**会话对象**

Session and Cookie

拓薪教育

刘洋

1. Cookie概述(熟练)
   1. 什么叫Cookie

Cookie翻译成中文是小甜点，小饼干的意思。在HTTP中它表示服务器送给客户端浏览器的小甜点。其实Cookie就是一个键和一个值构成的，随着服务器端的响应发送给客户端浏览器。然后客户端浏览器会把Cookie保存起来，当下一次再访问服务器时把Cookie再发送给服务器。

* 1. Cookie规范

你大可以放心，Cookie不会占满你的硬盘。因为一个Cookie最多只有4KB，并且浏览器最多可以保存300个Cookie。当然，在浏览器大战的今天，一些浏览器为了打败对手，可能对Cookie规范“扩展”了一些，例如每个Cookie的大小为8KB，最多可保存500个Cookie等！但也不会出现把你硬盘占满的可能！

不同的浏览器之间不能共享Cookie！！！

* 1. Cookie的作用

Cookie的作用可大了，但无论怎么夸大Cookie的作用都离不开“**跟踪客户端状态**”这句话。我们知道Cookie是服务器保存在客户端的信息，然后客户端会在下次请求时把Cookie在还给服务器，这样服务器就可以通过信息来识别客户端了。

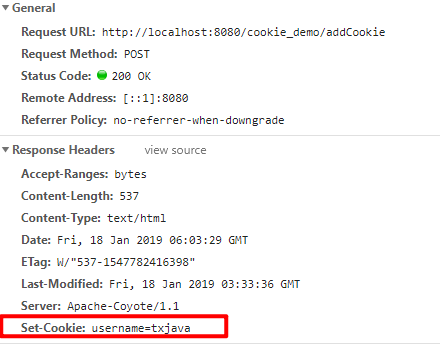
1. Cookie示例(精通)
   1. 保存Cookie到客户端

这是响应工作的一部分，所以这个方法是response对象的。并且Cookie是HTTP协议中的内容，所以保存Cookie是HttpServletResponse类的方法。

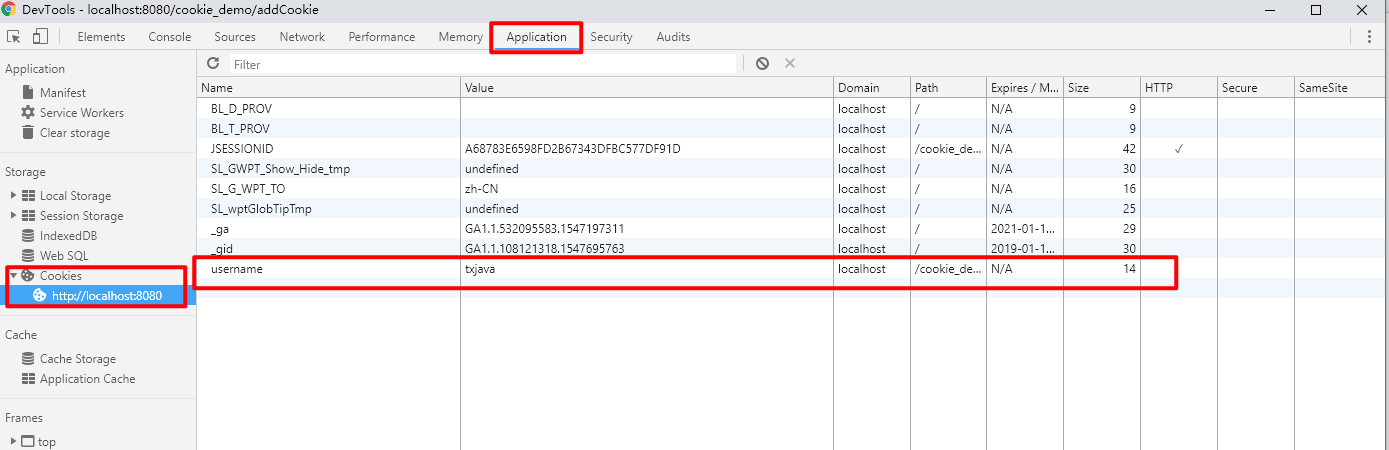
* void addCookie(Cookie c)：添加Cookie对象到当前response对象中，这个方法可以被调用多次，从而完成添加多个Cookie对象到response中。

|  |
| --- |
| ***Cookie cookie* = *new* Cookie("username", "txjava");** response**.addCookie(*cookie*);** |

通过浏览器的开发者工具我们可以看到,该请求的响应头中被添加了Set-Cookie的值:



同样根据浏览器的开发者工具我们可以看到<http://localhost:8080>的网址的cookie中被加入了username=txjava的信息:



这时我们再对服务器进行访问就会发现,访问服务器时请求时带着cookie的值的:



* 1. 服务器端读取Cookie

我们现在已经可以保存Cookie到客户端了，但还没有学习让服务器如何读取Cookie。

如果浏览器保存了Cookie，那么会在下一次请求时把Cookie放到请求头中发送给服务器，这时服务器需要在请求中读取Cookie。既然是在请求中读取，那么当然是使用request对象来读取了。

HttpServletRequest：Cookie[] getCookies()

示例:

|  |
| --- |
| ***Cookie*[] *cookies* =** request**.getCookies(); *if*(*cookies* != *null*){  *for*(*Cookie cookie* : *cookies*){  *System*.*out*.println(*cookie*.getName() +":"+ *cookie*.getValue());  } }** |

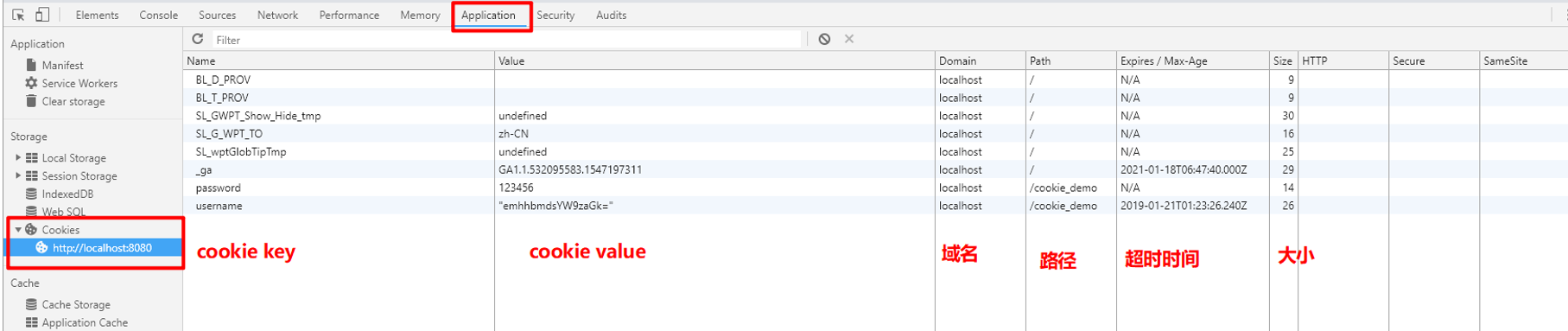
1. Cookie生命周期(熟练)

Cookie会在客户端存活多久呢？这就是Cookie的生命了。默认情况下，Cookie只在浏览器的**内存中**存活，也就是说，当你**关闭浏览器后，Cookie就会消失**！

可以使用Cookie#setMaxAge(int expiry)来设置Cookie的存活时间。参数expiry表示Cookie存活的秒数。

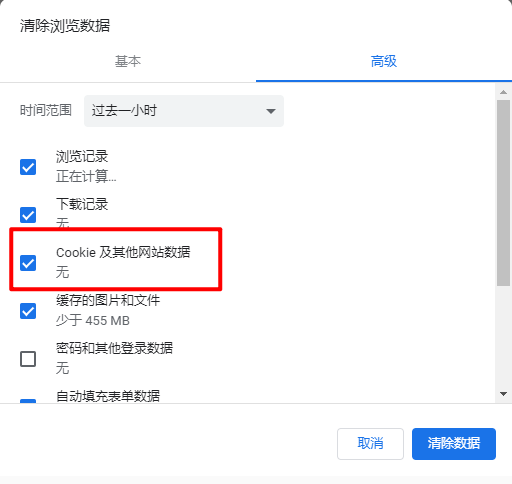
* cookie.setMaxAge(60\*60)：表示cookie对象可存活1小时。就算关闭浏览器，就算重启客户端电脑，cookie也会存活1小时。因为当maxAge大于0时，浏览器不仅会把cookie保存在浏览器内存中，还会把cookie保存到硬盘上。
* cookie.setMaxAge(-1)：cookie的maxAge属性的默认值就是-1（其实只要是负数都是一个意思），表示只在浏览器内存中存活。一旦关闭浏览器窗口，那么cookie就会消失。
* cookie.setMaxAge(0)：cookie被作废！表示cookie即不在内存中存活，也不在硬盘上存活，这样的cookie设置只有一个目的，那就是覆盖客户端原来的这个cookie，使其作废。

1. Cookie的浏览器管理(熟练)
   1. Google Chrom
      1. 查看cookie

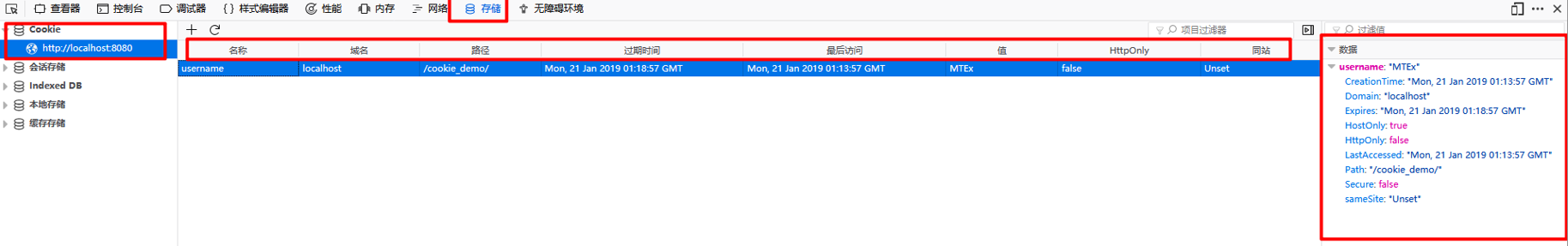


* + 1. 清理cookie

设置-隐私设置和安全性-清除浏览数据:



* 1. Firefox
     1. 查看Cookie



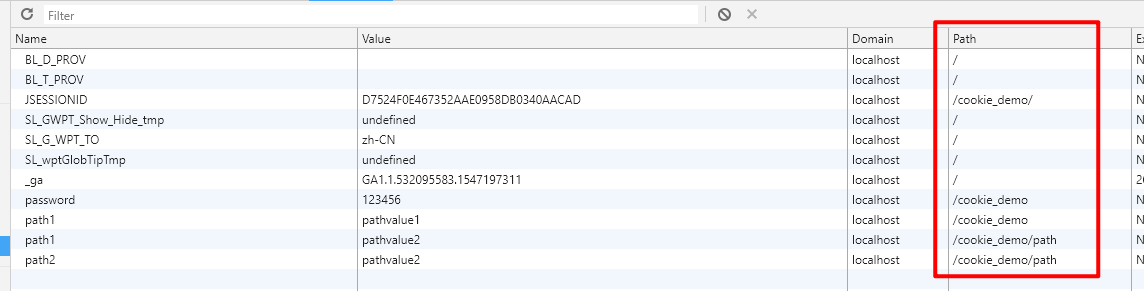
* + 1. 清理Cookie

选项 - 隐私与安全 - Cookie和网站数据 - 清除数据



1. Cookie的路径(熟练)
   1. Cookie的path属性

Cookie还有一个path属性，可以通过Cookie#setPath(String)方法来设置。你可以使用HttpWatch查看响应中的Set-Cookie中是否存在路径。下面是通过Chrome查看Cookie信息。



也就是说，就算你不设置Cookie的path，Cookie也是有路径的。这个路径就是请求的路径。例如在请求:

<http://localhost:8080/cookie_demo/path>

时，服务器响应了一个Cookie，那么这个Cookie的默认路径就是/cookie\_demo。

如果是在请求:

<http://localhost:8080/cookie_demo/path/son>

时，服务器响应了一个Cookie，那么这个Cookie的默认路径就是/cookie\_demo/path。

* 1. Cookie 路径的作用:

到现在我们还没说过Cookie的path有什么用，现在我们来聊聊path的作用。首先声明一点，path不是指Cookie在客户端存放的路径！！！不同的浏览器存放Cookie的路径是不同的！！！你不能通过Cookie的path来指定Cookie文件的存放路径！！！

那么Cookie的path是干什么的呢？假设你的浏览器当前已经有了两个Cookie：

* cookie1：name=path1;value=pathvalue1;path=/cookie\_demo；
* cookie2：name=path1;value=pathvalue2;path=/cookie\_demo/path；

当访问[http://localhost:8080/cookie\_demo/](http://localhost:8080/cookie_demo/path)\*时，请求头中会包含cookie1，而不会包含cookie2。

当访问[http://localhost:8080/cookie\_demo/path/](http://localhost:8080/cookie_demo/path/son)\*时，请求头中会包含cookie1和cookie2。

**也就是说，在访问子路径时，会包含其父路径的Cookie，而在访问父路径时，不包含子路径的Cookie。**

* 1. Cooki**e路径**的使用

如果你想在BServlet中设置的Cookie，在客户端访问AServlet时也包含在请求头中，那么就需要设置BServlet中的Cookie的path：

* c2.setPath(“/cookie\_demo”)：硬编码；
* c2.setPath(request.getContextpath() + “/”)：活编码。

这样就可以设置Cookie的路径，保存在访问AServlet时，也会包含BServlet中添加的Cookie。

Cookie的SetPath设置cookie的路径，这个路径直接决定服务器的请求是否会从浏览器中加载某些cookie。

首先默认情况如果不设置cookie的path，默认是 /项目名称/当前路径的上一层地址如：请求路径：/cookie\_demo/servlet/login， cookie的路径：/cookie\_demo/servlet

如果我们设置path，如果当前访问的路径包含了cookie的路径（当前访问路径在cookie路径基础上要比cookie的范围小）cookie就会加载到request对象之中。

1. Cookie保存中文(熟练)

Cookie中保存中文，次要。

Cookie中是不可以设置中文的，但可以使用URLEncodor.encode()方法编码后在存放到Cookie中。在获取Cookie时，需要先使用URLDecoder.decode()方法解码，再使用。

添加cookie:

|  |
| --- |
| ***Cookie cookie1* = *new* Cookie("username", *URLEncoder*.*encode*(*username*,"UTF-8"));** |

读取cookie:

|  |
| --- |
| ***Cookie*[] *cookies* =** request**.getCookies(); *if*(*cookies* != *null*){  *for*(*Cookie cookie* : *cookies*){  *if*(*cookie*.getName().equals("username"))  *username* = *URLDecoder*.*decode*(*cookie*.getValue(),"UTF-8");  *if*(*cookie*.getName().equals("password"))  *password* = *cookie*.getValue();  } }** |

1. Cookie的禁用(熟练)

默认情况下浏览器的cookie是被启用的，但是其实我们是可以手动的禁用cookie的，强烈不建议禁用cookie;

Cookie一旦被禁用掉绝大多数互联网的网站都无法登录，这个跟我们后续要讲解的session有关。

那么我们如何通过程序来通过程序判断用户的浏览器上的cookie是否被禁用了呢，其实很简单，我们可以通过去刚刚添加的cookie，如果没有取到，说明cookie被禁用。

1. HttpSession概述(熟练)

* 会话跟踪之session

session也是域对象之一，它的范围是在一个会话范围内有效。session既然是域对象，那么当然就要有getAttribute()和setAttribute()系列方法了。

在一个会话内共享一个session对象，所以session中可以保存一个会话内的数据。例如当前用户的信息。

session的范围大于request，可以在一个会话中多个请求之间共享数据。但session的范围小于ServletContext(application)，session不能在多个用户之间共享数据。

目前所学过的域对象的作用范围：

ServletContext > HttpSession > HttpServletRequest

* 获取session对象

使用request.getSession()方法就可以获取session对象。

有了session，就不用使用Cookie来跟踪会话了！但是session不能像Cookie那样长命，一旦用户关闭浏览器窗口，那么session就死掉了。

1. HttpSession原理（依赖Cookie）(精通)



我们都知道HTTP是无状态协议，但是为什么session可以跟踪会话状态呢？没错，session依赖Cookie。

当客户端第一次访问服务器时，服务器会为客户端创建一个session对象，然后把session对象放到session池中，在响应时把sessionId通过Cookie响应给客户端。注意，只有在第一次访问时，服务器才会创建session，给客户端响应sessionId。从此以后就不会了！

当客户端再次访问服务器时，会在请求中带着sessionId给服务器，服务器通过sessionId到session池中找到session对象，这就可以完成会话跟踪了。也就是说，服务器端保存的是session对象，而客户端只有sessionId。每次访问都需要通过客户端的sessionId来匹配服务器端的session对象！这样用户在session中保存的数据就可以再次被使用了。

sessionId是服务器通过Cookie发送给客户端浏览器的，这个Cookie的maxAge为-1，即只在浏览器内存中存在。如果你关闭所有浏览器窗口，那么这个Cookie就会消失了！

1. HttpSession失效(精通)

session失效有如下几个原因：

* session.invalidate()方法注销session
* session超时

|  |
| --- |
| **<*session-config*>** <!-- session的超时时间，以分钟为单位 -->  **<*session-timeout*>**1**</*session-timeout*> </*session-config*>** |

* Cookie被禁用

1. 面试题
   1. Session和Cookie的区别(精通)

* session 在服务器端，cookie 在客户端（浏览器）
* session 默认被存在在服务器的一个文件里（不是内存）
* session 的运行依赖 session id，而 session id 是存在 cookie 中的，也就是说，如果浏览器禁用了 cookie ，同时 session 也会失效（但是可以通过其它方式实现，比如在 url 中传递 session\_id）
* session 可以放在 文件、数据库、或内存中都可以。
* 用户验证这种场合一般会用 session 因此，维持一个会话的核心就是客户端的唯一标识，即 session id

1. 综合练习(精通)

使用cookie实现自动登陆

html代码:

|  |
| --- |
| **<!DOCTYPE html> <*html* lang="en"> <*head*>  <*meta* charset="UTF-8">  <*title*>**Title**</*title*> </*head*> <*body*>  <*form* action="login" method="post">** 用户:**<*input* type="text" name="username"/><*br*>** 密码:**<*input* type="password" name="password"/><*br*>  <*input* type="radio" value="1" name="auto"/>*&nbsp;***一天内自动登陆**<*br*>  <*input* type="submit" value="登陆"/> </*form*>  </*body*> </*html*>** |

Java代码:

|  |
| --- |
| ***package cn.tx.servlet*;  *import cn.tx.model.User*; *import sun.misc.BASE64Decoder*; *import sun.misc.BASE64Encoder*;  *import javax.servlet.ServletException*; *import javax.servlet.annotation.*WebServlet; *import javax.servlet.http.*\*; *import java.io.IOException*;** /\*\*  \* 模拟自动登陆方法  \*/ **@WebServlet(**name **= "LoginServlet",**urlPatterns **= "/login") *public class LoginServlet extends HttpServlet* {  *protected void* doPost(**HttpServletRequest request**,** HttpServletResponse response**) *throws ServletException*, *IOException* {** // 获取用户端提交的表单信息  ***String username* =** request**.getParameter("username");  *String password* =** request**.getParameter("password");  *String auto* =** request**.getParameter("auto");** // 统一设置response响应格式及编码  response**.setContentType("text/html;charset=utf-8");** response**.setCharacterEncoding("UTF-8");** // 判断是否提交用户信息  ***if*(*username* == *null* && *password* == *null* && *auto* == *null*){** // 用来接收cookie的value信息  ***String cookieValue* = *null*;** // 获取用户端cookies  ***Cookie*[] *cookies* =** request**.getCookies();** // 如果cookies不为null 尝试获取name为txjavac的cookie的value  ***if*(*cookies* != *null*){  *for*(*Cookie ck* : *cookies*){  *if*(*ck*.getName().equals("txjavac")){  *cookieValue* = *ck*.getValue();  }  }  }** // 如果没有获取到cookie的value 返回信息  ***if*(*cookieValue* == *null*){** response**.getWriter().write("您还未进行登陆,请进行登陆!!!");  *return*;  }*else*{**// 如果获取到  // 对该信息进行解码  ***BASE64Decoder decoder* = *new* BASE64Decoder();  *cookieValue* = *new* String(*decoder*.decodeBuffer(*cookieValue*));** // 对解码后的字符串进行切分  ***String*[] *split* = *cookieValue*.split(":");** // 获取用户名和密码  ***username* = *split*[*1*];  *password* = *split*[*2*];** // 创建用户对象  ***User user* = *new* User(*username*,*password*);** // 获取session  HttpSession ***session* =** request**.getSession();** // 把用户对象存储到session中  ***session*.setAttribute("user",*user*);** // 返回内容  response**.getWriter().write("尊敬的"+*username*+",欢迎您来到拓薪教育!!!");  *return*;  }  }** // 判断用户名或密码是否正确  ***if*(*username*!=*null* && *password*!=*null* && *username*.equals("admin") && *password*.equals("txjava")){** // 创建用户对象  ***User user* = *new* User(*username*,*password*);** // 获取session会话对象  HttpSession ***session* =** request**.getSession();** // 把用户存入session中  ***session*.setAttribute("user",*user*);** // 用户名密码正确的话,判断是否勾选了一天内自动登陆  ***if*(*auto* != *null* && *auto*.equals("1")){** // 拼接存储于cookie的value值  ***String value* = "txjava:" + *username* + ":" + *password* + ":" + *24*\**3600*;** // 进行BASE64编码  ***BASE64Encoder encoder* = *new* BASE64Encoder();  *value* = *encoder*.encode(*value*.getBytes());** // 创建cookie并且设置一天失效后添加到response  ***Cookie cookie* = *new* Cookie("txjavac",*value*);  *cookie*.setMaxAge(*24*\**3600*);** response**.addCookie(*cookie*);  }** response**.getWriter().write("尊敬的"+*username*+",欢迎您来到拓薪教育!!!");  *return*;  }*else* {** response**.getWriter().write("用户名或密码错误!!!");  *return*;  }   }   *protected void* doGet(**HttpServletRequest request**,** HttpServletResponse response**) *throws ServletException*, *IOException* {  doPost(**request**,**response**);  } }** |

1. 学习目标

* 熟练掌握Cookie的概念,使用,及生命周期;
* 能够使用Cookie存储中文并了解Cookie的路径作用;
* 掌握Session的原理及使用方法;
* 能够详细说明Cookie和Session的区别;
* 能够独立完成自动登陆