<https://www.cnblogs.com/x_wukong/p/3929654.html>

转自：http://www.2cto.com/kf/201206/135471.html

**一、术语session**  
在我的经验里，session这个词被滥用的程度大概仅次于transaction，更加有趣的是transaction与session在某些语境下的含义是相同的。  
   
session，中文经常翻译为会话，其本来的含义是指有始有终的一系列动作/消息，比如打电话时从拿起电话拨号到挂断电话这中间的一系列过程可以称之为一个session。有时候我们可以看到这样的话“在一个浏览器会话期间，...”，这里的会话一词用的就是其本义，是指从一个浏览器窗口打开到关闭这个期间 ①。最混乱的是“用户（客户端）在一次会话期间”这样一句话，它可能指用户的一系列动作（一般情况下是同某个具体目的相关的一系列动作，比如从登录到选购商品到结账登出这样一个网上购物的过程，有时候也被称为一个transaction），然而有时候也可能仅仅是指一次连接，也有可能是指含义①，其中的差别只能靠上下文来推断②。  
   
然而当session一词与网络协议相关联时，它又往往隐含了“面向连接”和/或“保持状态”这样两个含义，“面向连接”指的是在通信双方在通信之前要先建立一个通信的渠道，比如打电话，直到对方接了电话通信才能开始，与此相对的是写信，在你把信发出去的时候你并不能确认对方的地址是否正确，通信渠道不一定能建立，但对发信人来说，通信已经开始了。“保持状态”则是指通信的一方能够把一系列的消息关联起来，使得消息之间可以互相依赖，比如一个服务员能够认出再次光临的老顾客并且记得上次这个顾客还欠店里一块钱。这一类的例子有“一个TCP session”或者“一个POP3 session”③。  
   
而到了web服务器蓬勃发展的时代，session在web开发语境下的语义又有了新的扩展，它的含义是指一类用来在客户端与服务器之间保持状态的解决方案 ④。有时候session也用来指这种解决方案的存储结构，如“把xxx保存在session里”⑤。由于各种用于web开发的语言在一定程度上都提供了对这种解决方案的支持，所以在某种特定语言的语境下，session也被用来指代该语言的解决方案，比如经常把Java里提供的javax.servlet.http.HttpSession简称为session⑥。  
   
鉴于这种混乱已不可改变，本文中session一词的运用也会根据上下文有不同的含义，请大家注意分辨。  
在本文中，使用中文“浏览器会话期间”来表达含义①，使用“session机制”来表达含义④，使用“session”表达含义⑤，使用具体的“HttpSession”来表达含义⑥  
   
**二、HTTP协议与状态保持**  
HTTP协议本身是无状态的，这与HTTP协议本来的目的是相符的，客户端只需要简单的向服务器请求[下载](http://www.2cto.com/soft)某些文件，无论是客户端还是服务器都没有必要纪录彼此过去的行为，每一次请求之间都是独立的，好比一个顾客和一个自动售货机或者一个普通的（非会员制）大卖场之间的关系一样。  
   
然而聪明（或者贪心？）的人们很快发现如果能够提供一些按需生成的动态信息会使web变得更加有用，就像给有线电视加上点播功能一样。这种需求一方面迫使HTML逐步添加了表单、脚本、DOM等客户端行为，另一方面在服务器端则出现了CGI规范以响应客户端的动态请求，作为传输载体的HTTP协议也添加了文件上载、cookie这些特性。其中cookie的作用就是为了解决HTTP协议无状态的缺陷所作出的努力。至于后来出现的session机制则是又一种在客户端与服务器之间保持状态的解决方案。  
   
让我们用几个例子来描述一下cookie和session机制之间的区别与联系。笔者曾经常去的一家咖啡店有喝5杯咖啡免费赠一杯咖啡的优惠，然而一次性消费5杯咖啡的机会微乎其微，这时就需要某种方式来纪录某位顾客的消费数量。想象一下其实也无外乎下面的几种方案：  
1、该店的店员很厉害，能记住每位顾客的消费数量，只要顾客一走进咖啡店，店员就知道该怎么对待了。这种做法就是协议本身支持状态。  
2、发给顾客一张卡片，上面记录着消费的数量，一般还有个有效期限。每次消费时，如果顾客出示这张卡片，则此次消费就会与以前或以后的消费相联系起来。这种做法就是在客户端保持状态。  
3、发给顾客一张会员卡，除了卡号之外什么信息也不纪录，每次消费时，如果顾客出示该卡片，则店员在店里的纪录本上找到这个卡号对应的纪录添加一些消费信息。这种做法就是在服务器端保持状态。  
   
由于HTTP协议是无状态的，而出于种种考虑也不希望使之成为有状态的，因此，后面两种方案就成为现实的选择。具体来说cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。同时我们也看到，由于采用服务器端保持状态的方案在客户端也需要保存一个标识，所以session机制可能需要借助于cookie机制来达到保存标识的目的，但实际上它还有其他选择。  
   
**三、理解cookie机制**  
cookie机制的基本原理就如上面的例子一样简单，但是还有几个问题需要解决：“会员卡”如何分发；“会员卡”的内容；以及客户如何使用“会员卡”。  
   
正统的cookie分发是通过扩展HTTP协议来实现的，服务器通过在HTTP的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的cookie。然而纯粹的客户端脚本如JavaScript或者VBScript也可以生成cookie。  
   
而cookie的使用是由浏览器按照一定的原则在后台自动发送给服务器的。浏览器检查所有存储的cookie，如果某个cookie所声明的作用范围大于等于将要请求的资源所在的位置，则把该cookie附在请求资源的HTTP请求头上发送给服务器。意思是麦当劳的会员卡只能在麦当劳的店里出示，如果某家分店还发行了自己的会员卡，那么进这家店的时候除了要出示麦当劳的会员卡，还要出示这家店的会员卡。  
   
cookie的内容主要包括：名字，值，过期时间，路径和域。  
其中域可以指定某一个域比如.google.com，相当于总店招牌，比如宝洁公司，也可以指定一个域下的具体某台机器比如www.google.com或者froogle.google.com，可以用飘柔来做比。  
路径就是跟在域名后面的URL路径，比如/或者/foo等等，可以用某飘柔专柜做比。  
路径与域合在一起就构成了cookie的作用范围。  
如果不设置过期时间，则表示这个cookie的生命期为浏览器会话期间，只要关闭浏览器窗口，cookie就消失了。这种生命期为浏览器会话期的cookie被称为会话cookie。会话cookie一般不存储在硬盘上而是保存在内存里，当然这种行为并不是规范规定的。如果设置了过期时间，浏览器就会把cookie保存到硬盘上，关闭后再次打开浏览器，这些cookie仍然有效直到超过设定的过期时间。  
   
存储在硬盘上的cookie可以在不同的浏览器进程间共享，比如两个IE窗口。而对于保存在内存里的cookie，不同的浏览器有不同的处理方式。对于IE，在一个打开的窗口上按Ctrl-N（或者从文件菜单）打开的窗口可以与原窗口共享，而使用其他方式新开的IE进程则不能共享已经打开的窗口的内存cookie；对于Mozilla Firefox0.8，所有的进程和标签页都可以共享同样的cookie。一般来说是用javascript的window.open打开的窗口会与原窗口共享内存cookie。浏览器对于会话cookie的这种只认cookie不认人的处理方式经常给采用session机制的web应用[程序开发](http://www.2cto.com/kf)者造成很大的困扰。  
   
下面就是一个goolge设置cookie的响应头的例子  
HTTP/1.1 302 Found  
Location: http://www.google.com/intl/zh-CN/  
Set-Cookie: PREF=ID=0565f77e132de138:NW=1:TM=1098082649:LM=1098082649:S=KaeaCFPo49RiA\_d8; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com  
Content-Type: text/html  
   
这是使用HTTPLook这个HTTP Sniffer软件来俘获的HTTP通讯纪录的一部分  
   
浏览器在再次访问goolge的资源时自动向外发送cookie  
   
使用Firefox可以很容易的观察现有的cookie的值  
使用HTTPLook配合Firefox可以很容易的理解cookie的工作原理。  
   
IE也可以设置在接受cookie前询问  
   
这是一个询问接受cookie的对话框。  
   
**四、理解session机制**  
session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息。  
   
当程序需要为某个客户端的请求创建一个session的时候，服务器首先检查这个客户端的请求里是否已包含了一个session标识- 称为session id，如果已包含一个session id则说明以前已经为此客户端创建过session，服务器就按照session id把这个session检索出来使用（如果检索不到，可能会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中返回给客户端保存。 保存这个session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发挥给服务器。一般这个cookie的名字都是类似于SEEESIONID，而。比如weblogic对于web应用程序生成的cookie，JSESSIONID=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764，它的名字就是JSESSIONID。  
   
由于cookie可以被人为的禁止，必须有其他机制以便在cookie被禁止时仍然能够把session id传递回服务器。经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面，附加方式也有两种，一种是作为URL路径的附加信息，表现形式为http://...../xxx;jsessionid=ByOK ... 99zWpBng!-145788764  
另一种是作为查询字符串附加在URL后面，表现形式为http://...../xxx?jsessionid=ByOK ... 99zWpBng!-145788764  
这两种方式对于用户来说是没有区别的，只是服务器在解析的时候处理的方式不同，采用第一种方式也有利于把session id的信息和正常程序参数区分开来。  
为了在整个交互过程中始终保持状态，就必须在每个客户端可能请求的路径后面都包含这个session id。  
   
另一种技术叫做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把session id传递回服务器。比如下面的表单  
   
在被传递给客户端之前将被改写成  
   
这种技术现在已较少应用，笔者接触过的很古老的iPlanet6(SunONE应用服务器的前身)就使用了这种技术。  
实际上这种技术可以简单的用对action应用URL重写来代替。  
   
在谈论session机制的时候，常常听到这样一种误解“只要关闭浏览器，session就消失了”。其实可以想象一下会员卡的例子，除非顾客主动对店家提出销卡，否则店家绝对不会轻易删除顾客的资料。对session来说也是一样的，除非程序通知服务器删除一个session，否则服务器会一直保留，程序一般都是在用户做log off的时候发个指令去删除session。然而浏览器从来不会主动在关闭之前通知服务器它将要关闭，因此服务器根本不会有机会知道浏览器已经关闭，之所以会有这种错觉，是大部分session机制都使用会话cookie来保存session id，而关闭浏览器后这个session id就消失了，再次连接服务器时也就无法找到原来的session。如果服务器设置的cookie被保存到硬盘上，或者使用某种手段改写浏览器发出的HTTP请求头，把原来的session id发送给服务器，则再次打开浏览器仍然能够找到原来的session。  
   
恰恰是由于关闭浏览器不会导致session被删除，迫使服务器为seesion设置了一个失效时间，当距离客户端上一次使用session的时间超过这个失效时间时，服务器就可以认为客户端已经停止了活动，才会把session删除以节省存储空间。  
   
**五、理解javax.servlet.http.HttpSession**  
HttpSession是Java平台对session机制的实现规范，因为它仅仅是个接口，具体到每个web应用服务器的提供商，除了对规范支持之外，仍然会有一些规范里没有规定的细微差异。这里我们以BEA的Weblogic Server8.1作为例子来演示。  
   
首先，Weblogic Server提供了一系列的参数来控制它的HttpSession的实现，包括使用cookie的开关选项，使用URL重写的开关选项，session持久化的设置，session失效时间的设置，以及针对cookie的各种设置，比如设置cookie的名字、路径、域，cookie的生存时间等。  
   
一般情况下，session都是存储在内存里，当服务器进程被停止或者重启的时候，内存里的session也会被清空，如果设置了session的持久化特性，服务器就会把session保存到硬盘上，当服务器进程重新启动或这些信息将能够被再次使用，Weblogic Server支持的持久性方式包括文件、数据库、客户端cookie保存和复制。  
   
复制严格说来不算持久化保存，因为session实际上还是保存在内存里，不过同样的信息被复制到各个cluster内的服务器进程中，这样即使某个服务器进程停止工作也仍然可以从其他进程中取得session。  
   
cookie生存时间的设置则会影响浏览器生成的cookie是否是一个会话cookie。默认是使用会话cookie。有兴趣的可以用它来试验我们在第四节里提到的那个误解。  
   
cookie的路径对于web应用程序来说是一个非常重要的选项，Weblogic Server对这个选项的默认处理方式使得它与其他服务器有明显的区别。后面我们会专题讨论。  
   
关于session的设置参考[5] http://e-docs.bea.com/wls/docs70/webapp/weblogic\_xml.html#1036869  
   
**六、HttpSession常见问题**  
（在本小节中session的含义为⑤和⑥的混合）  
   
1、session在何时被创建  
一个常见的误解是以为session在有客户端访问时就被创建，然而事实是直到某server端程序调用HttpServletRequest.getSession(true)这样的语句时才被创建，注意如果JSP没有显示的使用 关闭session，则JSP文件在编译成Servlet时将会自动加上这样一条语句HttpSession session = HttpServletRequest.getSession(true);这也是JSP中隐含的session对象的来历。  
   
由于session会消耗内存资源，因此，如果不打算使用session，应该在所有的JSP中关闭它。  
   
2、session何时被删除  
综合前面的讨论，session在下列情况下被删除a.程序调用HttpSession.invalidate();或b.距离上一次收到客户端发送的session id时间间隔超过了session的超时设置;或c.服务器进程被停止（非持久session）  
   
3、如何做到在浏览器关闭时删除session  
严格的讲，做不到这一点。可以做一点努力的办法是在所有的客户端页面里使用javascript代码window.oncolose来监视浏览器的关闭动作，然后向服务器发送一个请求来删除session。但是对于浏览器崩溃或者强行杀死进程这些非常规手段仍然无能为力。  
   
4、有个HttpSessionListener是怎么回事  
你可以创建这样的listener去监控session的创建和销毁事件，使得在发生这样的事件时你可以做一些相应的工作。注意是session的创建和销毁动作触发listener，而不是相反。类似的与HttpSession有关的listener还有HttpSessionBindingListener，HttpSessionActivationListener和HttpSessionAttributeListener。  
   
5、存放在session中的对象必须是可序列化的吗  
不是必需的。要求对象可序列化只是为了session能够在集群中被复制或者能够持久保存或者在必要时server能够暂时把session交换出内存。在Weblogic Server的session中放置一个不可序列化的对象在控制台上会收到一个警告。我所用过的某个iPlanet版本如果session中有不可序列化的对象，在session销毁时会有一个Exception，很奇怪。  
   
6、如何才能正确的应付客户端禁止cookie的可能性  
对所有的URL使用URL重写，包括超链接，form的action，和重定向的URL，具体做法参见[6]  
http://e-docs.bea.com/wls/docs70/webapp/sessions.html#100770  
   
7、开两个浏览器窗口访问应用程序会使用同一个session还是不同的session  
参见第三小节对cookie的讨论，对session来说是只认id不认人，因此不同的浏览器，不同的窗口打开方式以及不同的cookie存储方式都会对这个问题的答案有影响。  
   
8、如何防止用户打开两个浏览器窗口操作导致的session混乱  
这个问题与防止表单多次提交是类似的，可以通过设置客户端的令牌来解决。就是在服务器每次生成一个不同的id返回给客户端，同时保存在session里，客户端提交表单时必须把这个id也返回服务器，程序首先比较返回的id与保存在session里的值是否一致，如果不一致则说明本次操作已经被提交过了。可以参看《J2EE核心模式》关于表示层模式的部分。需要注意的是对于使用javascript window.open打开的窗口，一般不设置这个id，或者使用单独的id，以防主窗口无法操作，建议不要再window.open打开的窗口里做修改操作，这样就可以不用设置。  
   
9、为什么在Weblogic Server中改变session的值后要重新调用一次session.setValue  
做这个动作主要是为了在集群环境中提示Weblogic Server session中的值发生了改变，需要向其他服务器进程复制新的session值。  
   
10、为什么session不见了  
排除session正常失效的因素之外，服务器本身的可能性应该是微乎其微的，虽然笔者在iPlanet6SP1加若干补丁的Solaris版本上倒也遇到过；浏览器插件的可能性次之，笔者也遇到过3721插件造成的问题；理论上防火墙或者代理服务器在cookie处理上也有可能会出现问题。  
出现这一问题的大部分原因都是程序的错误，最常见的就是在一个应用程序中去访问另外一个应用程序。我们在下一节讨论这个问题。  
   
**七、跨应用程序的session共享**  
   
常常有这样的情况，一个大项目被分割成若干小项目开发，为了能够互不干扰，要求每个小项目作为一个单独的web应用程序开发，可是到了最后突然发现某几个小项目之间需要共享一些信息，或者想使用session来实现SSO(single sign on)，在session中保存login的用户信息，最自然的要求是应用程序间能够访问彼此的session。  
   
然而按照Servlet规范，session的作用范围应该仅仅限于当前应用程序下，不同的应用程序之间是不能够互相访问对方的session的。各个应用服务器从实际效果上都遵守了这一规范，但是实现的细节却可能各有不同，因此解决跨应用程序session共享的方法也各不相同。  
   
首先来看一下Tomcat是如何实现web应用程序之间session的隔离的，从Tomcat设置的cookie路径来看，它对不同的应用程序设置的cookie路径是不同的，这样不同的应用程序所用的session id是不同的，因此即使在同一个浏览器窗口里访问不同的应用程序，发送给服务器的session id也可以是不同的。  
   
根据这个特性，我们可以推测Tomcat中session的内存结构大致如下。  
   
笔者以前用过的iPlanet也采用的是同样的方式，估计SunONE与iPlanet之间不会有太大的差别。对于这种方式的服务器，解决的思路很简单，实际实行起来也不难。要么让所有的应用程序共享一个session id，要么让应用程序能够获得其他应用程序的session id。  
   
iPlanet中有一种很简单的方法来实现共享一个session id，那就是把各个应用程序的cookie路径都设为/（实际上应该是/NASApp，对于应用程序来讲它的作用相当于根）。  
   
/NASApp  
   
需要注意的是，操作共享的session应该遵循一些[编程](http://www.2cto.com/kf)约定，比如在session attribute名字的前面加上应用程序的前缀，使得setAttribute("name", "neo")变成setAttribute("app1.name", "neo")，以防止命名空间冲突，导致互相覆盖。  
   
在Tomcat中则没有这么方便的选择。在Tomcat版本3上，我们还可以有一些手段来共享session。对于版本4以上的Tomcat，目前笔者尚未发现简单的办法。只能借助于第三方的力量，比如使用文件、数据库、JMS或者客户端cookie，URL参数或者隐藏字段等手段。  
   
我们再看一下Weblogic Server是如何处理session的。  
   
从截屏画面上可以看到Weblogic Server对所有的应用程序设置的cookie的路径都是/，这是不是意味着在Weblogic Server中默认的就可以共享session了呢？然而一个小实验即可证明即使不同的应用程序使用的是同一个session，各个应用程序仍然只能访问自己所设置的那些属性。这说明Weblogic Server中的session的内存结构可能如下  
   
对于这样一种结构，在session机制本身上来解决session共享的问题应该是不可能的了。除了借助于第三方的力量，比如使用文件、数据库、JMS或者客户端cookie，URL参数或者隐藏字段等手段，还有一种较为方便的做法，就是把一个应用程序的session放到ServletContext 中，这样另外一个应用程序就可以从ServletContext中取得前一个应用程序的引用。示例代码如下，  
   
应用程序A  
context.setAttribute("appA", session);  
   
应用程序B  
contextA = context.getContext("/appA");  
HttpSession sessionA = (HttpSession)contextA.getAttribute("appA");  
   
值得注意的是这种用法不可移植，因为根据ServletContext的JavaDoc，应用服务器可以处于安全的原因对于context.getContext("/appA");返回空值，以上做法在Weblogic Server 8.1中通过。  
   
那么Weblogic Server为什么要把所有的应用程序的cookie路径都设为/呢？原来是为了SSO，凡是共享这个session的应用程序都可以共享认证的信息。一个简单的实验就可以证明这一点，修改首先登录的那个应用程序的描述符weblogic.xml，把cookie路径修改为/appA访问另外一个应用程序会重新要求登录，即使是反过来，先访问cookie路径为/的应用程序，再访问修改过路径的这个，虽然不再提示登录，但是登录的用户信息也会丢失。注意做这个实验时认证方式应该使用FORM，因为[浏览器](http://www.2cto.com/os/liulanqi/)和web服务器对basic认证方式有其他的处理方式，第二次请求的认证不是通过session来实现的。具体请参看[7] secion 14.8 Authorization，你可以修改所附的示例程序来做这些试验。  
   
**八、总结**  
session机制本身并不复杂，然而其实现和配置上的灵活性却使得具体情况复杂多变。这也要求我们不能把仅仅某一次的经验或者某一个浏览器，服务器的经验当作普遍适用的经验，而是始终需要具体情况具体分析。  
   
  
   
session与cookie区别  
   
Session是由应用服务器维持的一个服务器端的存储空间，用户在连接服务器时，会由服务器生成一个唯一的SessionID,用该SessionID为标识符来存取服务器端的Session存储空间。而SessionID这一数据则是保存到客户端，用Cookie保存的，用户提交页面时，会将这一SessionID提交到服务器端，来存取Session数据。这一过程，是不用开发人员干预的。所以一旦客户端禁用Cookie，那么Session也会失效。  
   
服务器也可以通过URL重写的方式来传递SessionID的值，因此不是完全依赖Cookie。如果客户端Cookie禁用，则服务器可以自动通过重写URL的方式来保存Session的值，并且这个过程对程序员透明。  
   
可以试一下，即使不写Cookie，在使用request.getCookies();取出的Cookie数组的长度也是1，而这个Cookie的名字就是JSESSIONID，还有一个很长的二进制的字符串，是SessionID的值。  
   
Cookie是客户端的存储空间，由浏览器来维持。  
   
在一些投票之类的场合，我们往往因为公平的原则要求每人只能投一票，在一些WEB开发中也有类似的情况，这时候我们通常会使用COOKIE来实现，例如如下的代码：  
   
< % cookie[]cookies = request.getCookies();  
   
if (cookies.lenght == 0 || cookies == null)  
   
doStuffForNewbie();  
   
//没有访问过 www.2cto.com  
   
}  
   
else  
   
{  
   
doStuffForReturnVisitor(); //已经访问过了  
   
}  
   
% >  
   
这是很浅显易懂的道理，检测COOKIE的存在，如果存在说明已经运行过写入COOKIE的代码了，然而运行以上的代码后，无论何时结果都是执行doStuffForReturnVisitor()，通过控制面板-Internet选项-设置-察看文件却始终看不到生成的cookie文件，奇怪，代码明明没有问题，不过既然有cookie，那就显示出来看看。  
   
cookie[]cookies = request.getCookies();  
   
if (cookies.lenght == 0 || cookies == null)  
   
out.println("Has not visited this website");  
   
}  
   
else  
   
{  
   
for (int i = 0; i < cookie.length; i++)  
   
{  
   
out.println("cookie name:" + cookies[ i ].getName() + "cookie value:" +  
   
cookie[ i ].getValue());  
   
}  
   
}  
   
运行结果:  
   
cookie name:JSESSIONID cookie value:KWJHUG6JJM65HS2K6  
   
为什么会有cookie呢,大家都知道，http是无状态的协议，客户每次读取web页面时，服务器都打开新的会话，而且服务器也不会自动维护客户的上下文信息，那么要怎么才能实现网上商店中的购物车呢，session就是一种保存上下文信息的机制，它是针对每一个用户的，变量的值保存在服务器端，通过SessionID来区分不同的客户,session是以cookie或URL重写为基础的，默认使用cookie来实现，系统会创造一个名为JSESSIONID的输出cookie，我们叫做session cookie,以区别persistentcookies,也就是我们通常所说的cookie,注意sessioncookie是存储于浏览器内存中的，并不是写到硬盘上的，这也就是我们刚才看到的JSESSIONID。  
   
我们通常情是看不到JSESSIONID的，但是当我们把浏览器的cookie禁止后，web服务器会采用URL重写的方式传递Sessionid，我们就可以在地址栏看到sessionid=KWJHUG6JJM65HS2K6之类的字符串。  
   
明白了原理，我们就可以很容易的分辨出persistent cookies和sessioncookie的区别了，网上那些关于两者安全性的讨论也就一目了然了，session cookie针对某一次会话而言，会话结束sessioncookie也就随着消失了，而persistentcookie只是存在于客户端硬盘上的一段文本（通常是加密的），而且可能会遭到cookie欺骗以及针对cookie的跨站脚本攻击，自然不如session cookie安全了。  
   
通常sessioncookie是不能跨窗口使用的，当你新开了一个浏览器窗口进入相同页面时，系统会赋予你一个新的sessionid，这样我们信息共享的目的就达不到了，此时我们可以先把sessionid保存在persistentcookie中，然后在新窗口中读出来，就可以得到上一个窗口SessionID了，这样通过session cookie和persistentcookie的结合我们就实现了跨窗口的session tracking（会话跟踪）。  
   
在一些web开发的书中，往往只是简单的把Session和cookie作为两种并列的http传送信息的方式，sessioncookies位于服务器端，persistentcookie位于客户端，可是session又是以cookie为基础的，明白的两者之间的联系和区别，我们就不难选择合适的技术来开发webservice了、

**标签:**[web开发](http://www.cnblogs.com/x_wukong/tag/web%E5%BC%80%E5%8F%91/)