**Cookie and session**

**1. 概述**

简单来说，他们只是http中的一个配置项，在servlet规范中也只对应一个类。Session和cookie的作用都是为了保持访问用户和后端服务器的交互状态。

限制：http对cookie的数量和大小限制，cookie被盗、cookie伪造问题；如何解决session在多台服务器之间的共享问题。

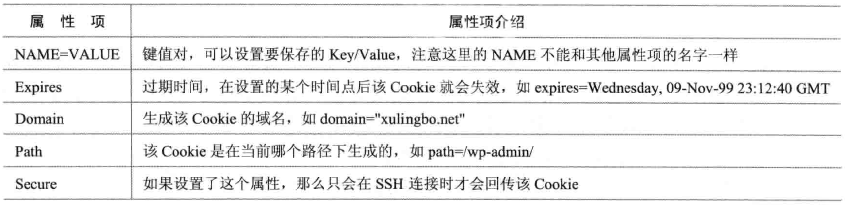
**2. cookie**

Cookie作用：通俗讲，当一个用户通过http访问一个服务器时。这个服务器会将一些key/value键值对方会给客户端浏览器，并给这些数据加上一些限制条件，在条件符合时，这个用户下次访问这个服务器时，数据又被完整的带回给服务器。

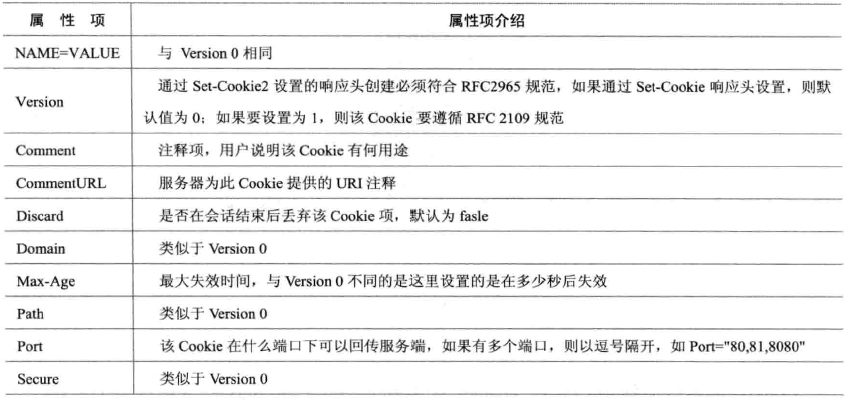
设计之初考虑的是，为了记录用户在一段时间内访问web应用的行为路径。由于http是一种无状态协议，当用户的一次访问请求结束后。后端服务器就无法知道下一次访问的还是不是上次访问的用户。当有了cookie之后，服务端就可以根据cookie值来划分访问的用户了。

**2.1 cookie的属性项**

Version 0：



Version 1：



**2.2 cookie是如何工作**

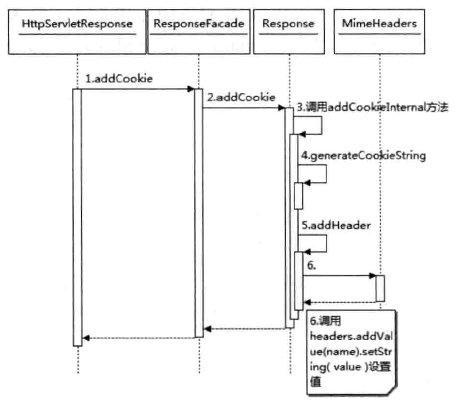


图1 Tomcat创建set-cookie响应头的时序图

由图1可知，真正构建cookie实在org.apache.catalina.connector.Response类中完成的，调用generateCookieString方法将Cookie对象构造成一个字符串，构造的字符串的格式如下：userName(cookie-key)=“yunfei”(cookie-value); Version=”1”; Domain=”baidu.com”; Max-Age=1000。然后将这个字符串命名为Set-Cookie添加到MimeHeaders中。

当我们通过response.addCookie()创建多个Cookie时，最终在请求返回时构造的http响应头是否将相同的Header标识的Set-Cookie值进行合并？在构建http返回字节流时是将Header中所有的项顺序的写出，而没有进行任何修改。所以可以想象浏览器在接受http返回的数据时是分别解析每一个Header项的。

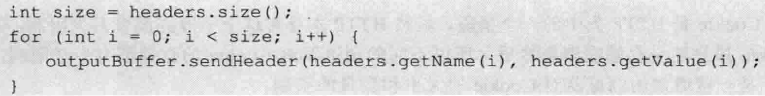


图2 tomcat最终构造http响应头的代码

当我们请求某个URL路径时，浏览器会根据这个URL路径将符合条件的Cookie放在Request请求头中传回给服务器，服务器通过reques.getCookie()来获取所有的cookie。

**2.3 cookie使用限制**

Cookie是http头中的一个字段，虽然http本身对这个字段并没有多少限制，但是cookie最终还是存储在浏览器里，所以不同浏览器对cookie的存储（数量和大小）都有一些限制。

**3 session**

前文讲述中，cookie可以让服务器追踪每个客户端的访问，但是每次客户端的访问都必须传回这些cookie，如果cookie很多，则增加了cs交互的数据传输量，而session的出现正是解决这个问题。

同一客户端每次和服务器交互时，不需要每次传回所有的cookie值，只需要传回一个id（客户端第一次访问服务器时生成的，每个客户端是唯一的），这样每个客户端就有了唯一的id，这个id通常是NAME为JESIONID的一个cookie。

**3.1 三种方式可以让session正常工作**

（1）基于URL Path Paramater，默认支持

（2）基于cookie，如果没有修改context容器的cookies标识，则默认也是支持的

（3）基于SSL，默认不支持，只有connector.getAttribute(“SSLEnabled”)为TRUE时才支持。

**3.2 session如何工作**

有了session ID，服务端就可以创建httpSession对象了，第一次触发通过request.getSession()方法。如果当前的session ID还没有对应httpSession对象，那么就创建一个新的，并将这个对象加到org.apache.catalina.Manager的sessions容器中保存。Manager将管理所有的Session的生命周期，Session过期将会被回收，服务器关闭，Session将被序列化到磁盘等。只要之歌httpSession对象存在，用户就可以通过session ID来获取这个对象，也就做到了对状态的保持。

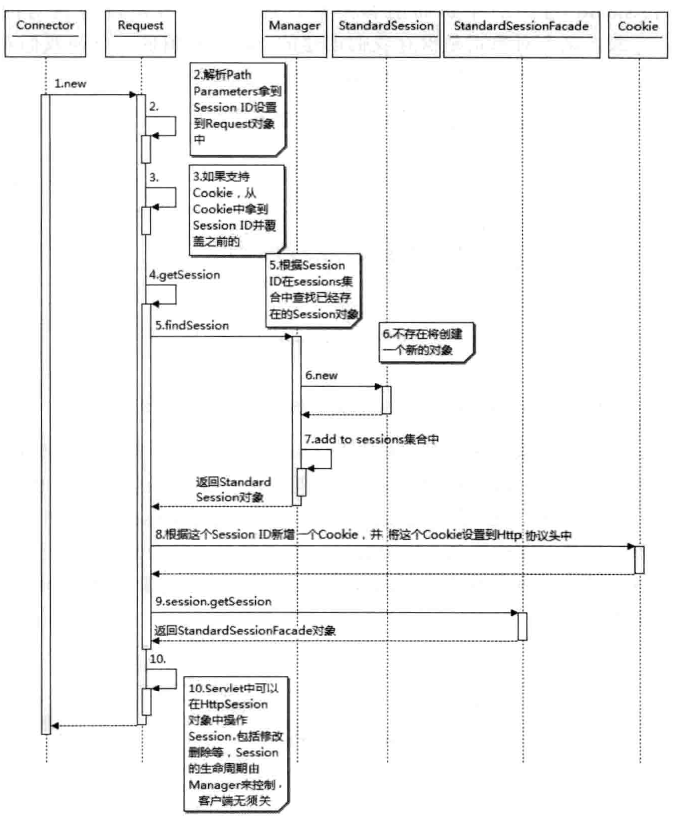


图3 session工作的时序图

其中，standManager类负责Servlet容器中所有的standardSession对象的生命周期管理。当servlet容器重启或者关闭时，standManger负责持久化没有过期的standSession对象，他将所有的standSession对象持久化到“SESSIONS.ser”文件中。Servlet容器重启时，他会重新读取这个文件解析出所有的Session对象，重新保存到StandardManager的Session集合中。

另外，StandardSession对象并不是永远保存的，否则内存将会耗尽，必须给每个session对象定义一个有效时间，超过这个时间则session对象将被清除。

**3.3 cookie和session的安全性对比**

Cookie把所有数据通过http的header从客户端传到服务端，又从服务端在传回客户端，所有的数据都存储在客户端的浏览器里，，所以这些cookie数据可以被访问到。

Session是将所有数据保存在服务端，只通过Cookie传递一个session ID，所以Session更适合存储用户隐私和重要数据。

**3.4 分布式Session框架**

大型互联网系统中，服务提供可能由于负载均衡的存在导致，在不同物理机上提供服务。那么如何解决“客户状态”共享问题？本着“谁家的孩子谁抱走”的原则，若采用cookie，各自客户机保存各自状态，在业务完全满足，但是cookie存在安全、和管理混乱问题。若采用session需解决session在不同服务器间的共享问题。

**3.4.1 实现思路**

我们需要一个服务订阅服务器，在应用启动时可以从这个订阅服务器订阅这个应用需要的可写session项和可写cookie项，这些配置的session和cookie可以限制应用能够使用哪些session和cookie，甚至操纵那些session和cookie可读或者可写。这样可以精确的控制哪些应用可以操作那些session和cookie，可以有效控制session的安全性和cookie的数量。

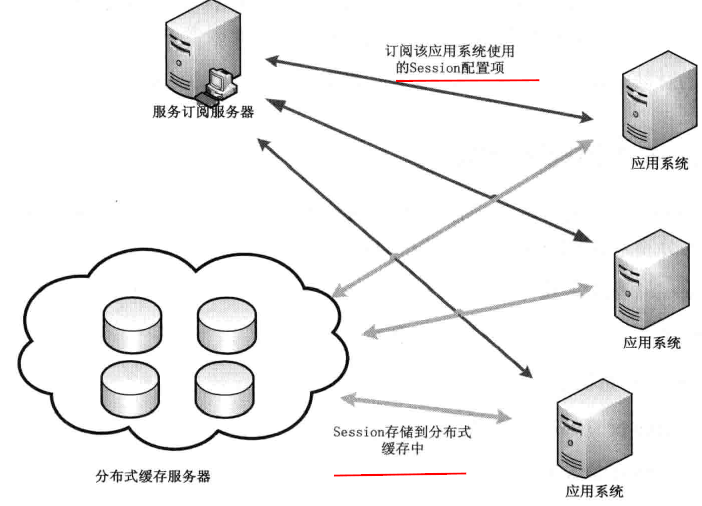


图4 session框架的架构图

**3.4.2 配置问题**

统一通过订阅服务器推送配置可以有效地集中管理资源，所以省去每个应用都来配置cookie，简化cookie的管理。如果应用需要使用一个新增的cookie，则可以通过统一的平台来申请，申请通过后才将这个配置项增加到订阅服务器。如果使用全局cookie，那么只需要将这个cookie通过订阅服务器统一推送过去就行了，省去了每个应用中手动增加cookie的配置。

**3.4.2 存储问题**

共享这些session必须将他们存储在一个分布式缓存中，可以随时的读入和存取，而且性能要很好才能满足要求。

**3.4.3 如何存取session和cookie**

既然是分布式session的处理框架，必然会重新实现httpSession的接口操作，使得应用操作Session的对象都是我们实现的InnerHttpSession对象，这个操作必须在我们进入应用前完成，所以我们可以配置一个filter拦截用户的请求。

在请求到达mvc框架之前封装httpServletRequest和httpServletResponse对象，碧昂创建我们子集的InnerHttpSession对象，我们把他设置到request和response对象中。这样应用通过request.getHttpSession()返回的就是我们创建的InnerHttpSession对象了，我们可以拦截response的addCookies设置的Cookie。

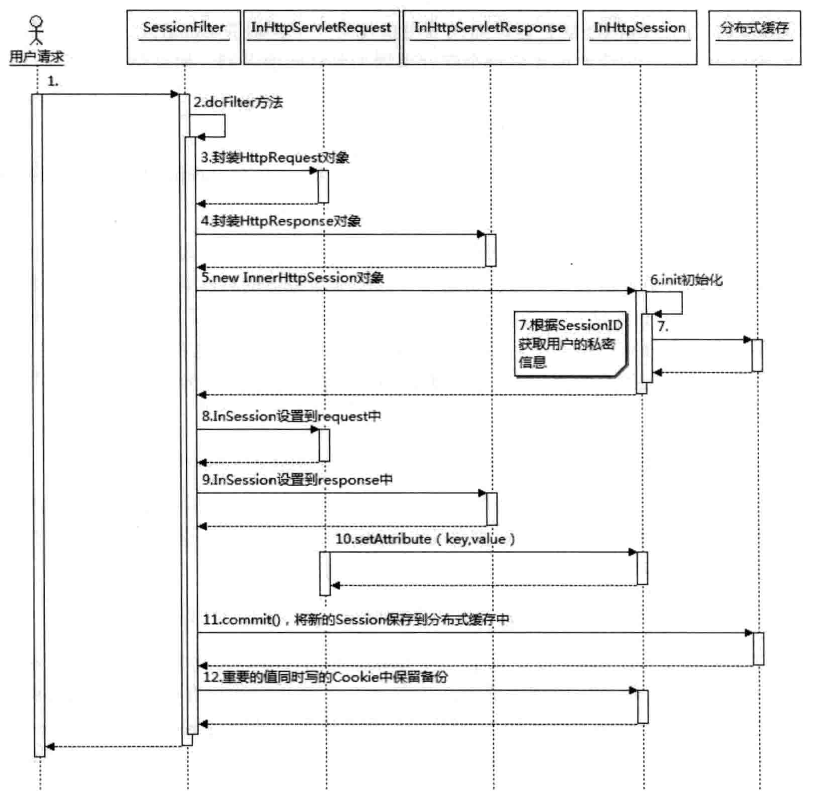


图5 HttpSession拦截请求时序图

**3.4.4 跨域名共享cookie问题**

要实现Session同步，需要另外一个跳转应用，这个应用可以背一个或者多个域名访问，他的主要功能是从一个域名下取得SessionID，然后将这个sessionID同步到另一个域名下。这个sessionID其实就相当于一个Cookie，所以要实现两个域名下的session同步，必须将用一个sessionID作为cookie写到两个域名下。