# Cookie和Session

**【学习目标】理解、了解、应用、记忆**

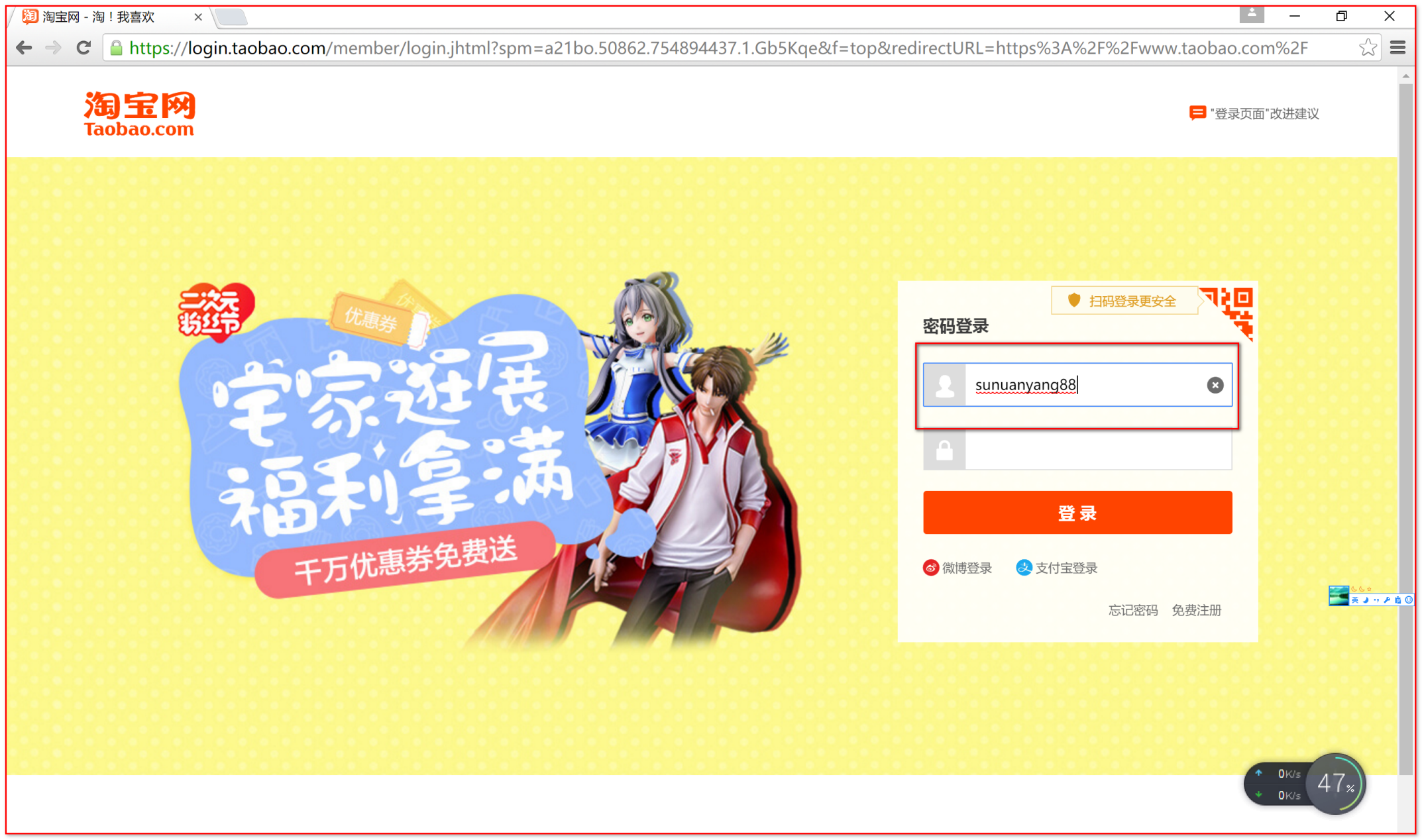
通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【理解】能够阐述Cookie技术**
2. **【应用】能够独立完成Cookie代码编写**
3. **【应用】能够阐述Session技术**
4. **【应用】能够编写代码完成session操作**
5. **【理解】能够编写代码实现Session应用--登陆验证码**

# Cookie的由来（浏览器端的会话技术——保存数据技术）

## Cookie的需求分析

场景：



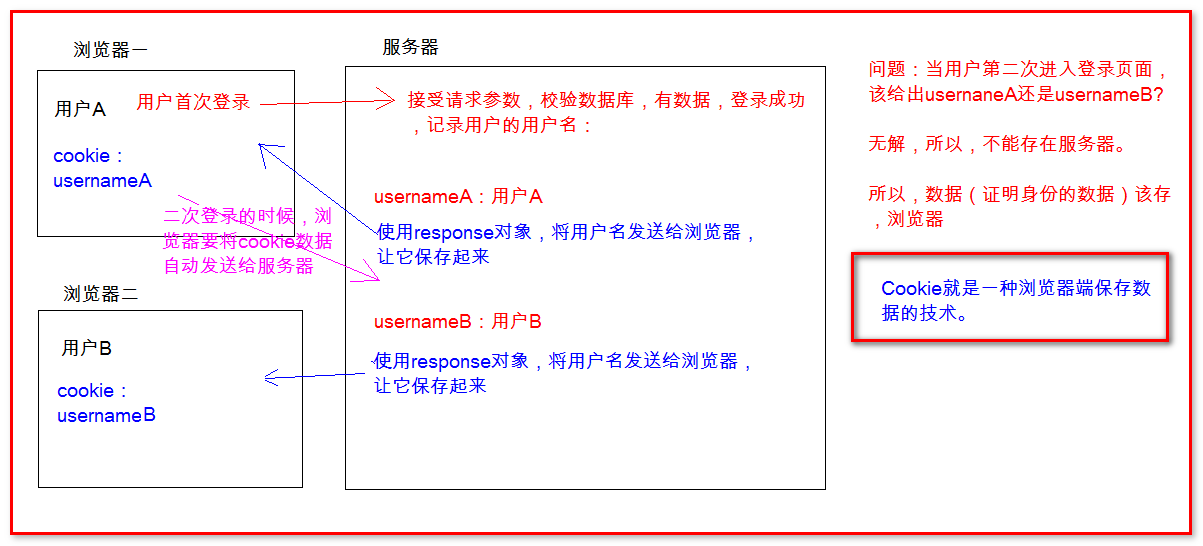
现象：进入淘宝的登陆页面，显示了我上次登录的用户名

显示上次登录用户名这个功能，如何实现？

1 上一次登录的时候，肯定将用户名记录下来

2 肯定是在服务器处理登录的时候，将用户名记录下来

3 那么这个数据到底存在哪里了呢？



# Cookie介绍&Cookie案例

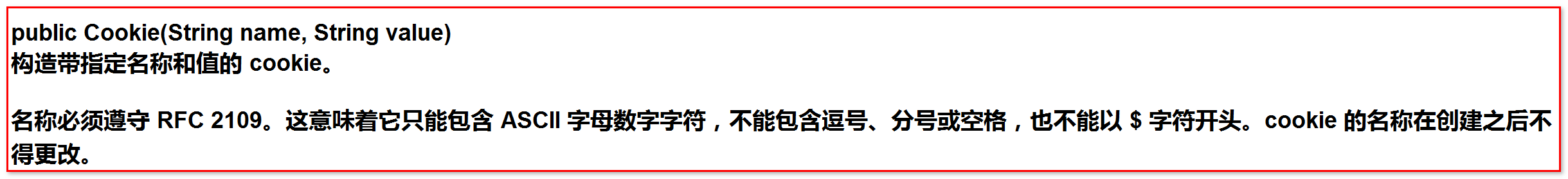
## Cookie快速入门（创建Cookie对象，发送给浏览器，服务器接收请求中cookie数据）

在哪里创建cookie对象？

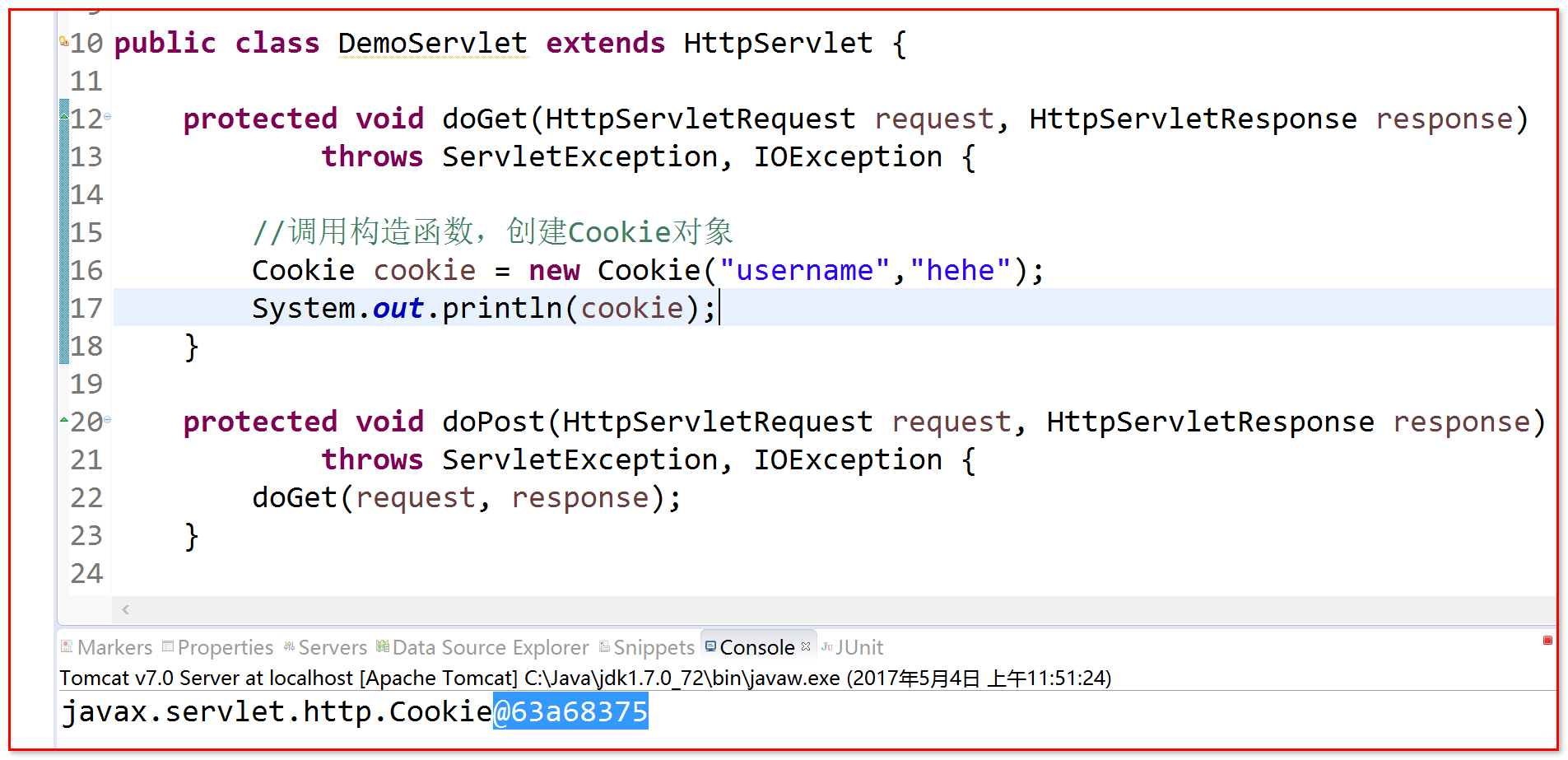
创建Servlet。



1 如何创建Cookie对象？查询构造函数



2 在哪里创建Cookie对象？根据之前的分析，记住用户名的动作发生在服务器端，所以在服务器创建Cookie



3 如何将cookie发送给浏览器？给浏览器发出响应的是response对象，将cookie交给response发送

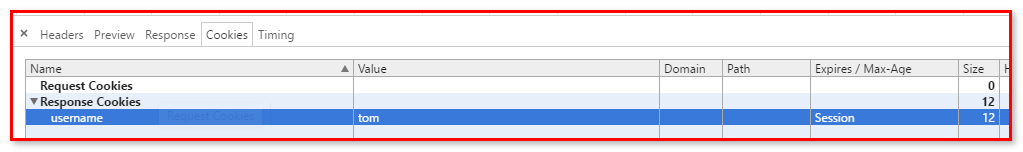
Response API：



代码操作：



效果：



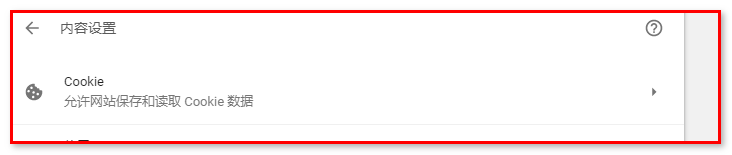
4 如何在浏览器查看cookie（第二种查看cookie的方式）？

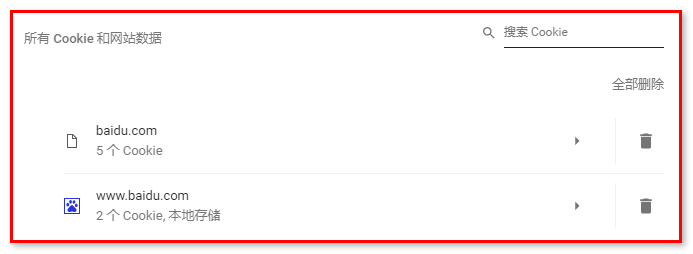
如何打开谷歌浏览器查看cookie?











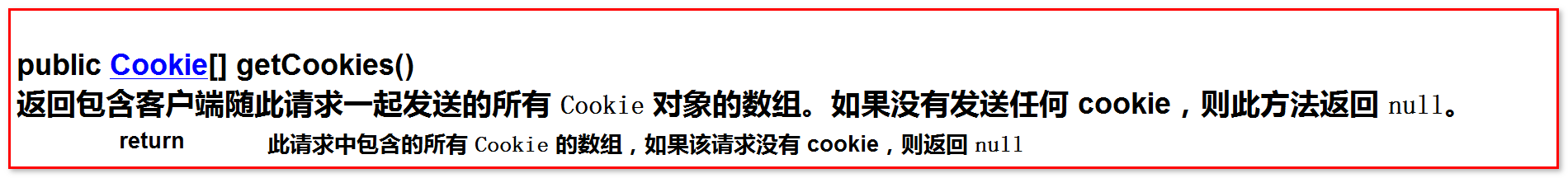
ok了

那么接下来，记录在cookie中的数据，服务器如何使用？

1 浏览器将cookie发送给服务器需要我们有特别的操作吗？不需要，发送http请求，从来都是服务器操作，跟我们没关系

2 浏览器自动发送cookie给服务器，服务器如何获取到这个cookie？通过request对象获取

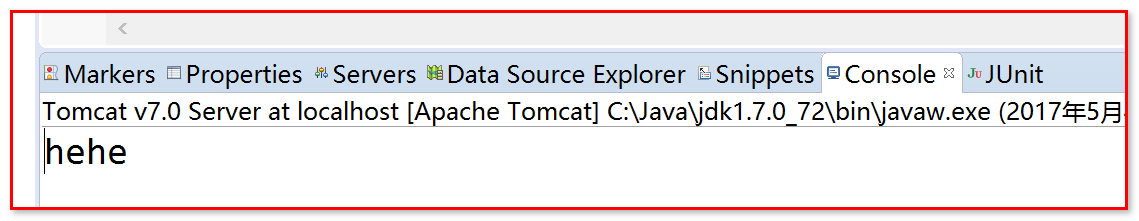
Request API：



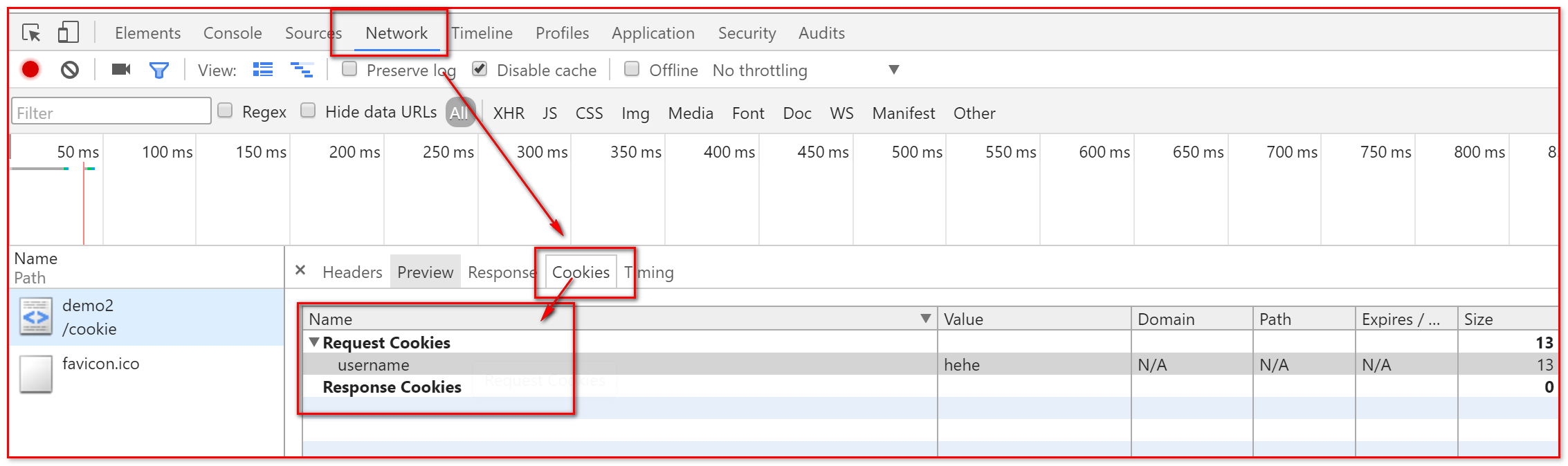
代码演示：



效果：



补充：通过浏览器开发者工具，查看cookie



## Cookie的生命周期

问题1: cookie可以记录数据，但是数据需要一直记录永远不删除吗？

答：数据不删除，数据量越来越大，影响存储空间，数据量越大网络传输cookie慢，服务器解析cookie，也是效率不高。

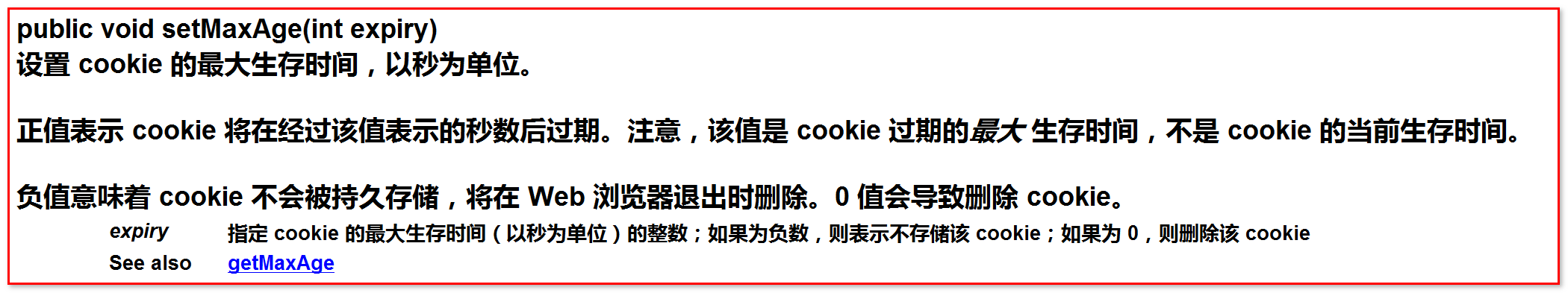
问题2：那么如何删除cookie数据？

答：我们在服务器创建cookie给浏览器，但是我们无法操作浏览器，因此，在cookie创建的时候设置cookie的生存时间，时间一到自动去死

如果不设置过期时间，默认是多少？

答：浏览会话结束时——浏览器关闭的时候

Cookie API：

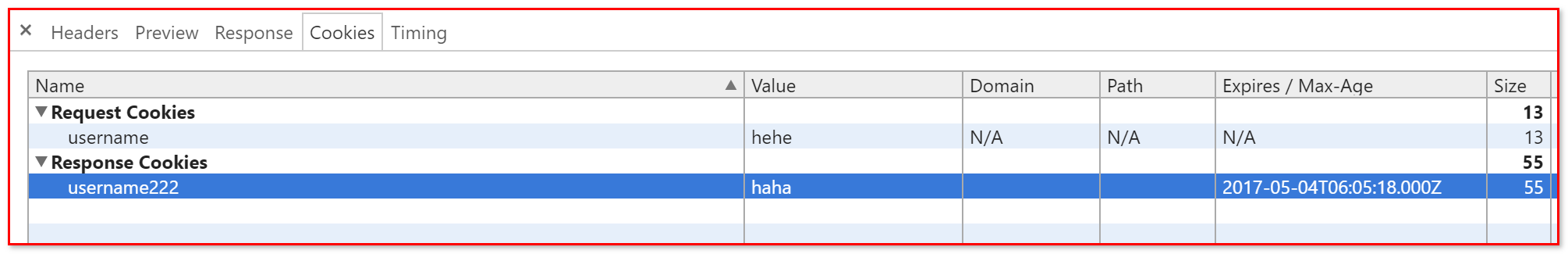


代码演示:



效果：

数据确实发送了



30秒后：username222消失了

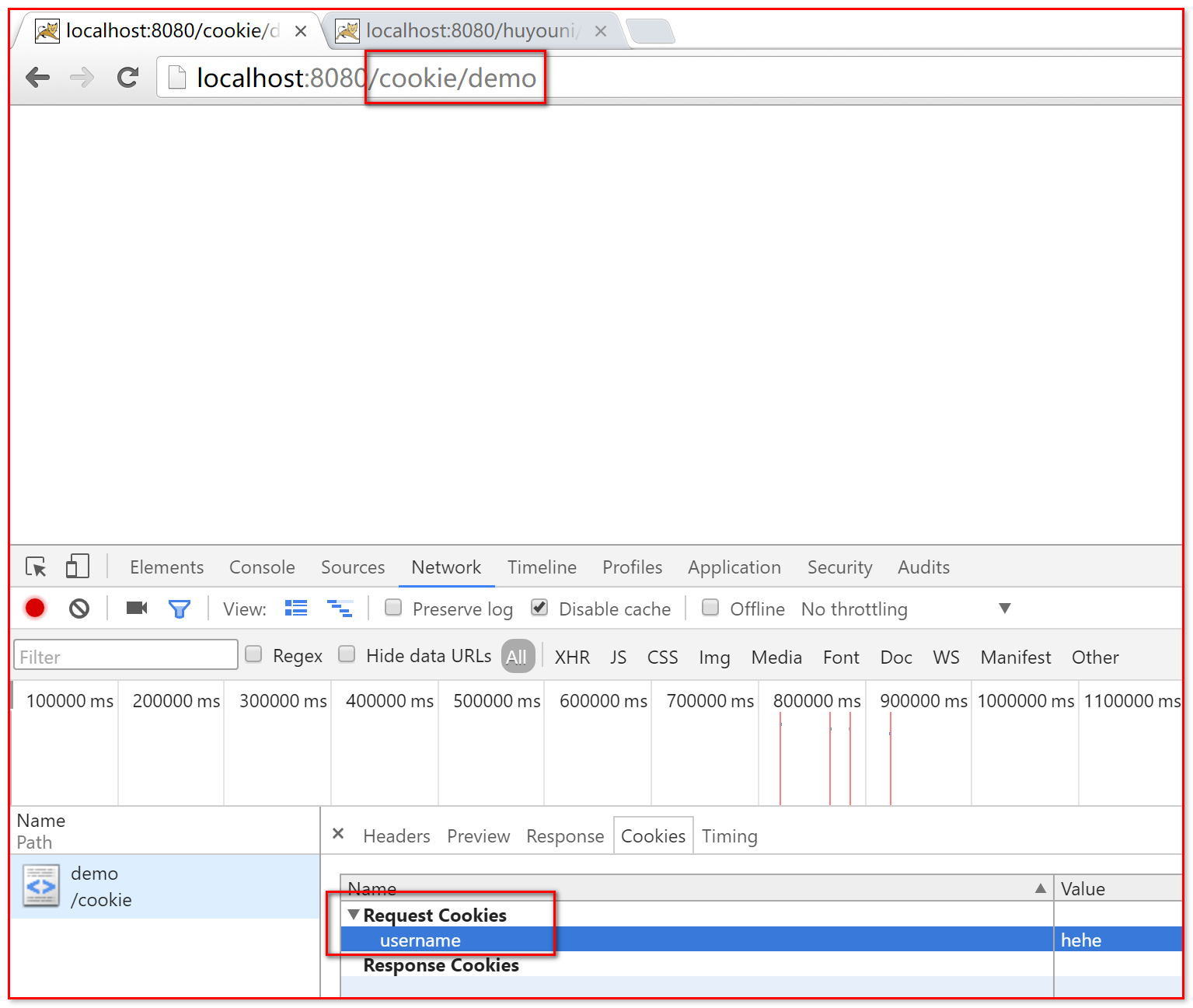


chrome://settings/content/cookies——打开查看谷歌浏览器cookie

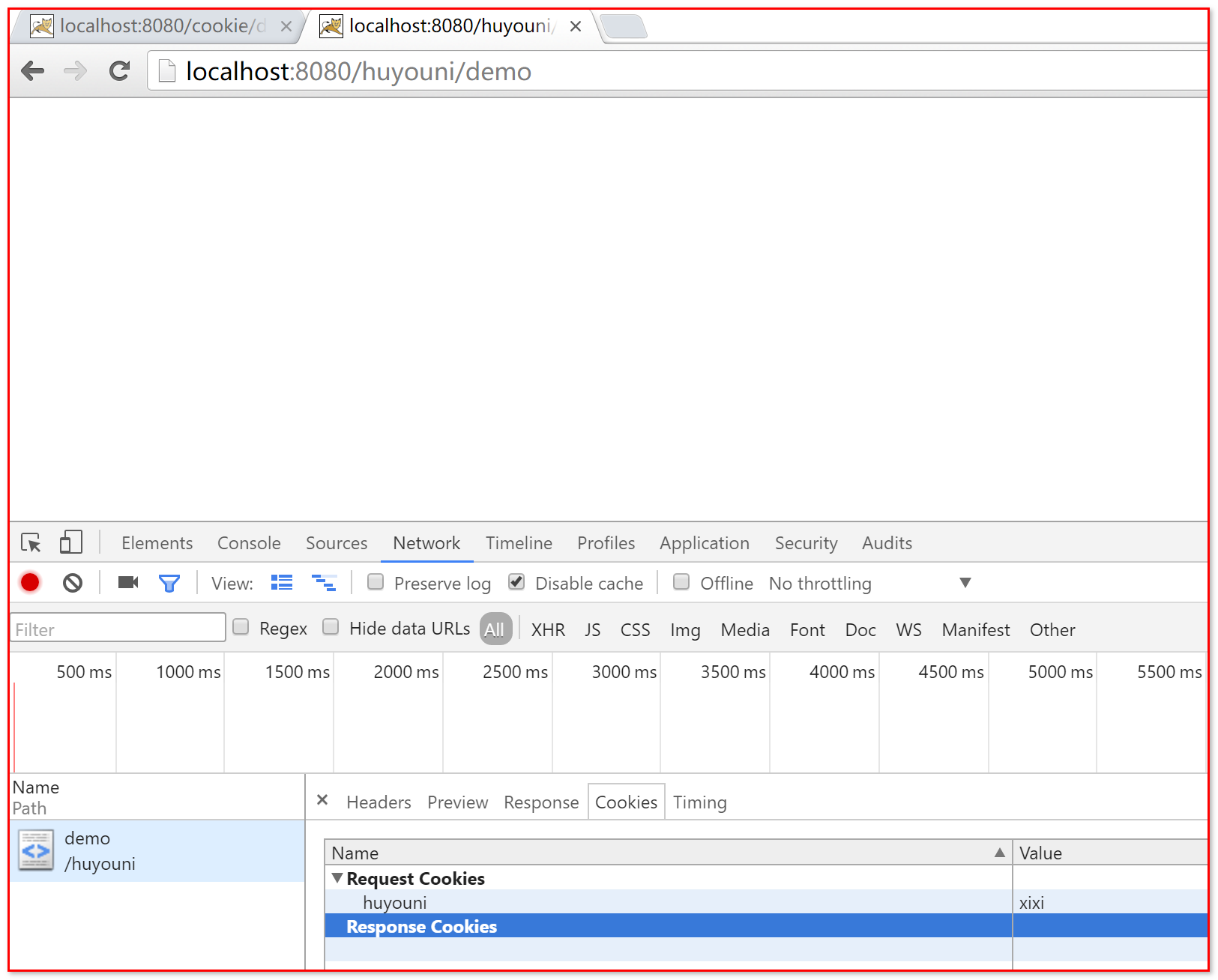
## Cookie的路径设置

现象：

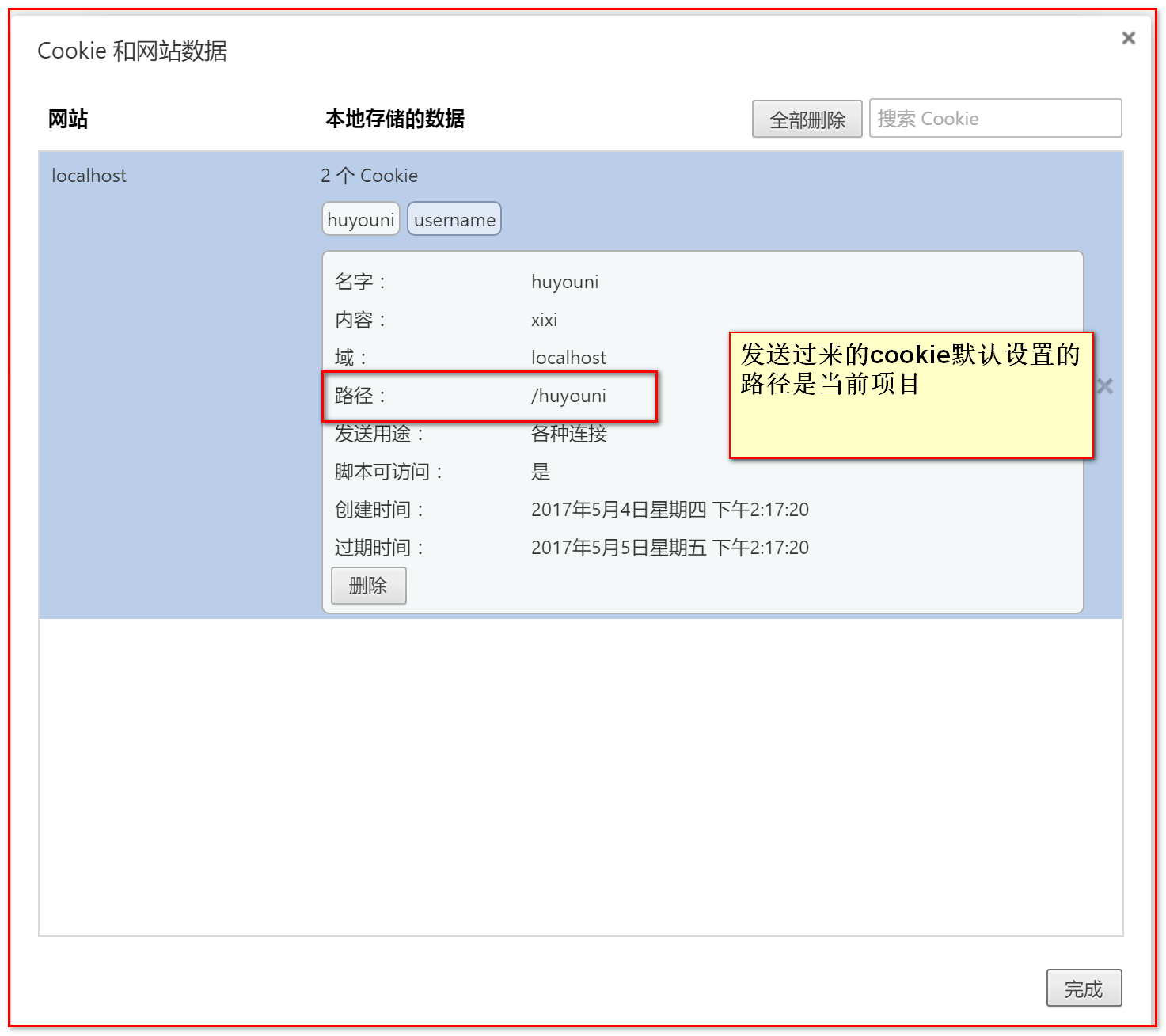
第一张图：



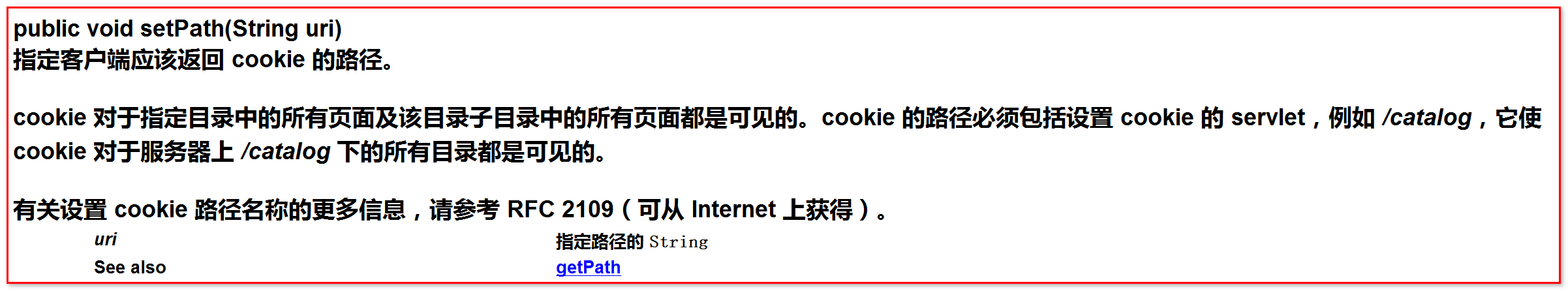
第二张图：



问题：cookie认路！！！为什么cookie认路？



那么这个默认的操作，调用的是哪个API呢？

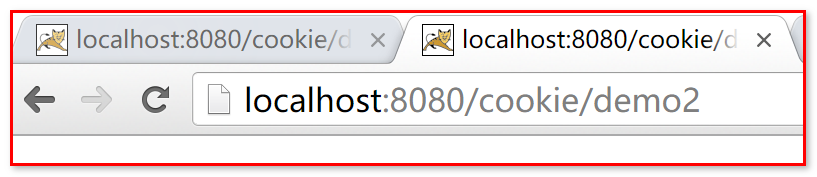


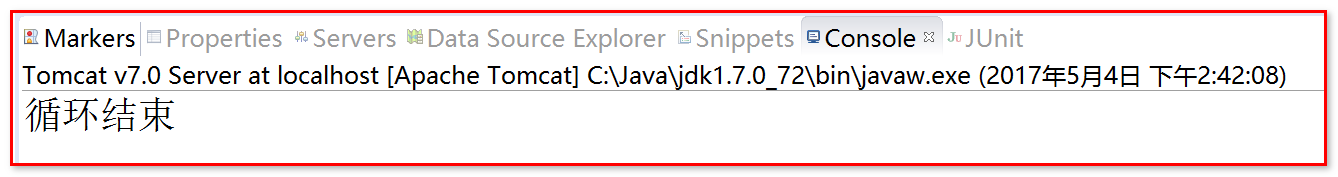
测试setPath方法一：同级别的不同目录是否可以获取cookie？



效果：获取不到cookie

访问demo2



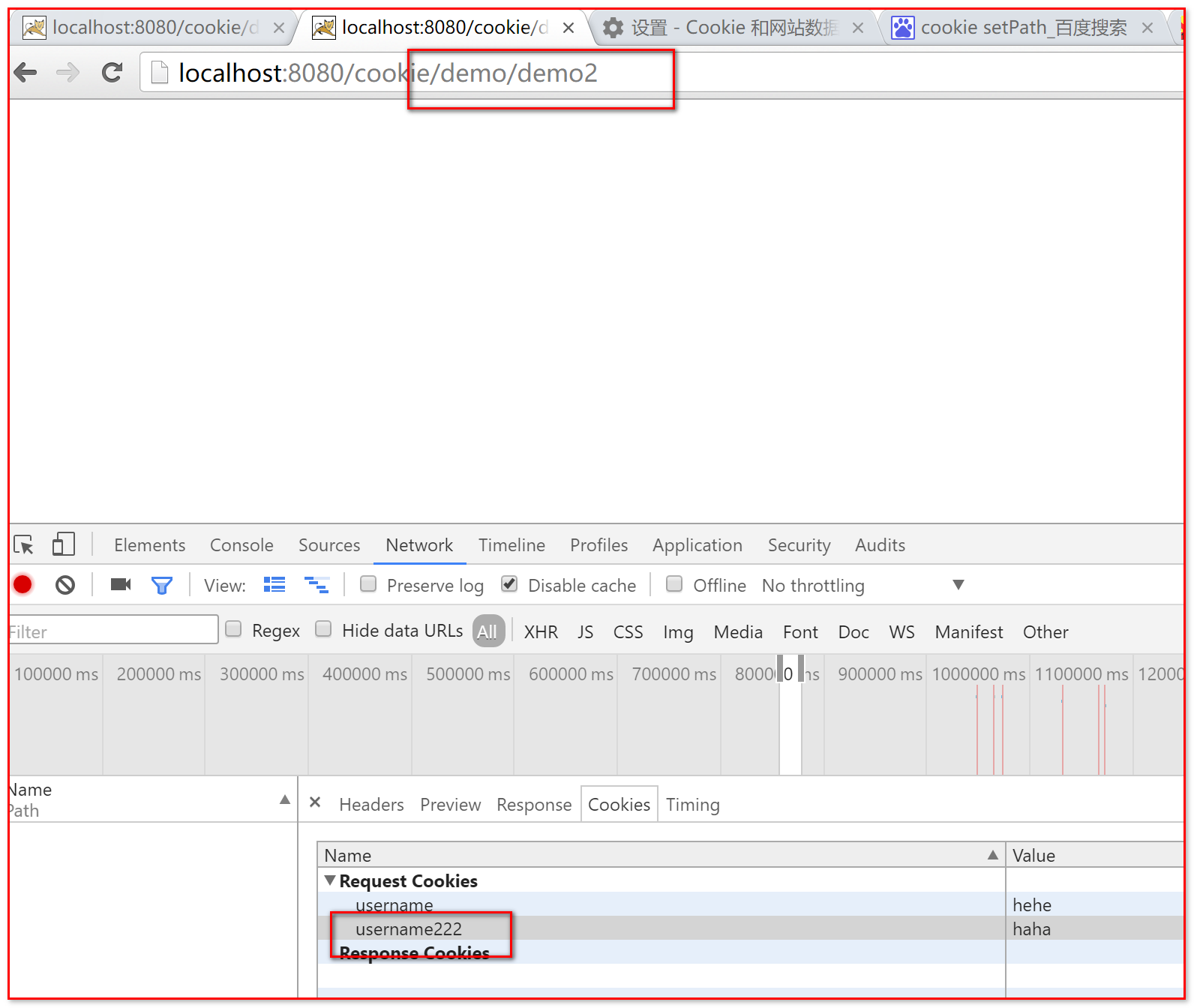


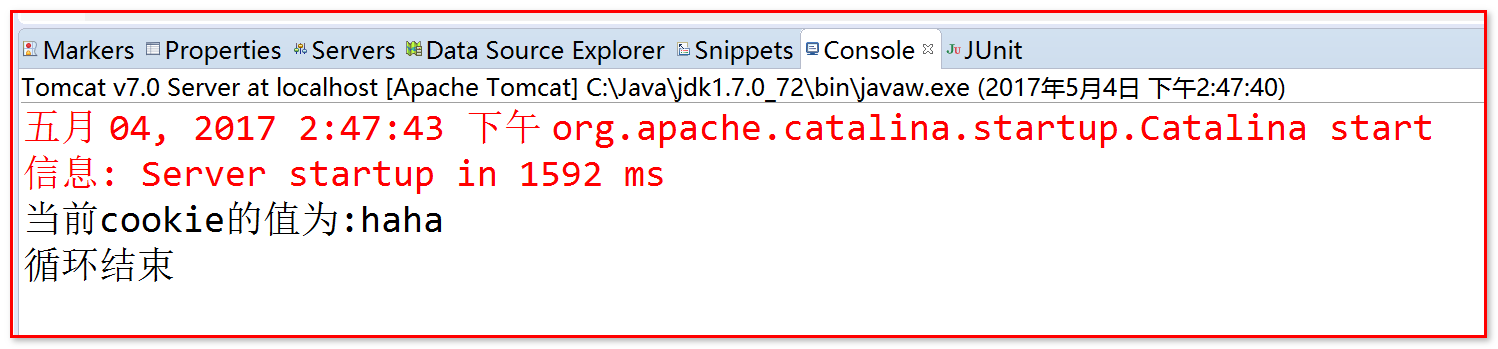
测试setPath方法二：当前目录的子目录是否可以获取数据？

设置成子目录：



测试效果：





结论：setPath方法可以设置当前目录和旗下子目录servlet都获取cookie，一般setPath(“/”),表示当前项目所有目录都可以获取cookie

一般路径不去设置，或者，设置为“/”;

### 删除Cookie（马上去死）

删除cookie其实是发送一个新的cookie，设置生成时间为0，而且设置数据为空字符或则null，通过response对象发送之后，会，覆盖之前的cookie

注意，删除cookie时，path必须一致，否则不会删除

指的是: 服务器通知浏览器 删除自己管理的cookie.

1、将cookie的name（key）保持一致，value 设置为 "";

cookie = new Cookie("username","")

2、设置存活时间为0，

cookie.setMaxAge(0)

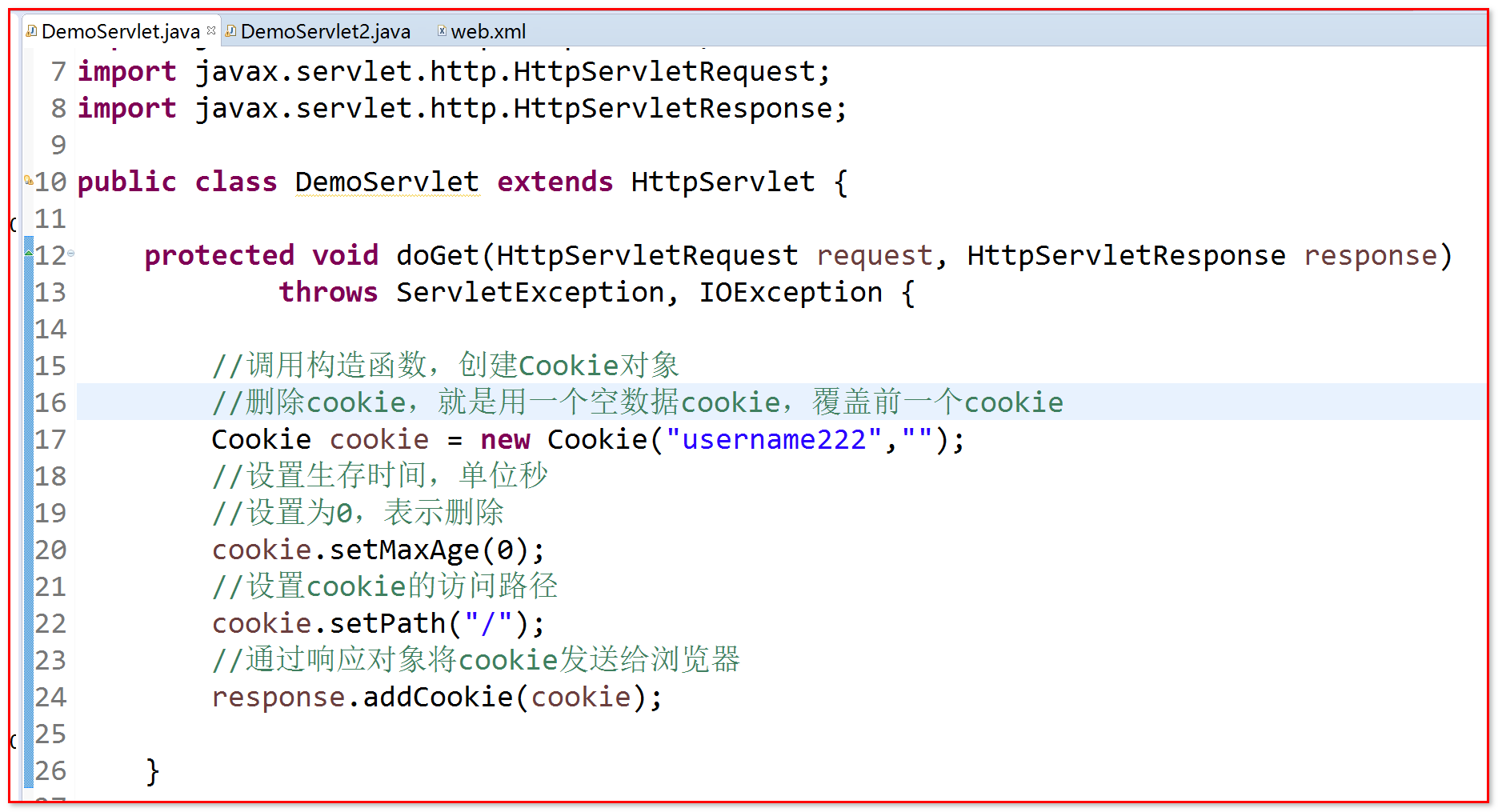
3、路径要发送cookie时保持一致，没有路径不需要设置。

cookie.setPath("/");

4、将cookie发送给浏览器。

response.addCookie(cookie)

需求：服务器让浏览器删除缓存中的cookie：



效果：再次获取时为空



## cookie案例一：显示用户上次访问的时间

当用户访问某些web应用的时候，经常会显示出上次的访问时间。

例如：QQ登录成功后，会显示用户上次的登录时间。

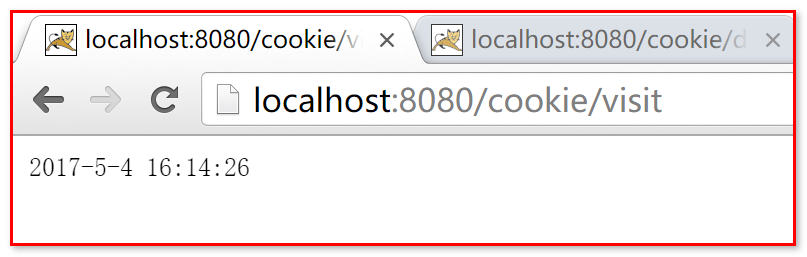
画图分析：



代码演示：



效果：



# Session（服务器端的会话技术）

Cookie技术可以将用户的信息保存在各自的浏览器中，

优点：很明显实现了同一个用户不同请求中数据共享。

缺点：黑客可以利用脚本等手段 窃取cookie中的重要数据，从而泄漏个人的隐私，存在巨大的安全隐患。

Cookie数据保存在用户电脑中，不是每个用户都是电脑高手，对数据安全措施，做的必然不完善。

诞生新技术：session，在服务器端保存用户的数据。（注意：session技术，还是依赖cookie技术）

会话：从第一次请求服务器开始，一直到关闭浏览器，这一段操作，称之为：会话。相当于平时打电话。

## 什么是session？（重点）

场景：

客户到KFC买东西，点了劲脆鸡腿堡、老北京鸡肉卷、可乐，服务员将三个菜品一次放入打包盒，然后让你带走。

浏览器 发送三次请求 服务器 数据 容器 后期获取数据的时候，从容器中获取

如果使用Cookie解决需求：

使用三个cookie保存数据——但是有问题：cookie数量增多网络传输慢，解析慢。

最好的方式：

1 需要一个容器来存放数据

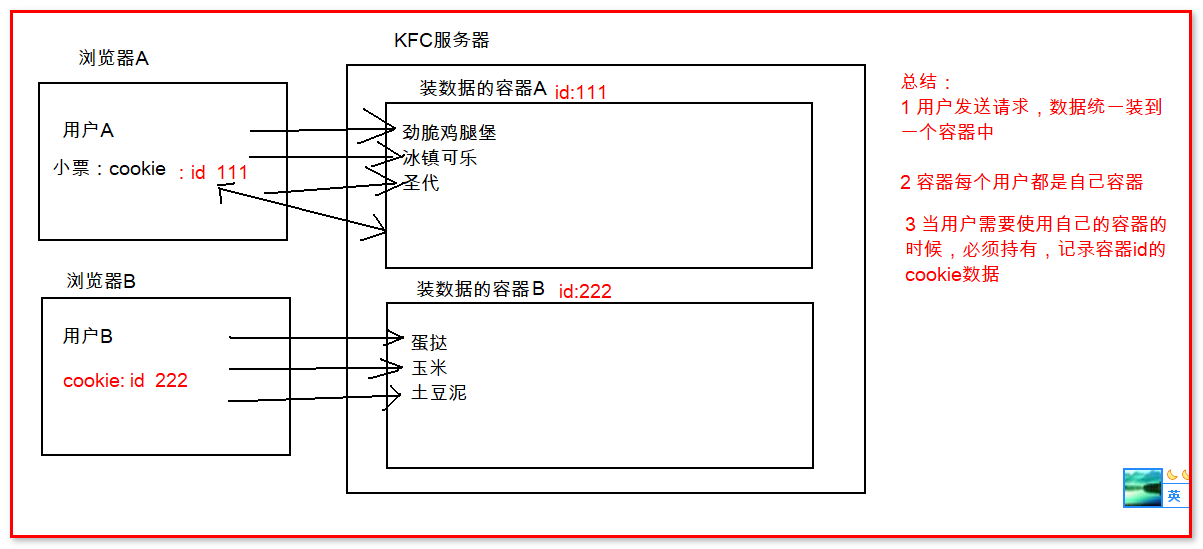
2 这个容器必然存在服务器

3 每个用户，需要分配一个容器

那么到底这个容器，在功能实现过程中，是怎么运行的？

这个技术，就是session：存储用户多次请求过程中的数据

Session技术原理图： 跟cookie有区别——cookie存数据（用户名称，用户密码：身份验证数据） session 存数据（普通业务内容），



总结：

1 session是服务器开辟的一个用来存储数据的空间

2 服务器为每个浏览器单独开辟一个session

3 服务器根据浏览器发送过来的cookie，来确认当前浏览器使用哪个session

补充说明：

* 在WEB开发中，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)。因此，在需要保存用户数据时，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。
* Session和Cookie的主要区别在于：

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（服务器端）。

* 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。

## Session的快速入门

### 获取Session对象

学习第一步：获取session对象

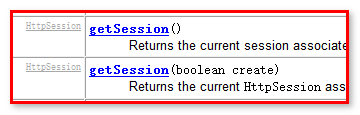


人话：提供了一个容器，将多次请求中的数据，可以都存储在这个容器中。

接下来，按照面向对象的思路，查询构造函数，创建对象。但是查阅API发现，没有构造函数使用——注意：httpsession对象，有服务器自己创建，不需要程序员手动操作！！！

需要通过一种方式，获取httpSession对象，通过一个对象，他的方法，获取httpSession

通过request对象来获取session对象： 为什么必须通过request来获取session

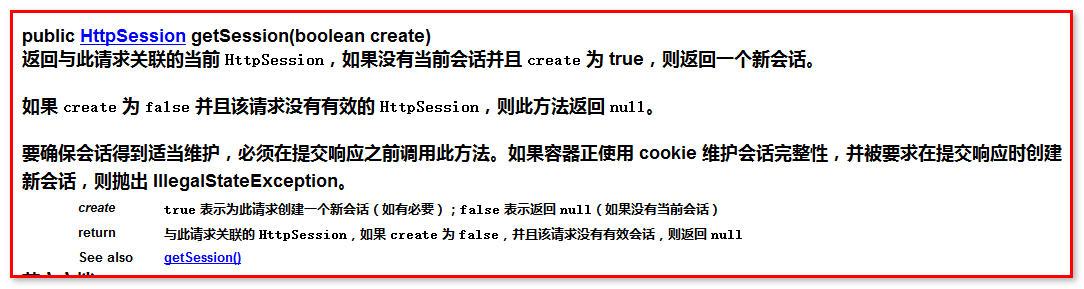


第一个空参数：

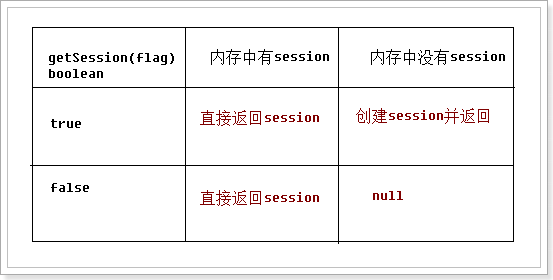


人话：获取当前服务器中给浏览器开辟的Session容器，如果以前没有，创建一个新的给你，如果以前有了，将以前的Session给你

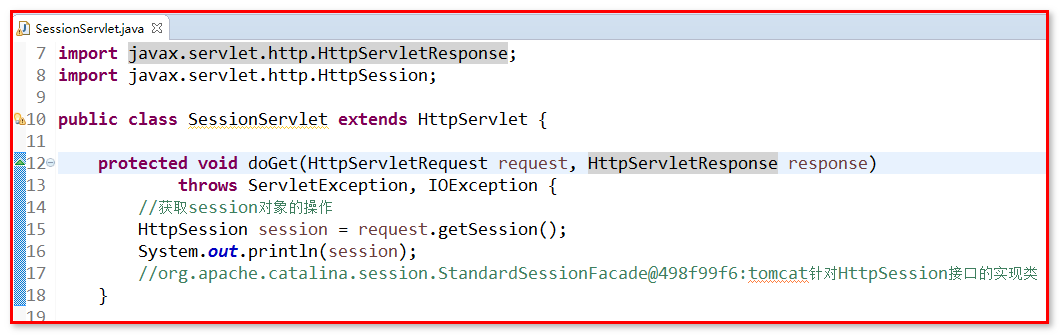
第二个带参数：



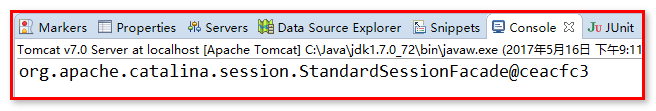
人话：如果这个函数参数为true和前面的getSession(),效果一致，如果参数为false，并且，以前没有Session对象，那么返回null。



代码演示：使用API



效果：



补充说明：

Session是基于用户的请求，而把用户的重要信息在服务器端针对这个用户（浏览器）创建了一个容器。

而这个Session容器是由web服务器(tomcat)帮助我们创建的，在程序中我们只能去获取到这个容器，然后给容器添加数据或者取出数据，或者删除数据，而我们是不能创建这个容器对象。

### 在Session中存取删数据（重点）

查询HttpSession对象中存数据，取数据，删数据的API

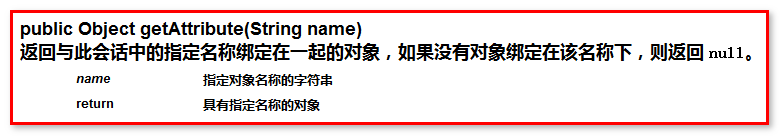
存数据：





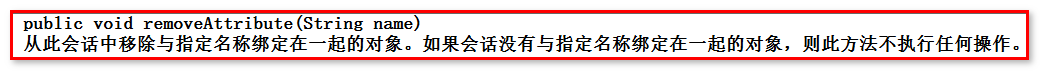
人话：指定一个对象保存到会话中，name参数，数据的名称，value参数：具体的值

取数据：



人话：获取指定名称的数据，如果没有，返回null

删数据：



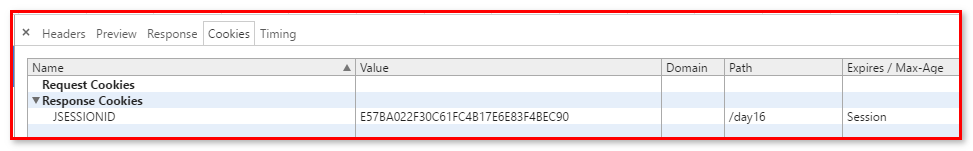
人话：删除Session容器中的数据，删除根据指定的名称（name参数），如果没有数据了，不做任何处理。

演示存取删操作：

|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** javax.servlet.http.HttpSession;  **public** **class** SessionServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  //演示获取session对象（容器，用来保存数据）  HttpSession session = request.getSession();  //org.apache.catalina.session.StandardSessionFacade@4838bd9d : 这个是httpsession接口的实现类，这个类由tomcat实现  System.***out***.println(session);  //设置数据的方法  /\*session.setAttribute("addr", "法国");  //获取数据的方法  String addr = (String)session.getAttribute("addr");  System.out.println(addr);  //删除数据的方法  session.removeAttribute("addr");  String addr1 = (String)session.getAttribute("addr");  System.out.println(addr1);\*/  }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

### 记录sessionid的cookie

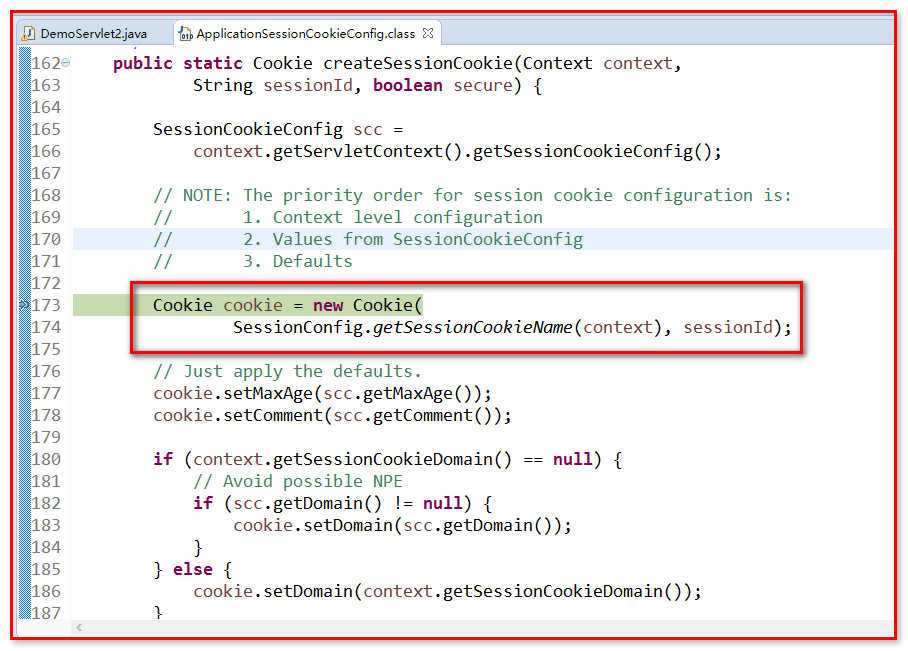
通过谷歌浏览器的工具查看，SessionId的cookie：



这个cookie明显是服务器创建的，那么是在哪里创建的呢？

测试前先清空cookie数据！

创建记录sessionid的cookie由tomcat中：ApplicationSessionCookieConfig



## 测试：关闭浏览器之后，session对象，还是同一个吗？

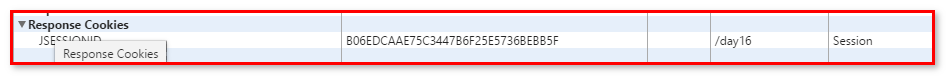
第一次访问：

org.apache.catalina.session.StandardSessionFacade@5e7eb260



关闭浏览器，再次访问:

org.apache.catalina.session.StandardSessionFacade@4107e577



问题：为什么，关闭浏览器之后，session对象就换了一个？

结论：关闭浏览器之后，重新访问项目，被分配一个新的session对象，原因——用来寻找session对象的cookie已经不存在了，随着浏览器关闭消失了。

需求：关闭浏览器之后，还要之前的session，如何实现呢？

解决方案：自己创建一个cookie，要求被浏览器持久化保存起来（setMaxAge(10000)）放便后期使用

代码演示自己手动持久化cookie：

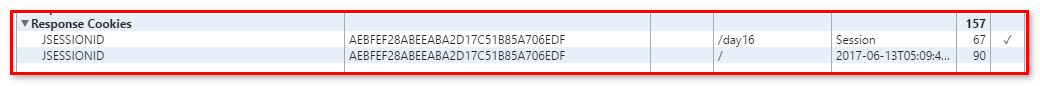
要使用的API:



人话：获取session的id。

|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.Cookie;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** javax.servlet.http.HttpSession;  **public** **class** SessionServlet2 **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  HttpSession session = request.getSession();  System.***out***.println(session);  //自己创建一个cookie，要求被浏览器持久化保存起来（setMaxAge(10000)）放便后期使用  Cookie cookie = **new** Cookie("JSESSIONID", session.getId());  //活的久一点  //cookie.setMaxAge(10000);  cookie.setPath("/");  response.addCookie(cookie);  }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

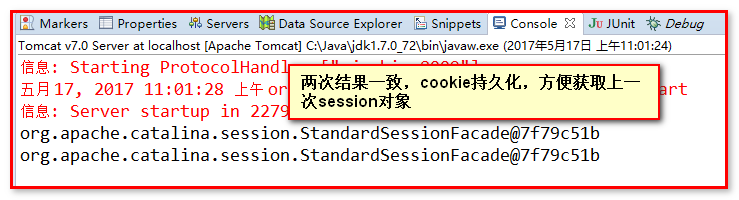
效果：



org.apache.catalina.session.StandardSessionFacade@9fd589a

org.apache.catalina.session.StandardSessionFacade@9fd589a

效果二：



总结：session容器的获取全部依赖于cookie，服务器自动解析cookie，根据cookie中jsessionid，获取指定的容器对象。

一般情况下，关闭浏览器之后，再次访问，是无法获取到Session中的数据的。

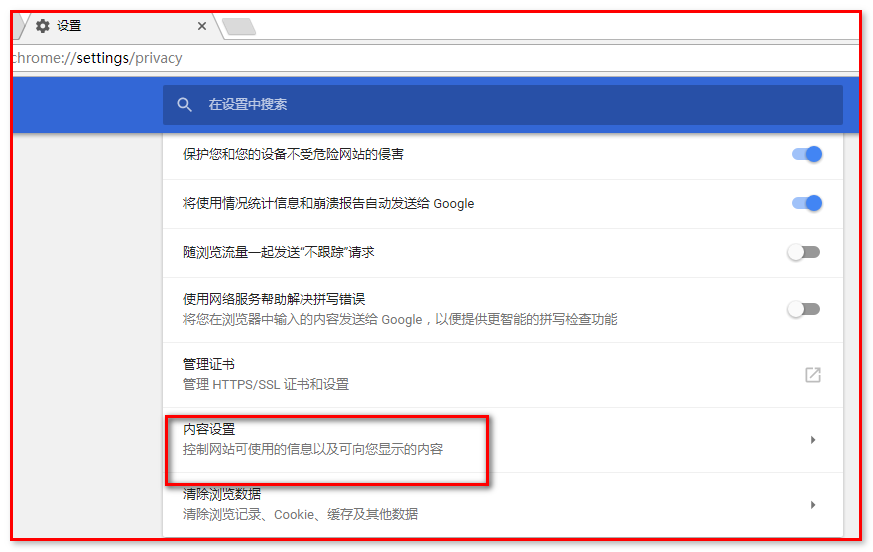
因此在服务器针对当前用户的第一次请求创建的唯一的Session容器对象，而在给这次请求的之后，服务器需要给用户响应数据，在响应的时候，服务器把当前Session容器对象的JSESSIONID以Cookie的形式发送给了浏览器。而这个Cookie并没有设置有效时间，那么这个Cookie就属于临时Cookie，在关闭浏览器之后，再次打开浏览器的时候，上次的Cookie已经消失了，用户虽然拿同一个浏览器访问服务器，可是这时没有JSESSIONID，服务器端的Session容器依然存在，但是没有JSESSIONID，服务器内部无法获取到这个session对象

把包含了JSESSIONID的Cookie在客户端持久化。

## 禁用Cookie后Session追踪

如何禁用cookie演示：演示谷歌浏览器禁用cookie







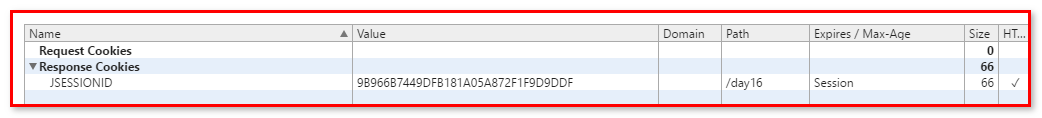


点击完成，设置OK。

测试，禁用cookie之后的效果：

刚才操作的都是浏览器，没有操作服务器，所以，

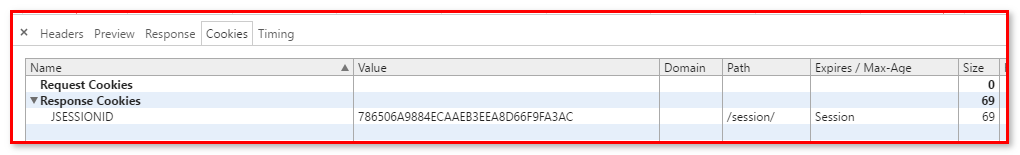
服务器依然会将cookie传递到浏览器：



效果：本地浏览器并不会将cookie保存下来



没cookie了，那么我们的session怎么办？



每次都在重新创建session！！！！！



现状：每次请求都创建新的session对象，不能完成多次请求数据集中到一个session中

解决方案：

一定要让我们每次请求，可以获取到同一个session！！！！！

以前cookie自动帮助我们，发送数据给服务器，现在，自己自力更生，手动在链接地址上拼接请求参数（jsessionid=XXXX）

测试以上方案：

|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** javax.servlet.http.HttpSession;  **public** **class** SessionServlet3 **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  //第一次请求的时候，创建session对象  HttpSession session = request.getSession();  System.***out***.println(session);    //为了后期，还能使用同一个session，设置一个链接，在链接中，拼接请求参数（jsessionid=XXXX）    response.setContentType("text/html;charset=utf-8");  //拼接参数，使用的是分号，不是问号。  //使用API，简化拼接sessionid的操作  String encodeURL = response.encodeURL("/day16/SessionServlet3");  String encodeRedirectURL = response.encodeRedirectURL("/day16/SessionServlet3");  System.***out***.println("encodeURL==="+encodeURL);  System.***out***.println("encodeRedirectURL==="+encodeRedirectURL);  //String a = "<a href=\"/day16/SessionServlet3;jsessionid="+session.getId()+"\">点击</a>";  String a = "<a href='"+encodeURL+"'>点击</a>";  response.getWriter().write(a);  }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

效果：



这个操作比较繁琐，有没有简便的方法？

有response对象有更好的方法：



人话：将路径添加jsessionid作为请求参数，如果，cookie数据没有禁用，那么，不做任何操作。

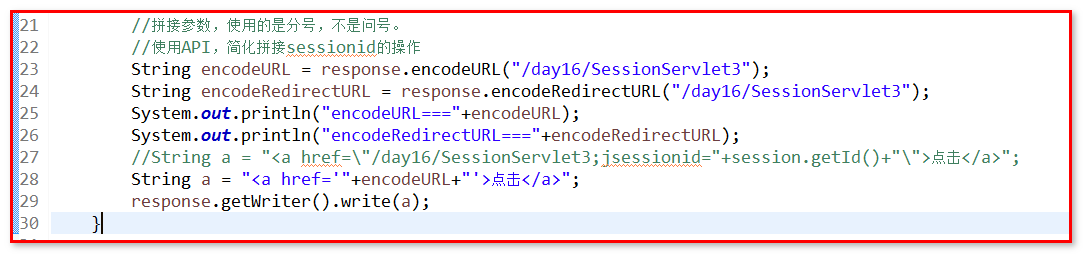
用于对表单action和超链接的url地址进行重写

在response对象中的提供的encodeURL方法它只能对页面上的超链接或者是form表单中的action中的路径进行重写（拼接JSESSIONID）。

如果我们使用的重定向技术，这时必须使用下面方法完成：



用于对sendRedirect方法后的url地址进行重写。



附：这两个方法是效果是一致的，在设置要转换的路径为空字符串的时候，encodeRedirectURL方法没有任何效果，encodeURL会继续拼接参数

总结：其实就是在路径后面拼接了 Session的唯一标识 JSESSIONID。

## Session的生命周期（面试）

使用session存取数据，必须在session对象存活的时候，才可以使用，因此，学习session生命周期（什么时候生，什么时候死）

Session对象的创建时间：

当第一次调用request.getSession()的时候创建session容器.

如果第一次访问jsp页面,也会创建session容器

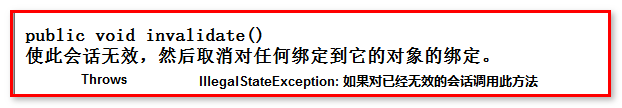
Session的销毁时间：

1. （自动去死）Session在服务器的存活时间。Session对象在服务器内部有默认的存活的时间。一般默认是30分钟。如果在30分钟内，用户没有再对当前这个服务器中的资源进行任何的访问操作，这时只要时间到达30分钟，服务器会自动的销毁这个session。

Session的存活时间我们可以在当前这个项目的web.xml中配置：

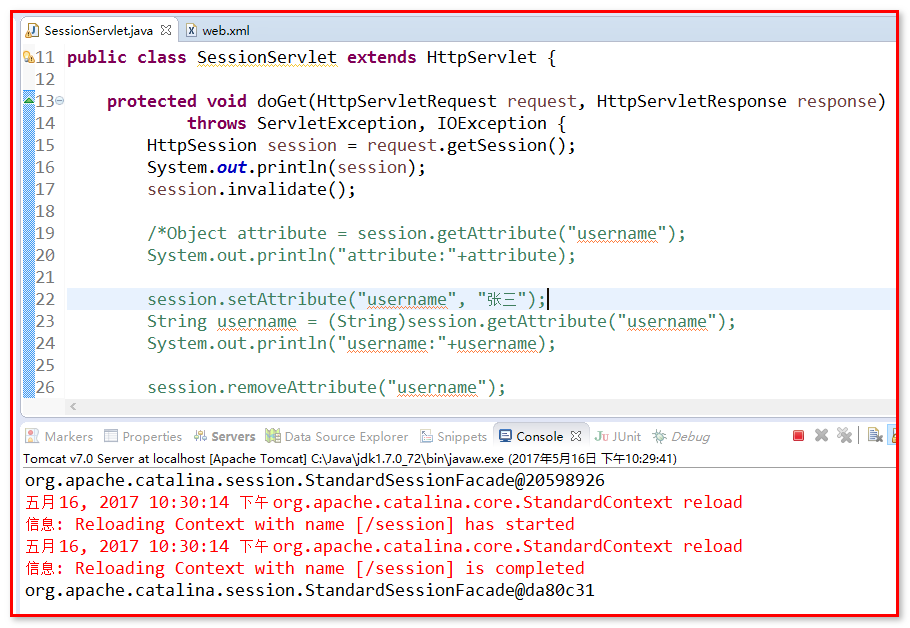


1. 在程序执行中,手动销毁session容器, 使用 session.invalidate()（重点）



人话：销毁session对象，只是让这个对象无效，下次在来访问，给一个新的session对象。

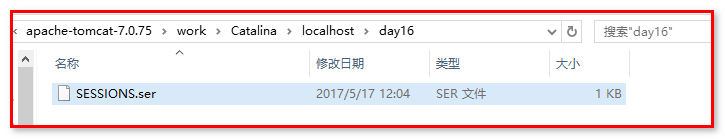
两次调用同一个servlet测试，有销毁方法，获取到不同的session对象



1. 不正常关闭服务器。直接点击控制台的红点，强制关闭tomcat

如果是正常关闭服务器，这时服务器内部会使用IO流中的序列化技术把这个Session对象保存在tomcat/work目录下面。

Session文件：

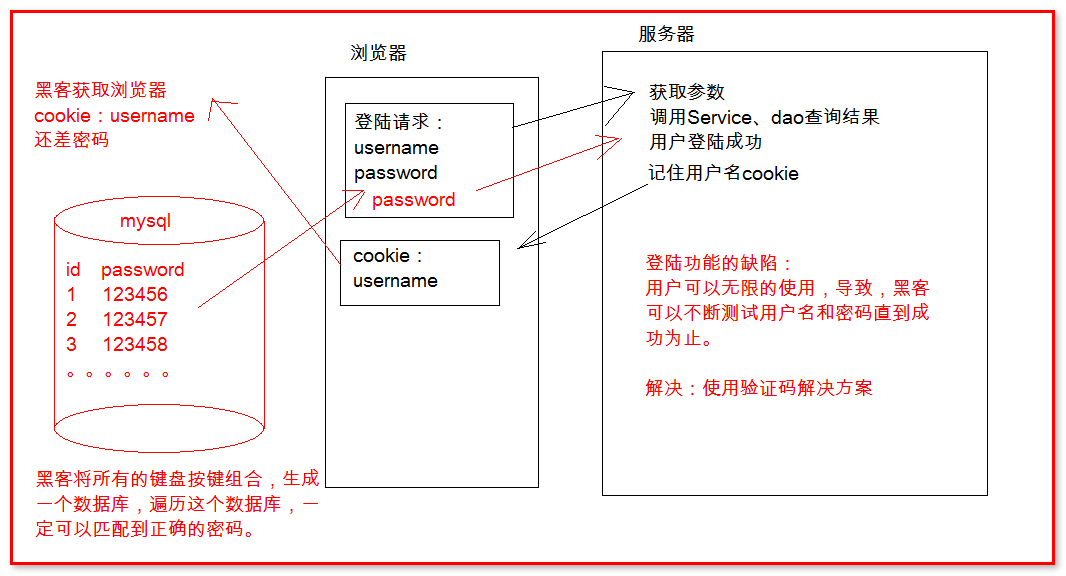


### Session应用：登陆验证码

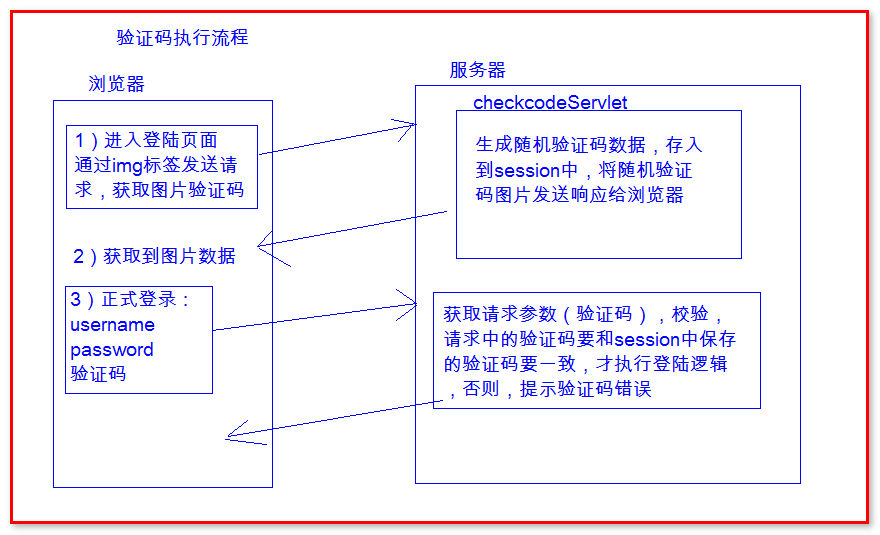
什么是登陆验证吗？

防止黑客暴力破解用户的密码的一种技术。

什么是暴力破解用户的密码？



黑客这样的暴力破解，有什么办法解决呢？验证码

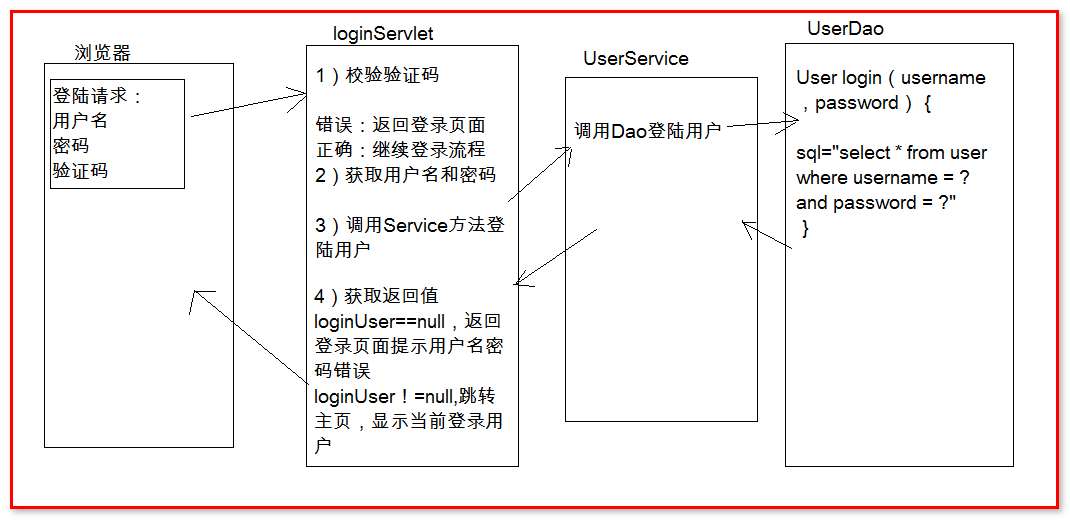


需求：在登录的时候，加上验证码校验

页面准备：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>login</title>  <script type=*"text/javascript"*>  **function** changeCode(){  //设置src属性的地址如果和前一次相同，默认从浏览器缓存的图片中获取数据，不再发请求到浏览器  //第一：地址不能变  //第二：必须让浏览器，察觉地址的一些内容，产生了变化  //第三：地址最后的参数，最好，每次都发生变化  //Date();获取当前的时间  //var date = new Date();  //获取当前时间的毫秒值  //var time = date.getMilliseconds()  document.getElementById("img").src = "/session/checkcode?r="+**new** Date().getTime();  /\*\*  为什么要使用验证码？    登陆功能——username password ——》登陆    用户名密码暴力破解：黑客有一个密码库  （一个数据库，存储了普通用户一般会使用的常见密码组合，甚至，将键盘上所有的字符组合通过遍历，一个一个存储到数据库）    当黑客知道你的用户名之后，（cookie，可能被黑客利用，获取用户名）  使用密码库中的密码，和用户的用户名组合，开始破解，用户的登陆    解决方案：限制黑客暴力破解密码，使用一个随机生成的验证码，用户登陆，必须，手动输入，验证码的内容，才允许执行登陆      \*/  }  </script>  </head>  <body>  <form action=*"/session/login"* 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=*"code"*></td></tr>  <!-- 通过向服务器发送请求，从服务器获取验证码数据 -->  <tr><td></td><td><img id=*"img"* src=*"/session/checkcode"* onclick="changeCode();"/><a href=*"javascript:;"* onclick="changeCode();">换一换</a></td></tr>  <tr><td></td><td><input type=*"submit"* value=*"登陆"*></td></tr>  </table>  </form>  </body>  </html> |

登陆流程分析：



**LoginServlet**

|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** cn.igeek.domain.User;  **import** cn.igeek.service.UserService;  **import** cn.igeek.service.impl.UserServiceImpl;  **public** **class** LoginServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  //用户请求中的验证码获取  String yanzhengma = request.getParameter("yanzhengma");    //与session中保存的验证码进行校验  String code\_session = (String)request.getSession().getAttribute("code\_session");  **if**(!code\_session.equalsIgnoreCase(yanzhengma)){  //验证码错误，告诉用户，页面提示  request.setAttribute("msg", "验证码错误");  request.getRequestDispatcher("/login.jsp").forward(request, response);  **return**;  }  //验证码正确，登录逻辑执行  //获取用户名和密码  String username = request.getParameter("username");  String password = request.getParameter("password");  //调用Service方法，登录用户  UserService userService = **new** UserServiceImpl();  //ctrl+shift+O自动解决一些常见问题，如果解决不了，留着程序员解决  User loginUser = userService.login(username,password);  **if**(loginUser == **null**){  request.setAttribute("msg", "用户名或则密码错误");  request.getRequestDispatcher("/login.jsp").forward(request, response);  **return**;  }**else**{  //登陆成功，跳转主页  response.sendRedirect(request.getContextPath());  //不写return，而且后面，还要可以执行的代码，有可能发送一个错误：response对象已经提交了，无法再次提交  **return**;  }  }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

**UserService**

|  |
| --- |
| **package** cn.igeek.service;  **import** cn.igeek.domain.User;  **public** **interface** UserService {  /\*\*  \* 文档注释 ：ctrl + shift + j  \* 用户登录的方法  \* **@param** username  \* **@param** password  \* **@return**  \*/  User login(String username, String password);  } |

UserServiceImpl

|  |
| --- |
| **package** cn.igeek.service.impl;  **import** cn.igeek.dao.UserDao;  **import** cn.igeek.dao.impl.UserDaoImpl;  **import** cn.igeek.domain.User;  **import** cn.igeek.service.UserService;  **public** **class** UserServiceImpl **implements** UserService {  @Override  **public** User login(String username, String password) {  UserDao userDao = **new** UserDaoImpl();  **return** userDao.login(username,password);  }  } |

**UserDao**

|  |
| --- |
| **package** cn.igeek.dao;  **import** cn.igeek.domain.User;  **public** **interface** UserDao {  /\*\*  \* 查询用户名和密码是否匹配的方法  \* **@param** username  \* **@param** password  \* **@return**  \*/  User login(String username, String password);  } |

UserDaoImpl

|  |
| --- |
| **package** cn.igeek.dao.impl;  **import** java.sql.Connection;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** cn.igeek.dao.UserDao;  **import** cn.igeek.domain.User;  **import** cn.igeek.utils.JDBCUtils;  **public** **class** UserDaoImpl **implements** UserDao {  @Override  **public** User login(String username, String password) {  //定义三个对象重复  //sql语句和设置参数，是变化的  //  Connection conn = **null**;  PreparedStatement stmt = **null**;  ResultSet rs = **null**;    **try** {  conn = JDBCUtils.*getConnection*();  String sql = "select \* from user where username = ? and password = ?";  stmt = conn.prepareStatement(sql);  stmt.setString(1, username);  stmt.setString(2, password);  rs = stmt.executeQuery();  //封装数据到对象，重复出现  **if**(rs.next()){  **return** **new** User();  }**else**{  **return** **null**;  }  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException("用户登陆失败");  }**finally** {  //释放资源，重复代码  JDBCUtils.*release*(conn, stmt, rs);  }    }  } |

## JavaWEB中三种数据范围（三种容器）

Request ServletContext Session

三个对象都有：setAttribute getAttribute方法，都可以存取数据

回答面试官问题套路：先回答基本概念，返回，跟上使用案例。

问题：什么时候使用request对象，保存数据？

答：一次请求中需要使用的数据，就保存在request。举例：商品数据，就存入请求中。

问题：什么时候使用ServletContext 对象，保存数据？

答：全局使用的数据，整个项目需要使用的数据，就是，要存入ServletContext 。举例：在线人数，存入ServletContext 。

问题：什么时候使用session对象，保存数据？

答：一次会话（多次请求——都发生在一次开启浏览器和关闭浏览器过程中）中需要使用的数据，都存入session。

举例：点餐商品