01-Cookie 和 Session

讲师：王振国

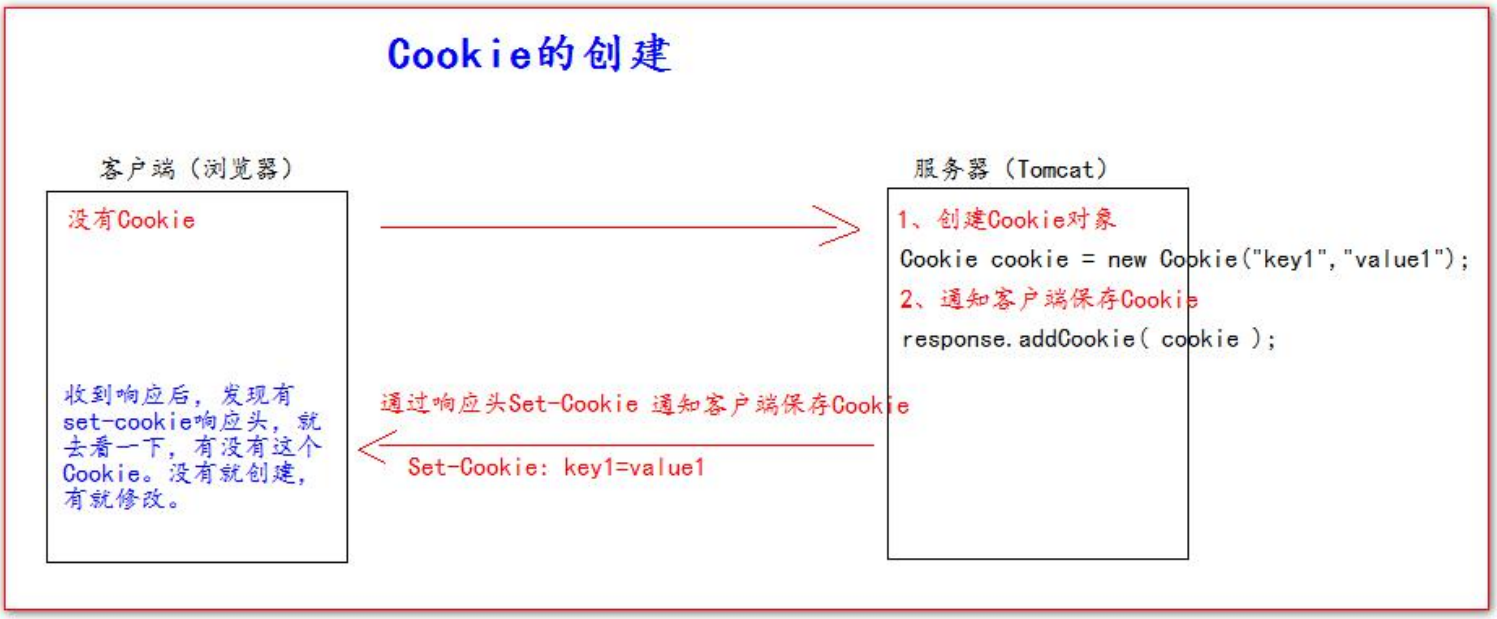
今日任务

1、Cookie 饼干

a)什么是 Cookie?

1、Cookie 翻译过来是饼干的意思。
2、Cookie 是服务器通知客户端保存键值对的一种技术。
3、客户端有了 Cookie 后，每次请求都发送给服务器。
4、每个 Cookie 的大小不能超过 4kb

b)如何创建 Cookie



Servlet 程序中的代码：

```
protected void createCookie(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    //1 创建 Cookie 对象
```

```
    Cookie cookie = new Cookie("key4", "value4");
    //2 通知客户端保存Cookie
    resp.addCookie(cookie);
    //1 创建Cookie 对象
    Cookie cookie1 = new Cookie("key5", "value5");
    //2 通知客户端保存Cookie
    resp.addCookie(cookie1);

    resp.getWriter().write("Cookie 创建成功");
}
```

c)服务器如何获取 Cookie

服务器获取客户端的 Cookie 只需要一行代码：req.getCookies():Cookie[]

Cookie 的工具类：

```
public class CookieUtils {
    /**
     * 查找指定名称的Cookie 对象
     * @param name
     * @param cookies
     * @return
     */
    public static Cookie findCookie(String name , Cookie[] cookies){
        if (name == null || cookies == null || cookies.length == 0) {
            return null;
        }

        for (Cookie cookie : cookies) {
            if (name.equals(cookie.getName())) {
                return cookie;
```

```
            }
        }

        return null;
    }

}
```

Servlet 程序中的代码：

```
    protected void getCookie(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {

        Cookie[] cookies = req.getCookies();

        for (Cookie cookie : cookies) {
            // getName 方法返回 Cookie 的 key（名）
            // getValue 方法返回 Cookie 的 value 值
            resp.getWriter().write("Cookie[" + cookie.getName() + "=" + cookie.getValue() + "] <br/>");
        }

        Cookie iWantCookie = CookieUtils.findCookie("key1", cookies);

//        for (Cookie cookie : cookies) {
//            if ("key2".equals(cookie.getName())) {
//                iWantCookie = cookie;
//                break;
//            }
//        }
        // 如果不等于 null，说明赋过值，也就是找到了需要的 Cookie
        if (iWantCookie != null) {
            resp.getWriter().write("找到了需要的 Cookie");
        }
    }
```

d)Cookie 值的修改

方案一：

1、先创建一个要修改的同名（指的就是 key）的 Cookie 对象

2、在构造器，同时赋于新的 Cookie 值。

3、调用 response.addCookie(Cookie);

```
//        方案一：
//        1、先创建一个要修改的同名的 Cookie 对象
//         2、在构造器，同时赋于新的 Cookie 值。
```

```
        Cookie cookie = new Cookie("key1","newValue1");
//        3、调用 response.addCookie( Cookie ); 通知 客户端 保存修改
        resp.addCookie(cookie);
```

方案二：

1、先查找到需要修改的 Cookie 对象

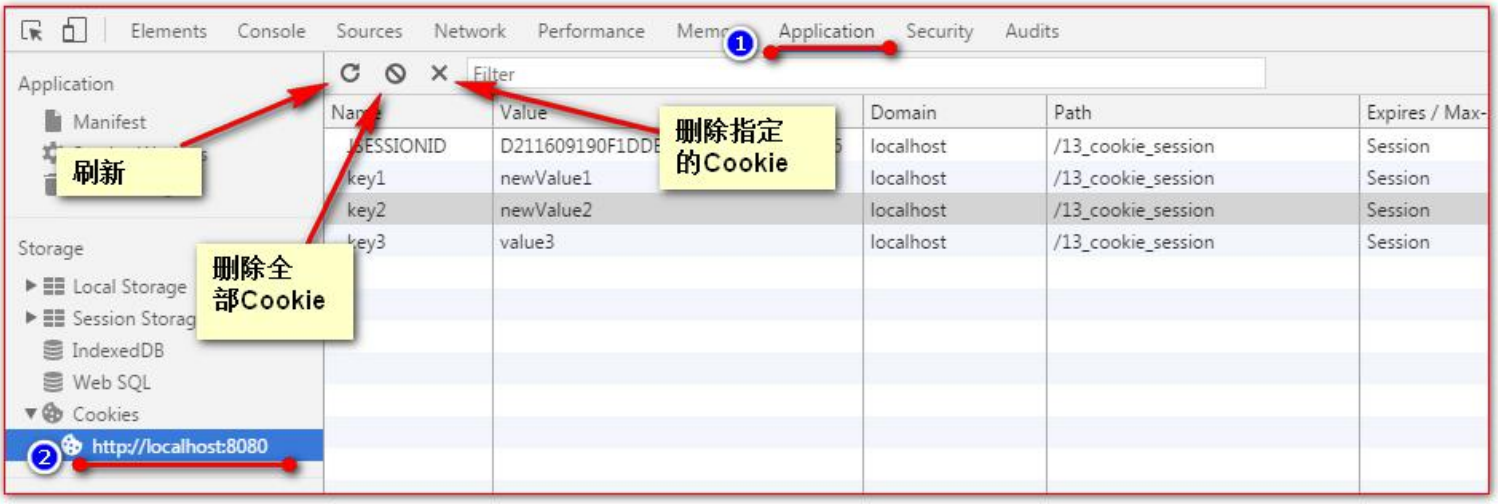
2、调用 setValue()方法赋于新的 Cookie 值。

3、调用 response.addCookie()通知客户端保存修改

```
//        方案二：
//        1、先查找到需要修改的 Cookie 对象
        Cookie cookie = CookieUtils.findCookie("key2", req.getCookies());
        if (cookie != null) {
//            2、调用 setValue()方法赋于新的 Cookie 值。
            cookie.setValue("newValue2");
//        3、调用 response.addCookie()通知客户端保存修改
            resp.addCookie(cookie);
        }
```

e)浏览器查看 Cookie：

谷歌浏览器如何查看 Cookie：



火狐浏览器如何查看 **Cookie**：

f) Cookie 生命控制

Cookie 的生命控制指的是如何管理 Cookie 什么时候被销毁（删除）

setMaxAge()

正数，表示在指定的秒数后过期

负数，表示浏览器一关，Cookie 就会被删除（默认值是-1）

零，表示马上删除 Cookie

```
/**
 * 设置存活1个小时的Cooie
 * @param req
 * @param resp
 * @throws ServletException
 * @throws IOException
 */
protected void life3600(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {

    Cookie cookie = new Cookie("life3600", "life3600");
    cookie.setMaxAge(60 * 60); // 设置Cookie一小时之后被删除。无效
    resp.addCookie(cookie);
    resp.getWriter().write("已经创建了一个存活一小时的 Cookie");

}

/**
 * 马上删除一个Cookie
 * @param req
 * @param resp
 * @throws ServletException
 * @throws IOException
 */
protected void deleteNow(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
```

```
    // 先找到你要删除的 Cookie 对象
    Cookie cookie = CookieUtils.findCookie("key4", req.getCookies());
    if (cookie != null) {
        // 调用 setMaxAge(0);
        cookie.setMaxAge(0); // 表示马上删除，都不需要等待浏览器关闭
        // 调用 response.addCookie(cookie);
        resp.addCookie(cookie);

        resp.getWriter().write("key4 的 Cookie 已经被删除");
    }

}

/**
 * 默认的会话级别的 Cookie
 * @param req
 * @param resp
 * @throws ServletException
 * @throws IOException
 */
protected void defaultLife(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    Cookie cookie = new Cookie("defalutLife","defaultLife");
    cookie.setMaxAge(-1);//设置存活时间
    resp.addCookie(cookie);
}
```

g)Cookie 有效路径 Path 的设置

Cookie 的 path 属性可以有效的过滤哪些 Cookie 可以发送给服务器。哪些不发。
path 属性是通过请求的地址来进行有效的过滤。

CookieA　　path=/工程路径
CookieB　　path=/工程路径/abc

请求地址如下：

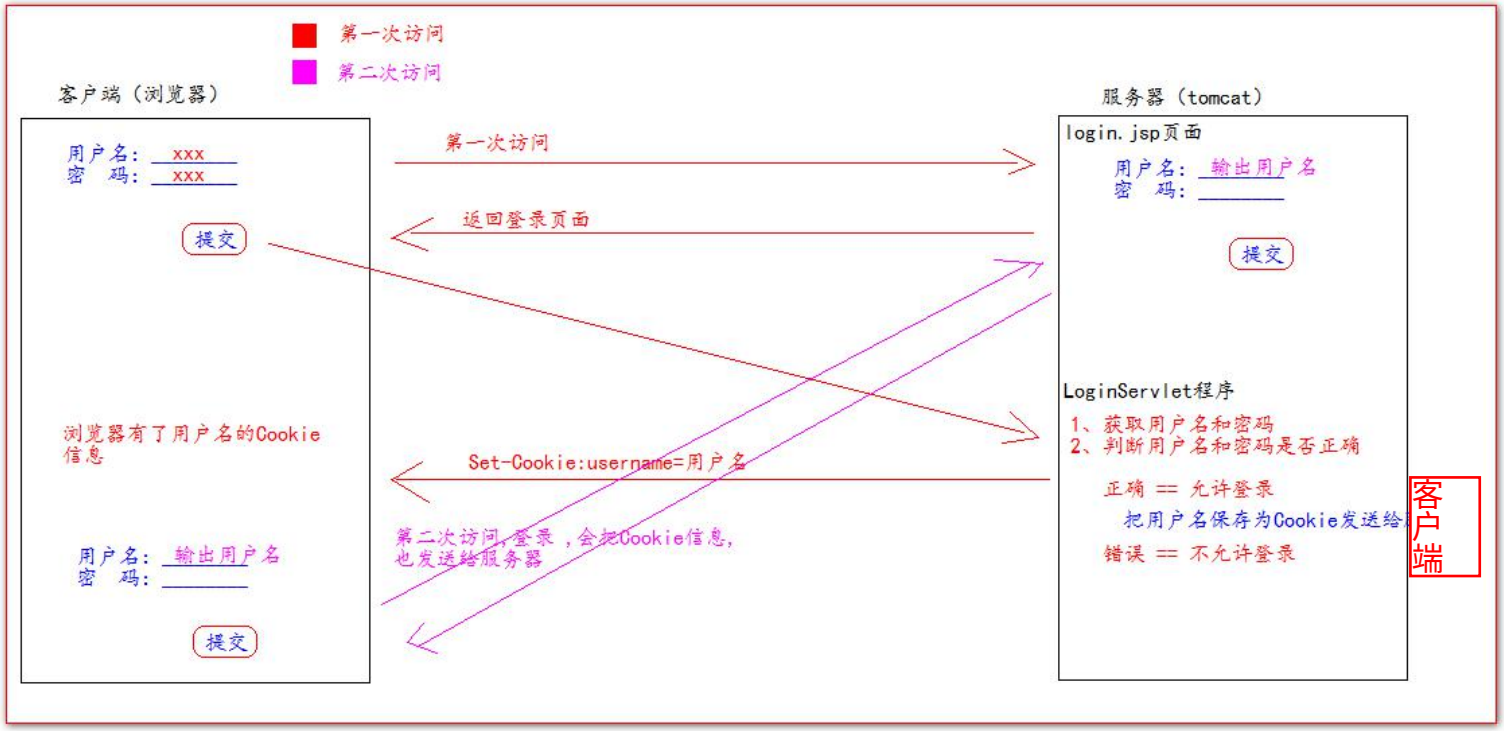
http://ip:port/工程路径/a.html
　　CookieA 发送
　　CookieB 不发送

http://ip:port/工程路径/abc/a.html
　　CookieA 发送
　　CookieB 发送

```
protected void testPath(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
```

```
    Cookie cookie = new Cookie("path1", "path1");
    // getContextPath() ===>>>>  得到工程路径
    cookie.setPath( req.getContextPath() + "/abc" ); // ===>>>>  /工程路径/abc
    resp.addCookie(cookie);
    resp.getWriter().write("创建了一个带有 Path 路径的 Cookie");
}
```

h) Cookie 练习---免输入用户名登录



login.jsp 页面

```
<form action="http://localhost:8080/13_cookie_session/loginServlet" method="get">
    用户名：<input type="text" name="username" value="${cookie.username.value}"> <br>
    密码：<input type="password" name="password"> <br>
    <input type="submit" value="登录">
</form>
```

LoginServlet 程序：

```
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");
```

```
        if ("wzg168".equals(username) && "123456".equals(password)) {
            //登录 成功
            Cookie cookie = new Cookie("username", username);
            cookie.setMaxAge(60 * 60 * 24 * 7);//当前 Cookie 一周内有效
            resp.addCookie(cookie);
            System.out.println("登录 成功");
        } else {
//            登录 失败
            System.out.println("登录 失败");
        }

    }
```

2、Session 会话

i) 什么是 Session 会话?

1、Session 就一个接口（HttpSession）。
2、Session 就是会话。它是用来维护一个客户端和服务器之间关联的一种技术。
3、每个客户端都有自己的一个 Session 会话。
4、Session 会话中，我们经常用来保存用户登录之后的信息。

j) 如何创建 Session 和获取(id 号,是否为新)

如何创建和获取 Session。它们的 API 是一样的。
request.getSession()
    第一次调用是：创建 Session 会话
    之后调用都是：获取前面创建好的 Session 会话对象。

isNew(); 判断到底是不是刚创建出来的（新的）
    true 表示刚创建
    false 表示获取之前创建

每个会话都有一个身份证号。也就是 ID 值。而且这个 ID 是唯一的。
getId() 得到 Session 的会话 id 值。

k)Session 域数据的存取

```
/**
 * 往 Session 中保存数据
 * @param req
```

```
 * @param resp
 * @throws ServletException
 * @throws IOException
 */
protected void setAttribute(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    req.getSession().setAttribute("key1", "value1");
    resp.getWriter().write("已经往 Session 中保存了数据");
}

/**
 * 获取 Session 域中的数据
 * @param req
 * @param resp
 * @throws ServletException
 * @throws IOException
 */
protected void getAttribute(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    Object attribute = req.getSession().getAttribute("key1");
    resp.getWriter().write("从 Session 中获取出 key1 的数据是：" + attribute);
}
```

I) Session 生命周期控制

public void setMaxInactiveInterval(int interval)　设置 Session 的超时时间（以秒为单位），超过指定的时长，Session 就会被销毁。

值为正数的时候，设定 Session 的超时时长。

负数表示永不超时（极少使用）

public int getMaxInactiveInterval()获取 Session 的超时时间

public void invalidate()　让当前 Session 会话马上超时无效。

Session 默认的超时时长是多少！

Session 默认的超时时间长为 30 分钟。

因为在 Tomcat 服务器的配置文件 web.xml 中默认有以下的配置，它就表示配置了当前 Tomcat 服务器下所有的 Session 超时配置默认时长为：30 分钟。

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

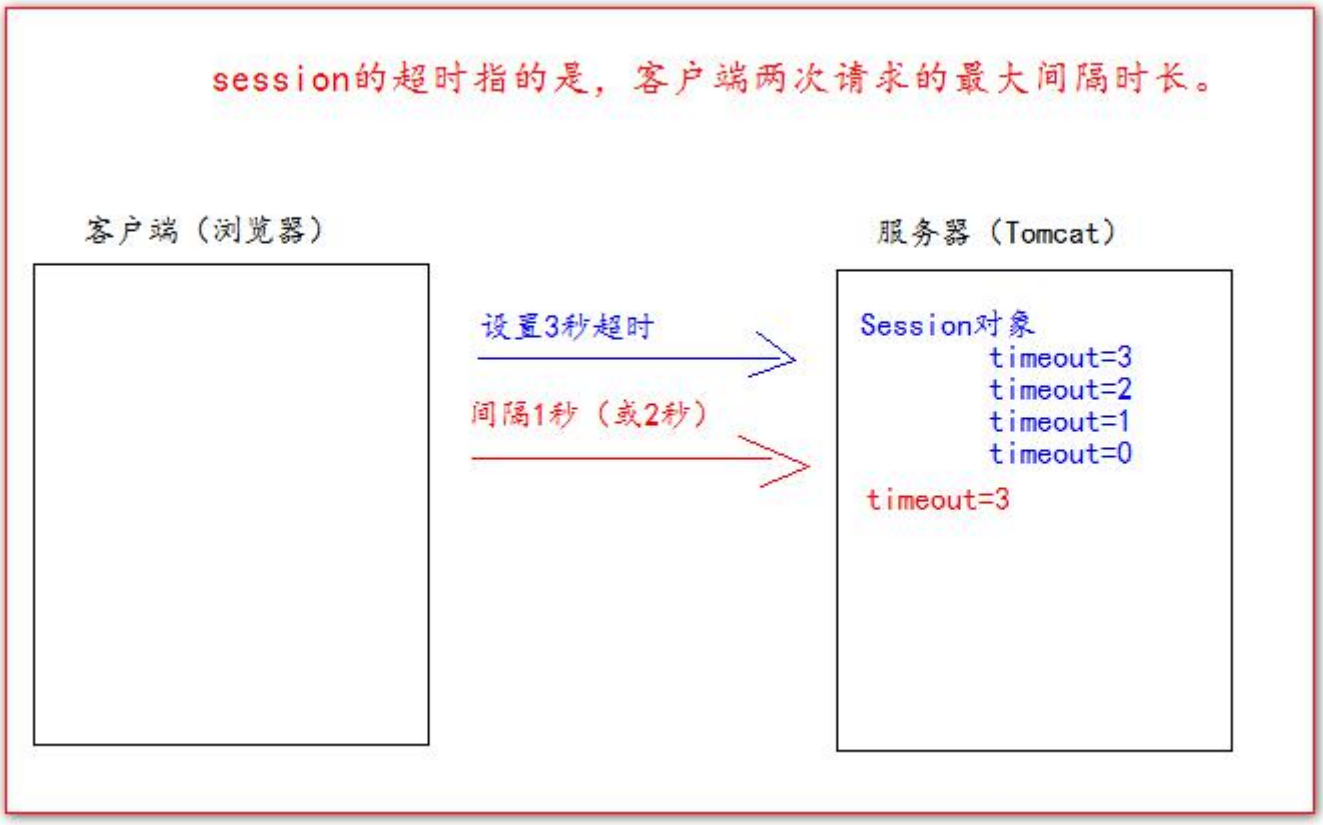
如果说。你希望你的 web 工程，默认的 Session 的超时时长为其他时长。你可以在你自己的 web.xml 配置文件中做以上相同的配置。就可以修改你的 web 工程所有 Seession 的默认超时时长。

```
<!--表示当前web 工程。创建出来 的所有Session 默认是 20 分钟 超时时长-->
<session-config>
    <session-timeout>20</session-timeout>
</session-config>
```

如果你想只修改个别 Session 的超时时长。就可以使用上面的 API。setMaxInactiveInterval(int interval)来进行单独的设置。

session.setMaxInactiveInterval(int interval)单独设置超时时长。

Session 超时的概念介绍：



示例代码：

```
protected void life3(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    // 先获取Session 对象
    HttpSession session = req.getSession();
    // 设置当前Session3 秒后超时
    session.setMaxInactiveInterval(3);

    resp.getWriter().write("当前 Session 已经设置为 3 秒后超时");
}
```

Session 马上被超时示例：

```
protected void deleteNow(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    // 先获取 Session 对象
    HttpSession session = req.getSession();
    // 让 Session 会话马上超时
    session.invalidate();

    resp.getWriter().write("Session 已经设置为超时（无效）");
}
```

m) 浏览器和 Session 之间关联的技术内幕

Session 技术，底层其实是基于 Cookie 技术来实现的。

3、项目第六阶段

3.1、登陆---显示用户名

UserServlet 程序中保存用户登录的信息

```
} else {
    // 登录 成功
    // 保存用户登录的信息到Session域中
    req.getSession().setAttribute( s: "user", loginUser);
    //跳到成功页面login_success.html
    req.getRequestDispatcher( s: "/pages/user/login_success.jsp").forward(req, resp);
}
```

修改 login_succuess_menu.jsp

```
<div>
    <span>欢迎<span class="um_span">${sessionScope.user.username}</span>光临尚硅谷书城</span>
    <a href="pages/order/order.jsp">我的订单</a>
    <a href="index.jsp">注销</a>
    <a href="index.jsp">返回</a>
</div>
```

还要修改首页 index.jsp 页面的菜单 ：

```
<div>
    <%--如果用户还没有登录，显示      【登录 和注册的菜单】 --%>
    <c:if test="${empty sessionScope.user}">
        <a href="pages/user/login.jsp">登录</a> |
        <a href="pages/user/regist.jsp">注册</a>
    </c:if>
    <%--如果已经登录，则显示 登录 成功之后的用户信息。--%>
    <c:if test="${not empty sessionScope.user}">
        <span>欢迎<span class="um_span">${sessionScope.user.username}</span>光临尚硅谷书城</span>
        <a href="pages/order/order.jsp">我的订单</a>
        <a href="index.jsp">注销</a>
    </c:if>
```

3.2、登出---注销用户

1、销毁 Session 中用户登录的信息（或者销毁 Session）
2、重定向到首页（或登录页面）。

UserServlet 程序中添加 logout 方法

```
    /**
     * 注销
     * @param req
     * @param resp
     * @throws ServletException
     * @throws IOException
     */
    protected void logout(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
//        1、销毁 Session 中用户登录的信息（或者销毁 Session）
        req.getSession().invalidate();
//        2、重定向到首页（或登录页面）。
        resp.sendRedirect(req.getContextPath());
    }
```

修改【注销】的菜单地址

```
<a href="userServlet?action=logout">注销</a>
```

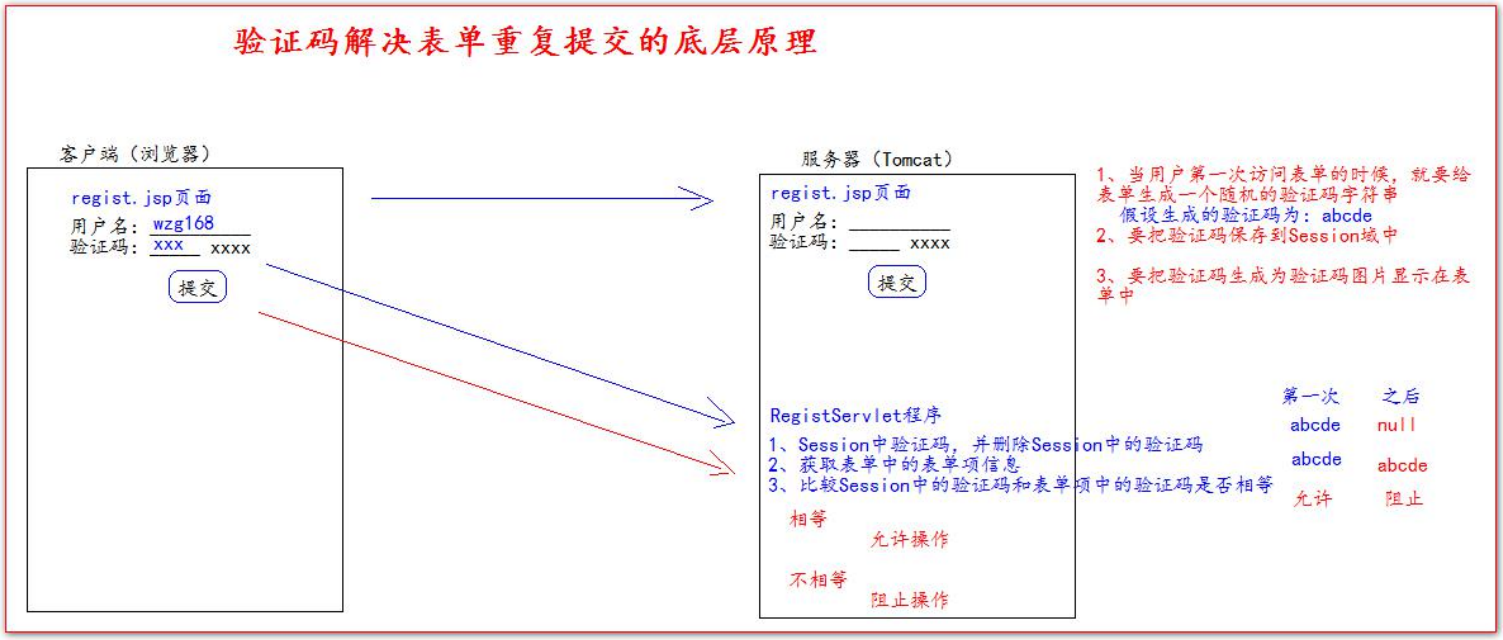
3.3、表单重复提交之-----验证码

表单重复提交有三种常见的情况：

一：提交完表单。服务器使用请求转来进行页面跳转。这个时候，用户按下功能键 F5，就会发起最后一次的请求。造成表单重复提交问题。解决方法：使用重定向来进行跳转

二：用户正常提交服务器，但是由于网络延迟等原因，迟迟未收到服务器的响应，这个时候，用户以为提交失败，就会着急，然后多点了几次提交操作，也会造成表单重复提交。

三：用户正常提交服务器。服务器也没有延迟，但是提交完成后，用户回退浏览器。重新提交。也会造成表单重复提交。



3.4、谷歌 kaptcha 图片验证码的使用

谷歌验证码 kaptcha 使用步骤如下：

1、导入谷歌验证码的 jar 包

kaptcha-2.3.2.jar

2、在 web.xml 中去配置用于生成验证码的 Servlet 程序

```
<servlet>
    <servlet-name>KaptchaServlet</servlet-name>
    <servlet-class>com.google.code.kaptcha.servlet.KaptchaServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>KaptchaServlet</servlet-name>
    <url-pattern>/kaptcha.jpg</url-pattern>
</servlet-mapping>
```

3、在表单中使用 img 标签去显示验证码图片并使用它

```
<form action="http://localhost:8080/tmp/registServlet" method="get">
  用户名：<input type="text" name="username" > <br>
  验证码：<input type="text" style="width: 80px;" name="code">
  <img src="http://localhost:8080/tmp/kaptcha.jpg" alt="" style="width: 100px; height: 28px;"> <br>
  <input type="submit" value="登录">
</form>
```

4、在服务器获取谷歌生成的验证码和客户端发送过来的验证码比较使用。

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    // 获取 Session 中的验证码
    String token = (String) req.getSession().getAttribute(KAPTCHA_SESSION_KEY);
    // 删除 Session 中的验证码
    req.getSession().removeAttribute(KAPTCHA_SESSION_KEY);


    String code = req.getParameter("code");
    // 获取用户名
    String username = req.getParameter("username");

    if (token != null && token.equalsIgnoreCase(code)) {
        System.out.println("保存到数据库：" + username);
        resp.sendRedirect(req.getContextPath() + "/ok.jsp");
    } else {
        System.out.println("请不要重复提交表单");
    }
}
```

切换验证码：

```
// 给验证码的图片，绑定单击事件
$("#code_img").click(function () {
   // 在事件响应的 function 函数中有一个 this 对象。这个 this 对象，是当前正在响应事件的 dom 对象
   // src 属性表示验证码 img 标签的 图片路径。它可读，可写
   // alert(this.src);
   this.src = "${basePath}kaptcha.jpg?d=" + new Date();
});
```