## 会话技术

基本概念: 指用户开一个浏览器，**访问一个网站,只要不关闭该浏览器，不管该用户点击多少个超链接，访问多少资源，直到用户关闭浏览器，整个这个过程我们称为一次会话**.。

每个用户在使用浏览器与服务器进行会话的过程中，不可避免各自会产生一些数据，**程序要想办法为每个用户保存这些数据。**

**保存会话数据的两种技术：**

**Cookie**

**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。**

**Session**

**Session是服务器端技术，利用这个技术，服务器在运行时可以为每一个用户的浏览器创建一个其独享的session对象，由于session为用户浏览器独享，所以用户在访问服务器的web资源时，可以把各自的数据放在各自的session中，当用户再去访问服务器中的其它web资源时，其它web资源再从用户各自的session中取出数据为用户服务。**

**为什么我们要使用会话技术？**

在论坛登陆的时候，很多时候会有一个小框框问你是否要自动登陆，当你下次登陆的时候就不用输入密码了

根据我以前浏览过的商品，猜我喜欢什么商品

### Session和Cookie的区别

* **从存储方式上比较**
  + Cookie只能存储字符串，**如果要存储非ASCII字符串还要对其编码。**
  + Session可以存储任何类型的数据，可以把Session看成是一个容器
* **从隐私安全上比较**
  + **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**
  + **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。
* **从有效期上比较**
  + 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
  + **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。**
* **从对服务器的负担比较**
  + 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。
  + 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。
* **从浏览器的支持上比较**
  + 如果浏览器禁用了Cookie，那么Cookie是无用的了！
  + 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。**
* **从跨域名上比较**
  + Cookie可以设置**domain属性**来实现跨域名
  + Session只在当前的域名内有效，不可夸域名

**Session、Cookie**

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。

服务器就**按照sessionid把这个session检索出来使用；**如果客户端请求不包含sessionid，则为此客户端创建一个session并且生成一个与此session相关联的sessionid，**并将这个sessionid在本次响应中返回给客户端保存。**

### Cookie和Session共同使用

* **如果仅仅使用Cookie或仅仅使用Session可能达不到理想的效果**。这时应该尝试一下同时使用Session和Cookie
* 那么，什么时候才需要同时使用Cookie和Session呢？
* 在上一篇博客中，我们使用了Session来进行简单的购物，功能也的确实现了。现在有一个问题：**我在购物的途中，不小心关闭了浏览器。当我再返回进去浏览器的时候，发现我购买过的商品记录都没了！！为什么会没了呢？原因也非常简单：服务器为Session自动维护的Cookie的maxAge属性默认是-1的，当浏览器关闭掉了，该Cookie就自动消亡了。当用户再次访问的时候，已经不是原来的Cookie了。**
* 我们现在想的是：**即使我不小心关闭了浏览器了，我重新进去网站，我还能找到我的购买记录**。
* 要实现该功能也十分简单，问题其实就在：服务器为Session自动维护的Cookie的maxAge属性是-1，**Cookie没有保存在硬盘中。**我现在要做的就是：**把Cookie保存在硬盘中，即使我关闭了浏览器，浏览器再次访问页面的时候，可以带上Cookie，从而服务器识别出Session。**
* **第一种方式：**只需要在处理购买页面上创建Cookie，Cookie的值是Session的id返回给浏览器即可

        Cookie cookie = new Cookie("JSESSIONID",session.getId());  
        cookie.setMaxAge(30\*60);

cookie.setPath("/ouzicheng/");

response.addCookie(cookie);

* **第二种方式：** **在server.xml文件中配置，将每个用户的Session在服务器关闭的时候序列化到硬盘或数据库上保存**。但此方法不常用，知道即可！

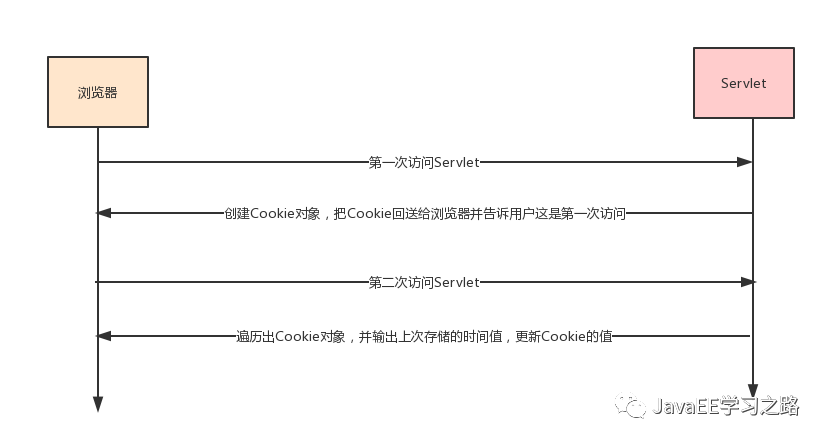
# Cookie

## 什么是Cookie

* 网页之间的交互是通过HTTP协议传输数据的，而**Http协议是无状态的协议**。无状态的协议是什么意思呢？一旦数据提交完后，浏览器和服务器的连接就会关闭，再次交互的时候需要重新建立新的连接。
* 服务器无法确认用户的信息，于是乎，W3C就提出了：给每一个用户都发一个通行证，无论谁访问的时候都需要携带通行证，这样**服务器就可以从通行证上确认用户的信息。通行证就是Cookie**

**Cookie 的作用呢？**

通俗的说就是当一个用户通过 HTTP 访问一个服务器时，这个服务器会将一些**Key/Value 键值对**返回给客户端浏览器，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，数据又被完整地带回给服务器。

**可以在电脑里找到COOKIE的位置。**

**Cookie的流程：浏览器访问服务器，如果服务器需要记录该用户的状态，就使用response向浏览器发送一个Cookie，浏览器会把Cookie保存起来。当浏览器再次访问服务器的时候，浏览器会把请求的网址连同Cookie一同交给服务器。**

**Cookie注意细节**

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（NAME）和设置值（VALUE）。**
2. **一个WEB站点可以给一个WEB浏览器发送多个Cookie，一个WEB浏览器也可以存储多个WEB站点提供的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，并将他发送到浏览器，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，则需要使用maxAge，并给出一个以秒为单位的时间。将最大时效设为0则是命令浏览器删除该cookie。**
5. **删除cookie的时候，path必须一致，否则不会删除。**

## Cookie的API

|  |  |  |  |
| --- | --- | --- | --- |
| Cookie类的主要方法 | | | |
| No. | 方法 | 类型 | 描述 |
| 1 | [Cookie](https://www.cnblogs.com/xdp-gacl/p/3803033.html#Cookie(java.lang.String,%20java.lang.String))([String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html) name, [String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html) value) | 构造方法 | 实例化Cookie对象，传入cooke名称和cookie的值 |
| 2 | public [String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html) getName() | 普通方法 | 取得Cookie的名字 |
| 3 | public [String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html) getValue() | 普通方法 | 取得Cookie的值 |
| 4 | public void setValue([String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html) newValue) | 普通方法 | 设置Cookie的值 |
| 5 | public void setMaxAge(int expiry) | 普通方法 | 设置Cookie的最大保存时间，即cookie的有效期。 |
| 6 | public int getMaxAge() | 普通方法 | 获取Cookies的有效期 |
| 7 | public void setPath([String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html) uri) | 普通方法 | 设置cookie的有效路径，比如把cookie的有效路径设置为"/xdp"，那么浏览器访问"xdp"目录下的web资源时，都会带上cookie，再比如把cookie的有效路径设置为"/xdp/gacl"，那么浏览器只有在访问"xdp"目录下的"gacl"这个目录里面的web资源时才会带上cookie一起访问，而当访问"xdp"目录下的web资源时，浏览器是不带cookie的 |
| 8 | public [String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html) getPath() | 普通方法 | 获取cookie的有效路径 |
| 9 | public void setDomain([String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html) pattern) | 普通方法 | 设置cookie的有效域 |
| 10 | public String getDomain() | 普通方法 | 获取cookie的有效域 |

response接口也中定义了一个**addCookie方法**，它用于在其响应头中增加一个**相应的Set-Cookie头字段**。 同样，request接口中也定义了一个getCookies方法，它用于获取客户端提交的Cookie。

## 简单使用Cookie

### 使用cookie记录用户上一次访问的时间

public class CookieDemo01 extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

**//设置服务器端以UTF-8编码进行输出**

**response.setCharacterEncoding("UTF-8");**

**//设置浏览器以UTF-8编码进行接收,解决中文乱码问题**

**response.setContentType("text/html;charset=UTF-8");**

PrintWriter out = response.getWriter();

//获取浏览器访问访问服务器时传递过来的cookie数组

**Cookie[] cookies = request.getCookies();**

//如果用户是第一次访问，那么得到的cookies将是null

if (cookies!=null) {

out.write("您上次访问的时间是：");

for (int i = 0; i < cookies.length; i++) {

Cookie cookie = cookies[i];

if (cookie.getName().equals("lastAccessTime")) {

Long lastAccessTime =Long.parseLong(cookie.getValue());

Date date = new Date(lastAccessTime);

out.write(date.toLocaleString());

}

}

}else {

out.write("这是您第一次访问本站！");

}

//用户访问过之后重新设置用户的访问时间，存储到cookie中，然后发送到客户端浏览器

**Cookie cookie = new Cookie**("lastAccessTime", System.currentTimeMillis()+"");//创建一个cookie，cookie的名字是lastAccessTime

//将cookie对象添加到response对象中，这样服务器在输出response对象中的内容时就会把cookie也输出到客户端浏览器

**response.addCookie(cookie);**

}

在上面的例子中，在程序代码中并没有使用setMaxAge方法设置cookie的有效期，所以当关闭浏览器之后，cookie就失效了，要想在关闭了浏览器之后，cookie依然有效，那么在创建cookie时，就要为cookie设置一个有效期。

//用户访问过之后重新设置用户的访问时间，存储到cookie中，然后发送到客户端浏览器

Cookie cookie = new Cookie("lastAccessTime", System.currentTimeMillis()+"");//创建一个cookie，cookie的名字是lastAccessTime

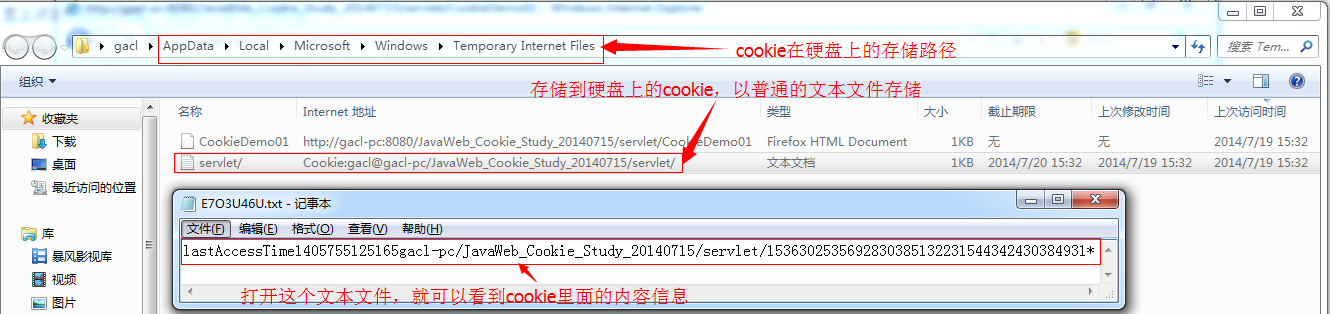
//设置Cookie的有效期为1天

cookie.setMaxAge(24\*60\*60);

//将cookie对象添加到response对象中，这样服务器在输出response对象中的内容时就会把cookie也输出到客户端浏览器

response.addCookie(cookie);

用户第一次访问时，服务器发送给浏览器的cookie就存储到了硬盘上，如下所示：



这样即使关闭了浏览器，下次再访问时，也依然可以通过cookie获取用户上一次访问的时间。

### Cookie保存中文

response.setContentType("text/html;charset=UTF-8");  
 PrintWriter printWriter = response.getWriter();  
 String name = "中国";  
 Cookie cookie = **new** Cookie("country", name);  
 cookie.setMaxAge(2000);  
 response.addCookie(cookie);  
 printWriter.write("我颁发了一个Cookie，值保存的是中文数据");

**访问Servlet1，好吧。出异常了！**

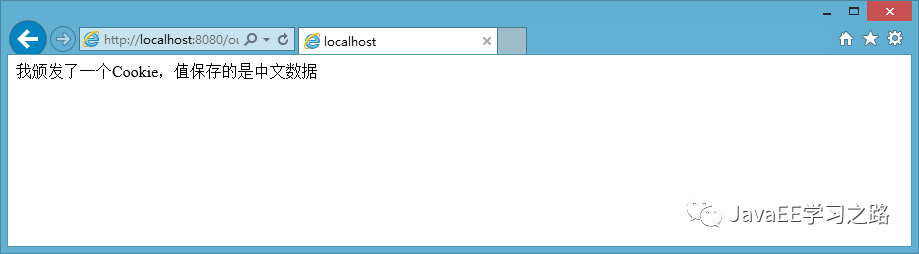
* 中文属于Unicode字符，英文数据ASCII字符，中文占4个字符或者3个字符，英文占2个字符。
* **Cookie 中的 key 和 value 不支持中文设置 Cookie 时需要对中文字符串进行编码**
* 解决：Cookie使用Unicode字符时需要对Unicode字符进行编码。

//对Unicode字符进行编码 **Cookie cookie = new Cookie("country", URLEncoder.encode(name, "UTF-8"));**

response.addCookie(cookie);

**使用URLEncoder类里面的encode(**[**String**](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLEncoder.class%E2%98%83URLEncoder~encode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String)**s,**[**String**](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLEncoder.class%E2%98%83URLEncoder~encode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String)**enc)方法进行中文转码**

* 再次访问Servlet1，已经把Cookie成功颁发给浏览器了

****

****

* **我们发现Cookie保存在硬盘的中文数据是经过编码的，那么我们在取出Cookie的时候要对中文数据进行解码**

Cookie[] cookies = request.getCookies();

for (int i = 0; cookies != null && i < cookies.length; i++) {

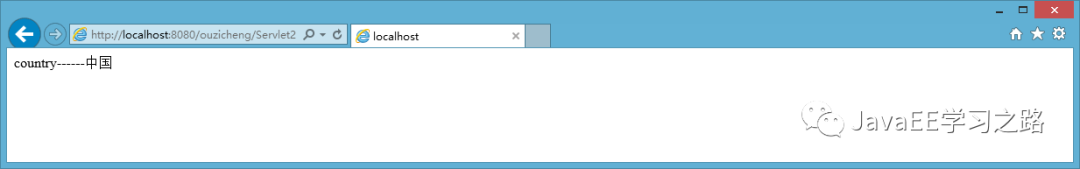
String name = cookies[i].getName();

//经过URLEncoding就要URLDecoding

**String value = URLDecoder.decode(cookies[i].getValue(), "UTF-8");**

printWriter.write(name + "------" + value);

}

****

### Cookie 的路径

Cookie 在创建的时候，会根据当前的Servlet的相对路径来设置自己的路径，比如 Servlet 的url-pattern为 /cookie/login，**相对路径则为：/cookie/**

* 出现的问题：  
  只有在访问路径为 /cookie/ 下面的资源的时候，**才会将该 Cookie 发送到服务器**
* 解决方案：  
  设置 Cookie 的路径：void setPath(String uri)  
  **Cookie对象.setPath("/");**表示当前应用中的所有的资源都能够共享该Cookie信息

### Cookie的有效期

**Cookie的有效期是通过setMaxAge()来设置的。**

* 如果MaxAge为正数，浏览器会把Cookie写到硬盘中，只要还在MaxAge秒之前，登陆网站时该Cookie就有效【不论关闭了浏览器还是电脑】
* 如果MaxAge为负数，\*\*Cookie是临时性的，仅在本浏览器内有效，关闭浏览器Cookie就失效了，Cookie不会写到硬盘中。Cookie默认值就是-1。这也就为什么在我第一个例子中，如果我没设置Cookie的有效期，在硬盘中就找不到对应的文件。
* **如果MaxAge为0，则表示删除该Cookie。**Cookie机制没有提供删除Cookie对应的方法，把MaxAge设置为0等同于删除Cookie

### Cookie的修改和删除

* 上面我们已经知道了Cookie机制没有提供删除Cookie的方法。其实细心点我们可以发现，Cookie机制也没有提供修改Cookie的方法。那么我们怎么修改Cookie的值呢？
* Cookie存储的方式类似于Map集合，**键对**
* **Cookie的名称相同，通过response添加到浏览器中，会覆盖原来的Cookie。**
* 以country为名保存的是%E4%B8%AD%E5%9B%BD，下面我再以country为名，把值改变一下。

****

String name = "看完博客就点赞"; //对Unicode字符进行编码  
 Cookie cookie = new Cookie("country", URLEncoder.encode(name, "UTF-8"));

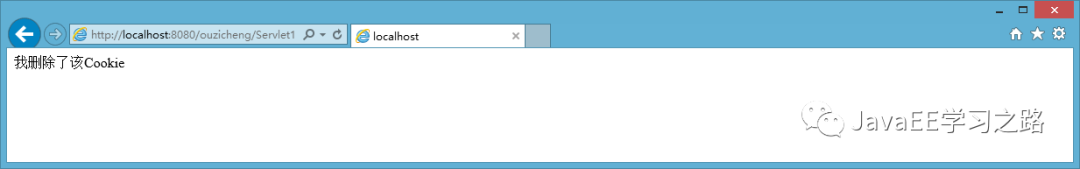
* 再次查看Cookie的时候，值已经改变了，但是文件还是那一份

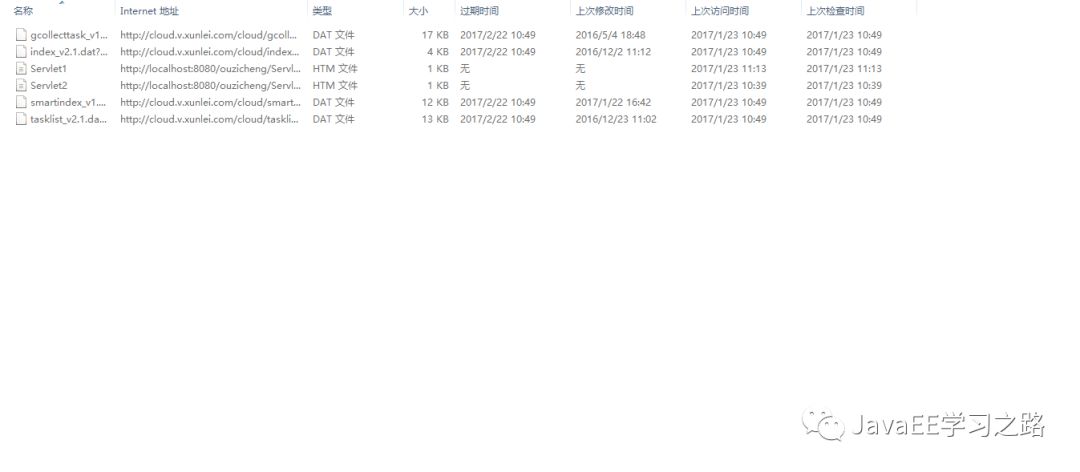
****

* 现在我要**删除该Cookie**，把MaxAge设置为0，**并添加到浏览器中即可**

String name = "看完博客就点赞";  
 //对Unicode字符进行编码  
 Cookie cookie = new Cookie("country", URLEncoder.encode(name, "UTF-8"));  
 //一定不要忘记添加到浏览器中  
 cookie.setMaxAge(0);  
 response.addCookie(cookie);  
 printWriter.write("我删除了该Cookie");

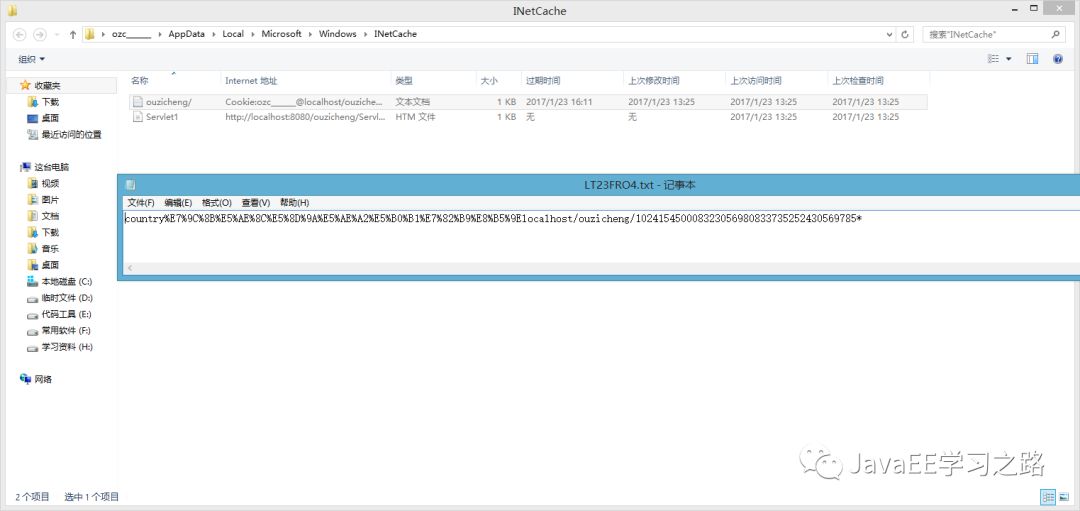
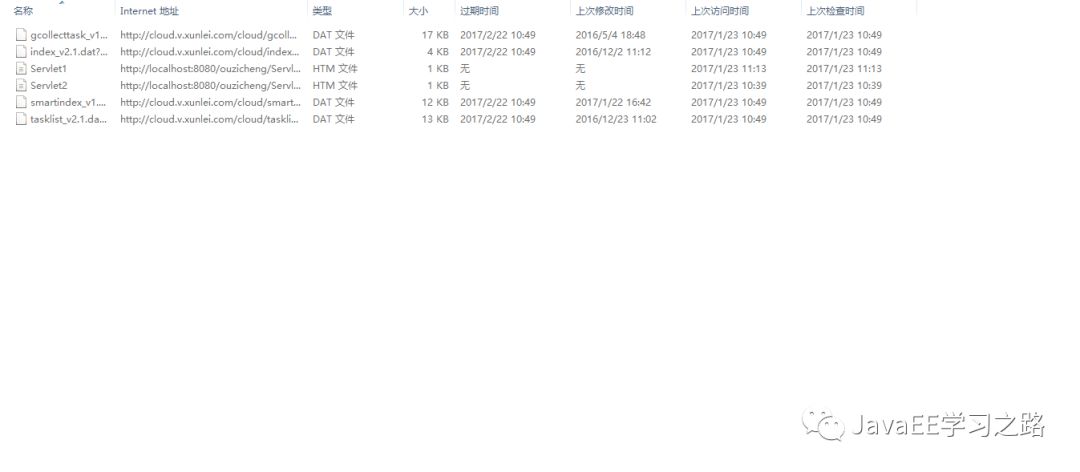
* 访问Servlet，在硬盘已经找不到Cookie的文件了！

****

****

* **注意：删除，修改Cookie时，新建的Cookie除了value、maxAge之外的所有属性都要与原Cookie相同。否则浏览器将视为不同的Cookie，不予覆盖，导致删除修改失败！**
* 我们来试验一下把。

String name = "看完博客就点赞";  
  
 //对Unicode字符进行编码  
 Cookie cookie = new Cookie("country", URLEncoder.encode(name, "UTF-8"));  
  
 //一定不要忘记添加到浏览器中  
 cookie.setMaxAge(10000);  
 response.addCookie(cookie);

****

* 上面新建了一个Cookie，我修改下Cookie的其他属性，再删除，看能否把Cookie删除掉

//一定不要忘记添加到浏览器中  
  
  **cookie.setPath("/ouzicheng");**  
 cookie.setMaxAge(0);  
 response.addCookie(cookie);  
  
 printWriter.write("删除一个Cookie");

* 结果Cookie还在硬盘中

****

### 显示上次浏览过商品

我就以书籍为例子了！**首先设计Book对象**

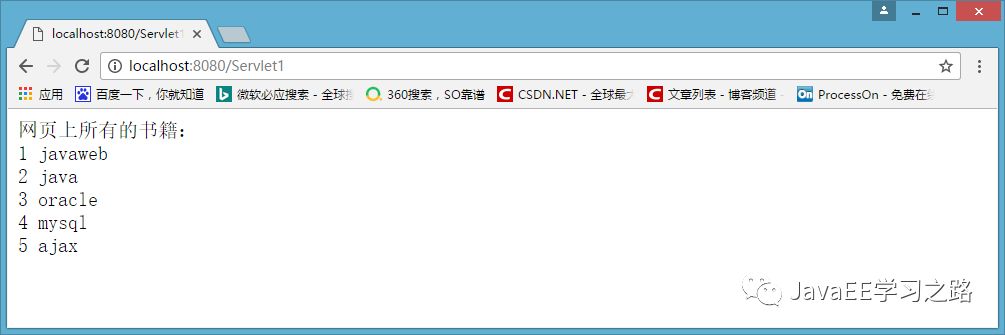
**private** String id ;  
 **private** String name ;  
 **private** String author;  
 **public** Book() { }  
 **public** Book(String id, String name, String author) {  
 **this**.id = id;  
 **this**.name = name;  
 **this**.author = author;  
 }  
 ...各种set、get方法

设计一个简单的数据库存储数据。**就用LinkedHashMap集合【根据商品的id找书籍所以用Map，删改较多所以用Linked（有序的）】**

**private** **static** LinkedHashMap<String, Book> linkedHashMap = **new** LinkedHashMap();  
 //简化开发复杂度，book的id和商品的id相同  
 **static** {  
 linkedHashMap.put("1", **new** Book("1", "javaweb", "zhong"));  
 linkedHashMap.put("2", **new** Book("2", "java", "fu"));  
 linkedHashMap.put("3", **new** Book("3", "oracle", "cheng"));  
 linkedHashMap.put("4", **new** Book("4", "mysql", "ou"));  
 linkedHashMap.put("5", **new** Book("5", "ajax", "zi"));  
 }  
 //获取到所有书籍  
 **public** **static** LinkedHashMap getAll() {  
 **return** linkedHashMap; }

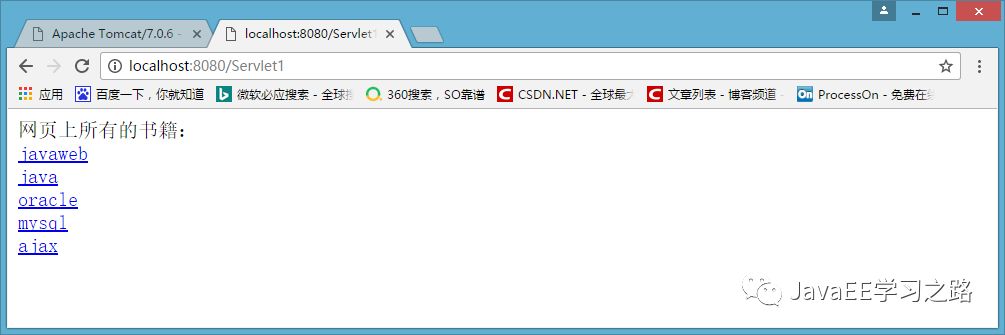
* **显示网页上所有的书籍【首页】**

printWriter.write("网页上所有的书籍："+"<br/>");  
 //拿到数据库所有的书  
 LinkedHashMap<String, Book> linkedHashMap = DB.getAll();  
 Set<Map.Entry<String, Book>> entry = linkedHashMap.entrySet();  
 //显示所有的书到网页上  
 **for** (Map.Entry<String, Book> stringBookEntry : entry) {  
 Book book = stringBookEntry.getValue();  
 printWriter.write(book.getId() +"           "+ book.getName()+"<br/>");  
 }



* 接着，我们要做的就是**给显示的书籍挂上一个超链接**，当用户点击想看的书籍时，就**跳转到该书籍的详细信息页面**
* **超链接应该把书的id传递过去，不然处理页面是不知道用户想看的是哪一本书的！**

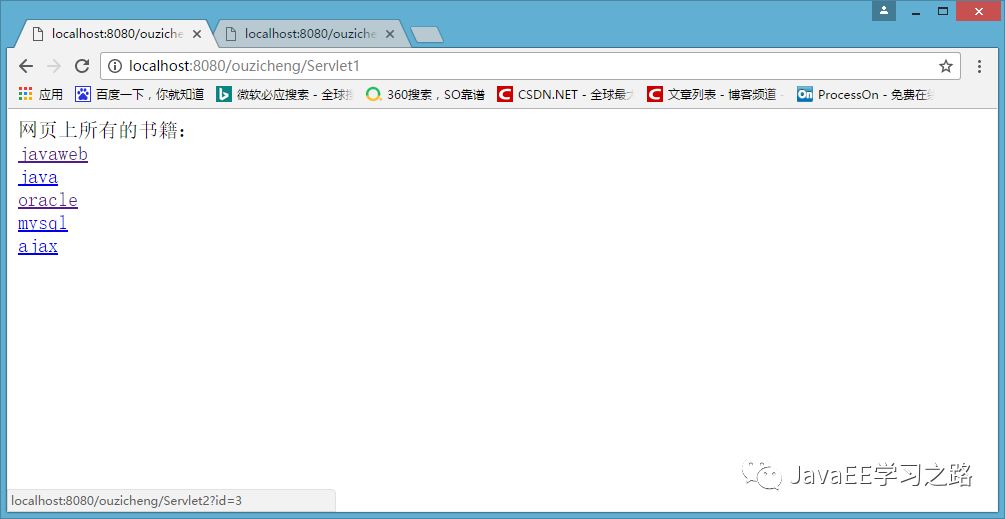
//显示所有的书到网页上  
 **for** (Map.Entry<String, Book> stringBookEntry : entry) {  
 Book book = stringBookEntry.getValue();  
 printWriter.write("<a href='/ouzicheng/Servlet2?id=" + book.getId() + "''target=\_blank' +" + book.getName() + "</a>");  
 printWriter.write("<br/>");  
 }



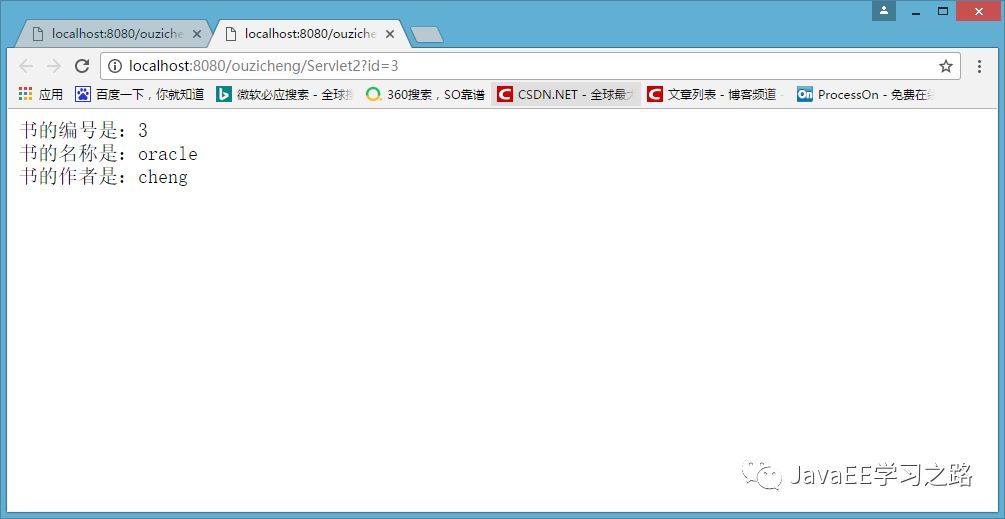
* **接收id，找到用户想要看哪一本书，输出该书的详细信息**

String id = request.getParameter("id");  
 //由于book的id和商品的id是一致的。获取到用户点击的书  
 Book book = (Book) DB.getAll().get(id);  
 //输出书的详细信息  
 printWriter.write("书的编号是：" + book.getId()+"<br/>");  
 printWriter.write("书的名称是：" + book.getName()+"<br/>");  
 printWriter.write("书的作者是：" + book.getAuthor()+"<br/>");

* 点击想要的书籍。



* 得到书籍的详细信息



* 既然用户点击了书籍，那么**服务器就应该颁发Cookie给浏览器，记住用户点击了该书籍**
* 现在问题来了，Cookie的值应该是什么呢？试想一下，**待会还要把浏览过的书籍显示出来，所以用书籍的id是最好不过的**。想到了用书籍的id作为Cookie的值，我们还要定义一些规则！
* 我们可能有非常多的书籍，不可能把用户浏览过的书籍都显示出来。所以我们定义**只能显示3本浏览过的书籍**
* 书籍的id都是数字，**如果不做任何修改，存到Cookie里边可能就是231，345，123此类的数字**，这样**取出某一个id的时候就十分费劲并且后面还要判断该书是否存在Cookie里边了**，所以我们要把存储到Cookie的书籍id分割起来。所以**我们定义”\_“作为分隔符**
* 按上面的应用，**我们的逻辑应该是：先遍历下Cookie，看下有没有我们想要的Cookie。如果找到想要的Cookie，那就取出Cookie的值**

String bookHistory = **null**;  
 Cookie[] cookies = request.getCookies();  
 **for** (**int** i = 0; cookies != **null** && i < cookies.length; i++) {  
 **if** (cookies[i].getName().equals("bookHistory")) {  
 bookHistory = cookies[i].getValue();  
 }  
 }

* 取出了Cookie的值也分几种情况
  1. Cookie的值为null【直接把传入进来的id当做是Cookie的值】
  2. Cookie的值长度有3个了【把排在最后的id去掉，把传进来的id排在最前边】
  3. Cookie的值已经包含有传递进来的id了【把已经包含的id先去掉，再把id排在最前面】
  4. Cookie的值就只有1个或2个，直接把id排在最前边

**if** (bookHistory == **null**) {  
 **return** id;  
 }  
 **//如果Cookie的值不是null的，那么就分解Cookie的得到之前的id。  
 String[] strings = bookHistory.split("\\\_");**  
  
 //为了增删容易并且还要判断id是否存在于该字符串内-----我们使用LinkedList集合装载分解出来的id  
 List list = Arrays.asList(strings);  
 LinkedList<String> linkedList = **new** LinkedList<>();  
 linkedList.addAll(list);  
 **if** (linkedList.contains(id)) {  
 linkedList.remove(id);  
 linkedList.addFirst(id);  
 }**else** {  
 **if** (linkedList.size() >= 3) {  
 linkedList.removeLast();  
 linkedList.addFirst(id);  
 } **else** {  
 linkedList.addFirst(id);  
 }  
 }

* 这么折腾完了，我们的Cookie值就在LinkedList集合里边了。接下来，**我们要做的就是把集合中的值取出来，拼接成一个字符串**

StringBuffer stringBuffer = **new** StringBuffer();  
 //遍历LinkedList集合，添加个下划线“\_”  
 **for** (String s : linkedList) {  
 stringBuffer.append(s + "\_");  
 }  
 //最后一个元素后面就不需要下划线了  
 **return** stringBuffer.deleteCharAt(stringBuffer.length() - 1).toString();

* 好的，我们现在已经完成了Cookie值了。接下来设置Cookie的生命周期，回送给浏览器即可

String bookHistory = makeHistory(request, id);  
 Cookie cookie = **new** Cookie("bookHistory", bookHistory);  
 cookie.setMaxAge(30000);  
 response.addCookie(cookie);

* 既然我们已经把Cookie回送给浏览器了。那么接下来我们**就在首页上获取Cookie的值，显示用户浏览过什么商品就行了**！

printWriter.write("您曾经浏览过的商品：");  
 printWriter.write("<br/>");  
 //显示用户浏览过的商品  
 Cookie[] cookies = request.getCookies();  
 **for** (**int** i = 0; cookies != **null** && i < cookies.length; i++) {  
 **if** (cookies[i].getName().equals("bookHistory")) {  
 //获取到的bookHistory是2\_3\_1之类的  
 String bookHistory = cookies[i].getValue();  
 //拆解成每一个id值  
 String[] ids = bookHistory.split("\\\_");  
 //得到每一个id值  
 **for** (String id : ids) {  
 //通过id找到每一本书  
 Book book = linkedHashMap.get(id);  
 printWriter.write(book.getName());  
 printWriter.write("<br/>");  
 }  
 **break**; } }

* 好的，我们来试验一下吧！！，第一次访问首页，并没有浏览过的商品

## Cookie细节

### Cookie不可跨域名性

* 很多人在初学的时候可能有一个疑问：在访问Servlet的时候浏览器**是不是把所有的Cookie都带过去给服务器**，会不会修改了别的网站的Cookie\*\*
* 答案是否定的。Cookie具有不可跨域名性。浏览器判断**一个网站是否能操作另一个网站的Cookie的依据是域名**。所以一般来说，**当我访问baidu的时候，浏览器只会把baidu颁发的Cookie带过去，而不会带上google的Cookie。**

### Cookie的安全属性

* HTTP协议不仅仅是无状态的，而且是不安全的！如果不希望Cookie在非安全协议中传输，可以**设置Cookie的secure属性为true，浏览器只会在HTTPS和SSL等安全协议中传输该Cookie**。
* 当然了，设置secure属性不会将Cookie的内容加密。如果想要保证安全，最好使用md5算法加密【后面有】。

### Cookie的修改和删除

* 上面我们已经知道了Cookie机制没有提供删除Cookie的方法。其实细心点我们可以发现，Cookie机制也没有提供修改Cookie的方法。那么我们**怎么修改Cookie的值呢**？
* Cookie存储的方式**类似于Map集合**
* Cookie的名称相同，通过response添加到浏览器中，会覆盖原来的Cookie。
* 以country为名保存的是%E4%B8%AD%E5%9B%BD，下面我再以country为名，把值改变一下。

String name = "看完博客就点赞";

//对Unicode字符进行编码

Cookie cookie = new Cookie("country", **URLEncoder.encode(name, "UTF-8"))**;

* 再次查看Cookie的时候，值已经改变了，但是文件还是那一份
* 现在我要删除该Cookie，把MaxAge设置为0，并添加到浏览器中即可

**注意**：删除，修改Cookie时，新建的Cookie除了value、maxAge之外的所有属性都要与原Cookie相同。否则浏览器将视为不同的Cookie，不予覆盖，导致删除修改失败！

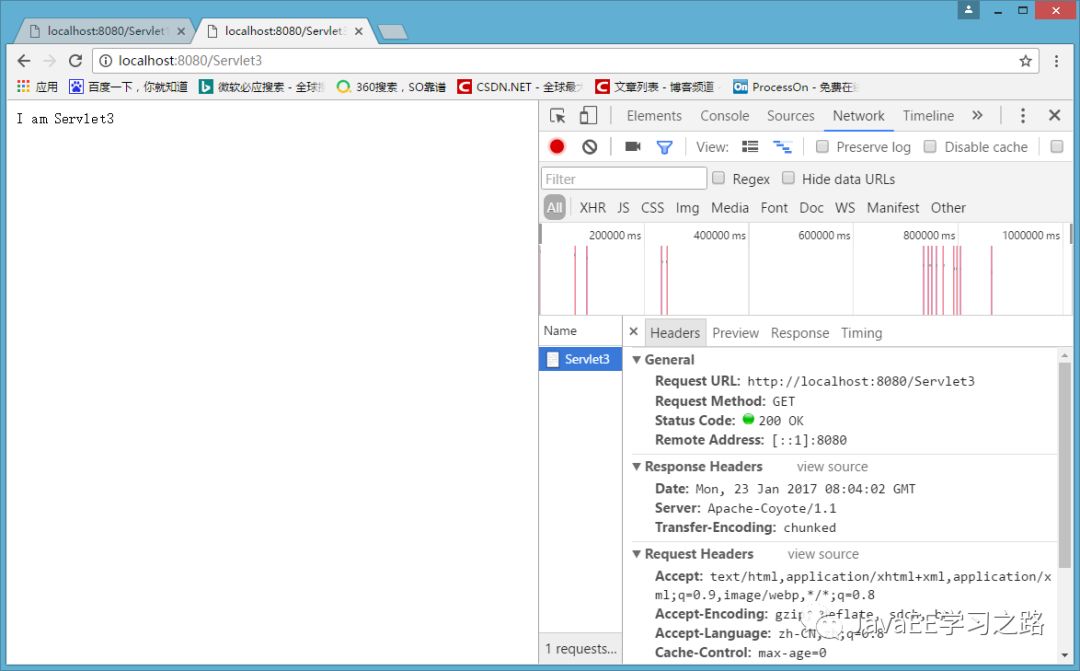
### Cookie的路径

**Cookie的path属性决定允许访问Cookie的路径**

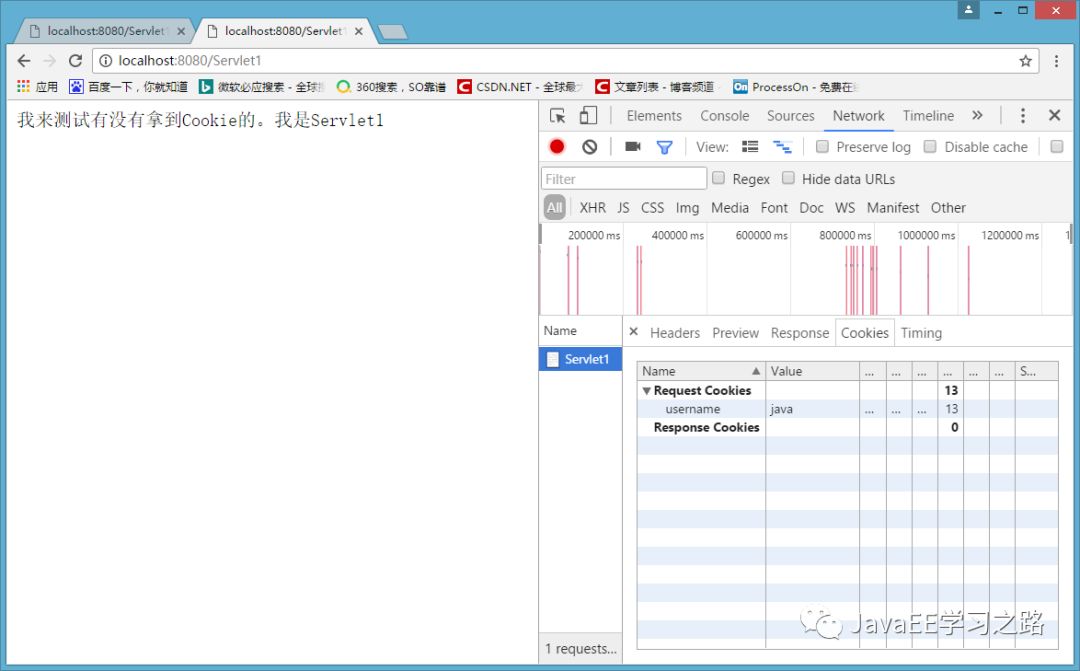
* 一般地，**Cookie发布出来，整个网页的资源都可以使用。现在我只想Servlet1可以获取到Cookie，其他的资源不能获取**。
* 使用Servlet2颁发一个Cookie给浏览器,设置路径为"/Servlet1"。

Cookie cookie = **new** Cookie("username", "java");  
 cookie.setPath("/Servlet1");  
 cookie.setMaxAge(1000);  
 response.addCookie(cookie);  
 printWriter.write("该Cookie只有Servlet1获取得到");

* 使用Servlet3访问服务器，看看浏览器是否把Cookie带上。显然，浏览器访问Servlet3并没有把Cookie带上。



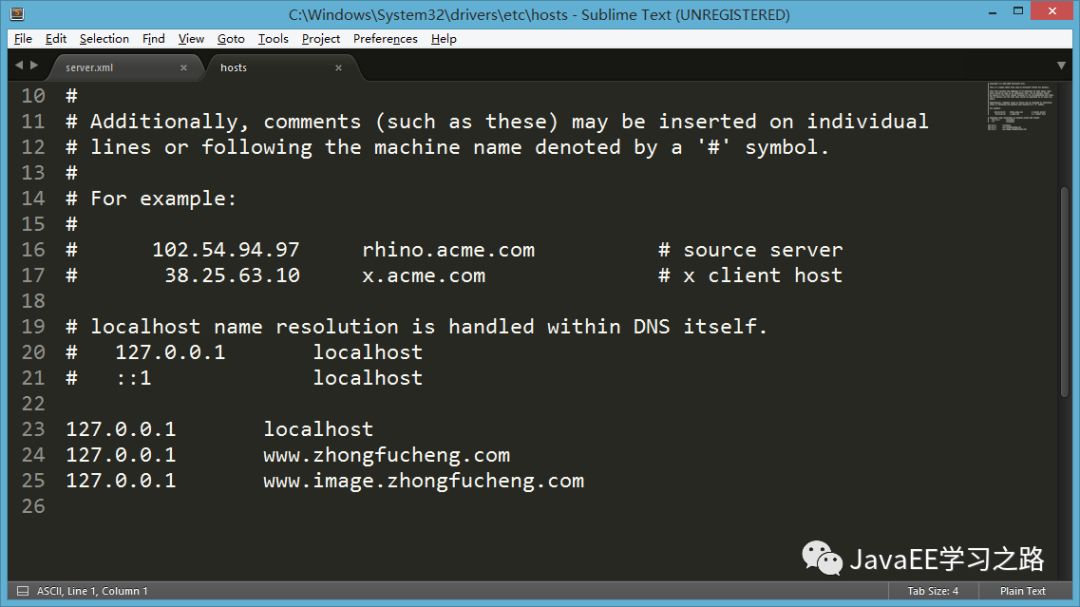
* 使用Servlet1访问服务器，看看浏览器是否把Cookie带上。答案是肯定的！



### Cookie的域名

Cookie的**domain属性决定运行访问Cookie的域名。domain的值规定为“.域名”**

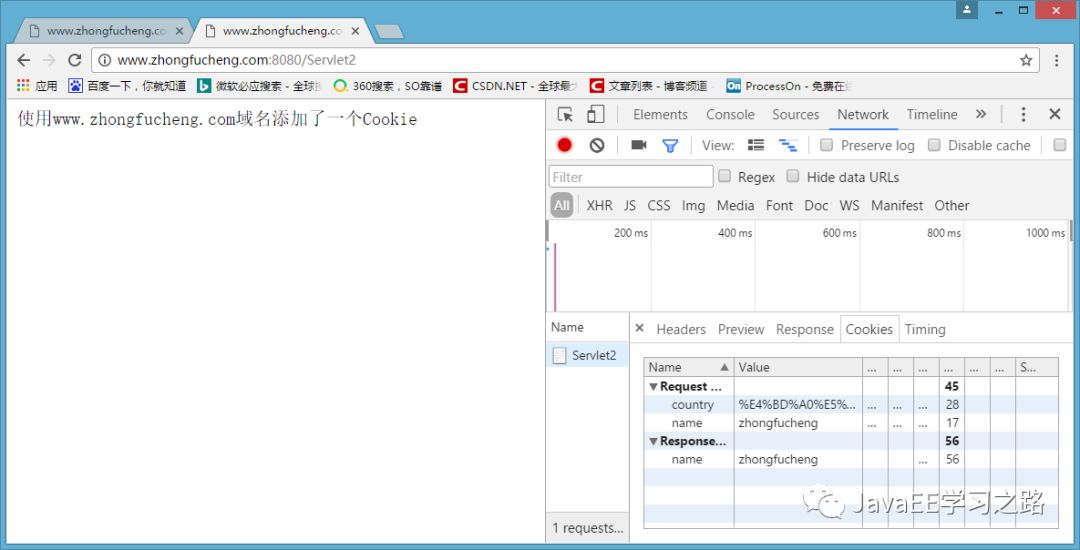
* Cookie的隐私安全机制决定Cookie是不可跨域名的。也就是说www.baidu.com和www.google.com之间的Cookie是互不交接的。**即使是同一级域名，不同二级域名也不能交接**，也就是说：www.goole.com和www.image.goole.com的Cookie也不能访问
* 我在本地上**配置了3个虚拟主机**，localhost,www.zhongfucheng.com,www.image.zhongfucheng.com【如果不知道怎么配置，在我Tomcat的博客有】



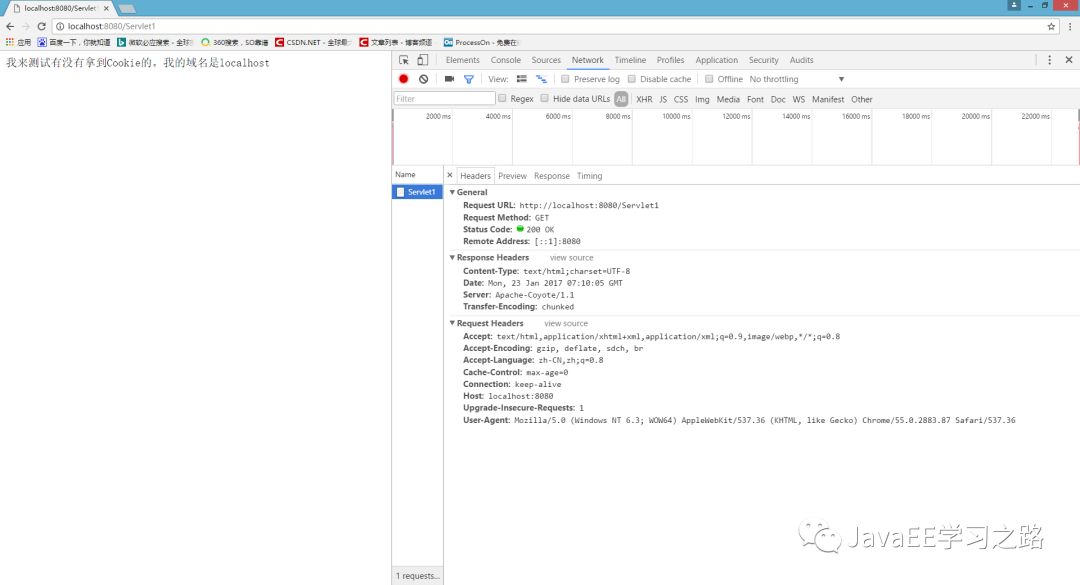


* 我用www.zhongfucheng.com域名发送了一个Cookie给浏览器

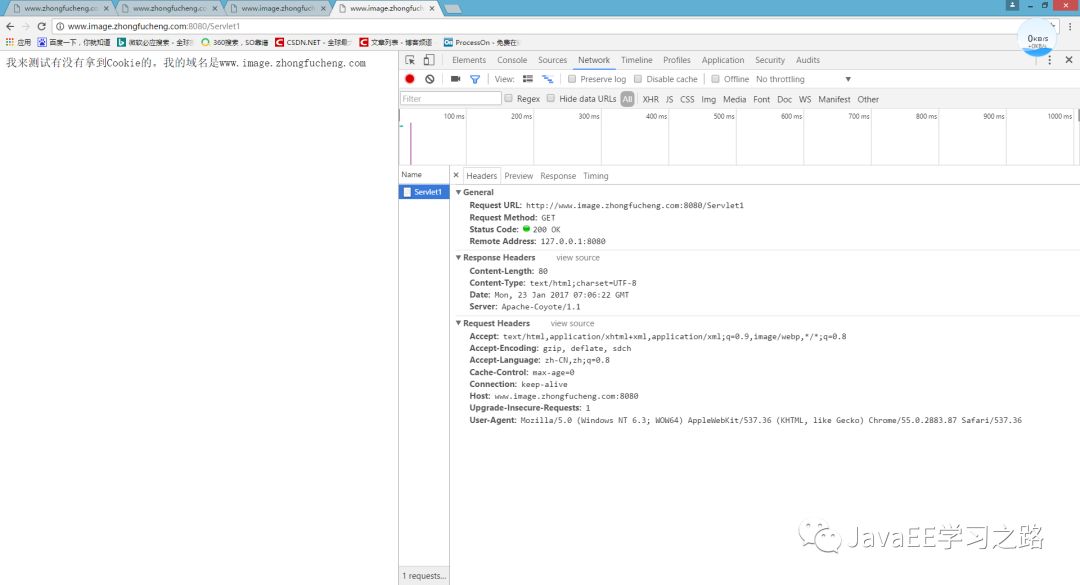
Cookie cookie = **new** Cookie("name", "zhongfucheng");  
 cookie.setMaxAge(1000);  
 response.addCookie(cookie);  
 printWriter.write("使用www.zhongfucheng.com域名添加了一个Cookie");



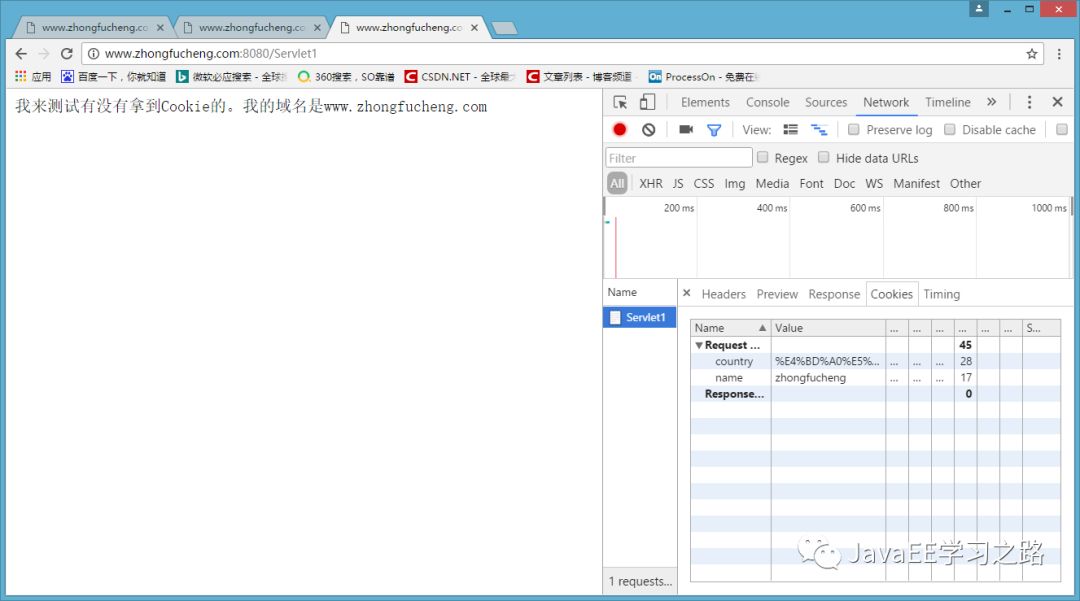
* 首先，证明了Cookie不可跨名性，localhost域名拿不到www.zhongfucheng.com颁发给浏览器的Cookie



* 再使用www.image.zhongfucheng.com域名访问,**证明即使一级域名相同，二级域名不同，也不能获取到Cookie**



* 当然，使用www.zhongfucheng.com当然能获取到Cookie，Cookie通过请求头带给服务器



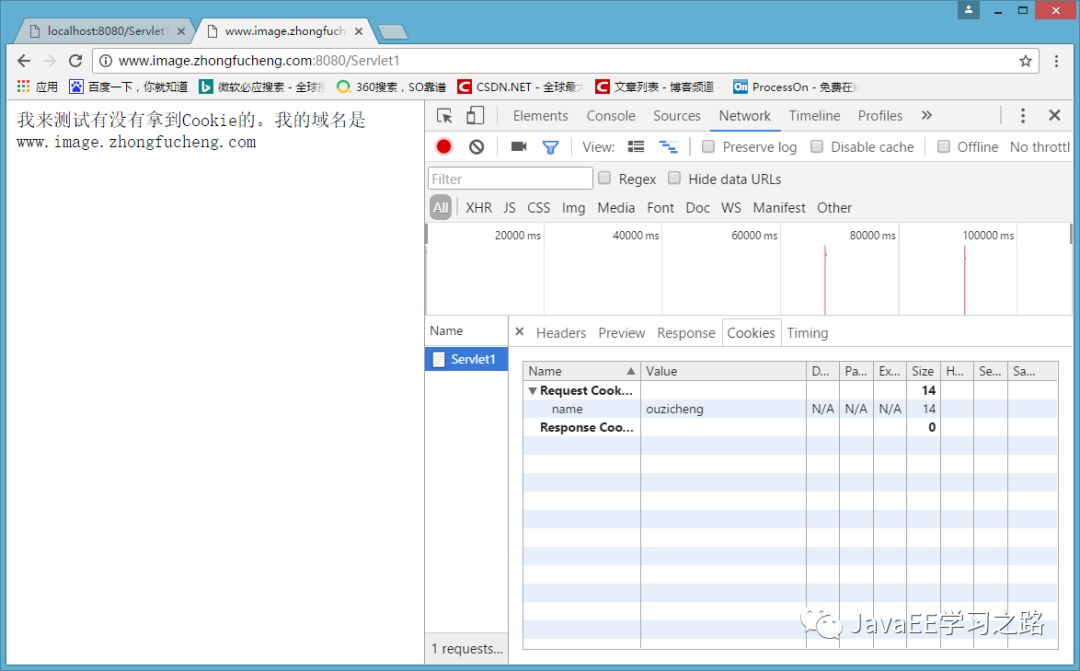
* 现在我希望**一级域名相同的网页Cookie之间可以相互访问。也就是说www.image.zhongfucheng.com可以获取到www.zhongfucheng.com的Cookie**就需要使用到domain方法。

Cookie cookie = **new** Cookie("name", "ouzicheng");  
 cookie.setMaxAge(1000);  
 **cookie.setDomain(".zhongfucheng.com");**  
 response.addCookie(cookie);  
 printWriter.write("使用www.zhongfucheng.com域名添加了一个Cookie,只要一级是zhongfucheng.com即可访问");

* 使用www.zhongfucheng.com发布一个Cookie



* 使用www.image.zhongfucheng.com域名访问一下。**发现可以获取到Cookie了**



# Session

## Session简单介绍

**服务器可以为每个用户浏览器**创建一个**会话对象（session对象**），注意：**一个浏览器独占一个session对象(默认情况下)。**

需要**保存用户数据与状态**时，服务器程序可以把用户数据写到用户浏览器独占的session中，当用户使用浏览器访问其它程序时，**其它程序可以从用户的session中取出该用户的数据**，为用户服务。（**购物车**）

是另一种**记录浏览器状态的机制**。

HTTP协议是无状态的，Session不能依据HTTP连接来判断是否为同一个用户。于是乎：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。其实Session依据Cookie来识别是否是同一个用户。这个 ID 就是客户端第一次访问服务器生成的，而且**每个客户端是唯一的。**

简单来说：Session 之所以可以识别不同的用户，依靠的就是Cookie

**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

**Session和Cookie的主要区别**

* Cookie是把用户的数据写给用户的浏览器。
* Session技术把**用户的数据写到用户独占的session中**。
* Session对象由服务器创建，开发人员可以调用request对象的getSession方法得到session对象。
* 不同的是Cookie保存在浏览器中，Session保存在服务器中。
* 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”。
* **Session可以存储对象，Cookie只能存储字符串。**

**Session API**

* long getCreationTime();【获取Session被创建时间】
* **String getId();【获取Session的id】**
* long getLastAccessedTime();【返回Session最后活跃的时间】
* ServletContext getServletContext();【获取ServletContext对象】
* **void setMaxInactiveInterval(int var1);【设置Session超时时间】**
* **int getMaxInactiveInterval();【获取Session超时时间】**
* **Object getAttribute(String var1);【获取Session属性**】
* Enumeration getAttributeNames();【获取Session所有的属性名】
* **void setAttribute(String var1, Object var2);【设置Session属性】**
* **void removeAttribute(String var1);【移除Session属性】**
* **void invalidate();【销毁该Session】**
* boolean isNew();【该Session是否为新的】

## Session 扩展

* **Seesion 中的共享数据的属性名的命名规范：**  
  通常为：XXX\_IN\_SESSION，例如：Session对象.setAttribute(“USER\_IN\_SESSION”,user)
* **序列化与反序列化：**  
  Session 中存储的对象通常需要**实现序列化接口**，因为在网络之间传输的数据格式为**二进制数据**：
  + **序列化：将对象转换成二进制数据**
  + **反序列化:将二进制数据转换成对象**
* **URL 重写**  
  **出现的问题：**  
  **当浏览器禁用Cookie之后,**那么我们的jsessionid就不能在浏览器中保存,那么**后面的请求**中就不会将 jsessionid 发送到服务器,服务器这面就**找不到数据**  
  **解决方案：**  
  **1.在url后手动的拼接上 jsessionid**  
  传递格式如 /path/Servlet;jsessionid=sessionid

"<a href='/session/list;jsessionid="+jsessionid+"'.  
2.使用响应对象中的encodeURL(String path)**实现 jsessionid 的自动拼接**  
**Servlet 容器能够重写客户请求的 URL，把 Session ID 添加到 URL 信息中，保证在客户端禁用或不支持COOKIE时，仍然可以使用Session**

**String url = resp.encodeURL("/session/list");**

该方法的实现机制为：

先判断当前的 Web 组件是否启用 Session，假如没有，直接返回参数 url。

再判断 客户端浏览器 是否支持 Cookie，假如支持，直接返回参数 url；假如不支持 ，就在参数 url 中**加入 Session ID 信息，然后返回修改后的 url。**

---------------------

* + **推荐方式：②**

## Session作为域对象

从上面的API看出，Session有着request和ServletContext类似的方法。其实**Session也是一个域对象**。Session作为一种记录浏览器状态的机制，**只要Session对象没有被销毁，Servlet之间就可以通过Session对象实现通讯**

### 服务器是如何实现一个session为一个用户浏览器服务的？

服务器创建session出来后，会把session的JSESSIONID号，以**cookie的形式**回写给客户机，这样，只要客户机的浏览器不关，再去访问服务器时，都会带着JSESSIONID号去，服务器发现客户机浏览器带JSESSIONID号过来了，**就会使用内存中与之对应的session为之服务**。可以用如下的代码证明：

public class SessionDemo1 extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setCharacterEncoding("UTF=8");

response.setContentType("text/html;charset=UTF-8");

//使用request对象的getSession()获取session，如果session不存在则创建一个

**HttpSession session = request.getSession();**

//将数据存储到session中

**session.setAttribute("data", "孤傲苍狼");**

//获取session的Id

String sessionId = session.getId();

//判断session是不是新创建的

if (session.isNew()) {

response.getWriter().print("session创建成功，session的id是："+sessionId);

}else {

response.getWriter().print("服务器已经存在该session了，session的id是："+sessionId);

}

}

public void doPost(HttpServletRequest request, HttpServletResponse response)

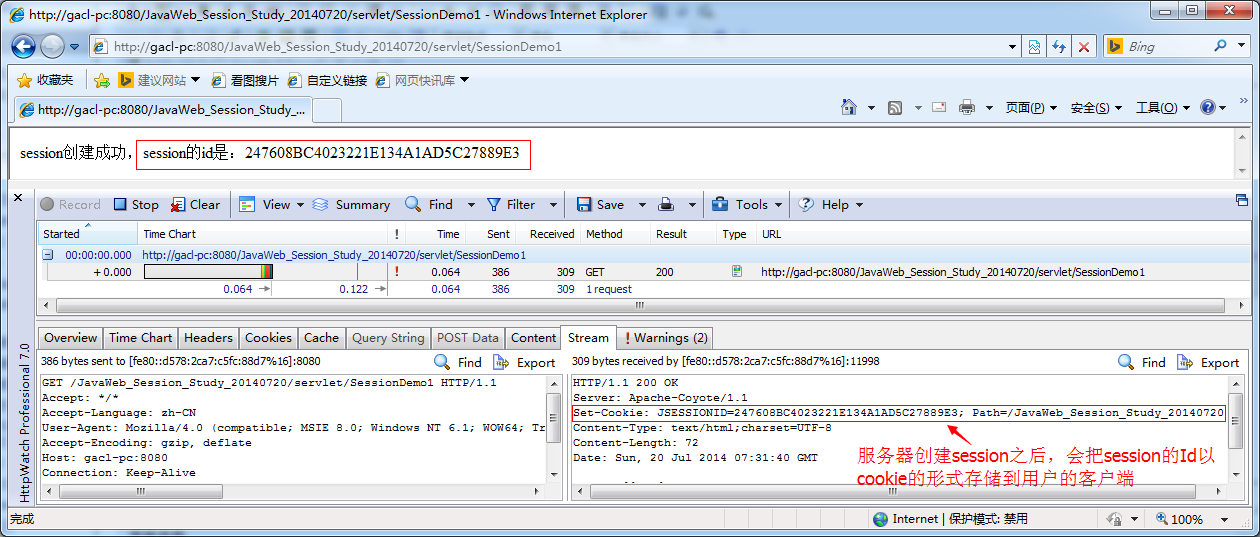
throws ServletException, IOException {

doGet(request, response);

}

}

第一次访问时，服务器会创建一个新的sesion，并且把**session的Id以cookie的形式发送给客户端浏览器**，如下图所示：



点击刷新按钮，再次请求服务器，此时就可以看到浏览器再请求服务器时，会把存储到cookie中的session的Id一起传递到服务器端了。

我**猜想**request.getSession()方法内部新创建了Session之后一定是做了**如下的处理**

//获取session的Id

String sessionId = session.getId();

//将session的Id存储到名字为**JSESSIONID的cookie中**

Cookie cookie = new Cookie("JSESSIONID", sessionId);

//设置cookie的有效路径

**cookie.setPath(request.getContextPath());?????????????????????????????**

response.addCookie(cookie);

一般来讲，当我们要存进的是**用户级别的数据就用Session**，那什么是用户级别呢？**只要浏览器不关闭，希望数据还在，就使用Session来保存**。

### Session的生命周期和有效期

* 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**，这也就为什么上面的例子可以**直接使用request对象获取得到Session对象**。
* 如果访问HTML,IMAGE等静态资源Session不会被创建。
* 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**。
* 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**。
* Session的超时时间默认是30分钟，有三种方式可以对Session的超时时间进行修改

**第一种方式：**在tomcat/conf/web.xml文件中设置，时间值为20**分钟**，**所有的WEB应用都有效**

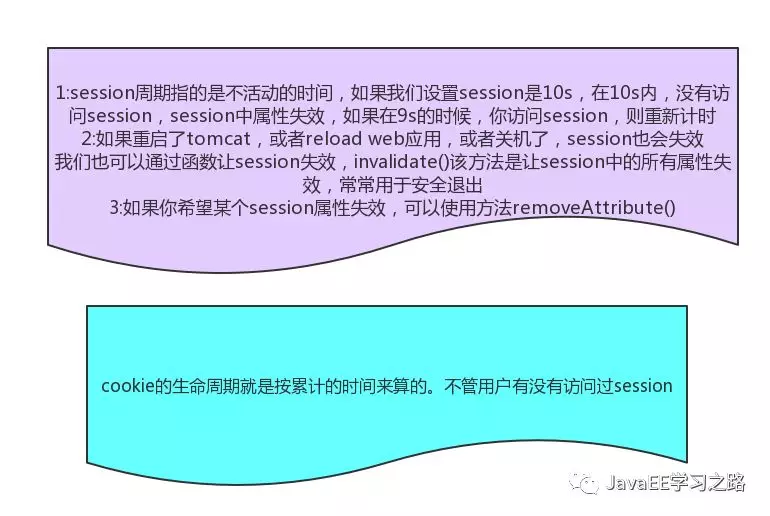
**第二种方式：**在单个的web.xml文件中设置，对单个web应用有效，**如果有冲突，以自己的web应用为准**。

<session-config>

<session-timeout>20</session-timeout>

</session-config>

**第三种方式：**通过setMaxInactiveInterval()方法设置

 //设置Session最长超时时间为**60秒**，这里的单位是秒  
 httpSession.setMaxInactiveInterval(60);  
 System.out.println(httpSession.getMaxInactiveInterval());

### 使用Session完成简单的购物功能

我们还是以书籍为例，所以可以copy“显示浏览过的商品“例子部分的代码

public class IndexServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

out.write("网页上所有的书籍：" + "<br/>");

// 拿到数据库所有的书

LinkedHashMap<String, Book> linkedHashMap = DB.getAll();

Set<Map.Entry<String, Book>> entry = linkedHashMap.entrySet();

// 显示所有的书到网页上

for (Map.Entry<String, Book> stringBookEntry : entry) {

Book book = stringBookEntry.getValue();

String url = "/day07/servlet/BuyServlet?id=" + book.getId();

out.write(book.getName());

out.write("<a href='" + url + "'>购买</a>");

out.write("<br/>");

}

}

class DB {

private static LinkedHashMap<String, Book> linkedHashMap = new LinkedHashMap();

// 简化开发复杂度，book的id和商品的id相同

static {// 加载DB类，加载put，执行一次

linkedHashMap.put("1", new Book("1", "javaweb", "zhong"));

linkedHashMap.put("2", new Book("2", "java", "fu"));

linkedHashMap.put("3", new Book("3", "oracle", "cheng"));

linkedHashMap.put("4", new Book("4", "mysql", "ou"));

linkedHashMap.put("5", new Book("5", "ajax", "zi"));

}

// 获取到所有书籍

public static LinkedHashMap getAll() {

return linkedHashMap;

}

}

class Book {

private String id;

private String name;

private String author;

在购物车页面上，获取到用户想买的书籍【用户可能不单想买一本书，于是乎，就用一个List容器装载书籍】，有了：**先遍历Cookie，再判断是否是第一次访问Servlet的逻辑思路**，我们就可以先获取到Session的属性，如果Session的属性为null，那么就是还没有该属性

       //得到用户想买书籍的id

       String id = request.getParameter("id");

       //根据书籍的id找到用户想买的书

       Book book = (Book) DB.getAll().get(id);

       //获取到Session对象

       HttpSession httpSession = **request.getSession();**

       //由于用户可能想买多本书的，所以我们用一个容器装着书籍

       List list = (List) httpSession.getAttribute("list");

       if (list == null) {

           list = new ArrayList();

           //设置Session属性

           httpSession.setAttribute("list",list);

       }

       //把书籍加入到list集合中

       list.add(book);

request.getRequestDispatcher("/servlet/ListCartServlet").forward(request, response);

* 按我们正常的逻辑思路：先创建一个ArrayList对象，把书加到list集合中，然后设置Session的属性。**这样是行不通的**。每次Servlet被访问的时候都会创建一个ArrayList集合，书籍会被分发到不同的ArrayList中去。所以下面的代码是不行的！

       //得到用户想买书籍的id

       String id = request.getParameter("id");

       //根据书籍的id找到用户想买的书

       Book book = (Book) DB.getAll().get(id);

       //获取到Session对象

       HttpSession httpSession = request.getSession();

       //创建List集合

       List list = new ArrayList();

       list.add(book);

       httpSession.setAttribute("list", list);

* 既然用户已经购买了书籍，那么也应该给提供页面显示用户购买过哪些书籍

 列出用户购买过的书籍

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

       //要得到用户购买过哪些书籍，得到Session的属性遍历即可

       HttpSession httpSession = request.getSession();

       List<Book> list = (List) httpSession.getAttribute("list");

       if (list == null || list.size() == 0) {

           printWriter.write("对不起，你还没有买过任何商品");

       } else {

           printWriter.write("您购买过以下商品：");

           printWriter.write("<br/>");

           for (Book book : list) {

               printWriter.write(book.getName());

               printWriter.write("<br/>");

           }

       }

### 浏览器禁用了Cookie，Session还能用吗？

上面说了Session是依靠Cookie来识别用户浏览器的。如果我的用户浏览器禁用了Cookie了呢？绝大多数的手机浏览器都不支持Cookie，那我的Session怎么办？

* 一看，Session好像不能用了。但是Java Web提供了解决方法：**URL地址重写**
* HttpServletResponse类提供了两个URL地址重写的方法：
  + **encodeURL(String url)**
  + **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【当然了，即使浏览器支持Cookie，第一次输出URL地址的时候还是会出现jsessionid（因为没有任何Cookie可带）】
* 下面我们就以上面“购物”的例子来做试验吧！**首先我们来看看禁用掉Cookie对原来的小例子有什么影响**。
* 无论点击多少次，都会直接提示我们有买过任何商品
* 原因也非常简单，没有Cookie传递给服务器，**服务器每次创建的时候都是新的Session**，导致最后获取到的List集合一定是空的。
* **不同Servlet获取到的Session的id号都是不同的**

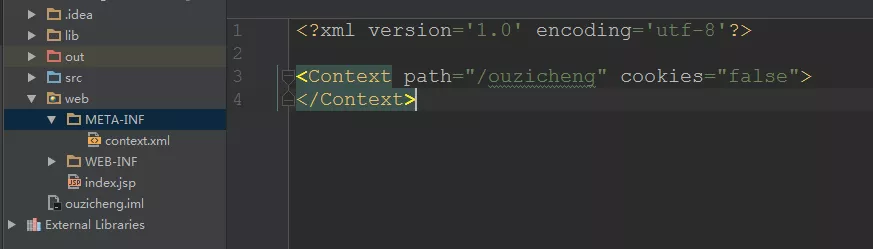
**response.sendRedirect(response.encodeURL(url));**

**URL地址重写的原理：将Session的id信息重写到URL地址中。服务器解析重写后URL，获取Session的id\*\*。这样一来，即使浏览器禁用掉了Cookie，但Session的id通过服务器端传递，还是可以使用Session来记录用户的状态。**

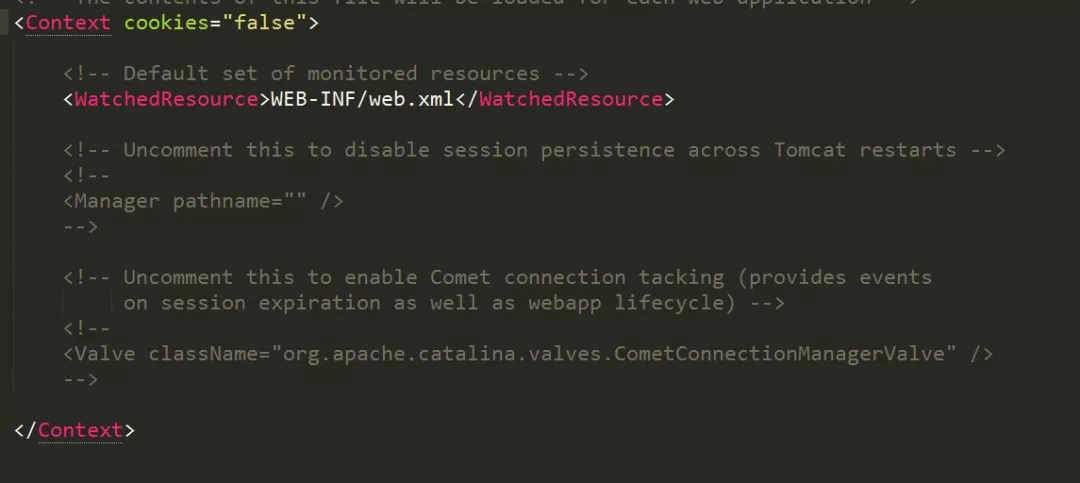
**Session禁用Cookie**

**Java Web规范支持通过配置禁用Cookie，禁用自己项目的Cookie**

**在META-INF文件夹下的context.xml文件中修改（没有则创建）**



**禁用全部web应用的Cookie**

**在conf/context.xml中修改**

**注意：该配置只是让服务器不能自动维护名为jsessionid的Cookie，并不能阻止Cookie的读写。**

### 使用Session完成用户简单登陆

* 先创建User类

private String username = null;

private String password = null;

public User() {

}

public User(String username, String password) {

this.username = username;

this.password = password;

}

....各种set、get方法

使用简单的集合模拟一个数据库

private static List<User> list = new ArrayList<>();

//装载些数据进数据库

static {

list.add(new User("aaa","111"));

list.add(new User("bbb","222"));

list.add(new User("ccc","333"));

}

//通过用户名和密码查找用户

public static User find(String username, String password) {

for (User user : list) {

if (user.getUsername().equals(username) && user.getPassword().equals(password)) {

return user;

}

}

return null;

}

表单提交的工作我就在jsp写了，如果在Servlet写太麻烦了！

<form action="/ouzicheng/LoginServlet" method="post">

用户名：<input type="text" name="username"><br/>

密码：<input type="password" name="password"><br/>

<input type="submit" value="提交">

</form>

**获取到表单提交的数据，查找数据库是否有相对应的用户名和密码**。如果没有就提示用户名或密码出错了，如果有就跳转到另外一个页面

String username = request.getParameter("username");

String password = request.getParameter("password");

User user = UserDB.find(username, password);

//如果找不到，就是用户名或密码出错了。

if (user == null) {

response.getWriter().write("you can't login");

return;

}

//标记着该用户已经登陆了！

HttpSession httpSession = request.getSession();

httpSession.setAttribute("user", user);

//跳转到其他页面，告诉用户成功登陆了。

**response.sendRedirect(response.encodeURL("index.jsp"));**

### 利用Session防止表单重复提交

* 重复提交的危害：
  + 在投票的网页上不停地提交，实现了刷票的效果。
  + 注册多个用户，不断发帖子，扰乱正常发帖秩序。

首先我们来看一下常见的重复提交。

* + 在处理表单的Servlet中刷新。
  + 后退再提交
  + 网络延迟，多次点击提交按钮

对于网络延迟造成的多次提交数据给服务器，其实是客户端的问题。于是，我们可以使用javaScript来防止这种情况

* **在处理表单的Servlet中刷新**和**后退再提交**这两种方式不能只靠客户端来限制了。也就是说javaScript代码无法阻止这两种情况的发生。
* 于是乎，我们就想得用其他办法来阻止表单数据重复提交了。我们现在学了Session，**Session可以用来标识一个用户是否登陆了**。Session的原理也说了：**不同的用户浏览器会拥有不同的Session**。而request和ServletContext为什么就不行呢？**request的域对象只能是一次http请求**，**提交表单数据的时候request域对象的数据取不出来**。ServletContext代表整个web应用，**如果有几个用户浏览器同时访问**，**ServletContext域对象的数据会被多次覆盖掉**，也就是说域对象的数据就毫无意义了。
* 可能到这里，我们会想到：在提交数据的时候，存进Session域对象的数据，在处理提交数据的Servlet中判断Session域对象数据????。究竟判断Session什么？判断Session域对象的数据不为null？**没用呀，既然已经提交过来了，那肯定不为null。**
* 此时，我们就想到了，在**表单中还有一个隐藏域，可以通过隐藏域把数据交给服务器**。
  + **判断Session域对象的数据和jsp隐藏域提交的数据是否对应**。
  + **判断隐藏域的数据是否为空**【如果为空，**就是直接访问表单处理页面的Servlet**】
  + **判断Session的数据是否为空**【servlet判断完是否重复提交，**最好能立马移除Session的数据**，不然还没有移除的时候，客户端那边儿的请求又来了，就又能匹配了，产生了重复提交。**如果Session域对象数据为空，证明已经提交过数据了！**】
* 我们向Session域对象的存入数据究竟是什么呢？简单的一个数字？好像也行啊。因为**只要Session域对象的数据和jsp隐藏域带过去的数据对得上号就行了呀，反正在Servlet上判断完是否重复提交，会立马把Session的数据移除掉的**。更专业的做法是：**向Session域对象存入的数据是一个随机数【Token--令牌**】。

**具体的做法：**在服务器端生成一个唯一的随机标识号，专业术语称为Token(令牌)，同时在当前用户的Session域中保存这个Token。然后将Token发送到客户端的Form表单中，在Form表单中使用隐藏域来存储这个Token，表单提交的时候连同这个Token一起提交到服务器端，然后在服务器端判断客户端提交上来的Token与服务器端生成的Token是否一致，如果不一致，那就是重复提交了，此时服务器端就可以不处理重复提交的表单。如果相同则处理表单提交，处理完后清除当前用户的Session域中存储的标识号。  
　　在下列情况下，服务器程序将拒绝处理用户提交的表单请求：

1. 存储Session域中的Token(令牌)与表单提交的Token(令牌)不同。
2. 当前用户的Session中不存在Token(令牌)。
3. 用户提交的表单数据中没有Token(令牌)。

**FormServlet**

/\*

\* 产生随机数就应该用一个对象来生成，这样可以避免随机数的重复。

\* 所以设计成单例

\* \*/

public class TokenProcessor {

   private TokenProcessor() {

   }

   private final static TokenProcessor TOKEN\_PROCESSOR = new TokenProcessor();

   public static TokenProcessor getInstance() {

       return TOKEN\_PROCESSOR;

   }

   public static String makeToken() {

       //这个随机生成出来的Token的长度是不确定的

       String token = String.valueOf(System.currentTimeMillis() + new Random().nextInt(99999999));

       try {

           //我们想要随机数的长度一致，就要获取到数据指纹

           MessageDigest messageDigest = MessageDigest.getInstance("md5");

           byte[] md5 = messageDigest.digest(token.getBytes());

           //如果我们直接 return  new String(md5)出去，**得到的随机数会乱码。**

           //因为随机数是任意的01010101010，在转换成字符串的时候，会查gb2312的码表，gb2312码表不一定支持该二进制数据，得到的就是乱码

           //于是乎经过base64编码成了明文的数据

           BASE64Encoder base64Encoder = new BASE64Encoder();

           return base64Encoder.encode(md5);

       } catch (NoSuchAlgorithmException e) {

           e.printStackTrace();

       return null;

* **创建FormServlet，用于生成Token(令牌)和跳转到form.jsp页面。创建Token随机数**

       //生出随机数

       TokenProcessor tokenProcessor = TokenProcessor.getInstance();

       String token = tokenProcessor.makeToken();

       //将随机数存进Session中

       request.getSession().setAttribute("token", token);

       //跳转到显示页面

       request.getRequestDispatcher("/login.jsp").forward(request, response);

* **jsp隐藏域获取到Session的值**

<form action="${pageContext.request.contextPath}/servlet/DoFormServlet" method="post">

   用户名：<input type="text" name="username">

   <input type="submit" value="提交" id="button">

   <%--使用EL表达式取出session中的Token--%>

   <input type="hidden" name="token" value="${token}" >

</form>

* DoFormServlet处理表单提交中判断：**jsp隐藏域是否有值带过来，Session中的值是否为空，Session中的值和jsp隐藏域带过来的值是否相等**

       String serverValue = (String) request.getSession().getAttribute("token");

       String clientValue = request.getParameter("token");

       if (serverValue != null && clientValue != null && serverValue.equals(clientValue)) {

           System.out.println("处理请求");

           //清除Session域对象数据

           request.getSession().removeAttribute("token");

       }else {

           System.out.println("请不要重复提交数据！");

       }

### 一次性校验码

**一次性校验码其实就是为了防止暴力猜测密码**

**在讲response对象的时候，我们使用response对象输出过验证码，但是没有去验证！**

**验证的原理也非常简单：生成验证码后，把验证码的数据存进Session域对象中，判断用户输入验证码是否和Session域对象的数据一致。**

**生成验证码图片，并将验证码存进Session域中**

**//在内存中生成图片**

**BufferedImage bufferedImage = new BufferedImage(80, 20, BufferedImage.TYPE\_INT\_RGB);**

**//获取到这张图片**

**Graphics2D graphics2D = (Graphics2D) bufferedImage.getGraphics();**

**//设置背景色为白色**

**graphics2D.setColor(Color.white);**

**graphics2D.fillRect(0, 0, 80, 20);**

**//设置图片的字体和颜色**

**graphics2D.setFont(new Font(null, Font.BOLD, 20));**

**graphics2D.setColor(Color.BLUE);**

**//生成随机数**

**String randomNum = makeNum();**

**//往这张图片上写数据,横坐标是0，纵坐标是20**

**graphics2D.drawString(randomNum, 0, 20);**

**//将随机数存进Session域中**

**request.getSession().setAttribute("randomNum", randomNum);**

**//控制浏览器不缓存该图片**

**response.setHeader("Expires", "-1");**

**response.setHeader("Cache-Control", "no-cache");**

**response.setHeader("Pragma", "no-cache");**

**//通知浏览器以图片的方式打开**

**response.setHeader("Content-type", "image/jpeg");**

**//把图片写给浏览器**

**ImageIO.write(bufferedImage, "jpg", response.getOutputStream());**

**生成随机数的方法：**

**private String makeNum()**

**Random random = new Random();**

**//生成0-6位的随机数**

**int num = random.nextInt(999999);**

**//验证码的数位全都要6位数，于是将该随机数转换成字符串，不够位数就添加**

**String randomNum = String.valueOf(num)；**

**//使用StringBuffer来拼凑字符串**

**StringBuffer stringBuffer = new StringBuffer();**

**for (int i = 0; i < 6 - randomNum.length(); i++) {**

**stringBuffer.append("0");**

**}**

**return stringBuffer.append(randomNum).toString();**

**}**

**jsp显示页面**

**<form action="/ouzicheng/Login2Servlet">**

**用户名：<input type="text" name="username"><br>**

**密码：<input type="password" name="password"><br>**

**验证码：<input type="text" name="randomNum">**

**<img src="/ouzicheng/ImageServlet" ><br><br>**

**<input type="submit" value="提交">**

**</form>**

**处理提交表单数据的Servlet，判断用户带过来验证码的数据是否和Session的数据相同。**

**//获取用户输入验证码的数据**

**String client\_randomNum = request.getParameter("randomNum");**

**//获取Session中的数据**

**String session\_randomNum = (String) request.getSession().getAttribute("randomNum");**

**//判断他俩数据是否相等，用户是否有输入验证码，Session中是否为空**

**if (client\_randomNum == null || session\_randomNum == null || !client\_randomNum.equals(session\_randomNum)) {**

**System.out.println("验证码错误了！！！");**

**return ;**

**}**

**//下面就是验证用户名和密码...................**