# 13.会话技术

## 13.1.什么是会话?

为了实现某一个功能, 浏览器和服务器之间可能会产生多次的请求和响应, 从浏览器访问服务器开始, 到最后访问服务器结束为止, 这期间产生的多次请求和响应加在一起, 就可以称之为浏览器和服务器之间的一次会话.

在一次会话中很可能会产生一些数据, 那么如何来保存会话中产生的数据??

~request对象作用范围太小了

~servletContext对象作用范围太大了

## 13.2.Cookie

### 13.2.1.Cookie概述

Cookie是将会话中产生的数据保存在客户端, 所以Cookie是一个客户端技术

Cookie保存数据的原理:

浏览器向服务器发送请求, 请求中包含着需要保存的数据, 服务器获取数据, 通过Set-Cookie响应头将数据再响应给浏览器, 让浏览器自己保存.

浏览器再次访问服务器时, 会在请求中, 通过一个Cookie请求头带着上次保存的数据, 服务器可以通过Cookie请求头来获取数据. 通过这种形式来保存会话中产生的数据!

由于Cookie技术是将会话产生的数据保存在客户端, 客户端各自保存各自的数据, 需要时再次带给服务器, 因此不会发生混乱!

### 13.2.2.实现Cookie

Cookie是基于 Set-Cookie响应头和Cookie请求头来保存会话中产生的数据

response.setHeader(“Set-Cookie”, “prod=小米手机”);

request.getHeader(“Cookie”) //prod=小米手机

**Cookie案例一：在页面中显示上次访问时间**

|  |
| --- |
| 创建web项目day12，创建包cn.tedu.cookie，创建servlet：CookieDemo1，并加入如下代码：  //解决乱码问题  response.setContentType("text/html;charset=utf-8");  //记住本次访问的时间  String dateStr = **new** Date().~~toLocaleString~~();  //将时间字符串通过Set-Cookie响应头发送给浏览器保存  response.setHeader("Set-Cookie", "time="+dateStr);  //同时获取第一次访问的时间，并将时间发送给浏览器显示在网页中  String dateStr2 = request.getHeader("Cookie");  response.getWriter().write("您上次访问的时间为："+dateStr2); |

SUN公司为了简化Cookie的操作, 提供了一套Cookie的API:

Cookie cookie = new Cookie(String name,String value);

cookie.getName(); //获取Cookie的名字

cookie.getValue(); //获取Cookie的value值

cookie.setValue(); //设置或修改cookie的value值

**添加Cookie到response响应中**

response.addCookie(cookie);//将Cookie信息添加到response响应中, 并发送给浏览器

该方法可以调用多次, 即可以将多个Cookie添加到response响应.

**获取Cookie**

Cookie[] cs = request.getCookies();//返回请求中所有的Cookie组成的数组

需要注意的是: 如果请求中没有cookie, 该方法返回值是null。

**Cookie案例一代码改造**

|  |
| --- |
| 在cookie包下创建servlet：CookieDemo2，并添加如下代码：  //解决乱码问题  response.setContentType("text/html;charset=utf-8");  //记住本次访问时间  String dateStr = **new** Date().~~toLocaleString~~();  //创建Cookie对象，将时间字符串保存在Cookie对象中  Cookie timeCookie = **new** Cookie("time",dateStr);  response.addCookie(timeCookie);  //获取上次访问时间，并在网页中显示上次访问时间  Cookie[] cookies = request.getCookies();  Cookie timeCookie2 = **null**;  **if**(cookies != **null**){  **for** (Cookie cookie : cookies) {  //获取当前遍历的cookie的名字  String name = cookie.getName();  **if** ("time".equals(name)) {  timeCookie2 = cookie;  }  }  }  **if**(timeCookie2 != **null**){  //获取上次访问的时间  String dateStr2 = timeCookie2.getValue();  response.getWriter().write("您上次访问的时间为："+dateStr2);  }**else**{  response.getWriter().write("您是第一次访问本网站");  } |

### 13.2.3.设置Cookie存活时间

setMaxAge();//该方法可以设置Cookie的最大生存时间。

如果不设置该方法, Cookie默认是会话级别的Cookie, Cookie是保存在浏览器的内存中, 如果浏览器关闭, 随着浏览器内存的释放, Cookie信息也会丢失.

如果设置了setMaxAge方法, 并且设置了一个有效的时间, Cookie将不会再保存到浏览器的内存中, 而是以文件的形式保存在浏览器的临时文件夹中(即本地硬盘上).

**代码示例：**

|  |
| --- |
| 在CookieDemo2中，创建好Cookie对象后，添加如下代码：  //创建Cookie对象，将时间字符串保存在Cookie对象中  Cookie timeCookie = **new** Cookie("time",dateStr);  //给cookie设置一个最大生存时间，让cookie信息以文件的形式，保存在浏览器的临时文件夹中，而不是内存中  timeCookie.setMaxAge(60\*60);//单位是秒，设置生存时间为1小时 |

### 13.2.4.设置Cookie路径

setPath();//设置Cookie路径, 如果不设置该方法, 默认浏览器会在访问

<http://localhost/day12/servlet/> 会带着Cookie

<http://localhost/day12/servlet/xxxx> 会带着Cookie

<http://localhost/day12/servlet/xxxx/xxx> 会带着Cookie

<http://localhost/day12/> 不会带着Cookie

也就说, 浏览器会在访问 <http://localhost/day12/servlet/> 或者其子孙路径时都会带着Cookie, 该路径是由发送Cookie的Servlet(CookieDemo2)所在的路径决定!（即http://localhost/day12/servlet/CookieDemo2所在路径为 http://localhost/day12/servlet/）

如果希望浏览器在访问当前应用(/day12)下任何一个资源时都能够带着Cookie, 我们应该把 <http://localhost/day12/servlet/> 改成 <http://localhost/day12/> 这个路径。

设置Cookie的路径：

//setPath(“http://localhost/day12/”);

//setPath(“[/day12/](http://localhost/day12/)”);

setPath(request.getContextPath()+“[/](http://localhost/day12/)”)

**代码示例：**

|  |
| --- |
| 在CookieDemo2中，创建好Cookie对象后，添加如下代码：  //创建Cookie对象，将时间字符串保存在Cookie对象中  Cookie timeCookie = **new** Cookie("time",dateStr);  //设置Cookie的路径，如果要设置为浏览器在访问当前应用下的任意资源时都带着Cookie，可以将Cookie的path设置为当前web应用的根路径  timeCookie.setPath(request.getContextPath()+"/"); |

### 13.2.5.设置domain

setDomain();//设置浏览器在访问哪一个主机时将会带着Cookie, 如果不设置, 浏览器在访问发送Cookie主机时将会带着Cookie

例如：

time localhost主机 cookie.setDomain(“http://www.baidu.com”)

//浏览器可以检测到并非baidu发的cookie，所以会拒绝接收。

time localhost主机 cookie.setDomain(“localhost”);

//默认就是自己，设置多此一举，而且若设置此方法，有的浏览器看到此方法也会拒绝接收

如果在创建Cookie之后, 设置了setDomain方法, 浏览器可能会拒绝接收Cookie, 所以最好不要设置!!!

### 13.2.6.删除Cookie

API中没有提供直接删除Cookie的方法!!

如果想删除Cookie, 可以向浏览器再次发送个 同名、同path、同domain的Cookie给浏览器, 并设置Cookie最大生存时间为0. 由于浏览器是根据Cookie的名字/path和domian来区分一个Cookie, 后发的Cookie会覆盖之前的Cookie, 又因为后发的Cookie最大生存为0, 浏览器收到之后也会删除后发的cookie, 通过这种方式可以删除一个Cookie!!

**代码示例：**

|  |
| --- |
| 在cookie包下创建servlet：CookieDemo3，并添加如下代码：  //删除Cookie  Cookie cookie = **new** Cookie("time","");  cookie.setPath(request.getContextPath()+"/");  cookie.setMaxAge(0);    response.addCookie(cookie);  response.getWriter().write("deleted ok..."); |

### 13.2.7.EasyMall登陆功能之记住用户名

代码如下：

|  |
| --- |
| 在WebRoot下login.jsp中，修改<form>标签属性action：  <form action=*"*<%= request.getContextPath() %>*/servlet/LoginServlet"* method=*"POST"*> |
| 在web包中创建servlet：LoginServlet，并添加如下代码：  //1.处理请求乱码  request.setCharacterEncoding("utf-8");  //2.获取请求参数  String username = request.getParameter("username");  String password = request.getParameter("password");  String remname = request.getParameter("remname");  //3.查询用户名和密码是否正确  Connection conn = **null**;  PreparedStatement ps = **null**;  ResultSet rs = **null**;  **try** {  conn = JDBCUtils.*getConnection*();  String sql = "select \* from user where username=? and password=?";  ps = conn.prepareStatement(sql);  ps.setString(1, username);  ps.setString(2, password);  rs = ps.executeQuery();  **if** (rs.next()) {//用户名密码正确  //4.当用户名密码正确时，实现记住用户名功能  **if**("true".equals(remname)){  /\*  \* 由于username要通过cookie头发送给浏览器，所以遵循http协议，  \* 但是浏览器不支持中文，会将数据进行url编码，例如：张飞-> %E5%BC%A0%E9%A3%9E  \* 所以给浏览器发送数据时，需要进行url编码  \* URLEncoder.encode(str,"utf-8");将中文字符进行编码  \* URLDecoder.decode(str,"utf-8");将中文字符进行解码  \*/  Cookie cookie = **new** Cookie("remname", URLEncoder.*encode*(username,"utf-8"));  cookie.setPath(request.getContextPath()+"/");  cookie.setMaxAge(60\*60\*24\*30);//设置30天  response.addCookie(cookie);  }**else**{//取消记住用户名（删除cookie）  Cookie cookie = **new** Cookie("remname","");  cookie.setPath(request.getContextPath()+"/");  cookie.setMaxAge(0);//设置30天  response.addCookie(cookie);  }  //5.去登陆（在session中保存用户的个人信息作为登录的标识）  //**TODO**    //6.登录成功后，跳转回主页（重定向）  response.sendRedirect( request.getContextPath()+ "/index.jsp");  }**else**{//用户名或密码不正确  //跳转回登录页面，并提示用户用户名或者密码错误  request.setAttribute("msg", "用户名或者密码错误");  request.getRequestDispatcher("/login.jsp").forward(request, response);  **return**;  }  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException();  }**finally**{  JDBCUtils.*close*(conn, ps, rs);  } |
| 在WebRoot下login.jsp中，获取提示消息：在<table>中最前面加入如下代码：  <table>  <tr>  <td colspan=*"2"* style="color:*red*;text-align: *center*">  <%= request.getAttribute("msg") == **null** ? "" : request.getAttribute("msg")%>  </td>  </tr> |
| 同样，在login.jsp中，如果cookie中保存了用户名，需要回显用户名，在用户名对应的<tr>中加入如下代码：  <tr>  <% //jsp片段，可以写多行java代码。  //1.获取请求中的所有Cookie信息  Cookie[] cookies = request.getCookies();  Cookie remCookie = **null**;  **if**(cookies != **null**){  **for**(Cookie cookie : cookies){  **if**("remname".equals(cookie.getName())){  remCookie = cookie;  }  }  }  String username = "";  **if**(remCookie != **null**){  username = remCookie.getValue();  //对用户名进行url解码(注意需要导包)  username = URLDecoder.decode(username,"utf-8");  }  %>  <td class=*"tdx"*>用户名：</td>  <td><input type=*"text"* name=*"username"* value=*"*<%= username %>*"*/></td>  </tr> |
| 最后，在login.jsp中，如果记住用户名之前是勾选的，说明Cookie中有值，以此可以判断记住用户名之前是否勾上，来进行回显。在记住用户名对应的<tr>中加入如下代码：  <tr>  <td colspan=*"2"*>  <input type=*"checkbox"* name=*"remname"* value=*"true"*  <%= remCookie == **null** ? "" : "checked = 'checked'" %>  />记住用户名  <input type=*"checkbox"* name=*"autologin"* value=*"true"*/>30天内自动登陆  </td>  </tr> |

## 13.3.Session

### 13.3.1.Session概述

Session是将会话中产生的数据保存在了服务器端, 是服务器端的技术。

**Session的原理**

浏览器向服务器发送请求, 将需要保存的数据带给服务器, 服务器通过请求获取数据, 接着去服务器内部检查有没有为当前浏览器服务的session, 如果有直接拿过来使用, 如果没有就创建一个session对象为当前浏览器服务. 将数据保存在session内部.

当浏览器再次访问服务器时, 服务器可以找到为当前浏览器服务的session, 从中取出数据, 通过这种方式也可以保存会话中产生的数据!

由于session技术是将会话中产生的数据保存在服务器内部的session中, 每一个浏览器都会有各自的session, 因此也不会发生混乱!

### 13.3.2.session是一个域对象

域对象: 如果一个对象具有可以被看见的范围, 利用对象上的map可以在整个范围内实现数据的共享!

session中提供了如下方法来操作域中的属性:

setAttribute(String name, Object value);

getAttribute(String name);

removeAttribute(String name);

getAttributeNames();

session域对象的特征:

**生命周期:**

**创建:** 第一次调用request.getSession()方法时创建session对象。

如果服务器中有session，就直接拿过来用，如果没有，则创建一个session拿过来用。

HttpSession session1 = request.getSession();

或

HttpSession session2 = request.getSession(true);

如果服务器中有session，就直接拿过来用，如果没有，并不会创建，而是返回null。

HttpSession session3 = request.getSession(false);

**销毁：**

超时死亡: 如果session 30分钟不使用就会超时销毁! 可以通过 web.xml中文件修改session的超时时间.

例如：在web.xml中的<web-app>标签中加入如下代码，其中时间设置的单位是分钟。

|  |
| --- |
| <session-config>  <session-timeout>1</session-timeout>  </session-config> |

**自杀:** 当调用session.invalidate()方法时会立即销毁session.

**意外身亡:** 当服务器意外关闭时(停电/宕机), session会立即销毁

当服务器正常关闭时, session并不会销毁, 而是以文件的形式保存到tomcat服务器中的work目录下, 这个过程叫做session的钝化. 当服务器再次开启时, 钝化的session还会恢复回来, 这个过程叫做session的活化.

**作用范围：**整个会话

**主要功能：**在整个会话范围内实现数据的共享!

**Session案例一：**

|  |
| --- |
| 在day12中创建包cn.tedu.session，创建servlet：SessionDemo1，并添加如下代码：  //1.当第一次调用request.getSession()时创建session对象  HttpSession session = request.getSession();  session.setAttribute("user", "张飞");  //重定向到首页  response.sendRedirect(request.getContextPath()+"/index.jsp"); |
| 在index.jsp中接收参数并判断：首先修改最上面第一行：由于jsp翻译成servlet后会默认生成Session，但是很多时候并不需要，所以可以在第一行设置session="false"，这样就不会默认创建，然后修改编码格式为utf-8  <%@ page language=*"java"* import=*"java.util.\*"* session=*"false"* pageEncoding=*"utf-8"*%> |
| 在index.jsp中接收参数并判断：<body>中添加如下代码：  <body>  <h1 style="color: *red*">  <!-- 如果用户未登录，则提示登录或注册 -->  <% **if**(request.getSession().getAttribute("user") == **null**){ %>  <a href=*"#"*>登录</a>  <a href=*"#"*>注册</a>  <!-- 如果用户已登录，则提示欢迎xxx回来 -->  <% }**else**{ %>  欢迎<%= request.getSession().getAttribute("user") %>回来!  <a href=*"#"*>退出</a>  <% } %>  </h1>  </body> |
| 在session包中创建servlet：SessionDemo2，并添加如下代码：  //立即销毁Session  request.getSession().invalidate(); |

### 13.3.3. Easymall登陆功能的实现

代码如下：

|  |
| --- |
| 因为有很多数据要放入到session中，所以可以创建一个User对象，将user对象保存到session中。  创建包cn.tedu.bean，创建类User，加入如下代码：（同时加入无参和有参构造，加入get和set方法）  **private** **int** id;  **private** String username;  **private** String password;  **private** String nickname;  **private** String email; |
| 在LoginServlet中的第5步，去登陆中加入如下代码：  //5.去登陆（在session中保存用户的个人信息作为登录的标识）  User user = **new** User();  user.setId(rs.getInt("id"));  user.setUsername(rs.getString("username"));  user.setPassword(rs.getString("password"));  user.setNickname(rs.getString("nickname"));  user.setEmail(rs.getString("email"));    //将user对象保存到session中  request.getSession().setAttribute("user", user); |
| 如果登陆，页面中应该显示已经登录。在\_head.jsp中的登录注册部分代码改造为：  <div id=*"content"*>  <!-- 如果用户未登录，则提示用户登录或注册 -->  <% **if**(request.getSession().getAttribute("user") == **null**){ %>  <a href=*"*<%= request.getContextPath() %>*/login.jsp"*>登录</a> &nbsp;&nbsp;|&nbsp;&nbsp;  <a href=*"*<%= request.getContextPath() %>*/regist.jsp"*>注册</a>  <!-- 如果用户已登录，则提示欢迎xxx回来 -->  <% }**else**{ %>  欢迎 <%= ((User)request.getSession().getAttribute("user")).getUsername() %> 回来，  <a href=*"*<%= request.getContextPath() %>*/servlet/LogoutServlet"*>退出</a>  <% } %>  </div> |
| 最后，需要实现退出功能。在web包中创建servlet：LogoutServlet，并添加如下代码：  //立即杀死session  request.getSession().invalidate();  //退出之后跳转回首页  response.sendRedirect(request.getContextPath()+"/index.jsp"); |

### 13.3.4.Session案例二：实现购物车

|  |
| --- |
| 在day12项目，WebRoot下创建sale.jsp，可以直接复制index.jsp并改名，在<body>中添加如下代码：  <body>  <a href=*"*<%= request.getContextPath() %>*/servlet/BuyServlet?prod=康帅傅方便面"*>康帅傅方便面</a> <br/><br/>  <a href=*"*<%= request.getContextPath() %>*/servlet/BuyServlet?prod=加乡宝凉茶"*>加乡宝凉茶</a> <br/><br/>  <a href=*"*<%= request.getContextPath() %>*/servlet/BuyServlet?prod=阿迪王皮鞋"*>阿迪王皮鞋</a> <br/><br/>  <a href=*"*<%= request.getContextPath() %>*/servlet/PayServlet"*>支付</a> <br/><br/>  </body> |
| 在day12项目，session包下创建servlet：BuyServlet，并添加如下代码：  response.setContentType("text/html;charset=utf-8");  //1.获取商品信息  String prod = request.getParameter("prod");  //2.解决乱码问题  prod = **new** String(prod.getBytes("iso8859-1"),"utf-8");  //3.获取session，并将商品信息加入到购物车  HttpSession session = request.getSession();  session.setAttribute("prod", prod);    response.getWriter().write("您已成功将["+prod+"]商品加入了购物车。"); |
| 在day12项目，session包下创建servlet：PayServlet，并添加如下代码：  response.setContentType("text/html;charset=utf-8");  //1.获取session（购物车）中的商品信息  String prod = (String) request.getSession().getAttribute("prod");  **if** (prod == **null**) {  response.getWriter().write("您还没有将任何商品加入到购物车...");  }**else**{  response.getWriter().write("您已成功为["+prod+"]商品支付了100元");  } |

测试后发现问题：session保存时间是30分钟，但是关闭浏览器后，session也消失了。

### 13.3.5.获取关闭之前的session

Session是基于一个名为JSESSIONID 的Cookie进行工作的

浏览器第一次访问服务器, 需要保存数据, 服务器会创建一个session对象为浏览器服务, 并且服务器会为session分配一个独一无二的编号, 就是session的id, 服务器在响应时会将session的id通过Cookie的形式发送给浏览器保存, 浏览器会将Cookie保存在内存中(即保存sessionid的Cookie是一个会话级别的Cookie), 这个Cookie会随着浏览器的关闭而销毁, 所以在浏览器关闭之前可以访问session, 关闭之后, 由于浏览器将session的id弄丢了, 所以服务器没办法根据id找到对应的Session。

解决方案:

我们可以向浏览器再发送一个名称为 JSESSIONID的Cookie, 值为session的id(session.getId()获取sessionid), 并且path为web应用的根路径, 并设置Cookie的最大生存时间为一个有效的时间, 这时服务器在响应时, 会发送两个保存sessionid的Cookie, 其中一个Cookie会保存在浏览器的临时文件夹中, 即使多次关闭浏览器, Cookie也不会销毁, session的id也不会销毁,服务器依然可以根据id找到对应的session

**代码示例：**

|  |
| --- |
| 在day12项目BuyServlet中，设置完session后添加如下代码：  //3.获取session，并将商品信息加入到购物车  HttpSession session = request.getSession();  session.setAttribute("prod", prod);    //设置保存sessionid的cookie的最大生存时间  Cookie cookie = **new** Cookie("JSESSIONID",session.getId());  cookie.setPath(request.getContextPath()+"/");  cookie.setMaxAge(60\*30);//设置生存时间为30分钟  response.addCookie(cookie);    response.getWriter().write("您已成功将["+prod+"]商品加入了购物车。"); |

### 13.3.6.Easymall验证码的校验

代码如下

|  |
| --- |
| 在EasyMall项目下，ValiImageServlet中最后添加如下代码：  //将图片保存到response缓冲区中，再响应给浏览器  vc.drawImage(response.getOutputStream());    //获取图片上的验证码  String code = vc.getCode();  //将验证码文本保存到session中，用于后期的校验  request.getSession().setAttribute("code", code); |
| 在RegistServlet中的验证码是否正确验证中加入如下代码：  //>>验证码是否正确  //从session中获取验证码内容  String code = (String) request.getSession().getAttribute("code");  **if** (!valistr.equalsIgnoreCase(code)) {  //将提示存入request域中，通过转发将消息带到regist.jsp中  request.setAttribute("msg", "验证码不正确！");  request.getRequestDispatcher("/regist.jsp").forward(request, response);  **return**;  } |

## 13.4.作业练习

1. 描述Cookie个数限制及大小？

答：各浏览器之间对cookie的不同限制：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | IE6.0 | IE7.0/8.0/9.0+ | Opera | FF | Safari | Chrome |
| cookie个数 | 每个域名为20个 | 每个域名为50个 | 每个域名为30个 | 每个域名为50个 | 没有个数限制 | 每个域名为53个 |
| cookie大小 | 4095字节 | 4095字节 | 4096字节 | 4097字节 | 4097字节 | 4097字节 |

总之，在进行页面cookie操作的时候，应该尽量保证cookie个数小于20个，总大小 小于4KB

1. cookie与session的区别？

答：cookie将数据保存在客户端，不会占用服务器资源，只能保存String类型数据，由于保存在本地，安全级别较低。

session将数据保存在服务器端，会占用服务器资源，可以保存任意类型数据，由于保存在服务器，安全级别高。

1. Session生命周期？

答：创建: 第一次调用request.getSession()方法时创建session对象。

销毁：

超时死亡: 默认如果session 30分钟不使用就会超时销毁!

自杀: 当调用session.invalidate()方法时会立即销毁session.

意外身亡: 当服务器意外关闭时(停电/宕机), session会立即销毁

1. session的原理？

答：浏览器向服务器发送请求, 将需要保存的数据带给服务器, 服务器通过请求获取数据, 接着去服务器内部检查有没有为当前浏览器服务的session, 如果有直接拿过来使用, 如果没有就创建一个session对象为当前浏览器服务. 将数据保存在session内部.

当浏览器再次访问服务器时, 服务器可以找到为当前浏览器服务的session, 从中取出数据, 通过这种方式也可以保存会话中产生的数据!