**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，它是通过服务器来保持状态的，使用上比Cookie简单一些，相应的也增加了服务器的存储压力。**

**由于Session这个词汇包含的语义很多，因此需要在这里明确一下 Session的含义。首先，我们通常都会把Session翻译成会话，因此我们可以把客户端浏览器与服务器之间一系列交互的动作称为一个 Session。从这个语义出发，我们会提到Session持续的时间，会提到在Session过程中进行了什么操作等等；**

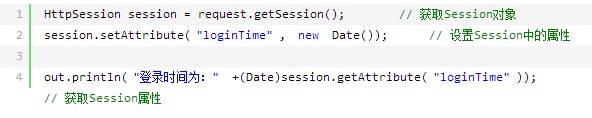
**其次，Session指的是服务器端为客户端所开辟的存储空间，在其中保存的信息就是用于保持状态。从这个语义出发，我们则会提到往Session中存放什么内容，如何根据键值从 Session中获取匹配的内容等。**

**要使用Session，第一步当然是创建Session了。那么Session在何时创建呢？而在Java中，当客户端第一次访问Servlet或jsp页面的时候自动创建(是否一定会创建了,不一定)，通过调用HttpServletRequest的getSession方法可以获得session对象。**

**在创建了Session的同时，服务器会为该Session生成唯一的Session id，而这个Session id在随后的请求中会被用来重新获得已经创建的Session；在Session被创建之后，就可以调用Session相关的方法往Session中增加内容了，而这些内容只会保存在服务器中，发到客户端的只有Sessionid(放在Cookie文件里)；当客户端再次发送请求的时候，会将这个Sessionid带上，服务器接受到请求之后就会依据Sessionid找到相应的Session，从而再次使用之。(默认情况下session也要依赖cookie机制来实现)**

**Session相当于程序在服务器上建立的一份客户档案，会给客户自动分配一个编号id，客户来访的时候只需要提供自己的id,就可以查询对应客户档案内容**

**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，例如：**



**request还可以使用getSession(boolean create)来获取Session。区别是如果该客户的Session不存在，request.getSession()方法会返回null，而getSession(true)会先创建Session再将Session返回。**

**Servlet中必须使用request来编程式获取HttpSession对象，而JSP中内置了Session隐藏对象，可以直接使用。如果使用声明了<%@page session="false" %>，则Session隐藏对象不可用。**

**注意：当多个客户端执行程序时，服务器会保存多个客户端的Session。获取Session的时候也不需要声明获取谁的Session。Session机制决定了当前客户只会获取到自己的Session，而不会获取到别人的Session。各客户的Session也彼此独立，互不可见。**

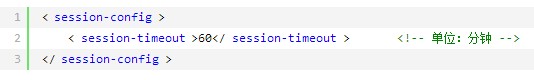
**提示：Session的使用比Cookie方便，但是过多的Session存储在服务器内存中，会对服务器造成压力。**

**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。**

**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。**

**Tomcat中Session的默认超时时间为20分钟。通过setMaxInactiveInterval(int seconds)修改超时时间。也可以修改web.xml改变Session的默认超时时间。例如修改为60分钟：**

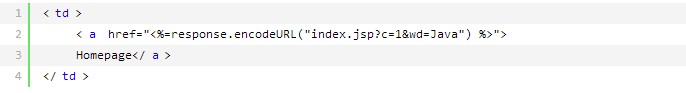


**在server.xml中定义context时采用如下定义（单位为秒）：**

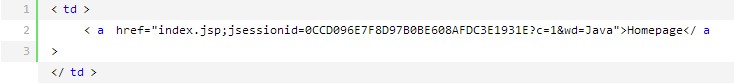


**URL地址重写**

**URL地址重写是对客户端不支持Cookie的解决方案(正常情况下Seesion要依靠Cookie来识别的)。URL地址重写的原理是将该用户Session的id信息重写到URL地址中。服务器能够解析重写后的URL获取Session的id。这样即使客户端不支持Cookie，也可以使用Session来记录用户状态。HttpServletResponse类提供了encodeURL(Stringurl)实现URL地址重写，例如：**

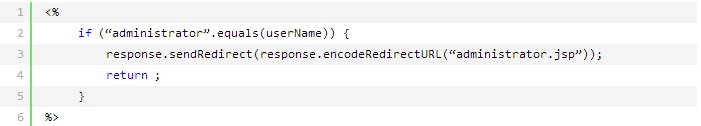


**该方法会自动判断客户端是否支持Cookie。如果客户端支持Cookie，会将URL原封不动地输出来。如果客户端不支持Cookie，则会将用户Session的id重写到URL中。重写后的输出可能是这样的：**



**即在文件名的后面，在URL参数的前面添加了字符串“;jsessionid=XXX”。其中XXX为Session的id。分析一下可以知道，增添的jsessionid字符串既不会影响请求的文件名，也不会影响提交的地址栏参数。用户单击这个链接的时候会把Session的id通过URL提交到服务器上，服务器通过解析URL地址获得Session的id。**

**如果是页面重定向（Redirection），URL地址重写可以这样写：**



**由于大部分的手机浏览器都不支持Cookie，程序都会采用URL地址重写来跟踪用户会话。索性禁止Session使用Cookie，统一使用URL地址重写会更好一些。Java Web规范支持通过配置的方式禁用Cookie。下面举例说一下怎样通过配置禁止使用Cookie。**

**打开项目根目录下的META-INF文件夹（跟WEB-INF文件夹同级，如果没有则创建），打开context.xml（如果没有则创建），编辑内容如下： /META-INF/context.xml：**



**部署后TOMCAT便不会自动生成名JSESSIONID的Cookie，Session也不会以Cookie为识别标志，而仅仅以重写后的URL地址为识别标志了**

**注意：该配置只是禁止Session使用Cookie作为识别标志，并不能阻止其他的Cookie读写。也就是说服务器不会自动维护名为JSESSIONID的Cookie了，但是程序中仍然可以读写其他的Cookie。**

**实现用户登录否的判断!**