# Day06

## 会话跟踪技术

### 1　什么是会话跟踪技术

我们需要先了解一下什么是会话！可以把会话理解为客户端与服务器之间的一次会晤，在一次会晤中可能会包含多次请求和响应。例如你给10086打个电话，你就是客户端，而10086服务人员就是服务器了。从双方接通电话那一刻起，会话就开始了，到某一方挂断电话表示会话结束。在通话过程中，你会向10086发出多个请求，那么这多个请求都在一个会话中。

在JavaWeb中，客户向某一服务器发出第一个请求开始，会话就开始了，直到客户关闭了浏览器会话结束。

在一个会话的多个请求中共享数据，这就是会话跟踪技术。例如在一个会话中的请求如下：

* 请求银行主页；
* 请求登录（请求参数是用户名和密码）；
* 请求转账（请求参数与转账相关的数据）；
* 请求信用卡还款（请求参数与还款相关的数据）。

在这上会话中当前用户信息必须在这个会话中共享的，因为登录的是张三，那么在转账和还款时一定是相对张三的转账和还款！这就说明我们必须在一个会话过程中有共享数据的能力。

### 2　会话路径技术使用Cookie或session完成

我们知道HTTP协议是无状态协议，也就是说每个请求都是独立的！无法记录前一次请求的状态。但HTTP协议中可以使用Cookie来完成会话跟踪！

在JavaWeb中，使用session来完成会话跟踪，session底层依赖Cookie技术。

## Cookie

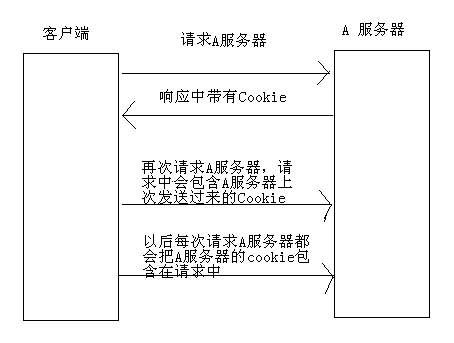
### 1　Cookie概述

#### 1.1　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。

客户使用哪一个浏览器向服务器发请求，服务器就给该浏览器写cookie

记住：cookie有浏览器之分的。



　　Cookie是由服务器创建，然后通过响应发送给客户端的一个键值对。客户端会保存Cookie，并会标注出Cookie的来源（哪个服务器的Cookie）。当客户端向服务器发出请求时会把所有这个服务器Cookie包含在请求中发送给服务器，这样服务器就可以识别客户端了！

#### 1.2　Cookie规范

* Cookie大小上限为4KB；4\*1024 对中文支持不好
* 一个服务器最多在客户端浏览器上保存20个Cookie；
* 一个浏览器最多保存300个Cookie；

上面的数据只是HTTP的Cookie规范，但在浏览器大战的今天，一些浏览器为了打败对手，为了展现自己的能力起见，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！

注意，不同浏览器之间是不共享Cookie的。也就是说在你使用IE访问服务器时，服务器会把Cookie发给IE，然后由IE保存起来，当你在使用FireFox访问服务器时，不可能把IE保存的Cookie发送给服务器。

#### 1.3　Cookie与HTTP头

Cookie是通过HTTP请求和响应头在客户端和服务器端传递的：

* Cookie：请求头，客户端发送给服务器端；
* 格式：Cookie: a=A; b=B; c=C。即多个Cookie用分号离开；
* Set-Cookie：响应头，服务器端发送给客户端；
* 一个Cookie对象一个Set-Cookie：

Set-Cookie: a=A

Set-Cookie: b=B

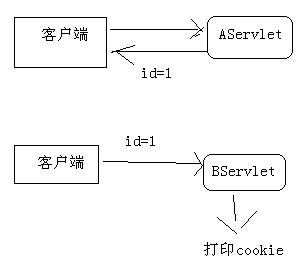
Set-Cookie: c=C

#### 1.4　Cookie的覆盖

　　如果服务器端发送重复的Cookie那么会覆盖原有的Cookie，例如客户端的第一个请求服务器端发送的Cookie是：Set-Cookie: a=A；第二请求服务器端发送的是：Set-Cookie: a=AA，那么客户端只留下一个Cookie，即：a=AA。

#### 1.5　Cookie第一例

我们这个案例是，客户端访问AServlet，AServlet在响应中添加Cookie，浏览器会自动保存Cookie。然后客户端访问BServlet，这时浏览器会自动在请求中带上Cookie，BServlet获取请求中的Cookie打印出来。



AServlet.java

|  |
| --- |
| **package** cn.kgc.servlet;  **import** java.io.IOException;  **import** java.util.UUID;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.Cookie;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  /\*\*  \* 给客户端发送Cookie  \* **@author** Administrator  \*  \*/  **public** **class** AServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  response.setContentType("text/html;charset=utf-8");    String id = UUID.*randomUUID*().toString();//生成一个随机字符串  Cookie cookie = **new** Cookie("id", id);//创建Cookie对象，指定名字和值  response.addCookie(cookie);//在响应中添加Cookie对象  response.getWriter().print("已经给你发送了ID");  }  } |

BServlet.java

|  |
| --- |
| **package** cn.kgc.servlet;  **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;  /\*\*  \* 获取客户端请求中的Cookie  \* **@author** Administrator  \*  \*/  **public** **class** BServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  response.setContentType("text/html;charset=utf-8");    Cookie[] cs = request.getCookies();//获取请求中的Cookie  **if**(cs != **null**) {//如果请求中存在Cookie  **for**(Cookie c : cs) {//遍历所有Cookie  **if**(c.getName().equals("id")) {//获取Cookie名字，如果Cookie名字是id  response.getWriter().print("您的ID是：" + c.getValue());//打印Cookie值  }  }  }  }  } |

### 2　Cookie的生命

#### 2.1　什么是Cookie的生命

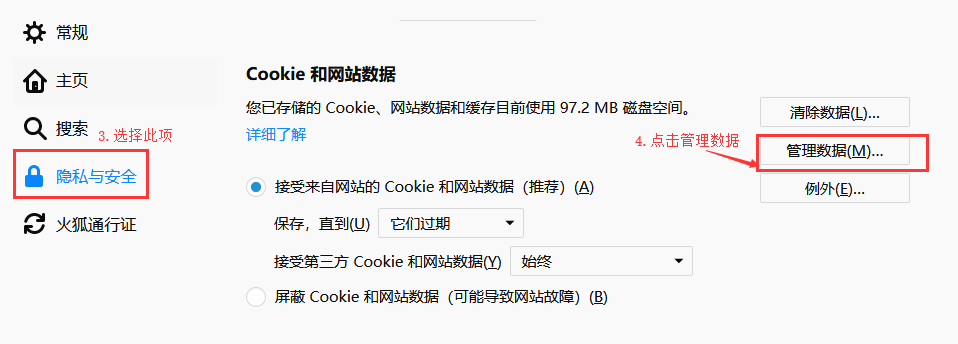
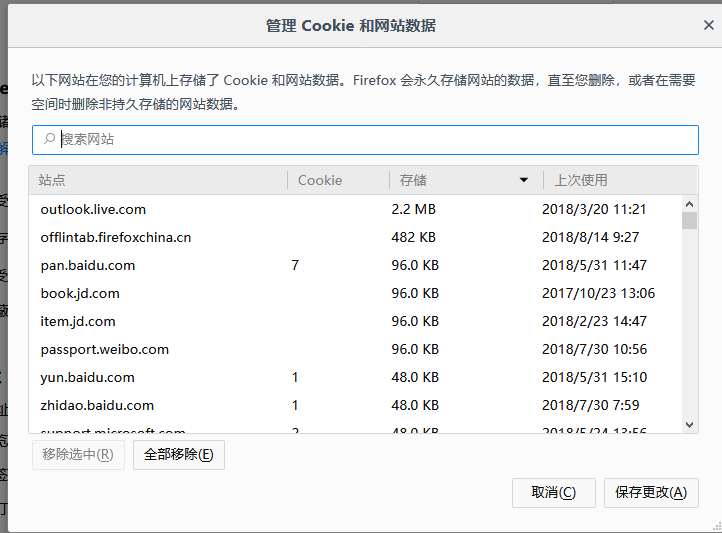
　　Cookie不只是有name和value，Cookie还是生命。所谓生命就是Cookie在客户端的有效时间，可以通过setMaxAge(int)来设置Cookie的有效时间。

* cookie.setMaxAge(-1)：cookie的maxAge属性的默认值就是-1，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(60\*60)：表示cookie对象可存活1小时。当生命大于0时，浏览器会把Cookie保存到硬盘上，就算关闭浏览器，就算重启客户端电脑，cookie也会存活1小时；
* cookie.setMaxAge(0)：cookie生命等于0是一个特殊的值，它表示cookie被作废！也就是说，如果原来浏览器已经保存了这个Cookie，那么可以通过Cookie的setMaxAge(0)来删除这个Cookie。无论是在浏览器内存中，还是在客户端硬盘上都会删除这个Cookie。

#### 2.2　浏览器查看Cookie

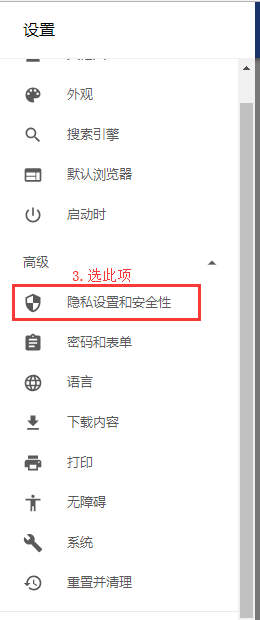
下面是浏览器查看Cookie的方式：

* IE查看Cookie文件的路径：C:\Users\用户名\AppData\Roaming\Microsoft\Windows\Cookies
* FireFox查看Cookie：

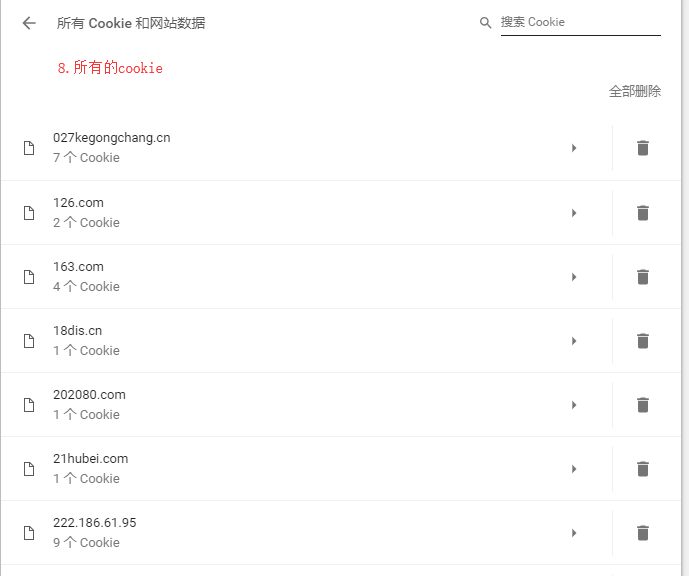
* Google查看Cookie：









#### 2.3　案例：显示上次访问时间

* 创建Cookie，名为lasttime，值为当前时间，添加到response中；
* 在AServlet中获取请求中名为lasttime的Cookie；
* 如果不存在输出“您是第一次访问本站”，如果存在输出“您上一次访问本站的时间是xxx”；

AServlet.java

|  |
| --- |
| **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  response.setContentType("text/html;charset=utf-8");  //创建Cookie对象，名为lasttime，值为当前时间  Cookie cookie = **new** Cookie("lasttime", **new** Date().toString());  //设置Cookie在客户端的有效时间为1小时  cookie.setMaxAge(60 \* 60);  //添加Cookie到response中  response.addCookie(cookie);  //获取请求中的Cookie  Cookie[] cs = request.getCookies();  String s = "您是首次访问本站！";  //如果请求中存在Cookie  **if**(cs != **null**) {  //循环遍历请求中的Cookie  **for**(Cookie c : cs) {  //如果Cookie名为lasttime  **if**(c.getName().equals("lasttime")) {  //设置变量s的值  s = "您上次的访问时间是：" + c.getValue();  }  }  }  //打印变量s的值到响应端  response.getWriter().print(s);  } |

### 3　Cookie的path

#### 3.1　什么是Cookie的路径

现在有WEB应用A，向客户端发送了10个Cookie，这就说明客户端无论访问应用A的哪个Servlet都会把这10个Cookie包含在请求中！但是也许只有AServlet需要读取请求中的Cookie，而其他Servlet根本就不会获取请求中的Cookie。这说明客户端浏览器有时发送这些Cookie是多余的！

可以通过设置Cookie的path来指定浏览器，在访问什么样的路径时，包含什么样的Cookie。

#### 3.2　Cookie路径与请求路径的关系

下面我们来看看Cookie路径的作用：

下面是客户端浏览器保存的3个Cookie的路径：

a:　/cookietest；

b:　/cookietest/servlet；

c:　/cookietest/jsp；

下面是浏览器请求的URL：

A:　http://localhost:8080/cookietest/AServlet；

B:　http://localhost:8080/cookietest/servlet/BServlet；

C:　http://localhost:8080/cookietest/servlet/CServlet；

* 请求A时，会在请求中包含a；
* 请求B时，会在请求中包含a、b；
* 请求C时，会在请求中包含a、c；

也就是说，请求路径如果包含了Cookie路径，那么会在请求中包含这个Cookie，否则不会请求中不会包含这个Cookie。

* A请求的URL包含了“/cookietest”，所以会在请求中包含路径为“/cookietest”的Cookie；
* B请求的URL包含了“/cookietest”，以及“/cookietest/servlet”，所以请求中包含路径为“/cookietest”和“/cookietest/servlet”两个Cookie；
* B请求的URL包含了“/cookietest”，以及“/cookietest/jsp”，所以请求中包含路径为“/cookietest”和“/cookietest/jsp”两个Cookie；

#### 3.3　设置Cookie的路径

设置Cookie的路径需要使用setPath()方法，例如：

cookie.setPath(“/cookietest/servlet”);

如果没有设置Cookie的路径，那么Cookie路径的默认值当前访问资源所在路径，例如：

* 访问http://localhost:8080/cookietest/AServlet时添加的Cookie默认路径为/cookietest；
* 访问http://localhost:8080/cookietest/servlet/BServlet时添加的Cookie默认路径为/cookietest/servlet；
* 访问http://localhost:8080/cookietest/jsp/BServlet时添加的Cookie默认路径为/cookietest/jsp；

### 4　Cookie中保存中文

Cookie的name和value都不能使用中文，如果希望在Cookie中使用中文，那么需要先对中文进行URL编码，然后把编码后的字符串放到Cookie中。

　向客户端响应中添加Cookie

|  |
| --- |
| //使用URL编码  String name = URLEncoder.*encode*("姓名", "UTF-8");  String value = URLEncoder.*encode*("张三", "UTF-8");  //编码后的字符串保存到Cookie中  Cookie c = **new** Cookie(name, value);  c.setMaxAge(3600);  response.addCookie(c); |

从客户端请求中获取Cookie

|  |
| --- |
| response.setContentType("text/html;charset=utf-8");  Cookie[] cs = request.getCookies();  **if**(cs != **null**) {  **for**(Cookie c : cs) {  //把Cookie的name和value使用URL解码后再打印。  String name = URLDecoder.*decode*(c.getName(), "UTF-8");  String value = URLDecoder.*decode*(c.getValue(), "UTF-8");  String s = name + ": " + value + "<br/>";  response.getWriter().print(s);  }  } |

### 5　显示曾经浏览过的商品

index.jsp

|  |
| --- |
| <body>  <h1>商品列表</h1>  <a href=*"/day06\_3/GoodServlet?name=ThinkPad"*>ThinkPad</a><br/>  <a href=*"/day06\_3/GoodServlet?name=Lenovo"*>Lenovo</a><br/>  <a href=*"/day06\_3/GoodServlet?name=Apple"*>Apple</a><br/>  <a href=*"/day06\_3/GoodServlet?name=HP"*>HP</a><br/>  <a href=*"/day06\_3/GoodServlet?name=SONY"*>SONY</a><br/>  <a href=*"/day06\_3/GoodServlet?name=ACER"*>ACER</a><br/>  <a href=*"/day06\_3/GoodServlet?name=DELL"*>DELL</a><br/>    <hr/>  您浏览过的商品：  <%  Cookie[] cs = request.getCookies();  **if**(cs != **null**) {  **for**(Cookie c : cs) {  **if**(c.getName().equals("goods")) {  out.print(c.getValue());  }  }  }  %>  </body> |

GoodServlet

|  |
| --- |
| **public** **class** GoodServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  String goodName = request.getParameter("name");  String goods = CookieUtils.*getCookValue*(request, "goods");    **if**(goods != **null**) {  String[] arr = goods.split(", ");  Set<String> goodSet = **new** LinkedHashSet(Arrays.*asList*(arr));  goodSet.add(goodName);  goods = goodSet.toString();  goods = goods.substring(1, goods.length() - 1);  } **else** {  goods = goodName;  }  Cookie cookie = **new** Cookie("goods", goods);  cookie.setMaxAge(1 \* 60 \* 60 \* 24);  response.addCookie(cookie);    response.sendRedirect("/day06\_3/index.jsp");  }  } |

CookieUtils

|  |
| --- |
| **public** **class** CookieUtils {  **public** **static** String getCookValue(HttpServletRequest request, String name) {  Cookie[] cs = request.getCookies();  **if**(cs == **null**) {  **return** **null**;  }  **for**(Cookie c : cs) {  **if**(c.getName().equals(name)) {  **return** c.getValue();  }  }  **return** **null**;  }  } |

## HttpSession

### HttpSession概述

#### 1.1　什么是HttpSesssion

javax.servlet.http.HttpSession接口表示一个会话，我们可以把一个会话内需要共享的数据保存到HttSession对象中！

Cookie保存用户电脑上（客户端）

Session保存服务器上

#### 1.2　获取HttpSession对象

* HttpSession request.getSesssion()：如果当前会话已经有了session对象那么直接返回，如果当前会话还不存在会话，那么创建session并返回；
  + 特点：有就直接用，没有帮创建
* HttpSession request.getSession(boolean)：当参数为true时，与requeset.getSession()相同。如果参数为false，那么如果当前会话中存在session则返回，不存在返回null；
  + True==========🡺getSession()
  + False=====🡺有就直接使用，没有直接null

#### 1.3　HttpSession是域对象

javaWeb一共有四个域对象：

其中三个用在Servlet：

ServletContext—application：tomcat不重启，数据就不会丢失

HttpSession---session：默认情况浏览器关闭，会话结束 session数据丢失了

HttpServletRequest—request：只要请求对象不发生变化，数据就不会丢失

另外一个对象用在JSP页面

只要是域对象，负责数据存和取，还提供移除作用：

setAttribute(key,value):存数据

getAttribute(key):取数据

removeAttribute(key):删除一个键值对

我们已经学习过HttpServletRequest、ServletContext，它们都是域对象，现在我们又学习了一个HttpSession，它也是域对象。它们三个是Servlet中可以使用的域对象，而JSP中可以多使用一个域对象，明天我们再讲解JSP的第四个域对象。

* HttpServletRequest：一个请求创建一个request对象，所以在同一个请求中可以共享request，例如一个请求从AServlet转发到BServlet，那么AServlet和BServlet可以共享request域中的数据；
* ServletContext：一个应用只创建一个ServletContext对象，所以在ServletContext中的数据可以在整个应用中共享，只要不启动服务器，那么ServletContext中的数据就可以共享；
* HttpSession：一个会话创建一个HttpSession对象，同一会话中的多个请求中可以共享session中的数据；

下面是session的域方法：

* void setAttribute(String name, Object value)：用来存储一个对象，也可以称之为存储一个域属性，例如：session.setAttribute(“xxx”, “XXX”)，在session中保存了一个域属性，域属性名称为xxx，域属性的值为XXX。请注意，如果多次调用该方法，并且使用相同的name，那么会覆盖上一次的值，这一特性与Map相同；
* Object getAttribute(String name)：用来获取session中的数据，当前在获取之前需要先去存储才行，例如：String value = (String) session.getAttribute(“xxx”);，获取名为xxx的域属性；
* void removeAttribute(String name)：用来移除HttpSession中的域属性，如果参数name指定的域属性不存在，那么本方法什么都不做；
* Enumeration getAttributeNames()：获取所有域属性的名称；

### 2　登录案例

需要的页面：

* login.jsp：登录页面，提供登录表单；
* index1.jsp：主页，显示当前用户名称，如果没有登录，显示您还没登录；
* index2.jsp：主页，显示当前用户名称，如果没有登录，显示您还没登录；

Servlet：

* LoginServlet：在login.jsp页面提交表单时，请求本Servlet。在本Servlet中获取用户名、密码进行校验，如果用户名、密码错误，显示“用户名或密码错误”，如果正确保存用户名session中，然后重定向到index1.jsp；

　　当用户没有登录时访问index1.jsp或index2.jsp，显示“您还没有登录”。如果用户在login.jsp登录成功后到达index1.jsp页面会显示当前用户名，而且不用再次登录去访问index2.jsp也会显示用户名。因为多次请求在一个会话范围，index1.jsp和index2.jsp都会到session中获取用户名，session对象在一个会话中是相同的，所以都可以获取到用户名！

login.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>login.jsp</title>  </head>    <body>  <h1>login.jsp</h1>  <hr/>  <form action=*"/day06\_4/LoginServlet"* method=*"post"*>  用户名：<input type=*"text"* name=*"username"* /><br/>  <input type=*"submit"* value=*"Submit"*/>  </form>  </body>  </html> |

index1.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>index1.jsp</title>  </head>    <body>  <h1>index1.jsp</h1>  <%  String username = (String)session.getAttribute("username");  **if**(username == **null**) {  out.print("您还没有登录！");  } **else** {  out.print("用户名：" + username);  }  %>  <hr/>  <a href=*"/day06\_4/index2.jsp"*>index2</a>  </body>  </html> |

index2.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>index2.jsp</title>  </head>    <body>  <h1>index2.jsp</h1>  <%  String username = (String)session.getAttribute("username");  **if**(username == **null**) {  out.print("您还没有登录！");  } **else** {  out.print("用户名：" + username);  }  %>  <hr/>  <a href=*"/day06\_4/index1.jsp"*>index1</a>  </body>  </html> |

LoginServlet

|  |
| --- |
| **public** **class** LoginServlet **extends** HttpServlet {  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  request.setCharacterEncoding("utf-8");  response.setContentType("text/html;charset=utf-8");  //获取表单参数username  String username = request.getParameter("username");  //如果用户为admin表示登录失败  **if**(username.equalsIgnoreCase("admin")) {  response.getWriter().print("用户名或密码错误！");  } **else** {  //获取session对象  HttpSession session = request.getSession();  //在session中保存用户名  session.setAttribute("username", username);  //重定向到index1.jsp  response.sendRedirect("/day06\_4/index1.jsp");  }  }  } |

### 3　session的实现原理

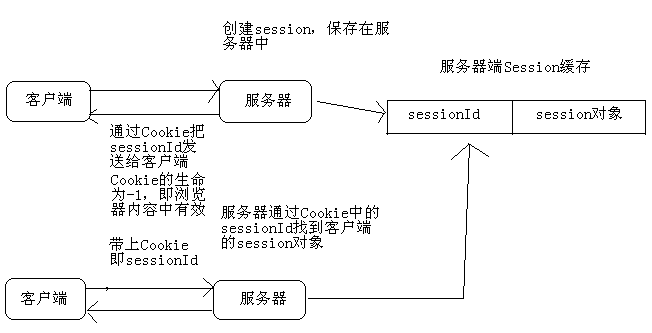
session底层是依赖Cookie的！我们来理解一下session的原理吧！

当我首次去银行时，因为还没有账号，所以需要开一个账号，我获得的是银行卡，而银行这边的数据库中留下了我的账号，我的钱是保存在银行的账号中，而我带走的是我的卡号。

当我再次去银行时，只需要带上我的卡，而无需再次开一个账号了。只要带上我的卡，那么我在银行操作的一定是我的账号！

当首次使用session时，服务器端要创建session，session是保存在服务器端，而给客户端的session的id（一个cookie中保存了sessionId）。客户端带走的是sessionId，而数据是保存在session中。

当客户端再次访问服务器时，在请求中会带上sessionId，而服务器会通过sessionId找到对应的session，而无需再创建新的session。



### 4　session与浏览器

生命周期：

产生时机：服务器执行request.getSession()代码创建session对象。

销毁时间：web.xml设置的session-timeout的时间，默认30分钟

数据有效期：session中保存的数据默认情况浏览器关闭，就丢失。

如果设置session中保存的数据，不因为浏览器关闭，而丢失，改进方案是：

1. 设置保存JSESSIONID的cookie的有效期

session保存在服务器，而sessionId通过Cookie发送给客户端，但这个Cookie的生命周期是-1，即只在浏览器内存中存在，也就是说如果用户关闭了浏览器，那么这个Cookie就丢失了。

当用户再次打开浏览器访问服务器时，就不会有sessionId发送给服务器，那么服务器会认为你没有session，所以服务器会创建一个session，并在响应中把sessionId中到Cookie中发送给客户端。

你可能会说，那原来的session对象会怎样？当一个session长时间没人使用的话，服务器会把session删除了！这个时长在Tomcat中配置是30分钟，可以在${CATALANA}/conf/web.xml找到这个配置，当然你也可以在自己的web.xml中覆盖这个配置！====不推荐

修改方案：在当前项目的web.xml添加配置节，控制服务器端的session的有效期：

web.xml

|  |
| --- |
| <session-config>  <session-timeout>30</session-timeout>  </session-config> |

session失效时间也说明一个问题！如果你打开网站的一个页面开始长时间不动，超出了30分钟后，再去点击链接或提交表单时你会发现，你的session已经丢失了！

### 5　session其他常用API

* String getId()：获取sessionId；
* int getMaxInactiveInterval()：获取session可以的最大不活动时间（秒），默认为30分钟。当session在30分钟内没有使用，那么Tomcat会在session池中移除这个session；
* void setMaxInactiveInterval(int interval)：设置session允许的最大**不活动时间**（秒），如果设置为1秒，那么只要session在1秒内不被使用，那么session就会被移除；
* long getCreationTime()：返回session的创建时间，返回值为当前时间的毫秒值；
* long getLastAccessedTime()：返回session的最后活动时间，返回值为当前时间的毫秒值；
* void invalidate()：让session失效！调用这个方法会被session失效，当session失效后，客户端再次请求，服务器会给客户端创建一个新的session，并在响应中给客户端新session的sessionId；
* boolean isNew()：查看session是否为新。当客户端第一次请求时，服务器为客户端创建session，但这时服务器还没有响应客户端，也就是还没有把sessionId响应给客户端时，这时session的状态为新。

### 6　URL重写

我们知道session依赖Cookie，那么session为什么依赖Cookie呢？因为服务器需要在每次请求中获取sessionId，然后找到客户端的session对象。那么如果客户端浏览器关闭了Cookie呢？那么session是不是就会不存在了呢？

其实还有一种方法让服务器收到的每个请求中都带有sessioinId，那就是URL重写！在每个页面中的每个链接和表单中都添加名为jsessionId的参数，值为当前sessionid。当用户点击链接或提交表单时也服务器可以通过获取jsessionId这个参数来得到客户端的sessionId，找到sessoin对象。

index.jsp

|  |
| --- |
| <body>  <h1>URL重写</h1>  <a href=*'/day06\_5/index.jsp;jsessionid=*<%=session.getId() %>*'* >主页</a>  <form action=*'/day06\_5/index.jsp;jsessionid=*<%=session.getId() %>*'* method=*"post"*>  <input type=*"submit"* value=*"提交"*/>  </form>  </body> |

也可以使用response.encodeURL()对每个请求的URL处理，这个方法会自动追加jsessionid参数，与上面我们手动添加是一样的效果。

|  |
| --- |
| <a href=*'*<%=response.encodeURL("/day06\_5/index.jsp") %>*'* >主页</a>  <form action=*'*<%=response.encodeURL("/day06\_5/index.jsp") %>*'* method=*"post"*>  <input type=*"submit"* value=*"提交"*/>  </form> |

使用response.encodeURL()更加“智能”，它会判断客户端浏览器是否禁用了Cookie，如果禁用了，那么这个方法在URL后面追加jsessionid，否则不会追加。

### 7　修改保存JSESSIONID的cookie的有效期参考代码

|  |
| --- |
| HttpSession session = request.getSession();//服务器创建session对象  //延长保存sessionid的cookie有效期  //1.cookie创建出来  Cookie cookie=new Cookie("JSESSIONID",session.getId());  //2.设置cookie有效期  cookie.setMaxAge(10\*60);  //3.浏览器回送cookie  response.addCookie(cookie); |

Session常见案例：

1. 登录成功，为了在不同网页显示登录账号，账号信息存session
2. 验证码，底层依赖session

## 案例：一次性图片验证码

### 验证码有啥用

防范暴力破解、暴力登录...

在我们注册时，如果没有验证码的话，可以使用while(true)来注册！那么服务器就废了！

While(true){

Location.href=”/loginServlet?username=自动生成&pwd=自动生成”

}

验证码可以去识别发出请求的是人还是程序！当然，如果聪明的程序可以去分析验证码图片！但分析图片也不是一件容易的事，因为一般验证码图片都会带有干扰线，人都看不清，那么程序一定分析不出来。

### 2　VerifyCode类

现在我们已经有了cn.kgc.utils.VerifyCode类，这个类可以生成验证码图片！下面来看一个小例子。

|  |
| --- |
| **public** **void** fun1() **throws** IOException {  // 创建验证码类  VerifyCode vc = **new** VerifyCode();  // 获取随机图片  BufferedImage image = vc.getImage();  // 获取刚刚生成的随机图片上的文本  String text = vc.getText();  System.*out*.println(text);  // 保存图片  FileOutputStream out = **new** FileOutputStream("F:/xxx.jpg");  VerifyCode.*output*(image, out);  } |

### 3　在页面中显示动态图片

我们需要写一个VerifyCodeServlet，在这个Servlet中我们生成动态图片，然后它图片写入到response.getOutputStream()流中！然后让页面的<img>元素指定这个VerifyCodServlet即可。

VerifyCodeServlet

|  |
| --- |
| **public** **class** VerifyCodeServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  VerifyCode vc = **new** VerifyCode();  BufferedImage image = vc.getImage();  String text = vc.getText();  System.*out*.println("text:" + text);  VerifyCode.*output*(image, response.getOutputStream());  }  } |

index.jsp

|  |
| --- |
| <script type=*"text/javascript"*>  **function** \_change() {  //获取<img>元素  **var** imgEle = document.getElementById("vCode");  //重新指定src！因为有些浏览器会对其进行缓存，所以需要添加不同的参数，保证每次URL的参数都不同。  imgEle.src = "/day06\_6/VerifyCodeServlet?t=" + **new** Date().getTime();  }  </script>  ...  <body>  <h1>验证码</h1>  <img id=*"vCode"* src=*"/day06\_6/VerifyCodeServlet"*/>  <a href=*"javascript:\_change()"*>看不清，换一张</a>  </body> |

### 4　在注册页面中使用验证码

|  |
| --- |
| <form action=*"/day06\_6/RegistServlet"* method=*"post"*>  用户名：<input type=*"text"* name=*"username"*/><br/>  验证码：<input type=*"text"* name=*"code"* size=*"3"*/>  <img id=*"vCode"* src=*"/day06\_6/VerifyCodeServlet"*/>  <a href=*"javascript:\_change()"*>看不清，换一张</a>  <br/>  <input type=*"submit"* value=*"Submit"*/>  </form> |

### 5　RegistServlet

修改VerifyCodeServlet

|  |
| --- |
| **public** **class** VerifyCodeServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  VerifyCode vc = **new** VerifyCode();  BufferedImage image = vc.getImage();  //在session中保存验证码文本  request.getSession().setAttribute("vCode", vc.getText());  VerifyCode.*output*(image, response.getOutputStream());  }  } |

RegistServlet

|  |
| --- |
| **public** **class** RegistServlet **extends** HttpServlet {  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  request.setCharacterEncoding("utf-8");  response.setContentType("text/html;charset=utf-8");    String username = request.getParameter("username");  //获取表单中的验证码，这是用户在表单中填写的。  String vCode = request.getParameter("code");  //获取session中的验证码，这是在生成验证码图片后，保存在session中的正确的验证码文本  String sessionVerifyCode = (String)request.getSession().getAttribute("vCode");  //比较用户输入的与真正的是否相同。  **if**(vCode.equalsIgnoreCase(sessionVerifyCode)) {  response.getWriter().print(username + ", 恭喜！注册成功！");  } **else** {  response.getWriter().print("验证码错误！");  }  }  } |

### 6　总结验证码案例

* VerifyCodeServlet：
* 生成验证码：VerifyCode vc = new VerifyCode(); BufferedImage image = vc.getImage()；
* 在session中保存验证码文本：request.getSession.getAttribute(“vCode”, vc.getText())；
* 把验证码输出到页面：VerifyCode.output(image, response.getOutputStream)；
* regist.jsp：
* 表单中包含username和code字段；
* 在表单中给出<img>指向VerifyCodeServlet，用来在页面中显示验证码图片；
* 提供“看不清，换一张”链接，指向\_change()函数；
* 提交到RegistServlet；
* RegistServlet：
* 获取表单中的username和code；
* 获取session中的vCode；
* 比较code和vCode是否相同；
* 相同说明用户输入的验证码正确，否则输入验证码错误。

## ServletContext（重要）

一个项目只有一个ServletContext对象！ServletContext实际指向当前WEB应用。Web应用程序。

我们可以在N多个Servlet中来获取这个唯一的对象，使用它可以给多个Servlet传递数据！

这个对象在Tomcat启动时就创建，在Tomcat关闭时才会销毁！

### ServletContext概述

指实际存在的一个Web应用，所有web程序发布到服务器上面，服务器就会产生一个ServletContext对象。服务器会为每个应用创建一个ServletContext对象：

* ServletContext对象的创建是在服务器启动时完成的；init()初始化的方法
* ServletContext对象的销毁是在服务器关闭时完成的；destroy()

　　 ServletContext对象的作用是在整个Web应用的动态资源之间共享数据！例如在AServlet中向ServletContext对象中保存一个值，然后在BServlet中就可以获取这个值，这就是共享数据了。

### 2　获取ServletContext

* ServletConfig#getServletContext()；
* GenericServlet#getServletContext();
* HttpSession#getServletContext()
* ServletContextEvent#getServletContext()

在Servlet中获取ServletContext对象：

* 在void init(ServletConfig config)中：ServletContext context = config.getServletContext();，ServletConfig类的getServletContext()方法可以用来获取ServletContext对象；

在GenericeServlet或HttpServlet中获取ServletContext对象：

* GenericServlet类有getServletContext()方法，所以可以直接使用this.getServletContext()来获取；

|  |
| --- |
| public class MyServlet implements Servlet {  public void init(ServletConfig config) {  ServletContext context = config.getServletContext();  }  …  } |
| public class MyServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response) {  ServletContext context = this.getServletContext();  }  } |

### 3　域对象的功能

ServletContext是JavaWeb四大域对象之一：

* PageContext；
* ServletRequest；
* HttpSession；
* ServletContext；

所有域对象都有存取数据的功能，因为域对象内部有一个Map，用来存储数据，下面是ServletContext对象用来操作数据的方法：

* void setAttribute(String name, Object value)：用来存储一个对象，也可以称之为存储一个域属性，例如：servletContext.setAttribute(“xxx”, “XXX”)，在ServletContext中保存了一个域属性，域属性名称为xxx，域属性的值为XXX。请注意，如果多次调用该方法，并且使用相同的name，那么会覆盖上一次的值，这一特性与Map相同；
* Object getAttribute(String name)：用来获取ServletContext中的数据，当前在获取之前需要先去存储才行，例如：String value = (String)servletContext.getAttribute(“xxx”);，获取名为xxx的域属性；
* void removeAttribute(String name)：用来移除ServletContext中的域属性，如果参数name指定的域属性不存在，那么本方法什么都不做；
* Enumeration getAttributeNames()：获取所有域属性的名称；

### 4　获取应用初始化参数

* Servlet也可以获取初始化参数，但它是局部的参数；也就是说，一个Servlet只能获取自己的初始化参数，不能获取别人的，即初始化参数只为一个Servlet准备！
* 可以配置公共的初始化参数，为所有Servlet而用！这需要使用ServletContext才能使用！

还可以使用ServletContext来获取在web.xml文件中配置的应用初始化参数！注意，应用初始化参数与Servlet初始化参数不同：

web.xml

|  |
| --- |
| <web-app ...>  ...  <context-param>  <param-name>paramName1</param-name>  <param-value>paramValue1</param-value>  </context-param>  <context-param>  <param-name>paramName2</param-name>  <param-value>paramValue2</param-value>  </context-param>  </web-app> |
| ServletContext context = **this**.getServletContext();  String value1 = context.getInitParameter("paramName1");  String value2 = context.getInitParameter("paramName2");  System.*out*.println(value1 + ", " + value2);    Enumeration names = context.getInitParameterNames();  **while**(names.hasMoreElements()) {  System.*out*.println(names.nextElement());  } |

### 5　练习：访问量统计

一个项目中所有的资源被访问都要对访问量进行累加！

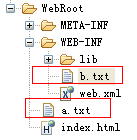
创建一个int类型的变量，用来保存访问量，然后把它保存到ServletContext的域中，这样可以保证所有的Servlet都可以访问到！

* 最初时，ServletContext中没有保存访问量相关的属性；
* 当本站第一次被访问时，创建一个变量，设置其值为1；保存到ServletContext中；
* 当以后的访问时，就可以从ServletContext中获取这个变量，然后在其基础之上加１。
* 获取ServletContext对象，查看是否存在名为count的属性，如果存在，说明不是第一次访问，如果不存在，说明是第一次访问；
* 第一次访问：调用Servletcontext的setAttribute()传递一个属性，名为count，值为1；
* 第2~N次访问：调用ServletContext的getAttribute()方法获取原来的访问量，给访问量加1，再调用Servletcontext的setAttribute()方法完成设置。

相信各位一定见过很多访问量统计的网站，即“本页面被访问过XXX次”。因为无论是哪个用户访问指定页面，都会累计访问量，所以这个访问量统计应该是整个项目共享的！很明显，这需要使用ServletContext来保存访问量。

|  |
| --- |
| //获取ServletContext对象  ServletContext application = **this**.getServletContext();  //获取ServletContext对象中的count属性  Integer count = (Integer)application.getAttribute("count");  **if**(count == **null**) {  //如果在ServletContext中不存在count属性，那么设置count的值为1，表示第一次被访问。  count = 1;  } **else** {  //如果在ServletContext中存在count属性，说明以前被访问过，那么让count在原来的基础上加1。  count++;  }  response.setContentType("text/html;charset=utf-8");  //向客户端响应本页面被访问的次数。  response.getWriter().print("<h1>本页面一共被访问" + count + "次！</h1>");  //保存count的值到ServletContext对象中。  application.setAttribute("count", count); |

## 获取资源相关方法



### 1　获取真实路径（\*\*\*\*\*）

还可以使用ServletContext对象来获取Web应用下的资源，例如在hello应用的根目录下创建a.txt文件，现在想在Servlet中获取这个资源，就可以使用ServletContext来获取。

* 获取a.txt的真实路径：String realPath = servletContext.getRealPath(“/a.txt”)，realPath的值为a.txt文件的绝对路径：D:\apache-tomcat-7.0.42\webapps\hello\a.txt；
* 获取b.txt的真实路径：String realPath = servletContext.getRealPath(“/WEB-INF/b.txt”)；

### 2　获取资源流

不只可以获取资源的路径，还可以通过ServletContext获取资源流，即把资源以输入流的方式获取：

* 获取a.txt资源流：InputStream in = servletContext.getResourceAsStream(“/a.txt”)；
* 获取b.txt资源流：InputStream in = servletContext.getResourceAsStream(“/WEB-INF/b.txt”)；

### 3　获取指定目录下所有资源路径

还可以使用ServletContext获取指定目录下所有资源路径，例如获取/WEB-INF下所有资源的路径：

|  |
| --- |
| Set set = context.getResourcePaths("/WEB-INF");  System.*out*.println(set); |
| [/WEB-INF/lib/, /WEB-INF/classes/, /WEB-INF/b.txt, /WEB-INF/web.xml] |

注意，本方法必须以“/”开头！！！

预习：

JSP和jsp内置对象

EL和jstl

作业：

1. 验证码实现登录===========》连接数据库
2. 基于实现登录，用户对象存session
3. 所有的jsp显示欢迎xxx 注销
4. 统计网站访问量