInfo[] Propertys = GetType().GetProperties();  foreach (PropertyInfo pi in Propertys)  {  try  {  cookieValue = HttpUtility.UrlDecode(HttpContext.Current.Request.Cookies[CookieName][pi.Name].ToString());  }  catch  {  cookieValue = string.Empty;  }  if (pi.CanWrite && cookieValue != null && cookieValue != string.Empty)  {  try  {  object obb = cookieValue;  Type type = pi.PropertyType;  obb = Convert.ChangeType(obb, type);  pi.SetValue(this, obb, null);  }  catch { }  }  }  }  #endregion  } |

•使用

首先说明如何使用此类。

为想要保存在Cookies中的类建立模型，并且继承自CookieInfo即可。比如下面建立了MyCookieInfo类，其中包含属性pkid，TestValue和TestDateTime：

|  |
| --- |
| /// <summary>  /// 保存Cookies的数据对象  /// </summary>  [System.Serializable]  public class MyCookieInfo : CookieInfo  {  private int m\_Pkid = 0;  public int Pkid  {  get  {  return m\_Pkid ;  }  set  {  m\_Pkid = value ;  }  }  private string m\_TestValue = "";  public string TestValue  {  get  {  return m\_TestValue;  }  set  {  m\_TestValue = value;  }  }  private DateTime m\_TestDateTime = DateTime.Now;  public DateTime TestDateTime  {  get  {  return m\_TestDateTime;  }  set  {  m\_TestDateTime = value;  }  }  } |

接下来就可以使用对象的Save和Load方法保存和读取Cookies：

•保存

Save方法有两个重载，不带参数的Save方法表示Cookies的过期时间与浏览器相同，即浏览器关闭则Cookies消失。否则需要传入Cookies过期时间。

|  |
| --- |
| MyCookieInfo testCookies = new MyCookieInfo();  testCookies.Pkid = 1;  testCookies.TestValue = "中文测试";  testCookies.Save(); |

•读取

|  |
| --- |
| MyCookieInfo testCookies = new MyCookieInfo();  testCookies.Load();  this.lblMsg.Text = "Pkid:" + testCookies.Pkid.ToString();  this.lblMsg.Text += ",TestValue:" + testCookies.TestValue.ToString();  this.lblMsg.Text += ",TestDateTime:" +  testCookies.TestDateTime.ToString("yyyy/MM/dd HH:mm:ss",  System.Globalization.DateTimeFormatInfo.InvariantInfo); |

现在我们已经可以将一个强类型的对象读取和保存Cookies了。

### 使用Javascript操作Cookies

在客户端我们同样需要操作Cookies。

下面是封装了的专门用于操作Cookies的jQuery工具函数。如果还有人不知道jQuery是什么，请参考我的“从零开始学习jQuery”系列教程：

http://www.cnblogs.com/zhangziqiu/archive/2009/04/30/jQuery-Learn-1.html

当然此工具函数稍加修改，就可以变成标准的Javascript函数。

下载地址：http://files.cnblogs.com/zhangziqiu/jquery.extend-lastest.js

工具函数说明：

方法签名： jQuery.cookie(name, subName, value, options)

方法说明：读取、写入、删除Cookies

方法参数：

名称 说明 举例

name cookies的主键值 读取主键：

$.cookie("singleKey")

写入cookies，值为字符串：

$.cookie("singleKey", "", "singleKey-value", { expires: 1, path: "/", secure: false })

subName 子键名称。在写入时请传递空或者null 读取子键：

$.cookie("multiKey", "subName1")

写入cookies，值为对象：

var subNameObj = { subName1: "aaa", subName2: "bbb", subName3: "ccc" };

$.cookie("multiKey", "", subNameObj, { expires: 1, path: "/", secure: false });

value Cookies值，可以是字符串或者对象。

如果是对象，则将对象的每个属性保存在Cookies子键。 参见上面实例。

options 参数：

expires：可以是数字或者Data类型的对象。

如果传入数字表示几天后过期。

path：路径，默认为域名根目录（“/”）。

secure：是否启用加密，默认为否。

指定过期时间：

var myDate = new Date();

myDate.setFullYear(2009, 10, 10);

$.cookie("singleKey", "", "singleKey-value", { expires: myDate, secure: false })

1天后过期：

var time = Date();

$.cookie("singleKey", "", "singleKey-value", { expires: 1, path: "/", secure: false })

## 主域和子域

主域的cookie，子域可以访问。浏览器在请求子域的页面的时候，会把主域的cookie也一块发送到后台服务器。

主域：http://www.baidu.com/；子域：<http://tieba.baidu.com/>

子域的cookie只能子域访问。子域在添加cookie的时候，可以设置域为主域。

## 路径

当cookie设置路径后，只有该路径下的网址才能获取该cookie。

|  |
| --- |
| Context.Response.Cookies["Date"].Path = "/html/"; |

在设置路径后，只有在根目录下html下的网址在请求时才发送cookie到服务器。

### 路径的使用场景

不设置路径时，所有的请求都会发送cookie，而设置Path与domain可以提高性能，只有该路径下的请求才会发送cookie。

## 使用场景

帮助网站记住使用当前浏览器的用户的相关信息。

1. 记住用户名和密码的话，cookie都会存一个可以，不要直接存用户名和密码。
2. 记录用户的浏览器商品的信息。
3. 广告营销。

## 保存与读取cookie

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  if(!IsPostBack)  {  //设置cookie：  Context.Response.Cookies["Date"].Value = DateTime.Now.ToString();  }else  {  //读取cookie：  Context.Response.Write(Context.Request.Cookies["Date"].Value);  }  } |

还能通过以下方式设置

|  |
| --- |
| HttpCookie aCookie = new HttpCookie("lastVisit");  aCookie.Value = DateTime.Now.ToString();  aCookie.Expires = DateTime.Now.AddDays(1); |

## 多值Cookie

可以在 Cookie 中存储一个值，如用户名和上次访问时间。也可以在一个 Cookie 中存储多个名称/值对。名称/值对称为子键。（二维数组）例如，不要创建两个名为 userName 和 lastVisit 的单独 Cookie，而可以创建一个名为 userInfo 的 Cookie，其中包含两个子键 userName 和 lastVisit。

您可能会出于多种原因来使用子键。首先，将相关或类似的信息放在一个 Cookie 中很方便。此外，由于所有信息都在一个 Cookie 中，所以诸如有效期之类的 Cookie 属性就适用于所有信息。（反之，如果要为不同类型的信息指定不同的到期日期，就应该把信息存储在单独的 Cookie 中。）

带有子键的 Cookie 还可帮助您限制 Cookie 文件的大小。正如前面“Cookie 的限制”一节中所提到的，Cookie 通常限制为 4096 字节，并且每个站点最多可存储 20 个 Cookie。使用带子键的单个 Cookie，使用的 Cookie 数就不会超过分配给站点的 20 个的限制。此外，一个 Cookie 会占用大约 50 个字符的系统开销（用于保存有效期信息等），再加上其中存储的值的长度，其总和接近 4096 字节的限制。如果存储五个子键而不是五个单独的 Cookie，便可节省单独 Cookie 的系统开销，节省大约 200 字节。

若要创建带子键的 Cookie，您可以使用编写单个 Cookie 的各种语法。下面的示例演示用于编写同一 Cookie 的两种方法，其中的每个 Cookie 都带有两个子键：

|  |
| --- |
| Response.Cookies["userInfo"]["userName"] = "patrick"; Response.Cookies["userInfo"]["lastVisit"] = DateTime.Now.ToString(); Response.Cookies["userInfo"].Expires = DateTime.Now.AddDays(1); HttpCookie aCookie = new HttpCookie("userInfo"); aCookie.Values["userName"] = "patrick"; aCookie.Values["lastVisit"] = DateTime.Now.ToString(); aCookie.Expires = DateTime.Now.AddDays(1); Response.Cookies.Add(aCookie); |

### 修改或删除子键

修改单个子键的方法与创建它的方法相同，如下面的示例所示：

|  |
| --- |
| Response.Cookies["userInfo"]["lastVisit"] = DateTime.Now.ToString();  Response.Cookies["userInfo"].Expires = DateTime.Now.AddDays(1); |

若要删除单个子键，可以操作 Cookie 的 Values 集合，该集合用于保存子键。首先通过从 Cookies 对象中获取 Cookie 来重新创建 Cookie。然后您就可以调用 Values 集合的 Remove 方法，将要删除的子键的名称传递给 Remove 方法。接着，将 Cookie 添加到 Cookies 集合，这样 Cookie 便会以修改后的格式发送回浏览器。下面的代码示例演示如何删除子键。在此示例中，要移除的子键的名称在变量中指定。

|  |
| --- |
| string subkeyName;  subkeyName = "userName";  HttpCookie aCookie = Request.Cookies["userInfo"];  aCookie.Values.Remove(subkeyName);  aCookie.Expires = DateTime.Now.AddDays(1);  Response.Cookies.Add(aCookie); |

## 过期时间

默认过期时间为Session。即会话期间有效，也就是关闭浏览器后过期。



设置过期时间

|  |
| --- |
| if (!IsPostBack)  {  //设置cookie：  if (Context.Response.Cookies["Date"].Value == null)  {  Context.Response.Cookies["Date"].Value = DateTime.Now.ToString();  //设置过期时间  Context.Response.Cookies["Date"].Expires = DateTime.Now.AddDays(7);  }  } |

查看cookie



关掉浏览器再打开网站，还能获取到cookie中的值。

## 修改和删除cookie

不能直接修改 Cookie。更改 Cookie 的过程涉及创建一个具有新值的新 Cookie，然后将其发送到浏览器来覆盖客户端上的旧版本 Cookie。下面的代码示例演示如何更改存储用户对站点的访问次数的 Cookie 的值：

|  |
| --- |
| int counter;if (Request.Cookies["counter"] == null) counter = 0;else { counter = int.Parse(Request.Cookies["counter"].Value); } counter++; Response.Cookies["counter"].Value = counter.ToString(); Response.Cookies["counter"].Expires = DateTime.Now.AddDays(1); |

删除 Cookie

删除 Cookie（即从用户的硬盘中物理移除 Cookie）是修改 Cookie 的一种形式。由于 Cookie 在用户的计算机中，因此无法将其直接移除。但是，可以让浏览器来为您删除 Cookie。该技术是创建一个与要删除的 Cookie 同名的新 Cookie，并将该 Cookie 的到期日期设置为早于当前日期的某个日期。当浏览器检查 Cookie 的到期日期时，浏览器便会丢弃这个现已过期的 Cookie。下面的代码示例演示删除应用程序中所有可用 Cookie 的一种方法：

|  |
| --- |
| HttpCookie aCookie;  string cookieName;  int limit = Request.Cookies.Count;  for (int i=0; i<limit; i++)  {  cookieName = Request.Cookies[i].Name;  aCookie = new HttpCookie(cookieName);  aCookie.Expires = DateTime.Now.AddDays(-1);  Response.Cookies.Add(aCookie);  } |

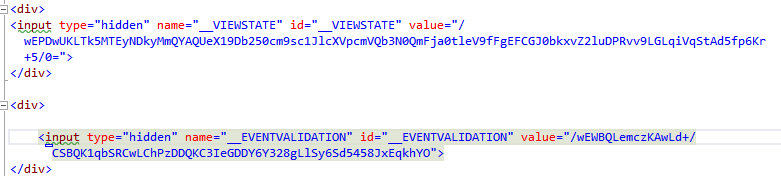
## 案例：Cookie保留登陆信息

### 界面如下：

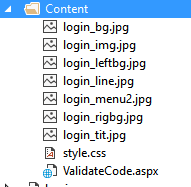


### 界面源代码复制到aspx页面上

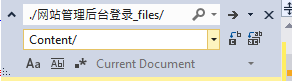
1. 删除隐藏域



1. 创建文件夹并把素材拷贝进去



1. 替换路径



1. 修改form。Aspx必须有一个form的runat=”Server”。



1. 在用户名输入框中展示上次登录的用户名



### 登陆处理

在后置代码中，进行编辑

|  |
| --- |
| public partial class Login : System.Web.UI.Page  {  //保存用户名  public string UName { get; set; }  protected void Page\_Load(object sender, EventArgs e)  {  if (!IsPostBack)  {  if (Request.Cookies["Name"].Value != null)  {  UName = Request.Cookies["Name"].Value;//从cookie中取值  }  }  else  {  string UserName = Request.Form["txtClientID"];  Response.Cookies["Name"].Value = UserName;//设置cookie的值  Response.Cookies["Name"].Expires = DateTime.Now.AddDays(5);  UName = UserName;  }  }  } |

# Session

Session是用于保存用户相关信息的服务端对象。第一次请求过来后，服务器端认为这是一个会话的开始。会话开始后，请求就会从服务器端的session池中个关联一个空闲的session对象（字典数组），服务器通过响应会把session对象的ID（AspNet\_SessionId）写到客户端的Cookie中，以后每次请求都会携带Session的ID，这样在这个会话中的所有请求都与同一个session关联到一起。

## 案例：Session的存取

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  if (!IsPostBack)  {  Context.Session["Time"] = DateTime.Now;//赋值  }  else  {  string t = Session["Time"].ToString();//取值  Response.Write(t);//取值  }  } |

## Session的使用场景

跟会话相关的数据都可以存在session中。在会话期间通过session多次与后台交互。

常用来制作验证码验证。

## 校验验证码

思路：

1. 一般处理程序生成验证码，验证码中生成的数字要写到session中。

|  |
| --- |
| Session["ValidateCode"] = str; |

2. 点击登陆时，用户输入的验证码通过请求报文传到服务器

|  |
| --- |
| string Validatecode = Request.Form["ValidateCode"]; |

3. 在后置代码中判断用户输入的代码是否与session中的一致。如果不一致，弹出提示。

|  |
| --- |
| string AlertValidate;  if (Validatecode != Session["ValidateCode"] as string)  {  AlertValidate = "<Script>alret('验证码错误')</Script>";  } |

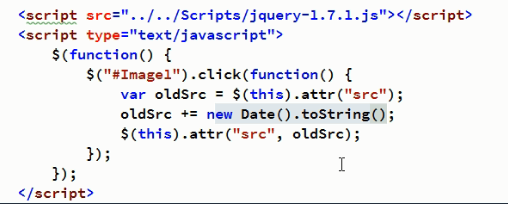
在前端代码最下面添加如下代码：



这样当验证码错误的时候，只有所有页面都加载完成后才会弹出提示。传统的使用Page.RegisterStartupScript方法有可能页面没有加载完就弹出提示，而阻塞页面继续加载。

1. 更换验证码（原理是只要Src地址不一样，浏览器会自动下载该地址的资源）。这里点击验证码图片即可生成新的验证码





## 判断是否登陆

思路

1. 用户登录成功后，把当前用户写入session中。

|  |
| --- |
| Session["UserInfo"] = modelList.FirstOrDefault(); |

2. 在跳转的页面中，查询是否sessiong值是否为空，为空则跳转到登陆页面

|  |
| --- |
| if(Session["UserInfo"] == null)  {  Response.Redirect("Login.aspx");  } |

步骤2中的代码在所有与登陆有关的代码中都需要写，代码重复了。我们可以定义一个BasePage类，所有其他页面继承自该类

|  |
| --- |
| public class BasePage:System.Web.UI.Page  {  //添加一个公共属性，保存用户信息  public Model.Class LoginUser { get; set; }  protected virtual void Page\_Init(object sender, EventArgs e)  {  if(Session["UserInfo"] ==null)  {  Response.Redirect("Login.aspx");  return;  }  //赋值  LoginUser = (Session["UserInfo"] as Model.Class;  }  } |

其他类从继承自System.Web.UI.Page改为继承自BasePage

|  |
| --- |
| public partial class Default :BasePage |

这样当其他页面被请求时，会首先执行Page\_Init方法再执行Page\_Load方法。

用公共属性也可以直接取得当前用户的信息

|  |
| --- |
| string name = LoginUser.Uname;  string age = LoginUser.Age; |

## Bug

校验验证码的时候，可以模拟请求不通过提交按钮，验证图片不会改变，这样就可以进行暴力破解了。

解决方法：请求一次后，session清空或移除。

|  |
| --- |
| Session.Remove("ValidateCode "); |

# 进程外session

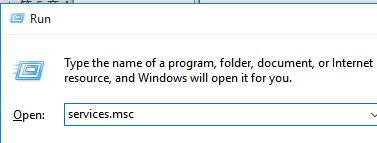
当访问量过大、错误积累到一定程度、进程处理性能下降的时候可能会导致工作进程重启；还能设置应用程序池多长时间回收。这时session就丢失了，造成所有用户下线。.net提供了进程外session来解决session丢失的问题。

进程外session用气来很简单，不需要修改任何代码，只需修改web.config。

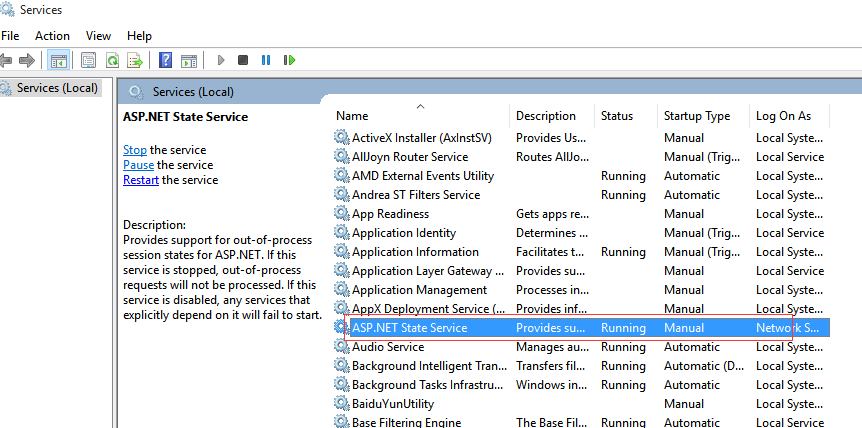
## 状态服务器（StateServer）

第一种方式:把session存到会话服务器中。

1. 打开服务



1. 开启状态服务



1. 配置web.config

|  |
| --- |
| <system.web>  <sessionState mode="StateServer" stateConnectionString="tcpip=localhost:42424">  </sessionState>  </system.web> |

1. 如果状态服务支持远程服务（既其他机器可以在本机上保存session），需要配置注册表

cmd → regedit.exe

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\aspnet\_state\Parameters\AllowRemoteConnection

设置成1

如果是1：那么代表其他机器也可以访问当前的状态服务。如果是0，那么只能我自己访问。

1. 如果会话中保存的是对象，那么对象必须可序列化

|  |
| --- |
| public partial class WebForm1 : System.Web.UI.Page  {  protected void Page\_Load(object sender, EventArgs e)  {  Session["Myclass"] = new Myclass() { Age = 12,Name="hao" };  }  }  //必须可序列化  [Serializable]  public class Myclass  {  public int Age { get; set; }  public string Name { get; set; }  } |

## 放到数据库里

1. 到aspnet\_regsql.exe所在目录下执行

aspnet\_regsql.exe -S 127.0.0.1 -U sa -P 123 -ssadd -sstype c -d SessionTest

1. 配置web.config

|  |
| --- |
| <system.web>  <sessionState mode="SQLServer" allowCustomSqlDatabase="true" sqlConnectionString="Data Source=.;Initial Catalog=SessionTest;uid=sa;pwd=123"></sessionState>  </system.web> |

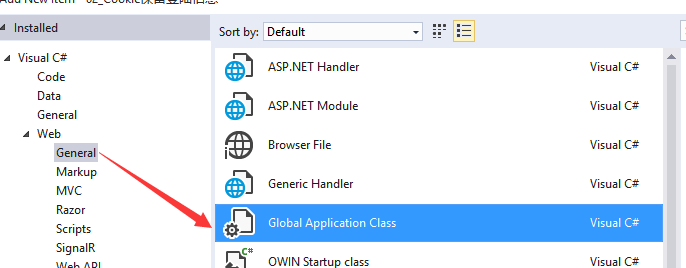
1. 对象必须可序列化

# Application

## 全局应用程序类

从Application这个单词上大致可以看出Application状态是整个应用程序全局的。在ASP时代我们通常会在Application中存储一些公共数据，而ASP.NET中Application的基本意义没有变：在服务器内存中存储数量较少又独立于用户请求的数据。由于它的访问速度非常快而且只要应用程序不停止，数据一直存在，我们通常在Application\_Start的时候去初始化一些数据，在以后的访问中可以迅速访问和检索。

我们可以来实践一下。首先，右键单击网站，选择“添加新项”命令，如下图所示，选择全局应用程序类。



Global.asax（通常我们不改名）是一个用来处理应用程序全局的事件。打开文件，系统已经为我们定义了一些事件的处理方法。

|  |
| --- |
| public class Global : System.Web.HttpApplication  {  // 在应用程序启动时运行的代码  protected void Application\_Start(object sender, EventArgs e)  {  }  //在新会话启动时运行的代码  protected void Session\_Start(object sender, EventArgs e)  {  }  //  protected void Application\_BeginRequest(object sender, EventArgs e)  {  }  protected void Application\_AuthenticateRequest(object sender, EventArgs e)  {  }  //在出现未处理的错误时运行的代码  protected void Application\_Error(object sender, EventArgs e)  {  }  //// 在会话结束时运行的代码。需要注意的是: 只有在 Web.config 文件中的 sessionstate 模式设置为InProc 时，才会引发 Session\_End 事件，如果会话模式设置为 StateServer 或 SQLServer，则不会引发该事件。  protected void Session\_End(object sender, EventArgs e)  {  }  //在应用程序关闭时运行的代码  protected void Application\_End(object sender, EventArgs e)  {  }  } |

通过这些注释我们可以看到，这些事件是整个应用程序的事件，和某一个页面没有关系。

## 使用Application统计网站访问

假设我们希望使用Application统计网站的访问情况。

· 页面单击数。页面被单击一次+1，不管是否是同一个用户多次单击页面。

· 用户访问数。来了一个用户+1，一个用户打开多个页面不会影响这个数字。

我们首先需要在Application\_Start中去初始化两个变量。

|  |
| --- |
| protected void Application\_Start(object sender, EventArgs e)  {  Application["PageClick"] = 0;  Application["UserVisit"] = 0;  } |

用户访问数根据Session来判断，因此可以在Session\_Start的时候去增加这个变量：

|  |
| --- |
| protected void Session\_Start(object sender, EventArgs e)  {  Application.Lock();  Application["UserVisit"] = (int)Application["UserVisit"] + 1;  Application.UnLock();  } |

我们看到，Application的使用方法和Session差不多。唯一要注意的是，Application的作用范围是整个应用程序，可能有很多用户在同一个时间访问Application造成并发混乱，因此在修改Application的时候需要先锁定Application，修改完成后再解锁。

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  if (!IsPostBack)  {  Application.Lock();  Application["PageClick"] = (int)Application["PageClick"] + 1;  Application.UnLock();  Response.Write(string.Format("页面单击数：{0}<br/>", Application["PageClick"]));  Response.Write(string.Format("用户访问数：{0}<br/>", Application["UserVisit"]));  }  } |

## Application总结

在ASP.NET 2.0中，Application已经变得不是非常重要了。因为Application的自我管理功能非常薄弱，它没有类似Session的超时机制。也就是说，Application中的数据只有通过手动删除或者修改才能释放内存，只要应用程序不停止，Application中的内容就不会消失。在下一节中，我们会看到，可以使用Cache实现类似Application的功能，同时Cache又有丰富而强大的自我管理机制。

总结一下Application的特性。

· 存储的物理位置。服务器内存。

· 存储的类型限制。任意类型。

· 状态使用的范围。整个应用程序。

· 存储的大小限制。任意大小。

· 生命周期。应用程序开始的时候创建（准确来说是用户第一次请求某URL的时候创建），应用程序结束的时候销毁。

· 安全与性能。数据总是存储在服务端，安全性比较高，但不易存储过多数据。

· 优缺点与注意事项。检索数据速度快，但缺乏自我管理机制，数据不会自动释放。

# 缓存

把数据放到内存里面，下次请求的时候，直接从内存读取数据返回给客户端就可以了，不用再去访问数据库或者磁盘。用内存空间换磁盘读的时间。

例如， 1秒钟来10000个请求查看当前新闻列表页面。 新闻列表10秒内没有变化第一个请求过来，跟数据交互一次，把新闻数据放到了一个全局变量。后续9999次请求，直接从全局变量里面拿到新闻的数据。

## 缓存的使用场景

短时间内访问量比较大，数据基本不变化，变化也不怎么影响就可以使用了。

## 页面缓存

把页面对象存到服务器内存中。第一次请求后，后续请求在一定时间内访问这个页面直接从内存中拿对象，不用从服务器中获取数据。

给页面添加<%@ OutputCache Duration=“15” VaryByParam=“\*”%>标签就可以启用页面缓存，这样整个页面的内容都会被缓存，页面中的ASP.Net代码、数据源在缓存期间都不会被运行，而是直接输出缓存的页面内容。 Duration表示缓存时间，以秒为单位，超过这个时间则缓存失效，再次生成以后会再缓存15秒，一般只有看帖、看新闻、看视频的页面才缓存，CRUD的页面没必要缓存。

缓存是针对所有这个页面的访问者。这样1个访问者和1万个访问者、一次访问和100万次访问对数据库的压力是一样的。

### 案例



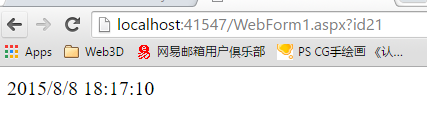
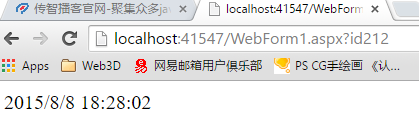
在上面的代码中，在10没内，如果数据没有变化，其他请求都是从内存中拿到数据。当后置代码输出时间时，10秒内没有变化。

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  this.Label1.Text = DateTime.Now.ToString();  } |

### VaryByParam

当值为“\*”时，只要有请求后任何变化就不会从缓存中读取数据。

例如，用上面的代码我们在请求时为其加上不同的get请求，测试发现每次数据都会不一样

当值为“none”时，不论请求怎么变化都从缓存中取得数据。

单独缓存。设置VaryByParam=“id”，表示对于相同页面的不同的id参数进行单独缓存。如果有多个确定缓存的参数，则将参数名用分号隔开即可，比如VaryByParam=“id;number”。

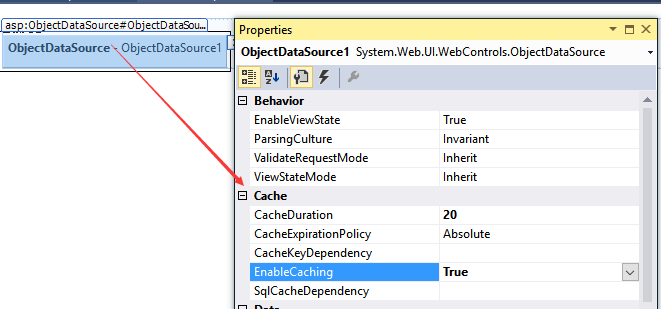
如下面的请求，当设置为“id”时，不论name如何变化都是从缓存中读取，但是id变化时，将不再从缓存中拿数据。

|  |
| --- |
| http://localhost:41547/WebForm1.aspx?id=2&name=jwew |

## 数据源缓存

不想对整个页面进行缓存，只缓存数据库中的数据时，可以使用数据源缓存。

只需要在objectDataSource的cache中设置即可。



设定ObjectDataSource的CacheDuration（缓存时间：秒），EnableCaching=true。这样每隔CacheDuration指定的时间段才调用SelectMethod指定的方法来执行数据库查询，其他时候都是直接返回缓存的数据。取数据的过程缓存，在缓存期间，绑定控件向ObjectDataSource要数据， ObjectDataSource直接将缓存的数据返回给控件，不再去向TypeName指向的类要数据

## 自定义缓存

### 永不过期

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  if (Cache["NowT"] == null)  {  //设置缓存。永久有效  Cache["NowT"] = DateTime.Now.ToString();  }  Response.Write(Cache["NowT"]);  } |

当其他的请求访问该网页时获得的是同一个值。

### 有过期时间的缓存

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  if (Cache["NowT"] == null)  {  //10秒后过期  Cache.Insert("NowT", DateTime.Now.ToString(), null, DateTime.Now.AddSeconds(10), TimeSpan.Zero);  }  Response.Write(Cache["NowT"]);  } |

### 滑动过期时间

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  if (Cache["NowT"] == null)  {  //10秒内无访问过期，如果一直有访问，则过期时间一直为没有访问后10秒过期  Cache.Insert("NowT", DateTime.Now.ToString(), null, DateTime.MaxValue, new TimeSpan(0,0,10));  }  Response.Write(Cache["NowT"]);  } |

### 依赖项的缓存

#### 依赖文件的缓存

缓存数据与文件相关，则可以用依赖文件的缓存。文件内容没有改变，缓存不会过期。

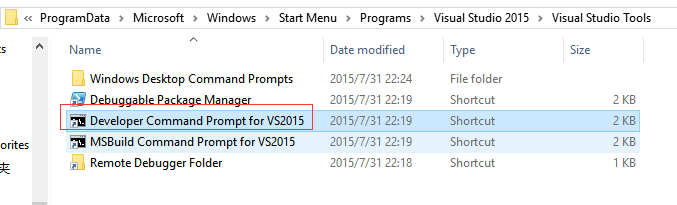
|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  string file = Request.MapPath("depend.txt");  if (Cache["NowT"] == null)  {  //10秒内无访问过期，如果一直有访问，则过期时间一直为没有访问后10秒过期  Cache.Insert("NowT", DateTime.Now.ToString(),new System.Web.Caching.CacheDependency(file));  }  Response.Write(Cache["NowT"]);  } |

#### 依赖数据库表的缓存

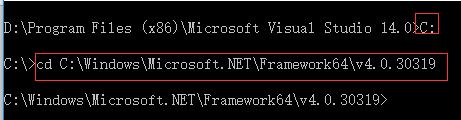
数据库中的表不改变，缓存不会过期。

##### 开启数据库缓存依赖

打开命令行提示工具



进入aspnet\_regsql所在目录

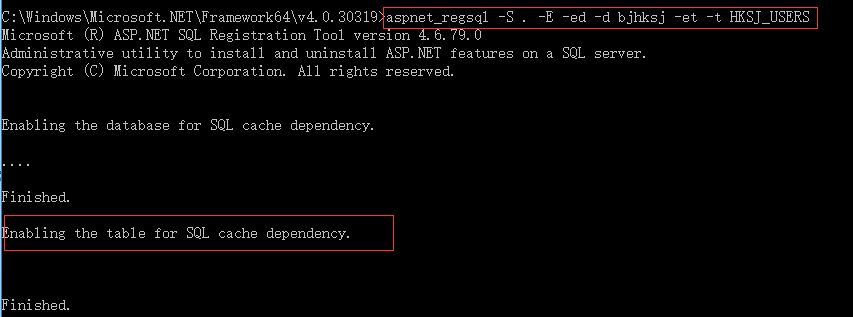


注册列表命令

|  |
| --- |
| aspnet\_regsql -S . -E -ed -d 数据库名 -et -t 表名 |
| -S服务器名称 -E集成身份验证 -ed启动 -d数据库名称 -et指定缓冲依赖的表名 -t表名 |

例如:

|  |
| --- |
| aspnet\_regsql -S . -E -ed -d bjhksj -et -t HKSJ\_USERS |



删除：aspnet\_regsql -S . -E -d 数据库名 -dt -t 表名  
取消数据库缓存依赖: aspnet\_regsql -S . -E -dd 数据库名  
数据库名 列出已注册表：aspnet\_regsql -S . -E -d 数据库名 –lt

##### 依赖于数据库的web.config配置

|  |
| --- |
| <configuration>  <connectionStrings>  <add connectionString="userInfo" name="server=.;database=bjhksj;uid=sa;password=123" />  </connectionStrings>  <system.web>  <!-------------------------数据库缓存依赖------------------------------>  <caching>  <sqlCacheDependency enabled="true">  <databases>  <!--pollTime为每隔多少毫秒向数据库中查询数据是否更新了-->  <add name="hk" connectionStringName="userInfo" pollTime="500"/>  </databases>  </sqlCacheDependency>  </caching>  <!-------------------------数据库缓存依赖------------------------------>  <compilation debug="true" targetFramework="4.5.2"/>  <httpRuntime targetFramework="4.5.2"/>  </system.web> |

##### 创建依赖数据库表的缓存

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  string file = Request.MapPath("depend.txt");  if (Cache["NowT"] == null)  {  //数据库依赖。SqlCacheDependency的第一个参数是web.config中设置的数据库的entryname。第二个参数是依赖的数据库表  Cache.Insert("NowT", DateTime.Now.ToString(), new SqlCacheDependency("hk", "HKSJ\_USERS"));  }  Response.Write(Cache["NowT"]);  } |

## 实例：自定义缓存的应用方式



# 全局应用程序

全局应用程序

## 日志处理

当程序出现错误是，会触发全局处理程序（Global.asax）中的Application\_Error方法。错误消息的处理可以在这里进行捕获。

### 错误写到日志文件中

|  |
| --- |
| protected void Application\_Error(object sender, EventArgs e)  {  string filename = Path.Combine("Log", DateTime.Now.ToString("yyyy-MM--dd") + ".txt");  string absolutefilename = Server.MapPath(filename);  var str = Server.GetLastError().ToString() + "\r\n";  if (!File.Exists(absolutefilename))  {  File.WriteAllText(absolutefilename, str);  }  else  {  File.AppendAllText(absolutefilename, str);  }  } |

当同时又多个用户访问网站出现错误时，7.1节中的处理方式就不合适了。因为有可能出现多个用户同时写入日志。

解决方法是上锁

### 防止多个用户同时写入日志

|  |
| --- |
| protected void Application\_Error(object sender, EventArgs e)  {  string filename = Path.Combine("Log", DateTime.Now.ToString("yyyy-MM--dd") + ".txt");  string absolutefilename = Server.MapPath(filename);  var str = Server.GetLastError().ToString() + "\r\n";  lock (lockobj)  {  using (FileStream fs = new FileStream(absolutefilename, FileMode.Append, FileAccess.Write))  {  byte[] data = Encoding.Default.GetBytes(str);  fs.Write(data, 0, data.Length);  }  }  } |

上锁后，只能处理1个用户的错误信息，其他用户都在等待，而文件读写速度比较慢，有的错误来不及处理，用户就等不及把网站关闭，错误也就捕获不到了。

### 用队列与多线程可以实现错误信息的快速处理

当错误消息过来时，把它放到错误消息队列中。

#### 新建信息处理类

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Text;  using System.Threading;  using System.Web;  namespace WebDemo.Common  {  public class LogHelper  {  public static Queue<string> ExcpetionInfoQueue = new Queue<string>();  public static string LogBasePath;  static LogHelper()  {  ThreadPool.QueueUserWorkItem(o =>  {  while (true)  {  if (ExcpetionInfoQueue.Count > 0)  {  string str = ExcpetionInfoQueue.Dequeue();  //写入错误消息。  string strFileName = DateTime.Now.ToString(@"yyyy-MM-dd") + ".txt";  string absoluteFileName = Path.Combine(LogBasePath, strFileName);  lock (ExcpetionInfoQueue)  {  using (FileStream fs = new FileStream(absoluteFileName, FileMode.Append, FileAccess.Write))  {  byte[] data = Encoding.Default.GetBytes(str);  fs.Write(data, 0, data.Length);  }  }  }  }  });  }  }  } |

#### 错误来时，放入队列

|  |
| --- |
| protected void Application\_Error(Object sender, EventArgs e)  {  if (string.IsNullOrEmpty(Common.LogHelper.LogBasePath))  {  Common.LogHelper.LogBasePath = Request.MapPath("/Log/");  } Common.LogHelper.ExcpetionInfoQueue.Enqueue(Server.GetLastError().ToString());  //整个网站中出息了未捕获的异常，一般就是跳转到一个错误页面，  //提醒错误，然后 隔5s跳回首页。  Response.Redirect("/Default.aspx");  } |

## 防盗链

防盗链放到Application\_BeginRequest方法中。Application\_BeginRequest是管道中的第一个事件，越早处理，越节省服务器资源。

|  |
| --- |
| void Application\_BeginRequest(object sender, EventArgs e)  {  if (Request.RawUrl.Contains("images/"))  {  if (Request.UrlReferrer == null || !IsSameDomain(Request.UrlReferrer, Request.Url))  {  ////  Response.ContentType = "image/jpeg";  string path = Request.MapPath("~/daolian.jpg");  Response.WriteFile(path);  //结束请求  Response.End();  }  }  }  //判断两个域名是否相等  bool IsSameDomain(Uri u1,Uri u2)  {  return Uri.Compare(u1, u2, UriComponents.HostAndPort, UriFormat.SafeUnescaped, StringComparison.CurrentCultureIgnoreCase) == 0 ? true : false;  } |

## 伪静态

在Application\_BeginRequest中可以进行伪静态处理。给客户的是一个静态页面，请求过来后，转换为动态的页面。

|  |
| --- |
| //管道当中的第一个事件  protected void Application\_BeginRequest(Object sender, EventArgs e)  {  //原来的请求，比如:127.0.0.1:80//a.html  string str = Request.Url.ToString();  //经过处理：用正则表达式等，变为动态网页：127.0.0.1:80//a.aspx  string str2 = str.Replace("htm", aspx);  //重指向  Context.RewritePath(str2);  } |

# 全局模块化处理程序

相对全局处理程序，全局模块化处理程序不需要重新编译，只设置web.config即可实现功能的切换。

## HttpModule

ttpModule是向实现类提供模块初始化和处置事件。当一个HTTP请求到达HttpModule时，整个ASP.NET Framework系统还并没有对这个HTTP请求做任何处理，也就是说此时对于HTTP请求来讲，HttpModule是一个HTTP请求的“必经之路”，所以可以在这个HTTP请求传递到真正的请求处理中心（HttpHandler）之前附加一些需要的信息在这个HTTP请求信息之上，或者针对截获的这个HTTP请求信息作一些额外的工作，或者在某些情况下干脆终止满足一些条件的HTTP请求，从而可以起到一个Filter过滤器的作用。

### 案例：为所有网站附加信息

1. 新建一个类，实现了IHttpModule接口。

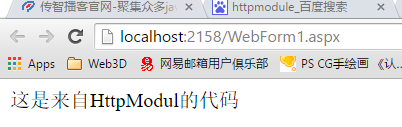
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  namespace WebApplication2  {  public class MyHttpModule : IHttpModule  {  public void Dispose()  {  throw new NotImplementedException();  }  public void Init(HttpApplication context)  {  //注册了HttpApplication第一个事件。  context.BeginRequest += context\_BeginRequest;  }  private void context\_BeginRequest(object sender, EventArgs e)  {  HttpApplication app = sender as HttpApplication;  app.Context.Response.Write("这是来自HttpModul的代码");  }  }  } |

1. 设置web.config

|  |
| --- |
| <system.webServer>  <modules>  <add name="DemoModule"  type="WebApplication2.MyHttpModule,WebApplication2"/>  </modules>  </system.webServer> |

Type中第一个字符串是实现了IHttpModule的类的全名，第二个值是程序集的名称。

1. 在任何网页中打开，都会输出下面一段信息



1. 注释掉web.config中的配置，既不会调用，从而实现不修改代码切换功能。

## HttpHandlers

可以对特定的请求方法，特定的文件进行处理。

1. 在web.config中设置

|  |
| --- |
| <system.webServer>  <handlers>  <add name="handlerDemo" verb="\*" path="\*.txt" type="WebApplication2.handlerDemo,WebApplication2" resourceType="Unspecified" />  </handlers>  </system.webServer> |

Verb表示特定的请求方式。如post，get。\*号表示支持所有类型的请求。

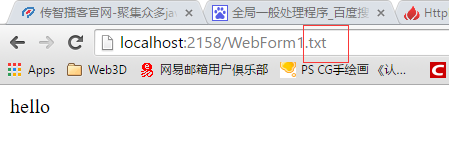
“\*.txt“表示任何后缀为txt的文件都交给type中第一个值所表示的一般处理程序处理。

Type：第一个值是一般处理程序的全名，第二个值是程序集名。

1. 一般处理程序

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  namespace WebApplication2  {  /// <summary>  /// Summary description for handlerDemo  /// </summary>  public class handlerDemo : IHttpHandler  {  public void ProcessRequest(HttpContext context)  {  //对特殊后缀的文件进行过滤，判断是否合法等。之后输出  context.Response.Write("hello");  }  public bool IsReusable  {  get  {  return false;  }  }  }  } |

1. 所有后缀为.txt的访问都会交给一般处理程序来处理

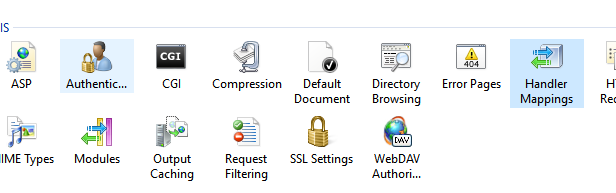


Handler可以用来进行防盗链等处理。

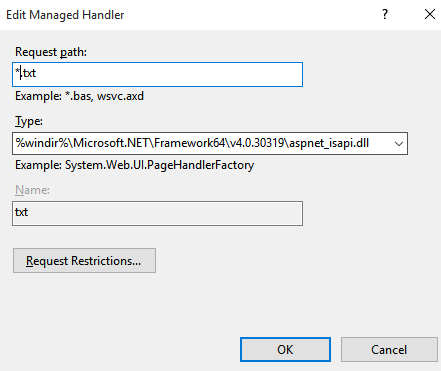
如果是非iis支持的文件类型，需要添加处理程序映射。

### 处理程序映射

1. 在iis中双击处理程序映射



1. 添加程序映射



这样txt后缀文件即可交给一般处理程序处理

# 自定义错误页

当页面发生错误的时候，ASP.Net会将错误信息展示出来（Sqlconnection的错误就能暴露连接字符串），这样一来不好看，二来会泄露网站的内部实现信息，给网站带来安全隐患，因此需要定制错误页，发生错误时显示开发人员定制的页面。404页面放点广告也是好的嘛。

配置web.config，配置customErrors区域：

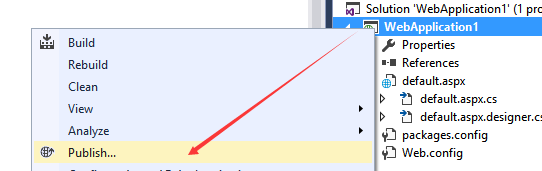
|  |
| --- |
| <system.web>  <customErrors mode="On" defaultRedirect="MyErrorPage.aspx">  <error statusCode="403" redirect="NoAccess.htm" />  <error statusCode="404" redirect="FileNotFound.htm" />  </customErrors>  </system.web> |

mode三个可选值：On：总是显示定制错误页面；Off：不显示定制错误界面，直接显示调用堆栈等异常信息；remoteonly：对于本机的访问显示调用堆栈等异常信息，对于外部用户的显示定制错误页面。一般设置为RemoteOnly，这样发生错误的话，管理员可以在服务器的浏览器中看详细错误信息，普通用户看不到。学习演示的时候mode设置为On，否则看不到定制页

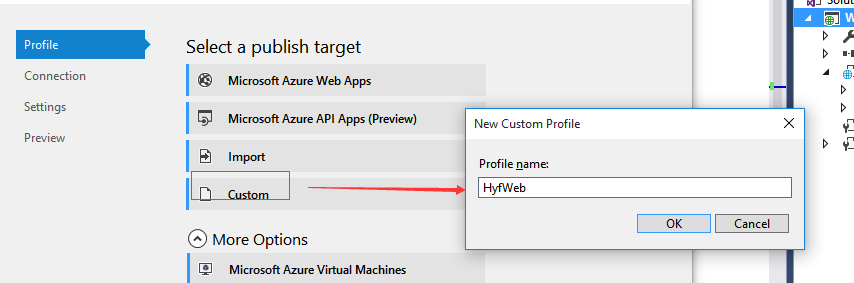
# 部署

## 发布网站

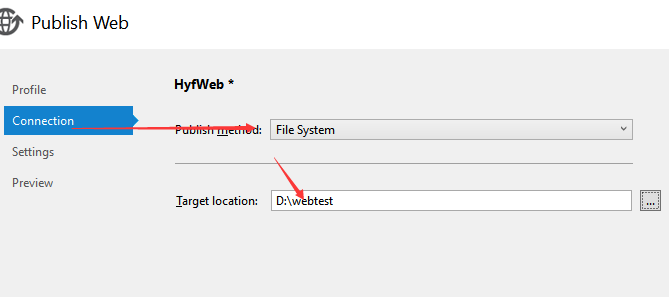
1. 在应用程序上右击选择发布



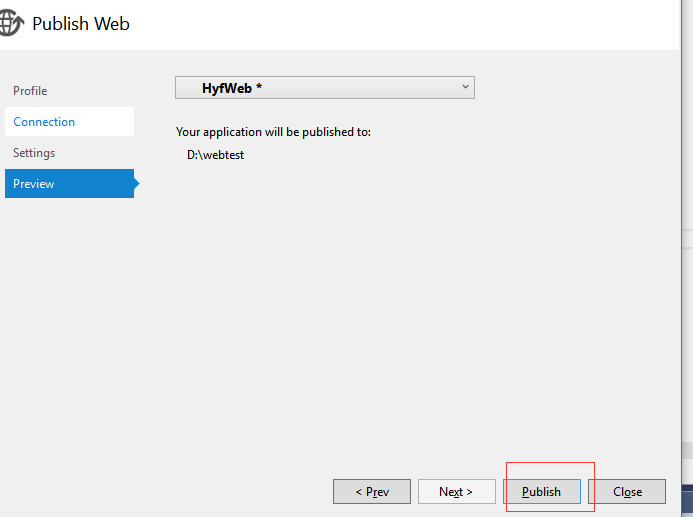
1. 新建profile name（简介）



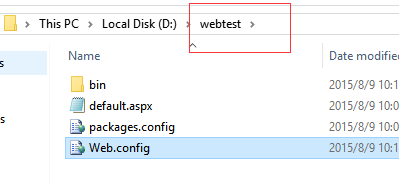
1. 创建发布方式，这里我们选择文件系统



1. 点发布

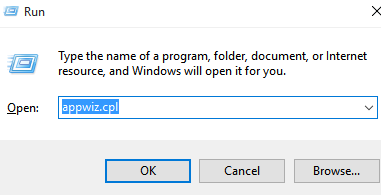


1. 发布成功

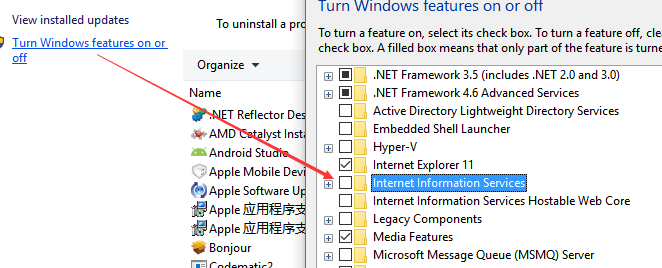


## IIS安装

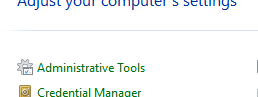
1. 启动“程序与功能”



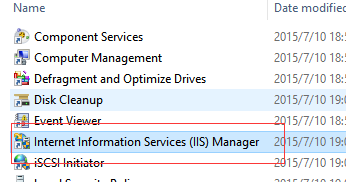
1. 开启internet information services服务



1. 在控制面板中找到管理工具

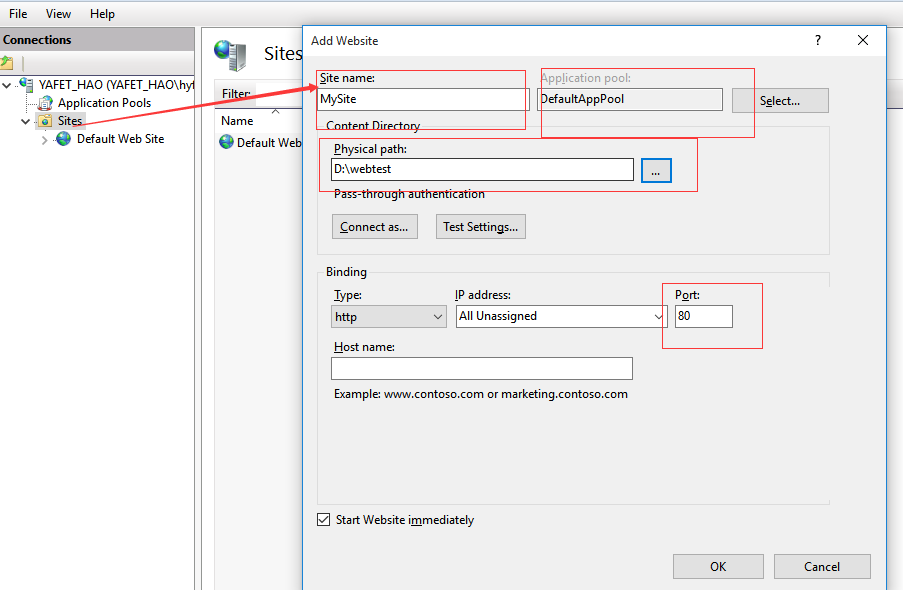


1. 启动iis管理器



## 部署网站

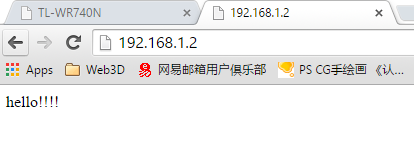
1. 创建网站



1. 设为默认网站



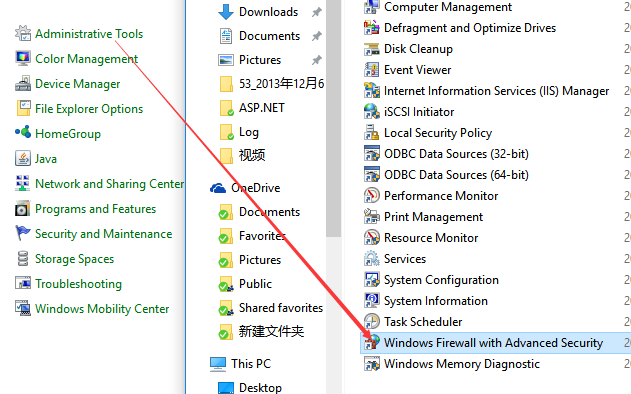
1. 访问网站



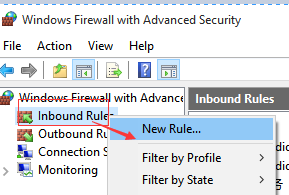
## 开启80端口

如果80端口关闭，可以在防火墙中进行设置。

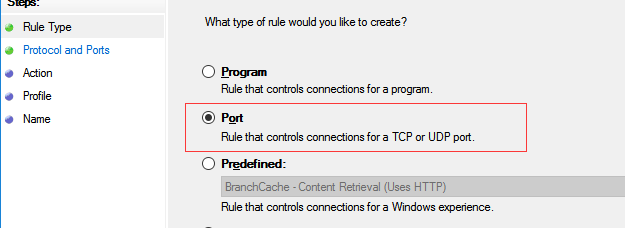
1. 在控制面板中，通过管理工具打开防火墙设置



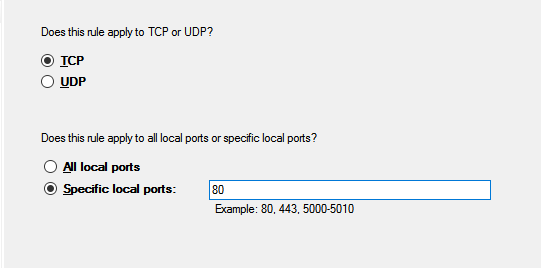
1. 在入站规则上右击创建新规则



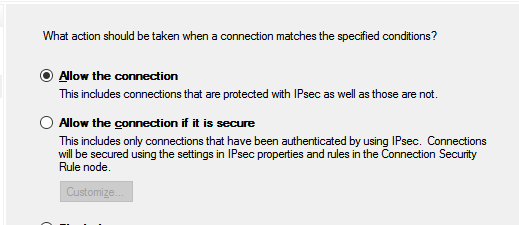
1. 选择port



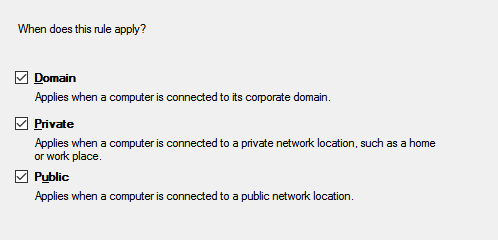
1. 设定开放的端口



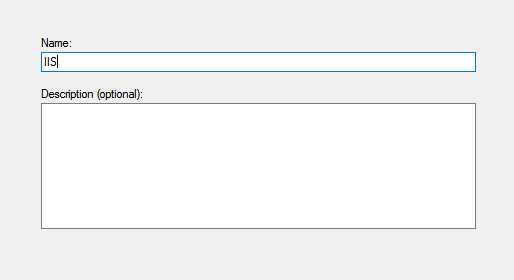
1. 运行连接



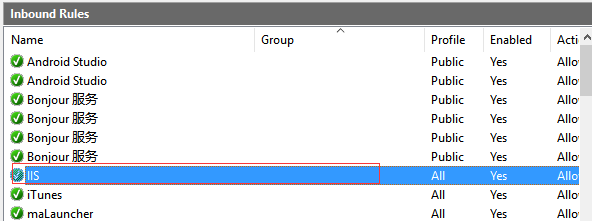
1. 选择全部



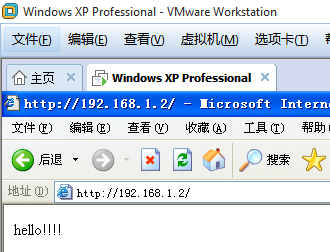
1. 自定义名称



1. 完成



1. 在其他计算机上访问



## 动态域名解析

使用花生壳、nat123、dnspod可以进行动态域名解析，这里用nat123.



通过hyfdbd.123nat.com可以进行外网访问。