## 1.1 Cookie机制

## 在程序中，会话跟踪是很重要的事情。理论上，****一个用户的所有请求操作都应该属于同一个会话****，而另一个用户的所有请求操作则应该属于另一个会话，二者不能混淆。例如，用户A在超市购买的任何商品都应该放在A的购物车内，不论是用户A什么时间购买的，这都是属于同一个会话的，不能放入用户B或用户C的购物车内，这不属于同一个会话。

而Web应用程序是使用HTTP协议传输数据的。**HTTP协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上跟踪会话**。即用户A购买了一件商品放入购物车内，当再次购买商品时服务器已经无法判断该购买行为是属于用户A的会话还是用户B的会话了。要跟踪该会话，必须引入一种机制。

Cookie就是这样的一种机制。它可以弥补HTTP协议无状态的不足。在Session出现之前，基本上所有的网站都采用Cookie来跟踪会话。

**1.1.1  什么是Cookie**

Cookie意为“甜饼”，是**由W3C组织提出**，最早由Netscape社区发展的一种机制。目前Cookie已经成为标准，所有的主流浏览器如IE、Netscape、Firefox、Opera等都支持Cookie。

　　由于HTTP是一种无状态的协议，服务器单从网络连接上无从知道客户身份。怎么办呢？就给客户端们颁发一个通行证吧，每人一个，无论谁访问都必须携带自己通行证。这样服务器就能从通行证上确认客户身份了。这就是Cookie的工作原理。

　　Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。

如果浏览器不支持Cookie（如大部分手机中的浏览器）或者把Cookie禁用了，Cookie功能就会失效。

### ****1.1.2  记录用户访问次数****

Java中把Cookie封装成了javax.servlet.http.Cookie类。每个Cookie都是该Cookie类的对象。服务器通过操作Cookie类对象对客户端Cookie进行操作。通过request.getCookie()获取客户端提交的所有Cookie（以Cookie[]数组形式返回），通过response.addCookie(Cookiecookie)向客户端设置Cookie。

Cookie对象使用key-value属性对的形式保存用户状态，一个Cookie对象保存一个属性对，一个request或者response同时使用多个Cookie。因为Cookie类位于包javax.servlet.http.\*下面，所以JSP中不需要import该类。

**1.1.3  Cookie的不可跨域名性**

很多网站都会使用Cookie。例如，Google会向客户端颁发Cookie，Baidu也会向客户端颁发Cookie。那浏览器访问Google会不会也携带上Baidu颁发的Cookie呢？或者Google能不能修改Baidu颁发的Cookie呢？

答案是否定的。Cookie具有不可跨域名性。根据Cookie规范，浏览器访问Google只会携带Google的Cookie，而不会携带Baidu的Cookie。Google也只能操作Google的Cookie，而不能操作Baidu的Cookie。

Cookie在客户端是由浏览器来管理的。浏览器能够保证Google只会操作Google的Cookie而不会操作Baidu的Cookie，从而保证用户的隐私安全。浏览器判断一个网站是否能操作另一个网站Cookie的依据是域名。Google与Baidu的域名不一样，因此Google不能操作Baidu的Cookie。

需要注意的是，虽然网站images.google.com与网站www.google.com同属于Google，但是域名不一样，二者同样不能互相操作彼此的Cookie。

### ****1.1.6  设置Cookie的所有属性****

除了name与value之外，Cookie还具有其他几个常用的属性。每个属性对应一个getter方法与一个setter方法。Cookie类的所有属性如表1.1所示。



### ****1.1.7  Cookie的有效期****

Cookie的maxAge决定着Cookie的有效期，单位为秒（Second）。Cookie中通过getMaxAge()方法与setMaxAge(int maxAge)方法来读写maxAge属性。如果maxAge属性为正数，则表示该Cookie会在maxAge秒之后自动失效。浏览器会将maxAge为正数的Cookie持久化，即写到对应的Cookie文件中。无论客户关闭了浏览器还是电脑，只要还在maxAge秒之前，登录网站时该Cookie仍然有效。下面代码中的Cookie信息将永远有效。

Cookie cookie = new Cookie("username","helloweenvsfei"); // 新建Cookie

cookie.setMaxAge(Integer.MAX\_VALUE); // 设置生命周期为MAX\_VALUE

response.addCookie(cookie); // 输出到客户端

如果maxAge为负数，则表示该Cookie仅在本浏览器窗口以及本窗口打开的子窗口内有效，关闭窗口后该Cookie即失效。maxAge为负数的Cookie，为临时性Cookie，不会被持久化，不会被写到Cookie文件中。Cookie信息保存在浏览器内存中，因此关闭浏览器该Cookie就消失了。Cookie默认的maxAge值为–1。

如果maxAge为0，则表示删除该Cookie。Cookie机制没有提供删除Cookie的方法，因此通过设置该Cookie即时失效实现删除Cookie的效果。失效的Cookie会被浏览器从Cookie文件或者内存中删除。

**1.1.8  Cookie的修改、删除**

Cookie并不提供修改、删除操作。如果要修改某个Cookie，只需要新建一个同名的Cookie，添加到response中覆盖原来的Cookie。

如果要删除某个Cookie，只需要新建一个同名的Cookie，并将maxAge设置为0，并添加到response中覆盖原来的Cookie。注意是0而不是负数。负数代表其他的意义。读者可以通过上例的程序进行验证，设置不同的属性。

注意：修改、删除Cookie时，新建的Cookie除value、maxAge之外的所有属性，例如name、path、domain等，都要与原Cookie完全一样。否则，浏览器将视为两个不同的Cookie不予覆盖，导致修改、删除失败。

**1.1.9  Cookie的域名**

Cookie是不可跨域名的。域名www.google.com颁发的Cookie不会被提交到域名www.baidu.com去。这是由Cookie的隐私安全机制决定的。隐私安全机制能够禁止网站非法获取其他网站的Cookie。

正常情况下，同一个一级域名下的两个二级域名如www.helloweenvsfei.com和images.helloweenvsfei.com也不能交互使用Cookie，因为二者的域名并不严格相同。如果想所有helloweenvsfei.com名下的二级域名都可以使用该Cookie，需要设置Cookie的domain参数。

### ****1.1.10  Cookie的路径****

domain属性决定运行访问Cookie的域名，而path属性决定允许访问Cookie的路径（ContextPath）。例如，如果只允许/sessionWeb/下的程序使用Cookie，可以这么写：

Cookie cookie = new Cookie("time","20080808"); // 新建Cookie

cookie.setPath("/session/"); // 设置路径

response.addCookie(cookie); // 输出到客户端

设置为“/”时允许所有路径使用Cookie。path属性需要使用符号“/”结尾。name相同但domain相同的两个Cookie也是两个不同的Cookie。

注意：页面只能获取它属于的Path的Cookie。例如/session/test/a.jsp不能获取到路径为/session/abc/的Cookie。使用时一定要注意。

## ****1.2****

## ****Session机制****

除了使用Cookie，Web应用程序中还经常使用Session来记录客户端状态。**Session是服务器端使用的一种记录客户端状态的机制**，使用上比Cookie简单一些，相应的也**增加了服务器的存储压力**。

### ****1.2.1  什么是Session****

Session是另一种记录客户状态的机制，不同的是Cookie保存在客户端浏览器中，而Session保存在服务器上。客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录在服务器上。这就是Session。客户端浏览器再次访问时只需要从该Session中查找该客户的状态就可以了。

如果说**Cookie机制是通过检查客户身上的“通行证”来确定客户身份的话，那么Session机制就是通过检查服务器上的“客户明细表”来确认客户身份。Session相当于程序在服务器上建立的一份客户档案，客户来访的时候只需要查询客户档案表就可以了。**

### ****1.2.2  实现用户登录****

　　Session对应的类为javax.servlet.http.HttpSession类。每个来访者对应一个Session对象，所有该客户的状态信息都保存在这个Session对象里。**Session对象是在客户端第一次请求服务器的时候创建的**。Session也是一种key-value的属性对，通过getAttribute(Stringkey)和setAttribute(String key，Objectvalue)方法读写客户状态信息。Servlet里通过request.getSession()方法获取该客户的Session

### ****1.2.3  Session的生命周期****

　　Session保存在服务器端。**为了获得更高的存取速度，服务器一般把Session放在内存里。每个用户都会有一个独立的Session。如果Session内容过于复杂，当大量客户访问服务器时可能会导致内存溢出。因此，Session里的信息应该尽量精简。**

**Session在用户第一次访问服务器的时候自动创建**。需要注意只有访问JSP、Servlet等程序时才会创建Session，只访问HTML、IMAGE等静态资源并不会创建Session。如果尚未生成Session，也可以使用request.getSession(true)强制生成Session。

**Session生成后，只要用户继续访问，服务器就会更新Session的最后访问时间，并维护该Session**。用户每访问服务器一次，无论是否读写Session，服务器都认为该用户的Session“活跃（active）”了一次。

**1.2.4  Session的有效期**

　　由于会有越来越多的用户访问服务器，因此Session也会越来越多。**为防止内存溢出，服务器会把长时间内没有活跃的Session从内存删除。这个时间就是Session的超时时间。如果超过了超时时间没访问过服务器，Session就自动失效了。**

Session的超时时间为maxInactiveInterval属性，可以通过对应的getMaxInactiveInterval()获取，通过setMaxInactiveInterval(longinterval)修改。

　　Session的超时时间也可以在web.xml中修改。另外，通过调用Session的invalidate()方法可以使Session失效。

### ****1.2.6  Session对浏览器的要求****

　　虽然Session保存在服务器，对客户端是透明的，它的正常运行仍然需要客户端浏览器的支持。这是因为Session需要使用Cookie作为识别标志。HTTP协议是无状态的，Session不能依据HTTP连接来判断是否为同一客户，因此服务器向客户端浏览器发送一个名为JSESSIONID的Cookie，它的值为该Session的id（也就是HttpSession.getId()的返回值）。Session依据该Cookie来识别是否为同一用户。

　　该Cookie为服务器自动生成的，它的maxAge属性一般为–1，表示仅当前浏览器内有效，并且各浏览器窗口间不共享，关闭浏览器就会失效。

　　因此同一机器的两个浏览器窗口访问服务器时，会生成两个不同的Session。但是由浏览器窗口内的链接、脚本等打开的新窗口（也就是说不是双击桌面浏览器图标等打开的窗口）除外。这类子窗口会共享父窗口的Cookie，因此会共享一个Session。

注意：新开的浏览器窗口会生成新的Session，但子窗口除外。子窗口会共用父窗口的Session。例如，在链接上右击，在弹出的快捷菜单中选择“在新窗口中打开”时，子窗口便可以访问父窗口的Session。

## ****3、cookie和session的区别****

**1.3.cookie和session的区别**

**1.3.1 cookie数据存放在客户的浏览器上，session数据放在服务器上.**

简单的说，当你登录一个网站的时候，如果web服务器端使用的是session,那么所有的数据都保存在服务器上面，

客户端每次请求服务器的时候会发送 当前会话的session\_id，服务器根据当前session\_id判断相应的用户数据标志，以确定用户是否登录，或具有某种权限。

由于数据是存储在服务器 上面，所以你不能伪造，但是如果你能够获取某个登录用户的session\_id，用特殊的浏览器伪造该用户的请求也是能够成功的。

session\_id是服务器和客户端链接时候随机分配的，一般来说是不会有重复，但如果有大量的并发请求，也不是没有重复的可能性。

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。

**1.3.2** **cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗考虑到安全应当使用session。**

**1.3.3 设置cookie时间可以使cookie过期。但是使用session-destory（），我们将会销毁会话。**

**1.3.4 session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能考虑到减轻服务器性能方面，应当使用cookie。**

**1.3.5 单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。(Session对象没有对存储的数据量的限制，其中可以保存更为复杂的数据类型)**

注意:

session很容易失效,用户体验很差;

虽然cookie不安全,但是可以加密 ;

cookie也分为永久和暂时存在的;

浏览器有禁止cookie功能 ,但一般用户都不会设置;

一定要设置失效时间,要不然浏览器关闭就消失了;

例如:

记住密码功能就是使用永久cookie写在客户端电脑，下次登录时，自动将cookie信息附加发送给服务端。

  application是全局性信息，是所有用户共享的信息，如可以记录有多少用户现在登录过本网站，并把该信息展示个所有用户。

两者最大的区别在于生存周期，一个是IE启动到IE关闭.(浏览器页面一关 ,session就消失了)，一个是预先设置的生存周期，或永久的保存于本地的文件。(cookie)

Session信息是存放在server端，但session id是存放在client cookie的，当然php的session存放方法是多样化的，这样就算禁用cookie一样可以跟踪

 Cookie是完全保持在客户端的如：IE firefox 当客户端禁止cookie时将不能再使用。