# Cookie和Session

# 学习目标

1.能够说出会话的概念

2.能够说出cookie的概念

3.能够创建、发送、接收、删除cookie

4.能够说出cookie执行原理

7.能够说出session的概念

8.能够获取session对象、添加、删除、获取session中的数据

9.能够完成登录验证码案例

反馈

|  |
| --- |
| 内容越来越多,越来越难了,宋哥讲慢点  坚持住,最后胜利会到来了. |
| ServletContext存储域对象，是否可以理解存储在服务器内存里面？  Web后端的对象都在服务器内存里,,还有request,  但是实际开发中这么多请求域对象，这么多服务器传递数据，感觉就不靠谱，在里面检索域对象的流程是怎么样的。  一次请求就是一个request,  由于request是请求域对象,如果页面跳转在一次请求域内需要传递数据使用请求转发,否则建议使用重定向。 那么，重定向不能通过域对象传递数据吗？为什么。  一次请求域内需要传递数据,使用请求转发  Session域,重定向也可以传递数据 |
| imgObj.src="CheckCodeServlet?t="+new Date().getTime(); 这个方法是怎么改变验证码的.  上面操作的是img标签src属性,浏览器对静态资源img标签,第一次访问完浏览器会缓存图片,只有src不变,浏览器以后都会访问缓存.验证码要更新,必须要让src属性改变,上面的就可以达到 |
| POST是不是会把数据分成header和data两个包发送两次? Get和POST的区别能详细讲一下吗?  http2协议,才会将分成header和data两个部分二进制的数据传递  http1.1,htt请求原始格式请求行\请求头\请求体  get请求  传递数据在地址栏请求行上,明文的,地址栏传递数据有数据大小限制1~18k,不安全  Post请求  传递的数据在请求体上,非明文,没有传递数据大小限制,相对安全 |
| 宇哥能讲讲开发中我们主要做哪些工作吗？具体一点的  偶尔我会分享一些经验, |

# 会话的基本概念

## 什么是会话

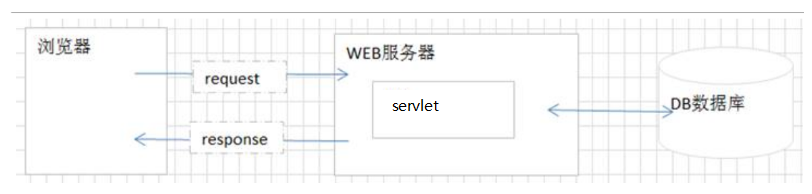
在日常生活中，从拨通电话到挂断电话之间的一连串的你问我答的过程就是一个会话。Web应用中的会话过程类似于生活中的打电话过程，它指的是一个客户端（浏览器）与Web服务器之间连续发生的一系列请求和响应过程,关闭浏览器会话结束.例如，一个用户在某网站上的整个购物过程就是一个会话。

在打电话过程中，通话双方会有通话内容，同样，在客户端与服务器端交互的过程中，也会产生一些数据。例如，用户甲和乙分别登录了购物网站，甲购买了一个Nokia手机，乙购买了一个Ipad，当这两个用户结账时，Web服务器需要对用户甲和乙的信息分别进行保存。在前面章节讲解的域对象中，HttpServletRequest对象和ServletContext对象都可以对数据进行保存，但是这两个对象都不可行，具体原因如下：

（1）客户端请求Web服务器时，针对每次HTTP请求，Web服务器都会创建一个HttpServletRequest对象，该对象只能保存本次请求所传递的数据。由于购买和结账是两个不同的请求，因此，在发送结账请求时，之前购买请求中的数据将会丢失。

（2）使用ServletContext对象保存数据时，由于同一个Web应用共享的是同一个ServletContext对象，因此，当用户在发送结账请求时，由于无法区分哪些商品是哪个用户所购买的，而会将该购物网站中所有用户购买的商品进行结算，这显然也是不可行的。

（3）为了保存会话过程中产生的数据，在Servlet技术中，提供了两个用于保存会话数据的对象，分别是Cookie和Session。关于Cookie和Session的相关知识，将在下面的小节进行详细讲解。



Servlet的会话技术主要有**cookie**与**session**两种 。

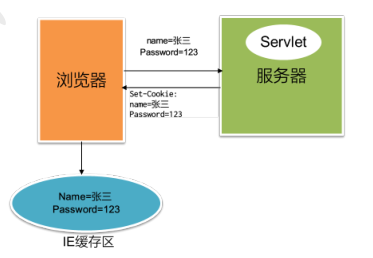
# cookie

## cookie的概念

### 什么是cookie

​ **浏览器端的会话技术，它通过在浏览器中记录一些服务器传递过来的数据(cookie存储在浏览器缓存目录中的文本文件)，通过cookie的有效期决定会话从什么时候开始，到什么时候结束。**

Cookie 就是浏览器写在你本地计算机上的一个文本文件，大小不超过 4K



## cookie的应用场景

1. 记住用户名



1. 自动登录（记住用户名和密码）



## cookie的基本使用

### cookie常用的方法介绍

#### API介绍

1. Cookie:  
       Cookie(String name, String value)  创建cookie对象  
       String getName()   获取cookie的名称  
       String getValue()  获取cookie的值  
       void setPath(String uri)   设置cookie的路径——浏览器根据这个路径判断那些cookie要发送给服务器
2. HttpServletResponse：  
    void addCookie(Cookie cookie)   将cookie发送给浏览器
3. HttpServletRequest：  
    Cookie[] getCookies()  获取浏览器发送的cookie

#### 使用步骤

1. 在CookieServlet1创建cookie
2. 在CookieServlet1设置cookie路径为当前项目根路径：“/day04”
3. 在CookieServlet1使用response对象将cookie发送给浏览器
4. 在CookieServlet2通过request对象获取浏览器发送的cookie
5. 打印在控制台上

#### 演示代码

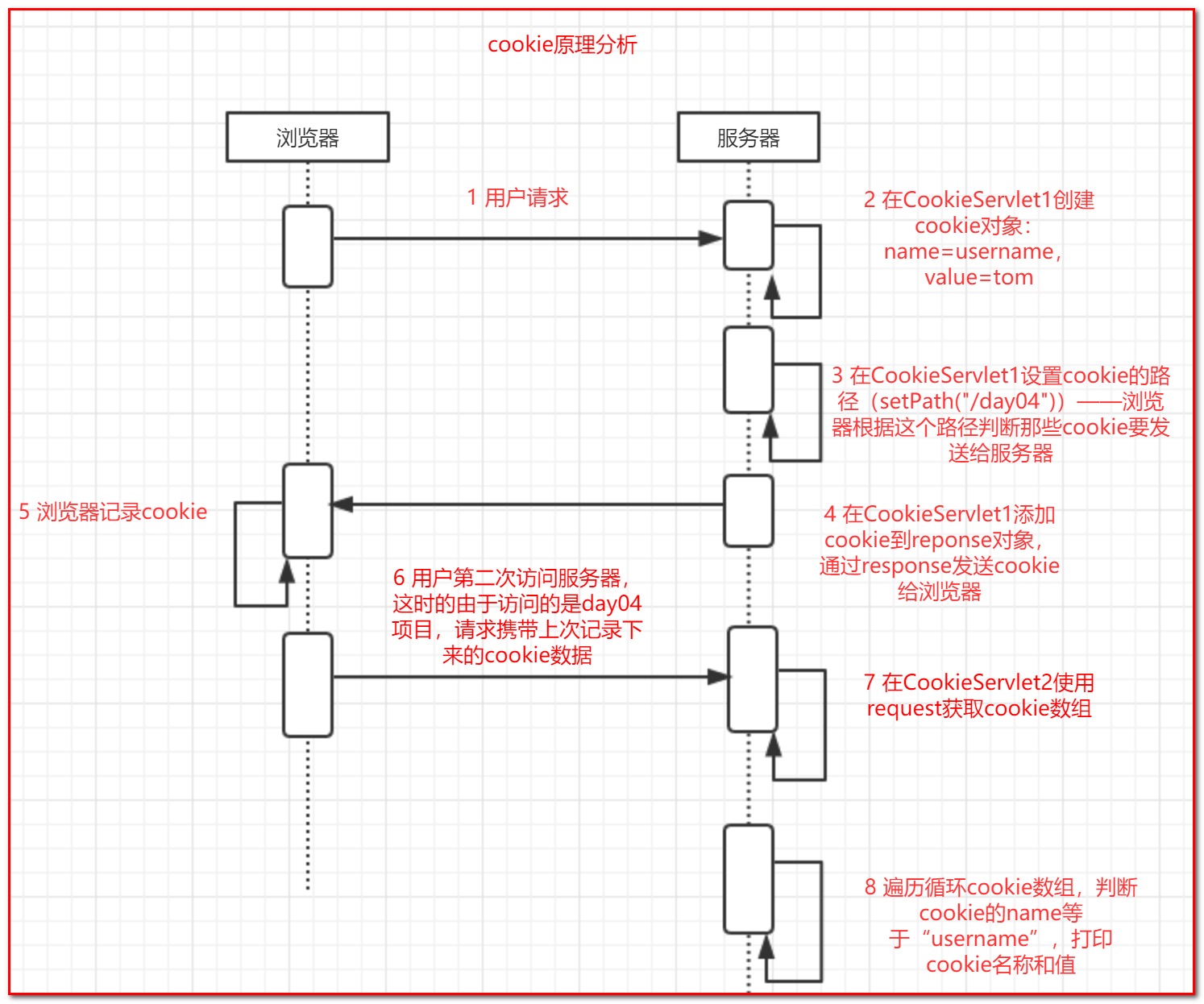
CookieServlet1：

|  |
| --- |
| **package** com.itheima.cookie;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.Cookie; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"AddCookieDataServlet"**, urlPatterns = **"/addCookieData"**) **public class** AddCookieDataServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {   */\*  \* Cookie(String name, String value) 创建cookie对象  String getName() 获取cookie的名称  String getValue() 获取cookie的值  void setPath(String uri) 设置cookie的路径——浏览器根据这个路径判断哪些cookie要发送给服务器  setMaxAge(秒);设置cookie有效期,单位秒  如果不设置有效期,默认是浏览器关闭cookie就是失效   HttpServletResponse对象  void addCookie(Cookie cookie) 将cookie发送给浏览器    cookie的特点  1.一个cookie对象存储一个键值对  2.cookie存储的值必须是string类型  3.cookie存储在客户端浏览器缓存目录中的一个文本文件  4.cookie是通过响应头传递给浏览器存储的,Set-Cookie: username=admin...  5.cookie存储在浏览器后,以后发送的请求.浏览器会自动根据有效路径携带cookie到服务器到  6.cookie如果使用默认有效期,浏览器缓存目录不会产生文本文件,cookie数据只会在浏览器内存中  \* \*/   //往cookie里面写入数据* Cookie cookie = **new** Cookie(**"username"**,**"admin"**);  *//Cookie cookie2 = new Cookie("username","admin");  //cookie设置有效期* cookie.setMaxAge(60\*60\*24\*7);*//7天,  //设置cookie的路径——浏览器根据这个路径判断哪些cookie要发送给服务器,路径格式:/工程名字/资源路径  //路径写法,浏览器使用的全路径需要从工程名字写起.如果服务器使用的全路径/资源路径  //cookie.setPath();//如果不设置默认day38,只要浏览器访问以/day38/开头,浏览器都会携带cookie到服务器端  //需求:只访问GetCookieDataServlet开头的,浏览器才携带cookie* cookie.setPath(request.getContextPath()+**"/GetCookieDataServlet"**);   *//将cookie的数据发送给浏览器* response.addCookie(cookie);  *//response.addCookie(cookie2);* } } |

CookieServlet2：

|  |
| --- |
| **package** com.itheima.cookie;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.Cookie; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"GetCookieDataServlet"**, urlPatterns = **"/GetCookieDataServlet"**) **public class** GetCookieDataServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {   *//读取cookie的数据的api  /\*  \* HttpServletRequest对象：  Cookie[] getCookies() 获取浏览器发送的cookie  \* \*/* Cookie[] cookies = request.getCookies();  *//判断数据有效性* **if**(cookies!=**null**){  *//遍历cookies数组读取里面的数据* **for**(Cookie cookie : cookies){  *//获取cookie里面存储name为username的数据* **if**(cookie.getName().equals(**"username"**)){  response.getWriter().write(cookie.getName()+**":"**+cookie.getValue());  }  }  }  } } |

#### cookie技术原理分析



### 删除cookie

#### 使用步骤

1. 创建cookie对象,名称保持一致
2. 设置生存时间为0
3. 设置cookie的路径,与原来保持一致
4. 发送cookie给浏览器,覆盖已有cookie

#### 注意事项

要删除已经存在的cookie，只需要创建一个一模一样的cookie覆盖已有cookie,新创建的cookie必须名称和路径与原来的cookie一致,value清空和有效期设置为0

#### 演示代码

|  |
| --- |
| **package** com.itheima.cookie;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.Cookie; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"DelCookieDataServlet"**, urlPatterns = **"/DelCookieDataServlet"**) **public class** DelCookieDataServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {   */\*删除浏览器端cookie  \* 创建一个新的cookie覆盖浏览器已有cookie  \* 新的cookie的name,path必须与已有的一样,设置有效期为0,清空value  \* \*/  //要求:删除掉客户端username的cookie* Cookie cookie = **new** Cookie(**"username"**,**""**);  cookie.setMaxAge(0);*//立即删除* cookie.setPath(request.getContextPath()+**"/GetCookieDataServlet"**);  *//输出到浏览* response.addCookie(cookie);  response.getWriter().write(**"del username cookie..."**);  } } |

### 案例：记录用户上一次访问时间

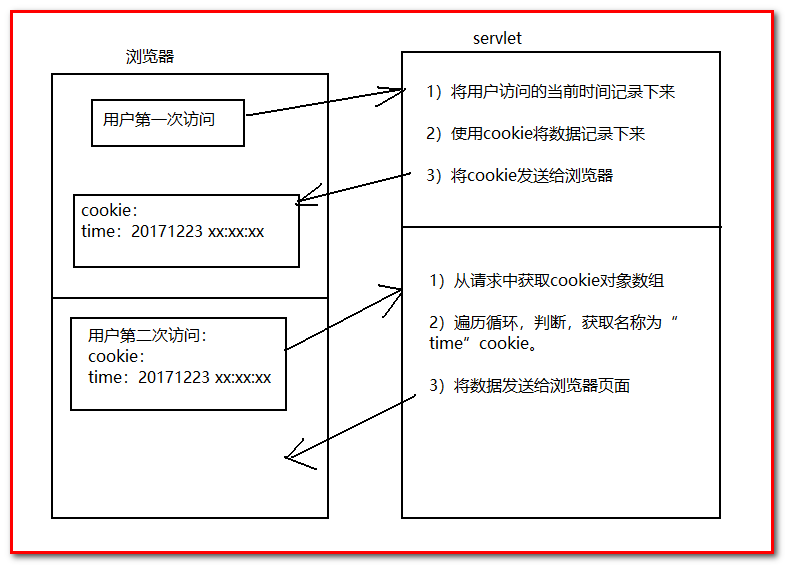
#### 案例需求

​ 展示用户上一次访问服务器的时间

#### 案例效果

​ 展示用户上一次访问服务器的时间

#### 案例分析



#### 实现代码

|  |
| --- |
| **package** com.itheima.cookie.demo;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.Cookie; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.text.SimpleDateFormat; **import** java.util.Date;  @WebServlet(name = **"LastTimeServlet"**, urlPatterns = **"/LastTimeServlet"**) **public class** LastTimeServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {   */\*显示用户上一次访问时间\*/   /\*用户第一次访问,每一次访问都要记录当前的时间\*/* SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat(**"yyyy-MM-dd/HH:mm:ss"**);  String date = simpleDateFormat.format(**new** Date());  *//将时间写入到cookie  //cookie存储的数据不能含有特殊字符",; ",否则会报java.lang.IllegalArgumentException: An invalid character [32] was present in the Cookie value* Cookie cookie = **new** Cookie(**"lastTime"**,date);  cookie.setMaxAge(60\*60\*24\*365);  response.addCookie(cookie);   */\*用户第二次访问\*/  //读取上一次的cookie数据* Cookie[] cookies = request.getCookies();  **if**(cookies!=**null**){  **for** (Cookie cookie1 :cookies){  **if**(cookie1.getName().equals(**"lastTime"**)){  response.setContentType(**"text/html;charset=utf-8"**);  response.getWriter().write(**"用户上一次访问的时间是:"**+cookie1.getValue());  }  }  }  } } |

#### 注意事项

​ 在**上面的案例中我们给cookie添加数据的时候需要注意数据的内容，上面提到的——在cookie值中不能使用分号（;）、逗号（,）、等号（=）以及空格。**

如果存入的数据不合法，会出现一个错误：

java.lang.IllegalArgumentException:   
 An invalid character [32] was present in the Cookie value

​ **那么如果我们一定要保存非法数据在cookie中，该如何操作呢？**

​ **答：cookie对于基本符号、数字、和字母是可以存储的，因此，我们只需要将非法数据转换成符号、数字、和字母形式存储，要使用的时候再转换成正常的中文（解码）即可。**

​ 那么，最后的问题就是，我们如何对中文数据进行编码和解码呢？我们需要来学习一些新的对象和方法。

##### API介绍

1. URLEncoder类：  
    static String encode(String s, String enc)  将指定的字符串，按指定的编码表编码
2. URLDecoder类：  
    static String decode(String s, String enc)  将指定的字符串，按指定的编码表解码

##### 使用步骤

1. 在创建cookie之前将数据编码
2. 将编码后的数据存入cookie
3. 获取到cookie之后，解码数据，获取正常中文内容

##### 注意事项

1. 编码解码两次操作要使用同一个编码表

##### 演示代码

|  |
| --- |
| **package** com.itheima.cookie.demo;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.Cookie; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.net.URLDecoder; **import** java.net.URLEncoder; **import** java.text.SimpleDateFormat; **import** java.util.Date;  @WebServlet(name = **"LastTimeServlet2"**, urlPatterns = **"/LastTimeServlet2"**) **public class** LastTimeServlet2 **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  */\*显示用户上一次访问时间  \* 要求时间格式为: XXXX年XX月XX日 XX时XX分XX秒  \* \*/   /\*用户第一次访问,每一次访问都要记录当前的时间\*/* SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat(**"yyyy年MM月dd日 HH时mm分ss秒"**);  String date = simpleDateFormat.format(**new** Date());  *//将时间写入到cookie  //cookie存储的数据不能含有特殊字符",; ",否则会报java.lang.IllegalArgumentException: An invalid character [32] was present in the Cookie value  //cookie是通过响应头传递数据给浏览器,浏览器不会解析这个数据,cookie是给服务器用,响应头不能直接传递中文数据,需要进行手动url编码  //cookie里面的中文需要进行普通的url编码就可以,因为是服务器写入和读取* date = URLEncoder.*encode*(date,**"utf-8"**);  Cookie cookie = **new** Cookie(**"lastTime"**,date);  cookie.setMaxAge(60\*60\*24\*365);  response.addCookie(cookie);   */\*用户第二次访问\*/  //读取上一次的cookie数据* Cookie[] cookies = request.getCookies();  **if**(cookies!=**null**){  **for** (Cookie cookie1 :cookies){  **if**(cookie1.getName().equals(**"lastTime"**)){  response.setContentType(**"text/html;charset=utf-8"**);  *//由于存的时候是url编码,所以取出来时候也需要url解码* String time = URLDecoder.*decode*(cookie1.getValue(),**"utf-8"**);  response.getWriter().write(**"用户上一次访问的时间是:"**+time);  }  }  }  } } |

## 总结cookie

服务器端生成cookie数据，通过响应头传递给浏览器，在浏览器端存储的文本文件。

浏览器每次请求都会携带与cookie有效路径开头的的cookie数据通过请求头带到服务器端。

cookie存储在客户端，数据是不安全的，对于敏感数据要进行加密后在存储，cookie文件大小有限制4KB.

# session

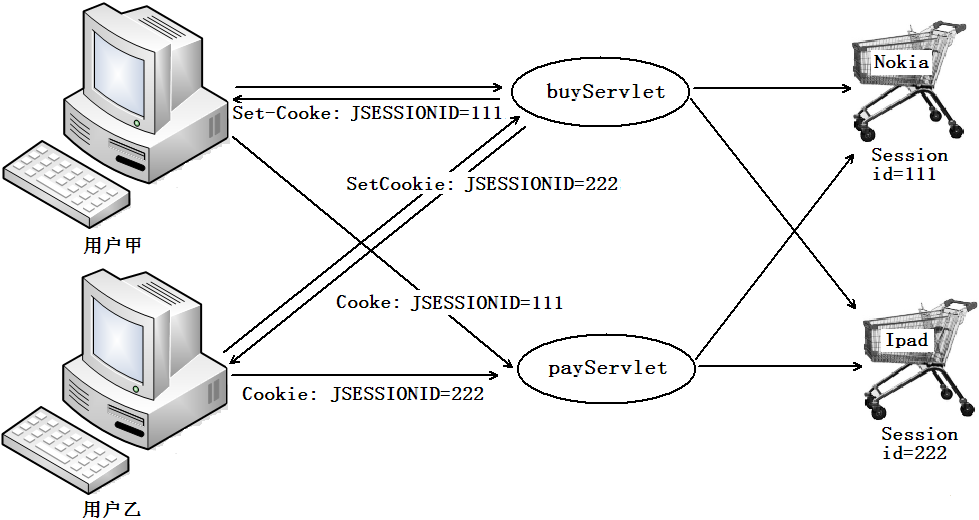
## session的基本概念

session是服务器端会话对象,是服务器为每个浏览器(每个用户)创建的单独私有存储空间.

当人们去医院就诊时，就诊病人需要办理医院的就诊卡，该卡上只有卡号，而没有其它信息。但病人每次去该医院就诊时，只要出示就诊卡，医务人员便可根据卡号查询到病人的就诊信息。Session技术就好比医院发放给病人的就医卡和医院为每个病人保留病例档案的过程。当浏览器访问Web服务器时，Servlet容器就会创建一个Session对象和ID属性，其中，Session对象就相当于病历档案，ID就相当于就诊卡号。当客户端后续访问服务器时，只要将标识号传递给服务器，服务器就能判断出该请求是哪个客户端发送的，从而选择与之对应的Session对象为其服务。

需要注意的是，由于客户端需要接收、记录和回送Session对象的ID，因此，通常情况下，Session是借助Cookie技术来传递ID属性的。

为了使读者更好的理解Session，接下来，以网站购物为例，通过一张图来描述Session保存用户信息的原理，具体如图1-1所示。



​ 图1-1 Session保存用户信息的过程

在图1-1中，用户甲和乙都调用buyServlet将商品添加到购物车，调用payServlet进行商品结算。由于甲和乙购买商品的过程类似，在此，以用户甲为例进行详细说明。当用户甲访问购物网站时，服务器为甲创建了一个Session对象（相当于购物车）。当甲将Nokia手机添加到购物车时，Nokia手机的信息便存放到了Session对象中。同时，服务器将Session对象的ID属性以Cookie (Set-Cookie: JSESSIONID=111)的形式返回给甲的浏览器。当甲完成购物进行结账时，需要向服务器发送结账请求，这时，浏览器自动在请求消息头中将Cookie (Cookie: JSESSIONID=111)信息回送给服务器，服务器根据ID属性找到为用户甲所创建的Session对象，并将Session对象中所存放的Nokia手机信息取出进行结算。

服务器会为每个客户端在服务器内存里创建单独的存储空间,存储空间每个客户端私有.

Session 技术的特点：

1) 运行在服务器端，由服务器为每个浏览器创建单独的存储空间对象，称为服务器会话对象。

2) 每个用户都有自己的会话对象，不同的浏览器代表不同的用户。

3) 会话对象也是一个域对象，主要用于在服务器端存储每个用户自己的数据，不同的用户之间数据不能共享。

 类似于超市的储物箱：



## session的与cookie的区别

|  | **cookie** | **session** |
| --- | --- | --- |
| 会话数据保存的位置 | 浏览器 | 服务器 |
| 数据的安全性 | 不安全 | 安全 |
| 存储数是否有限制 | 有(4kb) | 无 |

## session的应用场景

服务器端存储的用户私有数据都可以使用session存储

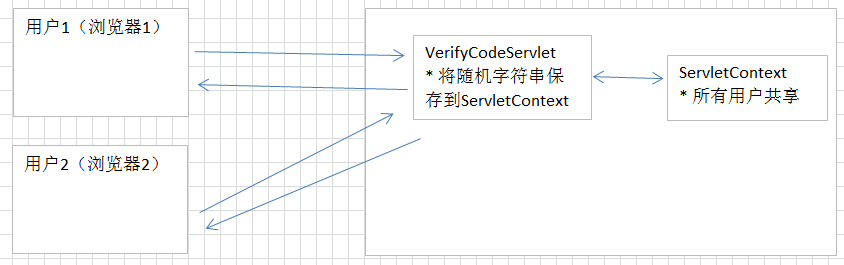
1. 保存购物车数据
2. 保存用户浏览器记录数据
3. 保存用户登录信息数据
4. 保存验证码

### 验证码为什么要存在Session

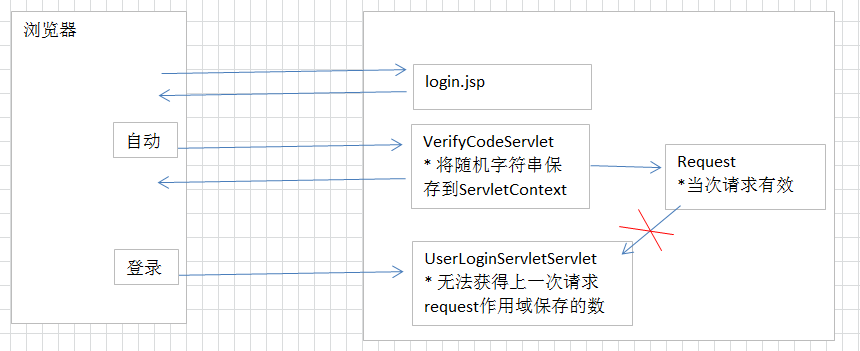
使用servlet生成验证码时，我们需要在服务器记录一份生成的随机字符，当用户提交填写的数据时，将用户输入的数据和服务器缓存的数据进行比对。

将随机字符串保存在ServletContext中或者request中是否可以?

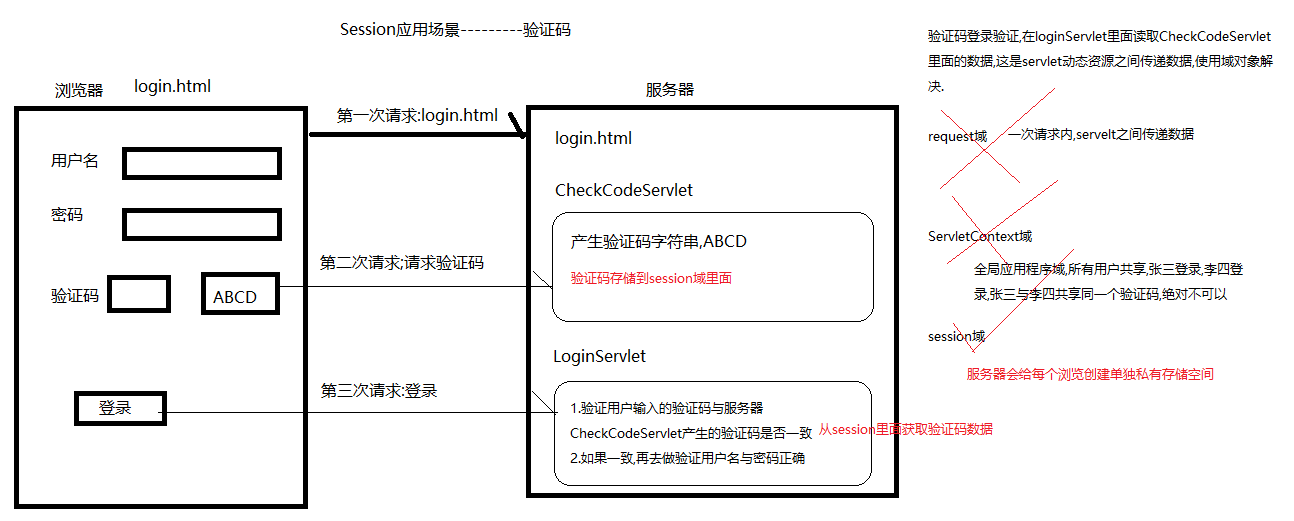
将数据存放到ServletContext，多个用户共享一个验证码。



将数据存放到request作用域，多次请求不能共享数据。



我们发现将数据保存到ServletContext和request中是存在问题的，那么就需要使用会话技术保存用户的私有信息.



## session的基本使用

### session是域对象(会话域对象)

#### 会话域对象介绍

域对象可以存储数据,session是会话域对象,一次会话过程内有效.

从客户端浏览器第一次访问服务器开始,到浏览器关闭,期间可以进行一系列的请求与响应,这个过程称作为一次会话过程.

#### 域对象API介绍

1. HttpServletRequest

HttpSession getSession()  获取session对象

1. HttpSession

void setAttribute(String name, Object value) 在session中保存数据  
   
 Object getAttribute(String name)  从session中获取数据  
   
 void removeAttribute(String name)  从session中移除数据

#### 使用步骤

1. 在AddSessionDataServlet通过request对象获取session对象
2. 在AddSessionDataServlet调用session的setAttribute保存数据
3. 在AddSessionDataServlet调用session的getAttribute获取数据
4. 在GetSessionDataServlet通过request对象获取session对象
5. 在GetSessionDataServlet调用session的removeAttribute移除数据
6. 在GetSessionDataServlet调用session的getAttribute获取数据

#### 注意事项

1. getAttribute方法返回值是Object，因此需要将返回值强制转换才能方便使用。

#### 演示代码

AddSessionDataServlet：

|  |
| --- |
| **package** com.itheima.session;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** javax.servlet.http.HttpSession; **import** java.io.IOException;  @WebServlet(name = **"AddSessionDataServlet"**, urlPatterns = **"/addSessionData"**) **public class** AddSessionDataServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {   */\*  \* session是一个域对象  \* void setAttribute(String name, Object value) 在session中保存数据   Object getAttribute(String name) 从session中获取数据   void removeAttribute(String name) 从session中移除数据  \* \*/   //获取session* HttpSession session = request.getSession();  *//存储数据* session.setAttribute(**"loginUser"**,**"admin"**);    *//每个session都有一个唯一的sessionid,session.getId();,用于标识唯一的session对象* response.getWriter().write(**"session write ok,sessionid="**+session.getId());   } } |

GetSessionDataServlet:

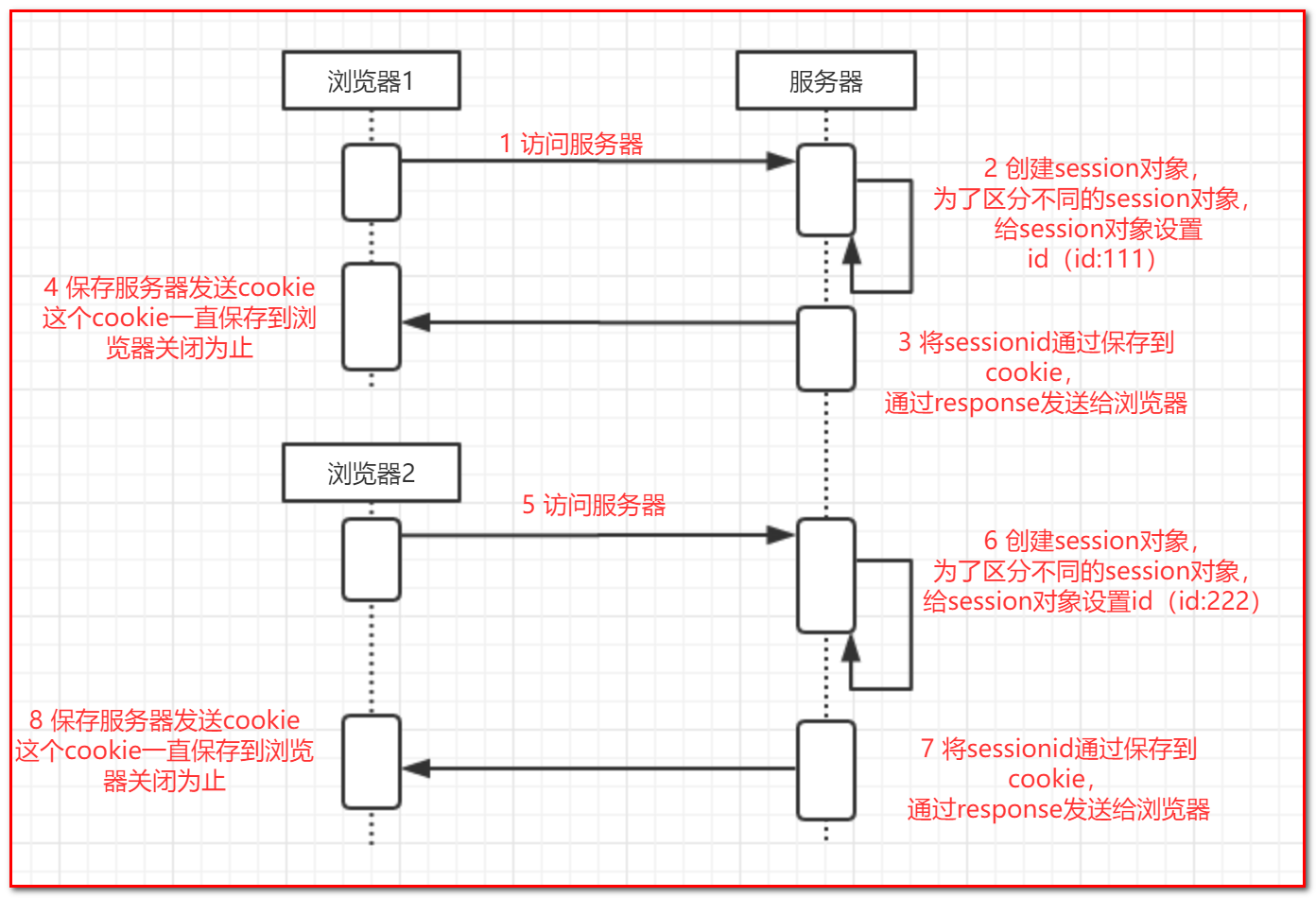
|  |
| --- |
| **package** com.itheima.session;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** javax.servlet.http.HttpSession; **import** java.io.IOException;  @WebServlet(name = **"GetSessionDataServlet"**, urlPatterns = **"/getSessionData"**) **public class** GetSessionDataServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  */\*  \* 获取session域里面的数据  \* \*/* HttpSession session = request.getSession();  Object object = session.getAttribute(**"loginUser"**);    **if**(object!=**null**){  String loginUser = object.toString();  response.getWriter().write(**"loginUser="**+loginUser);  }**else**{  response.getWriter().write(**"no data"**);  }  } } |

### session的原理分析

服务器为不同的浏览器创建了不同session对象。服务器是怎么识别不同浏览器？

浏览器访问服务器产生sessionid,浏览器不关闭sessionid保持不变,但是关闭浏览器后重新打开浏览器产生了新的sessionid,服务器怎么识别浏览器关闭的呢?

让我们一起学习session的运行原理





### session的有效期

#### 介绍

session的有效期是一次会话内，关闭了浏览器session就没用了，但是session对象依然存储在服务器内存中，session对象并没有销毁,session默认有效期30分钟.距离最后一次请求超过30分钟,session过期.

#### 改变会话过期的时间

方式一：在web.xml中配置session过期时间

1). 通过 web.xml 中的配置

<!-- 配置会话的时间，以分钟为单位 -->

<session-config>

<session-timeout>1</session-timeout>

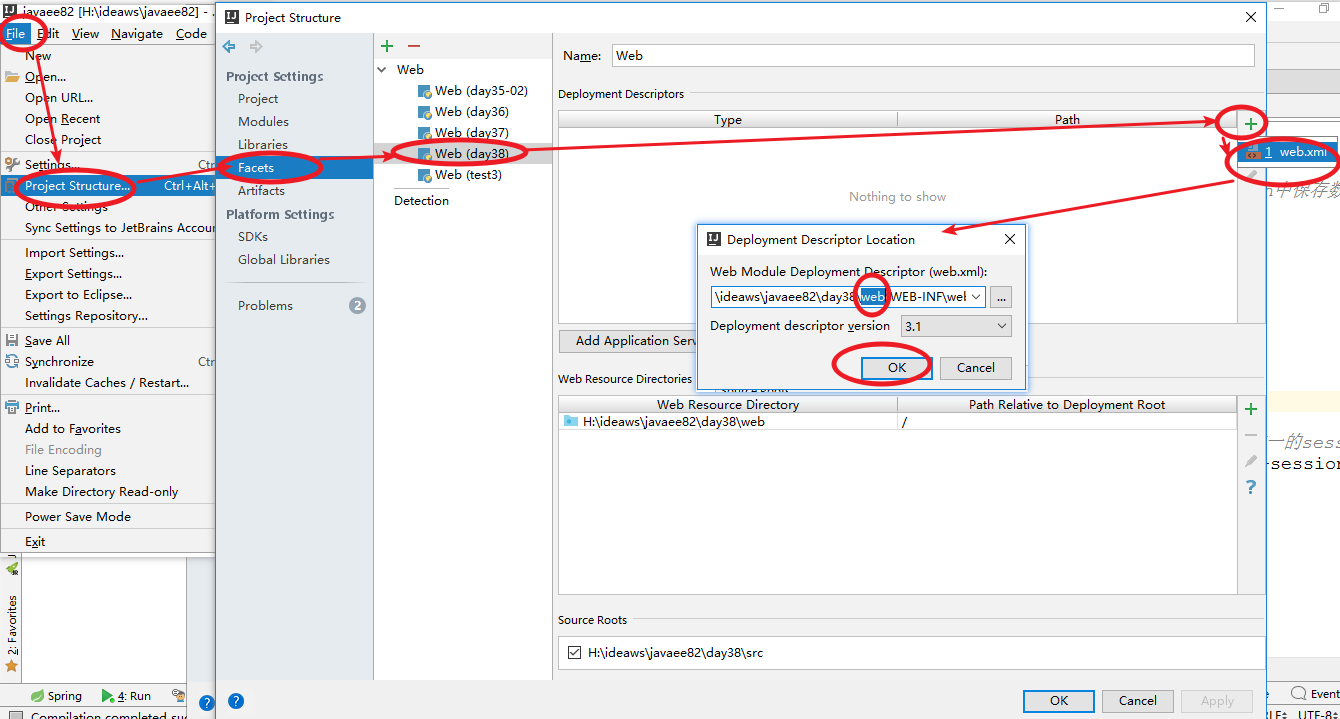
</session-config>

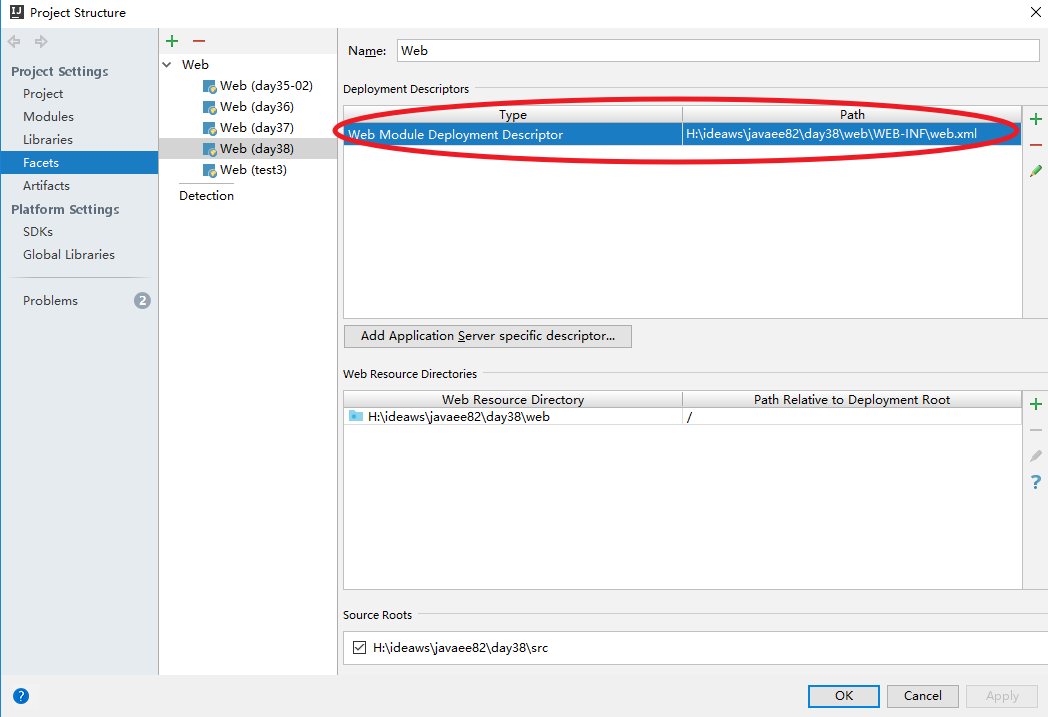
2). session-timeout 的解释：

如果设置为 0 或负数，则会话不过期，但受限于服务器的内存空间。

**注意：一般没有人会去更改session默认30分钟有效期。**

添加web.xml方式





3. 方式二：会话立刻失效（最有用）

session.invalidate();

应用场景，注销登录。就是让session立刻过期

session.removeAttribute(“loginUser”);也可以的

### session实现关闭浏览器继续可以访问数据

​ 通过上面的原理图，我们发现记录sessionid的cookie在关闭浏览器之后就到期了，那么如果我们希望——关闭浏览器继续可以访问session数据，该怎么做呢？

#### 案例需求

​ 关闭浏览器继续可以访问session数据

#### 案例效果

​ 关闭浏览器继续可以访问session数据

#### 案例分析

1. 创建一个servlet
2. 通过request对象获取session的id
3. 自定义一个cookie对象，保存session的id，注意cookie名称为"JSESSIONID"，设置一个比较长的生存时间
4. 发送给浏览器

#### 代码实现

|  |
| --- |
| **package** com.itheima.session;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.\*; **import** java.io.IOException;  @WebServlet(name = **"ChangeCookieTimeOutServlet"**, urlPatterns = **"/changeCookieTimeOutServlet"**) **public class** ChangeCookieTimeOutServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {   HttpSession session = request.getSession();  *//自己创建cookie覆盖默认的cookie,我们的cookie有效期要长一点* Cookie cookie = **new** Cookie(**"JSESSIONID"**,session.getId());  *//活的久一点* cookie.setMaxAge(10000);  response.addCookie(cookie);    response.getWriter().write(**"sessionId:"**+session.getId());  } } |

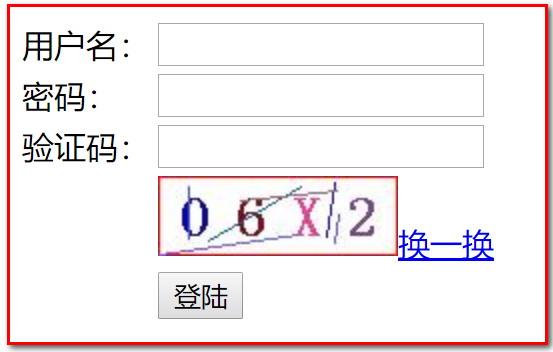
访问一次服务器，然后关闭浏览器，再访问一次：两次session对象一致

org.apache.catalina.session.StandardSessionFacade@49ef5434  
org.apache.catalina.session.StandardSessionFacade@49ef5434

## 案例:使用session存储验证码完成登录功能

### 案例需求

1. 在登录页面用户登录的时候要查看到验证码，如图所示：

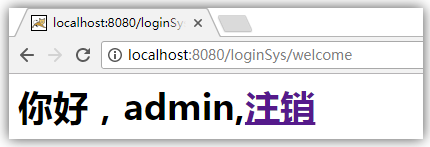


1. 在生成页面验证码图片的同时，使用session存储验证码
2. 在处理用户登录请求的时候，首先校验验证码
3. 校验通过才能执行登录操作,登录成功后跳转到主页WelcomeServlet输出欢迎登录用户和注销连接.如果没有登录,主页WelcomeServlet只输出”登录”连接
4. 登录成功后,可以点击注销链接,提交请求到LoginOutServlet处理注销登录,并跳转到WelcomeServlet只显示”登录”链接

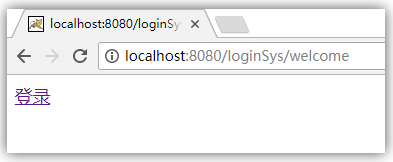
### 案例效果

​ 用户提交验证码，与session中保存验证码校验一致才能登录

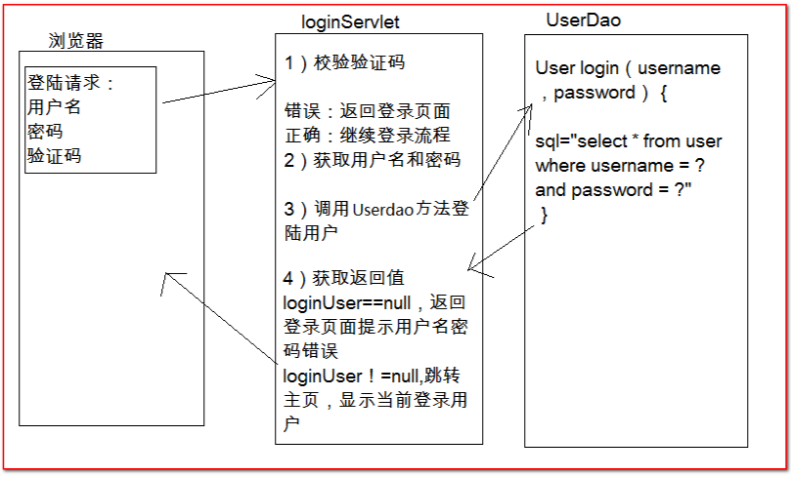
登录成功后访问WelcomServlet的效果

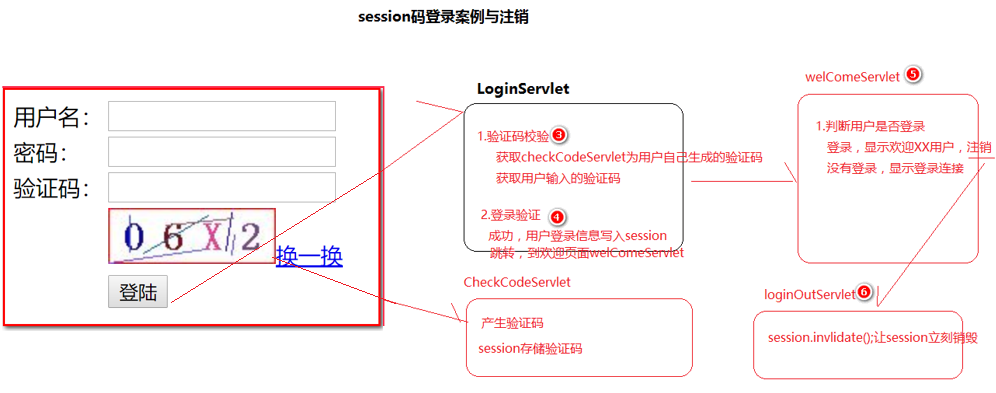


没有登录访问welcomeServlet的效果



### 案例分析





### 代码实现

1. 页面login.html

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**form action="LoginServlet" method="post"**>  <**table**>  <**tr**>  <**td**>用户名</**td**>  <**td**><**input type="text" name="username"**/></**td**>  </**tr**>  <**tr**>  <**td**>密码</**td**>  <**td**><**input type="password" name="password"**/></**td**>  </**tr**>  <**tr**>  <**td**>验证码：</**td**>  <**td**><**input type="text" name="checkcode"**/></**td**>  </**tr**>  <**tr**>  <**td**></**td**>  <**td**><**img id="img" onclick="***changeCode*(**this**);**" src="CheckCodeServlet"**/></**td**>  </**tr**>  <**tr**>  <**td**></**td**>  <**td**><**input type="submit" value="登录"**/></**td**>  </**tr**>  <**script type="text/javascript"**>  **function** *changeCode*(imgObj) {  *//点击验证码图片重写请求CheckCodeServlet  //浏览器对静态资源img,src值不变的话,浏览器会一直取缓存数据,只有src值改变,才会重新请求服务器获取资源* imgObj.**src**=**"CheckCodeServlet?t="**+**new *Date***().getTime();  }    </**script**>  </**table**> </**form**> </**body**> </**html**> |

​

1. 配置验证码CheckCodeServlet

|  |
| --- |
| **package** com.itheima.servlet;  **import** javax.imageio.ImageIO; **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.awt.\*; **import** java.awt.image.BufferedImage; **import** java.io.IOException; **import** java.util.Random;  @WebServlet(name = **"CheckCodeServlet"**, urlPatterns = **"/CheckCodeServlet"**) **public class** CheckCodeServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }    *//随机类* **private** Random **random** = **new** Random();   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  */\*验证码实现原理  \*  \* 输出一张内存图片,不是资源图片  \* \*/   //BufferedImage,内存图片对象  //new BufferedImage(int width, int height, int imageType);  // width,宽度  // height,高度  // imageType,图片类型,图片模式,RGB* BufferedImage image = **new** BufferedImage(90, 30,BufferedImage.***TYPE\_INT\_RGB***);  *//获取画笔* Graphics g= image.getGraphics();  *//设置画笔颜色* g.setColor(Color.***white***);  *//填充区域* g.fillRect(0,0,90,30);  *//设置边框线颜色* g.setColor(Color.***red***);  *//设置边框线* g.drawRect(0,0,89,29);   *//画干扰线,随机画2条* **for**(**int** i=0;i<2;i++) {  *//每条线改变画笔的颜色* g.setColor(getRandomColor());  **int** x1 = **random**.nextInt(89);  **int** y1 = **random**.nextInt(29);  **int** x2 = **random**.nextInt(89);  **int** y2 = **random**.nextInt(29);  g.drawLine(x1,y1,x2,y2);  }   *//画验证码  //准备验证码范围* String data = **"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890"**;  *//遍历字符串,随机取4个,并且将每个字符画到图片上去  //定义变量接收4个验证的字符串* StringBuilder checkCodeStr = **new** StringBuilder();  **for**(**int** i=0;i<4;i++){  **int** index = **random**.nextInt(data.length());  **char** item = data.charAt(index);  *//拼接* checkCodeStr.append(item);*//后面用,验证码验证  //一个字一个随机颜色* g.setColor(getRandomColor());  *//画到图片上去* g.drawString(item+**""**,10+i\*20,20);  }   *//将验证码写入到session* request.getSession().setAttribute(**"sysCheckCode"**,checkCodeStr.toString());    *//ImageIO对象将内存图片输出到浏览器* ImageIO.*write*(image,**"png"**,response.getOutputStream());  }   */\*\*  \* 随机颜色  \** ***@return*** *\*/* **private** Color getRandomColor(){  *//Color(int r, int g, int b)* **int** r=**random**.nextInt(256);*//red* **int** g=**random**.nextInt(256);;*//green* **int** b=**random**.nextInt(256);;*//blue* **return new** Color(r,g,b);  } } |

1. 登录LoginServlet

|  |
| --- |
| **package** com.itheima.servlet;  **import** com.itheima.dao.UserDao; **import** com.itheima.model.User; **import** org.apache.commons.beanutils.BeanUtils;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.util.Map;  @WebServlet(name = **"LoginServlet"**, urlPatterns = **"/LoginServlet"**) **public class** LoginServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   *//实例UserDao* **private** UserDao **userDao** = **new** UserDao();   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {   **try** {  *//解决post中文乱码* request.setCharacterEncoding(**"utf-8"**);  *//解决输出中文乱码* response.setContentType(**"text/html;charset=utf-8"**);   *//先验证验证码  //1.获取用户输入的验证码* String userCheckCode = request.getParameter(**"checkcode"**);  *//2.获取服务器生成的验证码* String sysCheckCode = request.getSession().getAttribute(**"sysCheckCode"**).toString();  *//3.对比* **if**(!sysCheckCode.equalsIgnoreCase(userCheckCode)){  *//给用户提示,验证码错误,返回登录页面* response.getWriter().write(**"<script>alert('验证码错误');history.go(-1);</script>"**);  *//程序到这里已经应该结束了,不可以再往下执行了* **return**;  }   *//获取登录请求数据封装到User里面  //原始方法:一个一个获取,再封装到User对象  // String username = request.getParameter("username");  // String password = request.getParameter("password");  // User user = new User(0,username,password);  //最新方法:使用BeanUtils工具类一次性将所有数据封装到User对象里面  //BeanUtil使用步骤  // 1.导入BeanUtils的jar包  // 2.使用BeanUtils工具类,BeanUtils.populate(java对象,map);  // java对象,就是要封装数据的对象  // map,是封装了所有java对象属性数据的集合* Map<String,String[]> map = request.getParameterMap();  User user = **new** User();  System.***out***.println(**"封装前user数据:"**+user.toString());   BeanUtils.*populate*(user,map);   System.***out***.println(**"封装后user数据:"**+user.toString());   *//访问dao层数据库查找对应用户对象* User loginUser = **userDao**.login(user);  *//判断用户是否为null* **if**(loginUser!=**null**) {  *//存在,输出登录成功  //response.getWriter().write("登录成功");   //登录成功,用户信息写入session中,用于其他资源判断用户是否登录* request.getSession().setAttribute(**"loginUser"**,loginUser);  *//登录成功,跳转到WelcomeServlet* response.sendRedirect(request.getContextPath()+**"/welcome"**);  }**else** {  *//不存在,将用户名存储到请求域,跳转到LoginFailServlet,获取请求域数据,输出XX用户登录失败  //将用户名存储到域对象里面* request.setAttribute(**"username"**,user.getUsername());  request.getRequestDispatcher(**"LoginFailServlet"**).forward(request,response);  }   } **catch** (Exception e) {  e.printStackTrace();  }  } } |

1. 登录成功Welcomeservlet

|  |
| --- |
| **package** com.itheima.servlet;  **import** com.itheima.model.User;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.io.PrintWriter;  @WebServlet(name = **"WelcomeServlet"**, urlPatterns = **"/welcome"**) **public class** WelcomeServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {   *//输出中文乱码* response.setContentType(**"text/html;charset=utf-8"**);  PrintWriter out = response.getWriter();  *// 1.判断用户是否登录* User user = (User)request.getSession().getAttribute(**"loginUser"**);   **if**(user!=**null**){  *// 2.有登录--显示,你好,xxx,注销连接* out.write(**"<div>你好,"**+user.getUsername()+**",<a href='loginOut'>注销</a></div>"**);  }**else** { *// 3.没有登录,显示,登录连接* out.write(**"<div><a href='login.html'>请登录</a></div>"**);  }   } } |

1. dao：

|  |
| --- |
| **package** com.itheima.dao;  **import** com.itheima.model.User; **import** com.itheima.util.JdbcUtils; **import** org.springframework.dao.DataAccessException; **import** org.springframework.jdbc.core.BeanPropertyRowMapper; **import** org.springframework.jdbc.core.JdbcTemplate;  **import** java.sql.SQLException;  */\*\*  \* 用户数据库访问类  \*/* **public class** UserDao {  *//核心类* **private** JdbcTemplate **jdbcTemplate** = **new** JdbcTemplate(JdbcUtils.*getDataSource*());   **public** User login(User user)**throws** SQLException {  *//定义sql* String sql=**"select** *\** **from user where username=? and password=?"**;  **try** {  *//执行sql* **return jdbcTemplate**.queryForObject(  sql,**new** BeanPropertyRowMapper<User>(User.**class**),  user.getUsername(),user.getPassword()  );  *//jdbcTemplate如果查不到数据会抛出异常,我们需要抓到* } **catch** (DataAccessException e) {  e.printStackTrace();  **return null**;  }  } } |