# 转发和重定向

## 应用背景

|  |  |  |  |
| --- | --- | --- | --- |
| 类别 | 概念 | 共享数据 | 应用 |
| Redict重定向  (浏览器URL改变) | 在**客户端**执行，重新加载一次新的URL,是一次新的请求。 | 本质上是**两个请求**，不能共享request中的数据。 | 一般用于用户注销登录返回主页面或者跳往站外网站。 |
| Forward转发  (浏览器URL不改变) | 在**服务器端**执行，转发到其他的web组件，可以是servlet或者jsp，请求为同一个。 | 本质上是**一个请求**，可以共享reuqest中的数据。 | 一般用于用户登录后根据角色转发到响应的模块。 |

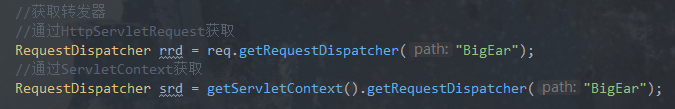
## 内部转发

### 转发过程

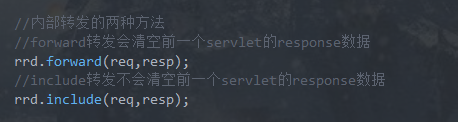
1. 客户浏览器发送 http 请求，web 服务器接受此请求。
2. 调用内部的一个方法在容器内部完成请求处理和转发动作，将请求内容和Attribute传输到下一个处理类中进行处理。
3. 将处理完结果及资源返回给浏览器，浏览器渲染返回结果。

### 具体实现

1. 获取转发器的两种方式



1. 转发的两种方式



### 实例





## 重定向

### 重定向过程

1. 客户浏览器发送 http 请求,服务器接收请求。
2. 服务器返回 302 状态码响应及对应新的 url地址给客户浏览器。
3. 客户浏览器发现是 302 响应，则自动再发送一个新的 http 请求，请求 url 是新的 location 地址。
4. 服务器根据此请求寻找资源并发送给客户，浏览器的地址会发生改变。

### 具体实现

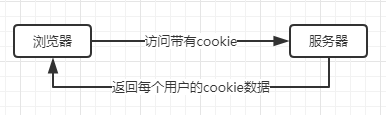


1. 重定向方法为HttpServletResponse.sendRedirect,地址必须为完全的URL，这个和转发不一样。

# Servlet读写Cookie

## 应用背景

1. 会话：用户开一个浏览器，点击多个超链接，访问服务器多个 Web 资源，然后关闭浏览器，整个过程称之为会话。
2. Cookie 是客户端技术，程序把每个用户的数据以 Cookie 的形式写给用户各自的浏览器。当用户使用浏览器再去访问服务器中的 Web 资源时，就会带着各自的数据去。这样，Web 资源处理的就是用户各自的数据了。



## 服务器端Cookie的基本使用

### Servlet中的Cookie类

public class Cookie implements Cloneable, Serializable

1. name：一个cookie的名称,只有getName()方法，只能通过构造方法赋值。
2. value：一个cookie的值，有setValue/getValue方法。
3. domain：可以访问此cookie的域名，有setDomain/getDomain方法。
4. path：可以访问此cookie的页面路径，有setPath/getPath方法。
5. http：Cookie的httponly属性，若此属性为True，则只有在http请求头中会有此Cookie信息，而不能通过document.cookie来访问此cookie。
6. secure：设置是否只能通过https来传递此条cookie。
7. expires/Max-Age：设置cookie超时时间。如果设置的值为一个时间，则当到达该时间时此cookie失效。不设置的话默认是session会话时间，意思是cookie会和session一起失效，当浏览器关闭（并不是浏览器标签关闭，而是整个浏览器关闭）后，cookie失效。
8. Servlet 设置 Cookie 后，默认 Cookie 的有效路径为 Servlet 所在目录。如：访问/servlet/ServletDemo01，在 ServletDemo01 中设置返回 Cookie，则该 Cookie 的默认有效路径为 /servlet，即当用户访问 /servlet 下的 Servlet 时，浏览器才会发送 Cookie 过去。

### 获取/返回Cookie





### Cookie要点

1. 一个Cookie 只能标识一种信息，它至少含有一个标识该信息的名称和设置值。返回同名的Cookie会覆盖之前的Cookie值。
2. 服务器端可以给一个浏览器发送多个Cookie，一个浏览器也可以存储多个服务器端提供的 Cookie。
3. 浏览器一般只允许存放 300 个 Cookie，每个站点最多存放 20 个 Cookie，每个 Cookie 的大小限制为 4kb。
4. 如果创建了一个 Cookie，并将它发送到浏览器，默认情况下它是一个会话级别的 Cookie（即存储在浏览器的内存中），用户退出浏览器之后即被删除。若希望浏览器将该 Cookie 存储在磁盘上，则需要设置 maxAge，并给出一个以秒为单位的时间。将最大失效设置为 0，则是命令浏览器删除该 Cookie。
5. 如果要删除Cookie，客户端直接清理缓存即可。服务器端不能直接删除客户端的Cookie，需要先获取客户端的Cookie，再将这个Cookie的maxAge设置为0，让客户端的cookie过期即可。另外要设置Cookie对应的Path,否则不会生效。

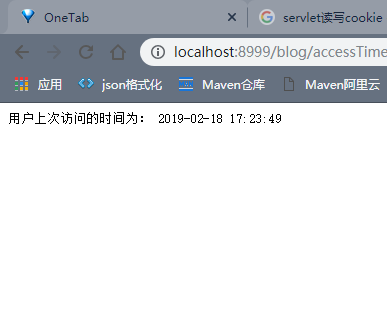
## 具体实例

### 利用Cookie记录用户上次登录时间

代码：



效果：



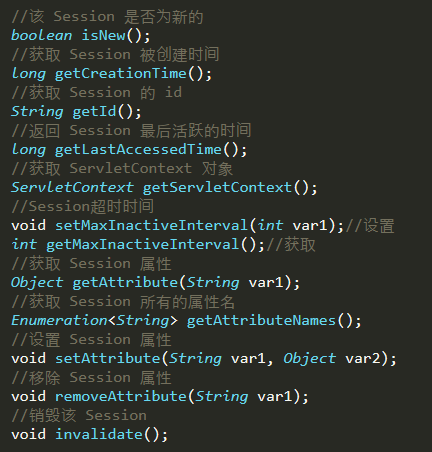
# Servlet中Session的使用

## 应用背景

### Session技术介绍

1. Session 是另一种记录浏览器状态的机制。不同的是 Cookie 保存在浏览器中，Session 保存在服务器中。用户使用浏览器访问服务器的时候，服务器把用户的信息以某种的形式记录在服务器，这就是 Session。
2. 如果说 Cookie 是检查用户身上的”通行证 “来确认用户的身份，那么 Session 就是通过检查服务器上的” 客户明细表“来确认用户的身份的。Session 相当于在服务器中建立了一份“客户明细表”。
3. Session 比 Cookie 使用方便，Session 可以解决 Cookie 解决不了的事情。Session 可以存储对象，Cookie 只能存储字符串。

### HttpSession 接口（API）



1. HttpSession只是一个接口，其具体实现是有服务器来完成的

比如tomcat，在tomcat的src中java\org\apache\catalina\session下的类StandardSession实现了HttpSession。

## 基本使用

### Session的生命周期

1. Session 在用户第一次访问服务器 Servlet，jsp 等**动态资源**就会被自动创建，Session 对象保存在内存里，如果访问 HTML,IMAGE 等**静态资源** Session 不会被创建。
2. 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，无论是否对 Session 进行读写，服务器都会认为 Session 活跃了一次。
3. 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。为了防止内存溢出，服务器会把长时间没有活跃的 Session 从内存中删除，这个时间也就是 Session 的超时时间,超时时间默认为30分钟。

### 获取Session

1. 无参数返回方法

public HttpSession getSession()

获取当前request关联的session,如果当前request没有session,创建一个session.

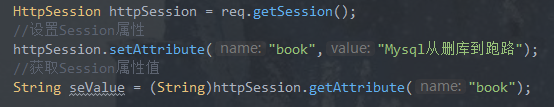
1. 有参数返回方法

public HttpSession getSession(boolean create);

如果为true，则和无参数返回方法一样。如果为false，当前session为空返回full，不为空返回当前session。

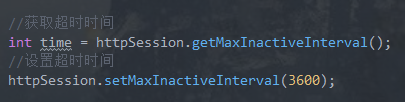
### Session相关属性

1. 设置属性

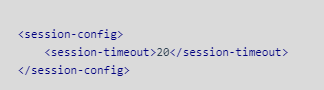


1. 超时时间

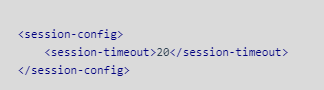
在servlet中设置，对该会话session有效



在web.xml中设置，对该web应用有效



在tomcat中conf/web.xml设置，对该tomcat下所有的web应用有效



(3) session的超时时间是指不活动时间，比如连续不活动30分钟才会销毁，如果中间重 新活动一次，则会重新计时。

### 2.4 销毁Session



### Session识别同一个浏览器的原理

1. 由于HTTP协议是无状态的协议，所以服务端需要记录用户的状态时，就需要用某种机制来识具体的用户，这个机制就是Session. 在服务端保存Session的方法很多，内存、数据库、文件都有。集群的时候也要考虑Session的转移，在大型的网站，一般会有专门的Session服务器集群，用来保存用户会话，这个时候 **Session 信息都是放在内存的**，使用一些缓存服务比如Memcached之类的来放 Session。
2. 服务端识别特定用户的原理：每次HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端。实际上大多数的应用都是用 Cookie 来实现Session跟踪的，第一次创建Session的时候，服务器向用户浏览器发送了一个名为 JESSIONID 的 Cookie，它的值是Session的ID值。。之后每次请求都会在cookie中带上这个sessionID，以便服务端识别该用户。
3. 该 Cookie 是服务器自动颁发给浏览器的，不用我们手工创建的。该 Cookie 的 maxAge 值默认是 - 1，也就是说仅当前浏览器使用，不将该 Cookie 存在硬盘中。
4. 如果浏览器禁用了Cookie，会使用一种叫做URL重写的技术来进行会话跟踪，即每次HTTP交互，URL后面都会被附加上一个诸如 sid=xxxxx 这样的参数，服务端据此来识别用户。
5. 重写URL

HttpServletResponse 类提供了两个 URL 地址重写的方法：

encodeURL(String url)和encodeRedirectURL(String url)

需要值得注意的是：这两个方法会自动判断该浏览器是否支持 Cookie，如果支持 Cookie，重写后的 URL 地址就不会带有 jsessionid 了。



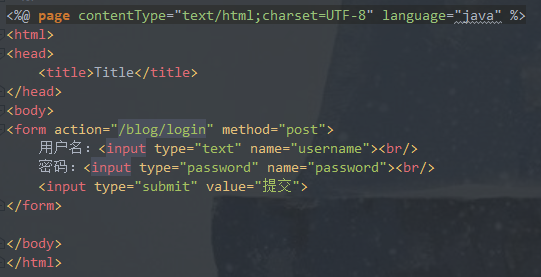
一般需要前一个servlet的respose来跳转到下一个servlet，这样才能进行url重写，如果是两次独立的访问不同的servlet，url重写也不能成功。

## 具体实例

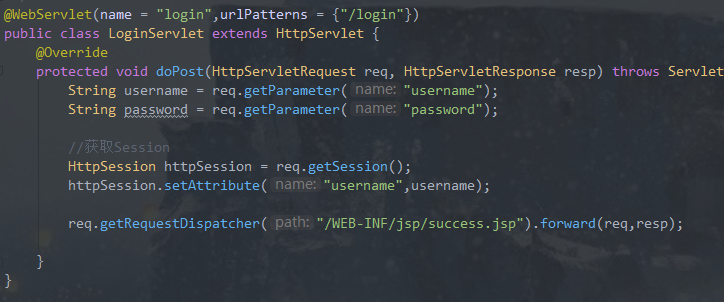
### Session实现用户的登录

1. 访问servlet跳转到登录页面(位于WEB-INF中的jsp页面)





1. 登录页面提交到对应的servlet，将用户信息写入session

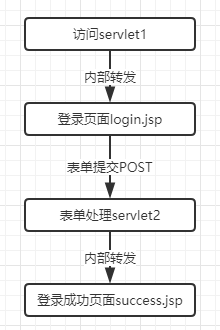


1. 登录成功后跳转到成功页面，并且读取session中的数据



### 3.2 Session-Token防止表单重复提交

3.2.1 表单重复提交的几种情况



1. 在登录页面重复点击 提交 按钮，可通过js解决。
2. 在成功页面不断刷新，导致表单重复提交。
3. 登录成功后，又回退到登录页面，重新提交表单。

3.2.2 利用token解决重复提交

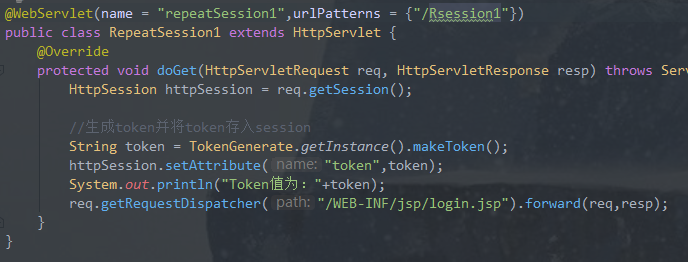
(1) 登录页面需要通过servlet1转发，在servlet1中生成一个唯一的随机标识号，专业术语称为Token(令牌),并保存到session中。

(2) 将该token保存到登录页面的表单隐藏域中，提交表单时将token一起提交。

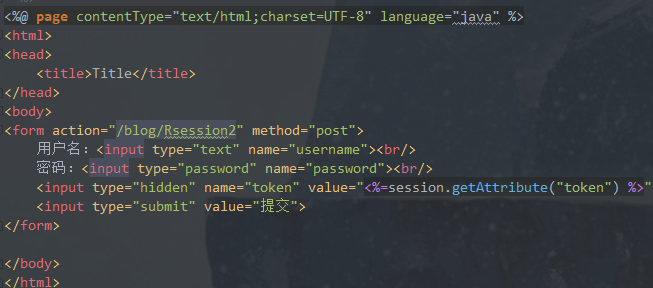
(3) 进入表单处理servlet2，判断服务器session中token是否与表单提交的token相等，如果相等则为第一次提交，并移除服务端session中的token。

(4) 如果服务端session中的token为空，或者浏览器表单中token为空，或者服务端与浏览器的token不相等，则判定为重复提交。

servlet1



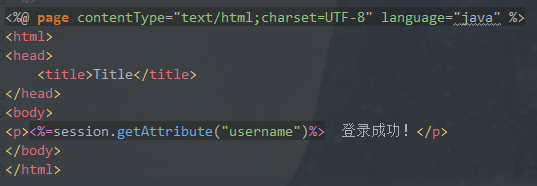
login.jsp



servlet2



success.jsp



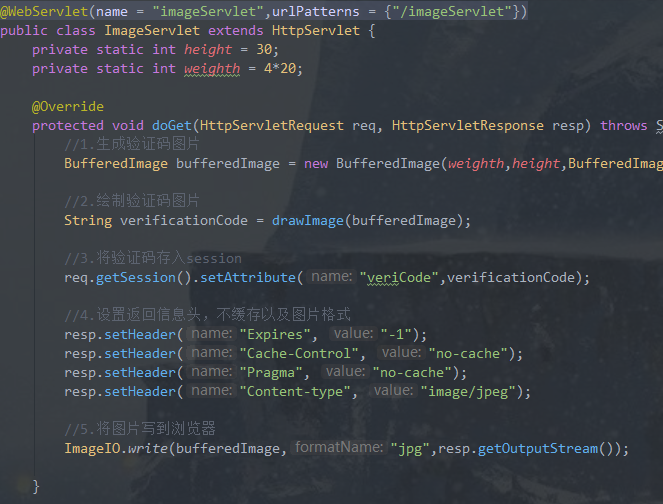
token生成类（单例模式）



### 一次性校验码

1. 在servlet生成验证码图片，并将验证码存入session，将图片输出给网页。
2. 表单提交后，比较浏览器输入验证码和session中验证码值。

验证码生成servlet



登录页面



登录处理servlet



## Session和Cookie区别

4.1 从存储方式上比较

(1) Cookie 只能存储字符串，如果要存储非 ASCII 字符串还要对其编码。

(2) Session 可以存储任何类型的数据，可以把 Session 看成是一个容器

4.2 从隐私安全上比较

1. Cookie 存储在浏览器中，对客户端是可见的。信息容易泄露出去。如果使用 Cookie，最好将 Cookie 加密
2. Session 存储在服务器上，对客户端是透明的。不存在敏感信息泄露问题。

4.3 从有效期上比较

1. Cookie 保存在硬盘中，只需要设置 maxAge 属性为比较大的正整数，即使关闭浏览器，Cookie 还是存在的
2. Session 的保存在服务器中，设置 maxInactiveInterval 属性值来确定 Session 的有效期。并且 Session 依赖于名为 JSESSIONID 的 Cookie，该 Cookie 默认的 maxAge 属性为 - 1。如果关闭了浏览器，该 Session 虽然没有从服务器中消亡，但也就失效了。

4.4 从对服务器的负担比较

1. Session 是保存在服务器的，每个用户都会产生一个 Session，如果是并发访问的用户非常多，是不能使用 Session 的，Session 会消耗大量的内存。
2. Cookie 是保存在客户端的。不占用服务器的资源。像 baidu、Sina 这样的大型网站，一般都是使用 Cookie 来进行会话跟踪。

4.5 从浏览器的支持上比较

1. 如果浏览器禁用了 Cookie，那么 Cookie 是无用的了！
2. 如果浏览器禁用了 Cookie，Session 可以通过 URL 地址重写来进行会话跟踪。

4.6 从跨域名上比较

1. Cookie 可以设置 domain 属性来实现跨域名
2. Session 只在当前的域名内有效，不可夸域名

# HTTP协议头部字段应用