**[分布式中使用Redis实现Session共享)](http://www.cnblogs.com/yanweidie/p/4678095.html)**

　　上一篇介绍了一些redis的安装及使用步骤，本篇开始将介绍redis的实际应用场景,先从最常见的session开始，刚好也重新学习一遍session的实现原理。在阅读之前假设你已经会使用[nginx+iis实现负载均衡](http://www.cnblogs.com/yanweidie/p/4658136.html)搭建负载均衡站点了，这里我们会搭建两个站点来验证redis实现的session是否能共享。

**阅读目录**

* [Session实现原理](http://www.cnblogs.com/yanweidie/p/4678095.html#_label0)
* [session共享实现方案](http://www.cnblogs.com/yanweidie/p/4678095.html#_label1)
* [问题拓展](http://www.cnblogs.com/yanweidie/p/4678095.html#_label2)
* [总结](http://www.cnblogs.com/yanweidie/p/4678095.html#_label3)

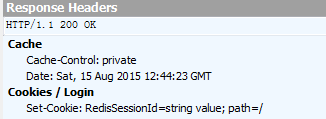
**Session实现原理**

　　session和cookie是我们做web开发中常用到的两个对象，它们之间会不会有联系呢？

**Cookie是什么？** 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。（**Cookie 会随每次HTTP请求一起被传递服务器端，排除js，css，image等静态文件，这个过程可以从fiddler或者ie自带的网络监控里面分析到，考虑性能的化可以从尽量减少cookie着手）**

**Cookie写入浏览器的过程**：我们可以使用如下代码在Asp.net项目中写一个Cookie 并发送到客户端的浏览器（为了简单我没有设置其它属性）。

HttpCookie cookie = new HttpCookie("RedisSessionId", "string value");Response.Cookies.Add(cookie);

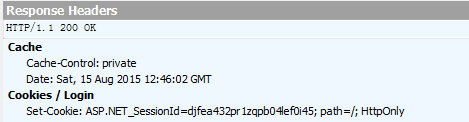


我们可以看到在服务器写的cookie，会通过响应头Set-Cookie的方式写入到浏览器。

**Session是什么？**Session我们可以使用它来方便地在服务端保存一些与会话相关的信息。比如常见的登录信息。

**Session实现原理？ HTTP协议是无状态的，对于一个浏览器发出的多次请求，WEB服务器无法区分 是不是来源于同一个浏览器。**所以服务器为了区分这个过程会通过一个sessionid来区分请求，而这个sessionid是怎么发送给服务端的呢？前面说了cookie会随每次请求发送到服务端，并且cookie相对用户是不可见的，用来保存这个sessionid是最好不过了，我们通过下面过程来验证一下。

Session["UserId"] = 123;



   通过上图再次验证了session和cookie的关系，服务器产生了一次设置cookie的操作，这里的sessionid就是用来区分浏览器的。为了实验是区分浏览器的，可以实验在IE下进行登录，然后在用chrome打开相同页面，你会发现在chrome还是需要你登录的，原因是chrome这时没有sessionid。httpOnly是表示这个cookie是不会在浏览器端通过js进行操作的，防止人为串改sessionid。

  asp.net默认的sessionid的键值是ASP.NET\_SessionId,可以在web.config里面修改这个默认配置

<sessionState mode="InProc" cookieName="MySessionId"></sessionState>

**服务器端Session读取？** 服务器端是怎么读取session的值呢 ，Session["键值"]。那么问题来了，为什么在Defaule.aspx.cs文件里可以获取到这个Session对象，这个Session对象又是什么时候被初始化的呢。

  为了弄清楚这个问题，我们可以通过转到定义的方式来查看。

   System.Web.UI.Page ->HttpSessionState(Session)

[复制代码](javascript:void(0);)

protected internal override HttpContext Context {

[System.Runtime.TargetedPatchingOptOut("Performance critical to inline across NGen image boundaries")]

　　get {

if (\_context == null) {

\_context = HttpContext.Current;

}

return \_context;

}

}

public virtual HttpSessionState Session {

get {

if (!\_sessionRetrieved) {

/\* try just once to retrieve it \*/

\_sessionRetrieved = true;

try {

\_session = Context.Session;

}

catch {

// Just ignore exceptions, return null.

}

}

if (\_session == null) {

throw new HttpException(SR.GetString(SR.Session\_not\_enabled));

}

return \_session;

}

}

[复制代码](javascript:void(0);)

　　上面这一段是Page对象初始化Session对象的，可以看到Session的值来源于HttpContext.Current，而HttpContext.Current又是什么时候被初始化的呢，我们接着往下看。

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

public sealed class HttpContext : IServiceProvider, IPrincipalContainer

{

internal static readonly Assembly SystemWebAssembly = typeof(HttpContext).Assembly;

private static volatile bool s\_eurlSet;

private static string s\_eurl;

private IHttpAsyncHandler \_asyncAppHandler; // application as handler (not always HttpApplication)

private AsyncPreloadModeFlags \_asyncPreloadModeFlags;

private bool \_asyncPreloadModeFlagsSet;

private HttpApplication \_appInstance;

private IHttpHandler \_handler;

[DoNotReset]

private HttpRequest \_request;

private HttpResponse \_response;

private HttpServerUtility \_server;

private Stack \_traceContextStack;

private TraceContext \_topTraceContext;

[DoNotReset]

private Hashtable \_items;

private ArrayList \_errors;

private Exception \_tempError;

private bool \_errorCleared;

[DoNotReset]

private IPrincipalContainer \_principalContainer;

[DoNotReset]

internal ProfileBase \_Profile;

[DoNotReset]

private DateTime \_utcTimestamp;

[DoNotReset]

private HttpWorkerRequest \_wr;

private VirtualPath \_configurationPath;

internal bool \_skipAuthorization;

[DoNotReset]

private CultureInfo \_dynamicCulture;

[DoNotReset]

private CultureInfo \_dynamicUICulture;

private int \_serverExecuteDepth;

private Stack \_handlerStack;

private bool \_preventPostback;

private bool \_runtimeErrorReported;

private PageInstrumentationService \_pageInstrumentationService = null;

private ReadOnlyCollection<string> \_webSocketRequestedProtocols;

｝

View Code

     我这里只贴出了一部分源码，HttpContext包含了我们常用的Request,Response等对象。HttpContext得从ASP.NET管道说起，以IIS 6.0为例，在工作进程w3wp.exe中，利用Aspnet\_ispai.dll加载.NET运行时（如果.NET运行时尚未加载）。IIS 6.0引入了应用程序池的概念，一个工作进程对应着一个应用程序池。一个应用程序池可以承载一个或多个Web应用，每个Web应用映射到一个IIS虚拟目录。与IIS 5.x一样，每一个Web应用运行在各自的应用程序域中。如果HTTP.SYS接收到的HTTP请求是对该Web应用的第一次访问，在成功加载了运行时后，会通过AppDomainFactory为该Web应用创建一个应用程序域（AppDomain）。随后，一个特殊的运行时IsapiRuntime被加载。IsapiRuntime定义在程序集System.Web中，对应的命名空间为System.Web.Hosting。IsapiRuntime会接管该HTTP请求。IsapiRuntime会首先创建一个IsapiWorkerRequest对象，用于封装当前的HTTP请求，并将该IsapiWorkerRequest对象传递给ASP.NET运行时：HttpRuntime，从此时起，HTTP请求正式进入了ASP.NET管道。根据IsapiWorkerRequest对象，HttpRuntime会创建用于表示当前HTTP请求的上下文（Context）对象：HttpContext。  
     至此相信大家对Session初始化过程，session和cookie的关系已经很了解了吧，下面开始进行Session共享实现方案。

**Session共享实现方案**

**一.StateServer方式**

　　这种是asp.net提供的一种方式，还有一种是SQLServer方式(不一定程序使用的是SQLServer数据库，所以通用性不高，这里就不介绍了)。也就是将会话数据存储到单独的内存缓冲区中，再由单独一台机器上运行的Windows服务来控制这个缓冲区。状态服务全称是“ASP.NET State Service ”（aspnet\_state.exe）。它由Web.config文件中的stateConnectionString属性来配置。该属性指定了服务所在的服务器，以及要监视的端口。

<sessionState mode="StateServer" stateConnectionString="tcpip=127.0.0.1:42424" cookieless="false" timeout="20" />

  在这个例子中，状态服务在当前机器的42424端口（默认端口）运行。要在服务器上改变端口和开启远程服务器的该功能，可编辑HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\services\aspnet\_state\Parameters注册表项中的Port值和AllowRemoteConnection修改成1。 显然，使用状态服务的优点在于进程隔离，并可在多站点中共享。 使用这种模式，会话状态的存储将不依赖于iis进程的失败或者重启，然而，一旦状态服务中止，所有会话数据都会丢失（这个问题redis不会存在，重新了数据不会丢失）。

  这里提供一段bat文件帮助修改注册表，可以复制保存为.bat文件执行

[复制代码](javascript:void(0);)

reg add "HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\services\aspnet\_state\Parameters" /v "AllowRemoteConnection" /t REG\_DWORD /d 1 /f

reg add "HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\services\aspnet\_state\Parameters" /v "Port" /t REG\_DWORD /d 42424 /f

net stop aspnet\_state

net start aspnet\_state

pause

[复制代码](javascript:void(0);)

http://images0.cnblogs.com/blog2015/336693/201508/152151544427606.png

   完成这些配置以后还是不能实现共享,虽然站点间的SessionId是一致的，但只有一个站点能够读取的到值，而其它站点读取不到。下面给出解决方案，在Global文件里面添加下面代码

[复制代码](javascript:void(0);)

public override void Init()

{

base.Init();

foreach (string moduleName in this.Modules)

{

string appName = "APPNAME";

IHttpModule module = this.Modules[moduleName];

SessionStateModule ssm = module as SessionStateModule;

if (ssm != null)

{

FieldInfo storeInfo = typeof(SessionStateModule).GetField("\_store", BindingFlags.Instance | BindingFlags.NonPublic);

FieldInfo configMode = typeof(SessionStateModule).GetField("s\_configMode", BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Static);

SessionStateMode mode = (SessionStateMode)configMode.GetValue(ssm);

if (mode == SessionStateMode.StateServer)

{

SessionStateStoreProviderBase store = (SessionStateStoreProviderBase)storeInfo.GetValue(ssm);

if (store == null)//In IIS7 Integrated mode, module.Init() is called later

{

FieldInfo runtimeInfo = typeof(HttpRuntime).GetField("\_theRuntime", BindingFlags.Static | BindingFlags.NonPublic);

HttpRuntime theRuntime = (HttpRuntime)runtimeInfo.GetValue(null);

FieldInfo appNameInfo = typeof(HttpRuntime).GetField("\_appDomainAppId", BindingFlags.Instance | BindingFlags.NonPublic);

appNameInfo.SetValue(theRuntime, appName);

}

else

{

Type storeType = store.GetType();

if (storeType.Name.Equals("OutOfProcSessionStateStore"))

{

FieldInfo uribaseInfo = storeType.GetField("s\_uribase", BindingFlags.Static | BindingFlags.NonPublic);

uribaseInfo.SetValue(storeType, appName);

object obj = null;

uribaseInfo.GetValue(obj);

}

}

}

break;

}

}

}

[复制代码](javascript:void(0);)

**二.redis实现session共享**

   下面我们将使用redis来实现共享，首先要弄清楚session的几个关键点,过期时间，SessionId，一个SessionId里面会存在多组key/value数据。基于这个特性我将采用Hash结构来存储，看看代码实现。用到了上一篇提供的RedisBase帮助类。

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.SessionState;

using ServiceStack.Redis;

using Com.Redis;

namespace ResidSessionDemo.RedisDemo

{

public class RedisSession

{

private HttpContext context;

public RedisSession(HttpContext context, bool IsReadOnly, int Timeout)

{

this.context = context;

this.IsReadOnly = IsReadOnly;

this.Timeout = Timeout;

//更新缓存过期时间

RedisBase.Hash\_SetExpire(SessionID, DateTime.Now.AddMinutes(Timeout));

}

/// <summary>

/// SessionId标识符

/// </summary>

public static string SessionName = "Redis\_SessionId";

//

// 摘要:

// 获取会话状态集合中的项数。

//

// 返回结果:

// 集合中的项数。

public int Count

{

get

{

return RedisBase.Hash\_GetCount(SessionID);

}

}

//

// 摘要:

// 获取一个值，该值指示会话是否为只读。

//

// 返回结果:

// 如果会话为只读，则为 true；否则为 false。

public bool IsReadOnly { get; set; }

//

// 摘要:

// 获取会话的唯一标识符。

//

// 返回结果:

// 唯一会话标识符。

public string SessionID

{

get

{

return GetSessionID();

}

}

//

// 摘要:

// 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时间（以分钟为单位）。

//

// 返回结果:

// 超时期限（以分钟为单位）。

public int Timeout { get; set; }

/// <summary>

/// 获取SessionID

/// </summary>

/// <param name="key">SessionId标识符</param>

/// <returns>HttpCookie值</returns>

private string GetSessionID()

{

HttpCookie cookie = context.Request.Cookies.Get(SessionName);

if (cookie == null || string.IsNullOrEmpty(cookie.Value))

{

string newSessionID = Guid.NewGuid().ToString();

HttpCookie newCookie = new HttpCookie(SessionName, newSessionID);

newCookie.HttpOnly = IsReadOnly;

newCookie.Expires = DateTime.Now.AddMinutes(Timeout);

context.Response.Cookies.Add(newCookie);

return "Session\_"+newSessionID;

}

else

{

return "Session\_"+cookie.Value;

}

}

//

// 摘要:

// 按名称获取或设置会话值。

//

// 参数:

// name:

// 会话值的键名。

//

// 返回结果:

// 具有指定名称的会话状态值；如果该项不存在，则为 null。

public object this[string name]

{

get

{

return RedisBase.Hash\_Get<object>(SessionID, name);

}

set

{

RedisBase.Hash\_Set<object>(SessionID, name, value);

}

}

// 摘要:

// 判断会话中是否存在指定key

//

// 参数:

// name:

// 键值

//

public bool IsExistKey(string name)

{

return RedisBase.Hash\_Exist<object>(SessionID, name);

}

//

// 摘要:

// 向会话状态集合添加一个新项。

//

// 参数:

// name:

// 要添加到会话状态集合的项的名称。

//

// value:

// 要添加到会话状态集合的项的值。

public void Add(string name, object value)

{

RedisBase.Hash\_Set<object>(SessionID, name, value);

}

//

// 摘要:

// 从会话状态集合中移除所有的键和值。

public void Clear()

{

RedisBase.Hash\_Remove(SessionID);

}

//

// 摘要:

// 删除会话状态集合中的项。

//

// 参数:

// name:

// 要从会话状态集合中删除的项的名称。

public void Remove(string name)

{

RedisBase.Hash\_Remove(SessionID,name);

}

//

// 摘要:

// 从会话状态集合中移除所有的键和值。

public void RemoveAll()

{

Clear();

}

}

}

[复制代码](javascript:void(0);)

   下面是实现类似在cs文件中能直接使用Session["UserId"]的方式，我的MyPage类继承Page实现了自己的逻辑主要做了两件事  1:初始化RedisSession  2:实现统一登录认证，OnPreInit方法里面判断用户是否登录，如果没有登录了则跳转到登陆界面

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

namespace ResidSessionDemo.RedisDemo

{

/// <summary>

/// 自定义Page 实现以下功能

/// 1.初始化RedisSession

/// 2.实现页面登录验证，继承此类，则可以实现所有页面的登录验证

/// </summary>

public class MyPage:Page

{

private RedisSession redisSession;

/// <summary>

/// RedisSession

/// </summary>

public RedisSession RedisSession

{

get

{

if (redisSession == null)

{

redisSession = new RedisSession(Context, true, 20);

}

return redisSession;

}

}

protected override void OnPreInit(EventArgs e)

{

base.OnPreInit(e);

//判断用户是否已经登录，如果未登录，则跳转到登录界面

if (!RedisSession.IsExistKey("UserCode"))

{

Response.Redirect("Login.aspx");

}

}

}

}

[复制代码](javascript:void(0);)

 我们来看看Default.aspx.cs是如何使用RedisSession的，至此我们实现了和Asp.netSession一模一样的功能和使用方式。

RedisSession.Remove("UserCode");

   相比StateServer,RedisSession具有以下优点

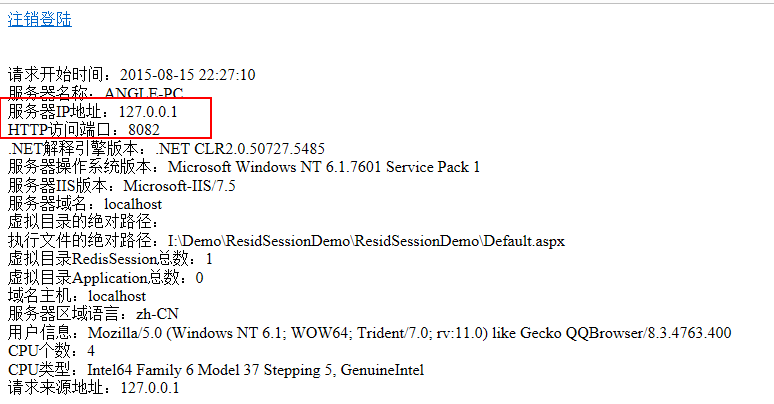
**1.redis服务器重启不会丢失数据  2.可以使用redis的读写分离个集群功能更加高效读写数据**

**测试效果，**使用nginx和iis部署两个站点做负载均衡，iis1地址127.0.0.1:8002 iis2地址127.0.0.1:9000  nginx代理服务地址127.0.0.1:8003,不懂如何配置的可以去阅读我的[nginx+iis实现负载均衡](http://www.cnblogs.com/yanweidie/p/4658136.html)这篇文章。我们来看一下测试结果。

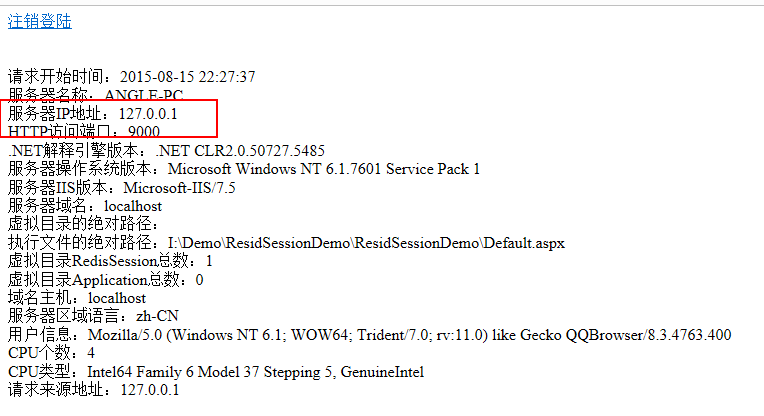
**访问127.0.0.1:8003 需要进行登录   用户名为admin  密码为123**



**登录成功以后，重点关注端口号信息**

****

**刷新页面，重点关注端口号信息**

****

可以尝试直接访问iis1地址127.0.0.1:8002 iis2地址127.0.0.1:9000 这两个站点，你会发现都不需要登录了。至此我们的redis实现session功能算是大功告成了。

**总结**

　　本篇从实际应用讲解了redis，后面应该还会有几篇继续介绍redis实际应用，敬请期待！

      本篇文章用到的资源打包下载地址：[redis\_demo](http://files.cnblogs.com/files/yanweidie/ResidSessionDemo.rar" \t "_blank)

　　svn下载地址：<http://code.taobao.org/svn/ResidSessionDemo/>