

java web

张海宁¹

May 2, 2018

¹hnzhang1@gzu.edu.cn

Contents

1	cookie and session	1
1.1	cookie	1
1.1.1	cookie 的常用方法	3
1.2	session	3
1.2.1	session 的常用方法	4
1.3	cookie 和 session 的生命周期	4
1.4	应用	4
1.5	Other	5
2	servlet	7
2.1	servlet	7
2.1.1	jsp 页面	7
2.1.2	servlet 代码	8
2.2	filter	8
2.2.1	建立 filter	8
2.2.2	部署 filter	10
2.3	listener	10
2.3.1	新建 listener	10
2.3.2	部署 listener	14
3	Login	17
3.1	目的	17
3.2	目标	17
3.3	过程	17
3.3.1	使用 Session 实现登陆	17
3.3.2	Cookie 和 Session 配合实现登陆	20

Chapter 1

cookie and session

在计算机领域，如果一个程序如果能够记忆用户的操作历史或者用户的交互历史，那么这个程序就是有状态的。这些被记住的信息就称为系统的状态。

Http 协议是无状态 的，每次的客户端请求都会被当做一个新的请求，服务端不会记住你曾经访问过，那么就无法追踪用户¹。为了实现有状态的通信，人们提出了 cookie 和 session 技术²。这两种技术都是用来保存与用户有关的信息的。

1.1 cookie

Cookie³是由服务器端生成并发送给客户端保存的一小段文本数据。Cookie 中不能包含空格，值的存储形式是一系列的“属性-值”。

以下为两次请求同一个 jsp 网站，抓取到的相关 http 请求信息。可以看到：

1. 第一次请求的 request header 中并无 cookie 相关信息，但是 response header 中有一个 set-cookie，也就是说，服务器看到请求中没有 cookie，那么就自动创建了一个 cookie 给客户端（这个和具体的 web 容器有关）；
2. 第二次请求的时候，request header 直接带上了这个 cookie 去访问服务器，服务器看到这个 cookie 就知道是这个特定的用户又来访问了。

第一次访问网站：

General

Request URL: http://localhost:8080/lab-java/

Request Method: GET

Status Code: 200

Remote Address: [::1]:8080

Referrer Policy: no-referrer-when-downgrade

¹一个比较典型的场景就是电商网站的购物车

²<https://tools.ietf.org/html/rfc6265>

³https://en.wikipedia.org/wiki/HTTP_cookie

Response Headers

HTTP/1.1 200

Set-Cookie: JSESSIONID=8DF2DE79C1951FF63FAC3B89D47FD44F; Path=/lab-java; HttpOnly

Content-Type: text/html; charset=UTF-8

Content-Length: 1300

Date: Fri, 20 Apr 2018 10:06:57 GMT

Request Headers

GET /lab-java/ HTTP/1.1

Host: localhost:8080

Connection: keep-alive

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.117 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8

DNT: 1

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7

第二次访问网站:**General**

Request URL: http://localhost:8080/lab-java/

Request Method: GET

Status Code: 200

Remote Address: [::1]:8080

Referrer Policy: no-referrer-when-downgrade

Response Headers

HTTP/1.1 200

Content-Type: text/html; charset=UTF-8

Content-Length: 1300

Date: Fri, 20 Apr 2018 10:14:19 GMT

Request Headers

GET /lab-java/ HTTP/1.1

Host: localhost:8080

Connection: keep-alive

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.117 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8

DNT: 1

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7

Cookie: JSESSIONID=8DF2DE79C1951FF63FAC3B89D47FD44F

1.1.1 cookie 的常用方法

1. 获取 cookie

```
Cookie[] ck = request.getCookies();
```

一个服务器可能会向一个客户端设置若干个 cookie

2. 获取某个 cookie 对象的名字和值

```
ckName=ck[i].getName(); ckValue=ck[i].getValue();
```

3. 向客户端添加 cookie

(a) 新建 cookie

```
Cookie my2ndCookie = new Cookie("username","unknown.yes#or#no?");
```

(b) 向客户端发送 cookie

```
response.addCookie(my2ndCookie);
```

1.2 session

一个客户端与服务器端之间的通讯过程称为一次会话 (session)。

Session⁴是保存在服务器端的，是一个容器，其中保存的是一对一的数据 (属性-值)。在 jsp 中，每当一个客户端的请求到来时，服务器端都会检查其请求中是否包含 cookie，若有，则从 cookie 中读取数据 (JSESSIONID)；若无，则为这一个用户端与服务器端的会话过程创建一个 session。这个 session 会在用户关闭了这个会话之后就消失了，下一次这个用户再来的时候，服务器会把他当做一个全新用户来对待。如何让服务器记住某个特定的用户呢？方法有若干种，这里介绍其中一种最常用的。

在1.1部分介绍了客户端第一次访问服务器的时候，会得到一个服务器端生成的 cookie，这个 cookie 的名字是 JSESSIONID，值是一串十六进制数。这个 cookie 是用来保存用户 session 信息的，那个 JSESSIONID 就是服务器为那次会话生成的一个标识符，通常叫做 session id。客户端在这之后再访问服务器的话，就会带上这个 cookie 来访问，服务器看到这个 cookie 里的 session id 就会知道某个特定的用户来了。cookie 和 session 的这种配合使用，就使得保存在客户端的信息和保存在服务器端的信息联系起来了。

⁴[https://en.wikipedia.org/wiki/Session_\(computer_science\)](https://en.wikipedia.org/wiki/Session_(computer_science))

Method	Description
<code>int getMaxAge()</code>	Returns the maximum age of the cookie, specified in seconds. By default, -1 indicating the cookie will persist until browser shutdown.
<code>void setMaxAge(int expiry)</code>	Sets the maximum age of the cookie in seconds. an integer specifying the maximum age of the cookie in seconds; if negative, means the cookie is not stored; if zero, deletes the cookie.

Table 1.1: cookie 时效相关的方法

1.2.1 session 的常用方法

1. 获取 session

session 是 servlet 的一个内置对象，可以像 request 一样直接使用，一个客户端和一个服务器在一段时间内只拥有一个 session

2. 往 session 里写数据

```
session.setAttribute("id", id);
```

注意是一对一的写，与 cookie 类似

3. 读 session 里的数据

```
session.getAttribute("id");
```

4. 销毁 session

```
session.invalidate();
```

注销登陆

1.3 cookie 和 session 的生命周期

cookie 和 session 是有其生命周期的。

- 有些网站会提供一个记住密码，多长时间不用再重新登陆的功能，使用的方法就是设置 cookie 的过期时间；
- 有时候停留在某个页面长时间不进行操作的话，再去操作就可能会被提示登陆超时，请重新登陆，这个功能就很可能是 session 超时了。

cookie 和 session 与时效相关的一些方法如 Table1.1和 Table1.2所示。

1.4 应用

1. 使用 cookie 和 session 实现登陆功能

Method	Description
<code>long getCreationTime()</code>	Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
<code>long getLastAccessedTime()</code>	Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.
<code>int getMaxInactiveInterval()</code>	Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
<code>void setMaxInactiveInterval(int interval)</code>	Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. A negative time indicates the session should never timeout.

Table 1.2: session 时效相关的一些方法

2. 使用 filter 和 session 实现统计在线人数的功能

1.5 Other

分享 stanford 的一个关于网站方面的资源, <https://crypto.stanford.edu/cs142/lectures/>; 还有 cookie 和 session 方面的<https://web.stanford.edu/~ouster/cgi-bin/cs142-fall10/lecture.php?topic=cookie>。

Chapter 2

servlet

servlet 是运行在 web 容器中的 java 程序，很适合用来进行处理 web 应用中的业务逻辑。

通过 servlet 和 jsp 的配合使用，可以使其各负其责：

- jsp
负责显示页面
- servlet
负责处理业务逻辑

jsp 和 servlet 的使用场景就是：jsp 负责页面显示、与用户交互，jsp 页面中一些需要处理的数据就送到 servlet 来处理。

2.1 servlet

通过编写一个负责处理注册信息的 servlet 来展示 servlet 的使用方法。

在 eclipse 中，右键当前的项目，选择新建，找到 servlet，即可建立一个 servlet(这是一个 java 类)。

建立完成后，打开刚才建立的 servlet，可以看到其中会有一些 eclipse 自动填写的代码，我们关注的是其中的两个方法 doGet 和 doPost。一般来说，servlet 是用来处理 jsp 页面传递过来的数据，而传递数据的方式通常情况下是通过表单来进行的，表单中的数据提交通常使用的是 post 方法。本部分涉及到两个文件：teacher.jsp 和 Reg.java，其中 teacher.jsp 负责页面的显示以及数据的收集，Reg.java 负责处理 teacher.jsp 页面传递过来的数据。

2.1.1 jsp 页面

teacher.jsp 页面中的关键代码如列表2.1所示。从列表2.1中可以看出，当前 form 的 action 是指向的 teacher_add.jsp 页面，这是在学习 sevlet 之前的数据处理方式，从现在开始需要把 action 的值改为新建的 servlet，即：action="Reg"。

```
<form action="teacher_add.jsp" method="post">
<table>
<tr>
  <td>姓名: </td>
  <td><input type="text" name="nm"></td>
</tr>
<tr>
  <td>职工号: </td>
  <td><input type="number" name="staffid" min=20060001 ></td>
</tr>
<tr>
  <td><input type="submit"></td>
  <td><input type="reset"></td>
</tr>
</table>
</form>
```

Figure 2.1: teacher.jsp 中的关键代码

2.1.2 servlet 代码

因为 teacher.jsp 页面中 form 的数据提交方式为 post, 所以我们要完成 Reg.java 中的 doPost() 方法。Reg.java 中的关键代码如列表2.2所示。

2.2 filter

filter 是 servlet 规范中定义的一种特殊类。filter 可以理解成介于客户端和目标资源之间的一个过滤器, 即它会对客户端的请示进行过滤后才可以到达服务器上的目标资源, 或者访问到目标资源后, 对服务器端产生的响应进行处理后才送回客户端 (这两个活动可以在一个过滤器中同时进行, 即双向过滤)。filter 的示意图如 Figure 2.3所示。在用户注册的时候, 如果想禁止某个姓名被使用, 可以通过部署一个 filter 介于注册页面和业务逻辑处理模块之间。接下来的例子中, 注册页面为 teacher.jsp, 业务逻辑处理模块为 Reg.java (请注意这是一个 **servlet**)。

2.2.1 建立 filter

与建立一个 servlet 的方法类似, 可以建立一个 filter, 将其命名为 FilterReg.java。可以看到这个新建的 java 文件中, 有若干方法, 其中的 doFilter 方法是我们需要关注的, 因为过滤功能就是在此处实现的。Filter 文件 FilterReg.java 的代码如 Figure 2.4所示。Figure 2.4中的代码会判断用户提交的姓名, 如果姓名为

```

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // 设置request 传递过来值的编码，并获取传递值
    request.setCharacterEncoding("utf-8");

    String id = request.getParameter("staffid");
    String name = request.getParameter("nm");

    //get current date and time
    //LabDate ld = new LabDate();
    //String time = ld.getDtTm();

    //write to database
    //String[] fields= {"id","name","logDate"};
    //String[] values= new String[3];
    //values[0]=id;
    //values[1]=name;
    //values[2]=time;
    //Db db = new Db();
    //int i = db.writeDb("teachers", fields, values);
    //db.getClose();
    //String rz="";
    //if(i==1) {
    //    rz = "done! Will return to the former page in 3 seconds.";
    //} else {
    //    rz = "Something wrong! Will return in 3 seconds.";
    //}
    //response.getWriter().print(rz);
    //response.setHeader("refresh","3,URL=teacher.jsp");
}

```

Figure 2.2: Reg.java 中的关键代码

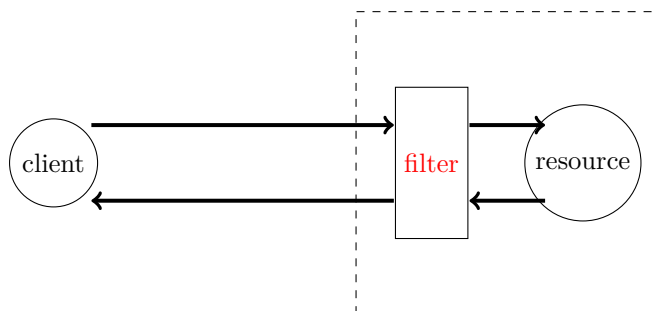


Figure 2.3: the position and function of a filter

“zq”，则会截断请求，使得用户请求不能到达 Reg 这个 servlet，从而不能完成注册，达到了禁止特定用户名注册的目的。

2.2.2 部署 filter

在上一小节中，讲述了如何建立一个 filter，其实要使 filter 发挥作用还必须进行正确的配置。从 servlet 3.0 开始就不需要在 web.xml 文件中进行配置了，只需要在 servlet 和 filter 的源代码文件中进行声明即可，如 Figure 2.5和 Figure 2.6所示。

特别要注意的是 filter 和 servlet 里声明的 `urlPatterns` 需要保持一致：filter 里的 `@WebFilter(filterName = "/FilterReg",urlPatterns = "/Reg")` 和 servlet 里的 `@WebServlet(name="reg",urlPatterns= "/Reg")`。

2.3 listener

listener 是 servlet 规范中定义的一种特殊类。listener 是监听器，其作用就是用来监听 servlet 容器中一些事件 (event) 的发生。从大的分类上来说，listener 可以监听以下三个对象的事件：

1. ServletContext
2. HttpSession
3. ServletRequest

listener 和 event 的对应关系，如 Table 2.1所示。

监听器通常可以被用来统计在线人数。接下来以此为例展示监听器的使用方法。

2.3.1 新建 listener

与 servlet 和 filter 的新建方式一样，新建一个 listener。这里需要注意的是，在下一步选择监听事件的时候，选择 `HttpSessionEvent` 里面的 `lifecycle`，如 Figure 2.7所示，因为这里统计在线人数是通过统计当前活动的 session 来计数的。

```

public void doFilter(ServletRequest request ,
    ServletResponse response , FilterChain chain)
    throws IOException , ServletException {
    // TODO Auto-generated method stub
    // place your code here
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse resp = (HttpServletResponse) response;
    String id = request.getParameter("staffid");
    String name = request.getParameter("nm");
    //System.out.println("staffid is: "+id);
    System.out.println("filter says: the name is "+name);
    if(name.equals("zq")) {
        resp.getWriter().println("The name "+name+" is forbidden!");
    }else {
        // pass the request along the filter chain
        chain.doFilter(request , response);
        resp.setHeader("refresh", "3,URL=teacher.jsp");
    }
}

```

Figure 2.4: FilterReg.java 中的关键代码

```

package cs;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Reg
 */
@WebServlet(name="reg",urlPatterns= {"/Reg"})
public class Reg extends HttpServlet {

```

Figure 2.5: servlet 中的声明 @WebServlet

```

package filter;

import java.io.IOException;
import javax.servlet.DispatcherType;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet Filter implementation class FilterReg
 */
@WebFilter(filterName = "/FilterReg", urlPatterns = {"/Reg"})

```

Figure 2.6: filter 中的声明 @WebFilter

Listener 接口	Event 类
ServletContextListener	ServletContextEvent
ServletContextAttributeListener	ServletContextAttributeEvent
HttpSessionListener	HttpSeesionEvent
HttpSessionActivationListener	
HttpSessionAttributeListener	HttpSessionBindingEvent
HttpSessionBindingListener	
ServletRequestListener	ServletRequestEvent
ServletRequestAttributeListener	ServletRequestAttributeEvent

Table 2.1: Listener 接口与 Event 类

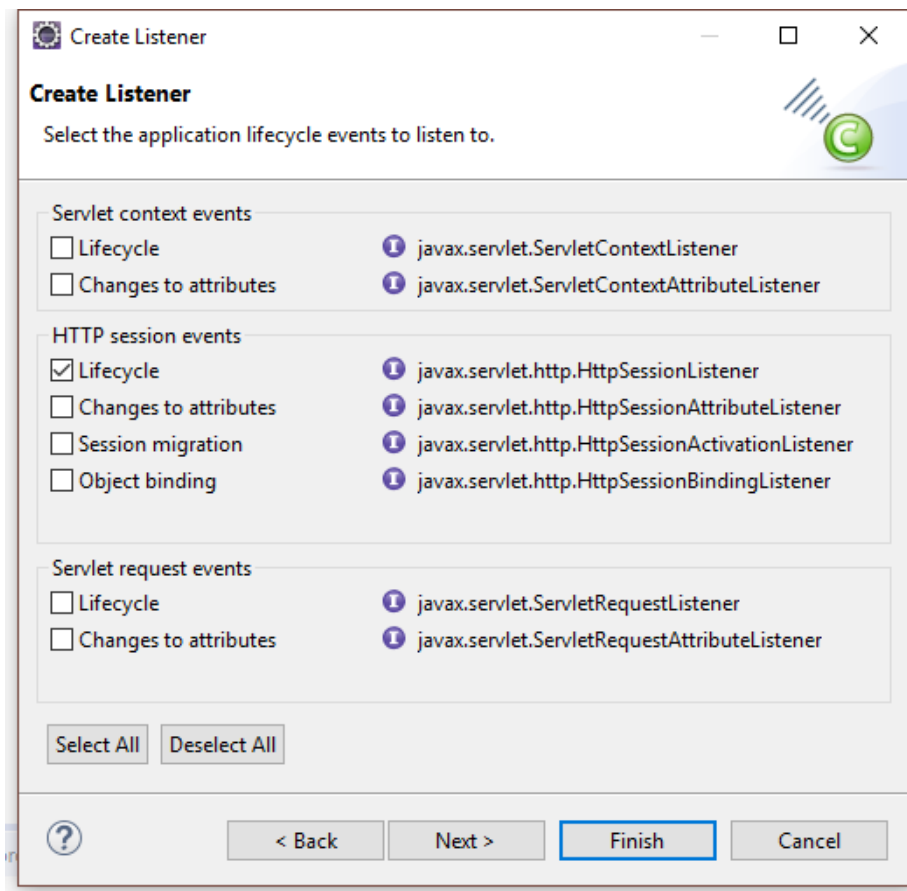


Figure 2.7: 选择 listener 监听事件

其代码如 Figure 2.8所示。

2.3.2 部署 listener

listener 的部署很简单，不需要另外配置，在创建 listener 的时候 eclipse 会自动帮助创建一条注释：@WebListener，这条注释就完成了 listener 的部署。

通常，每个 http 请求都会与一个 session 关联在一起。如果该请求没有与之相关联的 session，那么服务器会为其创建一个 session，此时会触发 sessionCreated 事件；该 session 会在用户不再活动的一定时间之后失效，或者用户主动注销（调用 session.invalidate() 方法），此时会触发 sessionDestroyed 事件，这两个事件会被监听器监听到。

```

//OnlineNum.java
@WebListener
public class OnlineNum implements HttpSessionListener {
    private int count;
    public int getNum() {
        return count;
    }
    public synchronized void setCount(int c) {
        count+=c;
    }

    public OnlineNum() {
        // TODO Auto-generated constructor stub
        count=0;
        System.out.println("当前在线人数被初始化为0");
    }

    /**
     * @see HttpSessionListener#sessionCreated(HttpSessionEvent)
     */
    public void sessionCreated(HttpSessionEvent arg0) {
        // TODO Auto-generated method stub
        String dt = new LabDate().getDtTm();
        String sid = arg0.getSession().getId();
        setCount(1);
        System.out.println("at "+dt+","+sid+" coming, 当前在线人数为: "+count);
    }

    /**
     * @see HttpSessionListener#sessionDestroyed(HttpSessionEvent)
     */
    public void sessionDestroyed(HttpSessionEvent arg0)
    {
        // TODO Auto-generated method stub
        setCount(-1);
        String dt = new LabDate().getDtTm();
        String sid = arg0.getSession().getId();
        System.out.println("at "+dt+","+sid+" leave, 当前在线人数为: "+count);
    }
}

```

Figure 2.8: listener 代码

Chapter 3

Login

3.1 目的

练习 cookie 和 session 的使用，加深对它们的理解。

3.2 目标

使用 cookie 和 session 为项目增加登陆功能。

3.3 过程

3.3.1 使用 Session 实现登陆

1. 前台界面

如 Figure 3.1所示，此前台界面位于 content_left.jsp 这个 div 内。

2. 前台代码

如 Figure 3.2所示。注意这里使用了 *EL* 和 *JSTL* 表达式。

3. 控制登陆的 servlet

其代码如 Figure 3.3所示。

4. 注销

用户登陆之后要为其提供主动注销的功能，因为这里的登陆是通过 session 来实现的，所以要达到注销的目的，只需要使 session 无效即可。可以设置一个按钮，点下此按钮会提交一个 post 请求到 logout.jsp 页面，在 logout.jsp 页面中调用 session.invalidate(), 使 session 变得无效。

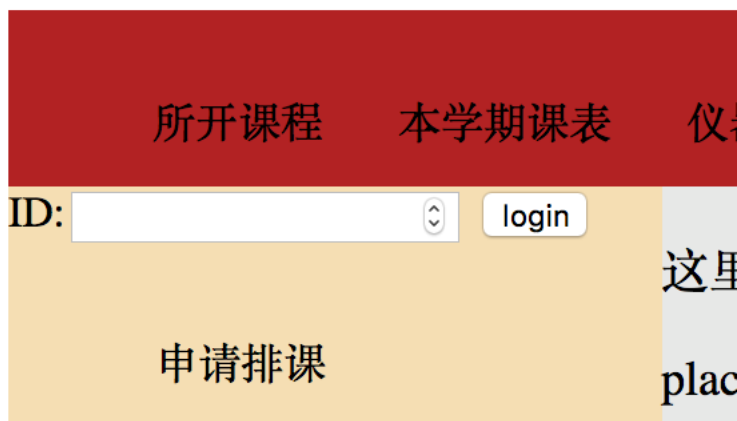


Figure 3.1: Login 界面

```

<!--content_left.jsp-->
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<div class="content left">
<c:choose>
<c:when test="${not empty sessionScope.userName }">
你好, ${sessionScope.userName }。
<form action="logout.jsp" method="post">
<input type="submit" value="logout"></form>
</c:when>
<c:otherwise>
<form action="Login" method="post">
ID:<input type="number" min=20060001 name="id">
<input type="submit" value="login">
</form>
</c:otherwise>
</c:choose>

```

Figure 3.2: 根据 session 信息决定如何显示登陆信息的代码

```

//Login.java
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    int id = Integer.parseInt(request.getParameter("id"));
    Db db = new Db();
    Connection conn = db.getConnection();
    String sql = "select name from teachers where id=?";
    try {
        PreparedStatement pst = conn.prepareStatement(sql);
        pst.setInt(1, id);
        ResultSet rs=pst.executeQuery();
        if(rs.next()) {
            String name = rs.getString(1);
            HttpSession s = request.getSession();
            s.setAttribute("id", id);
            s.setAttribute("userName", name);
            response.sendRedirect("index.jsp");
        }else {
            PrintWriter out = response.getWriter();
            out.print("Wrong ID!\nWill return to index.");
            response.setHeader("refresh","3;URL=index.jsp");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            conn.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

Figure 3.3: 负责处理登陆的 servlet 部分代码

```
$tomcat_home$/conf/web.xml
<!-- ===== Default Session Configuration ===== -->
602  <!-- You can set the default session timeout (in minutes) for all newly -->
603  <!-- created sessions by modifying the value below.      -->
604  <!-- default: 30-->
605
606  <session-config>
607      <session-timeout>30</session-timeout>
608  </session-config>
```

Figure 3.4: tomcat 的 Session 默认生存周期

```
//Login.java
HttpSession s = request.getSession();
s.setAttribute("id", id);
s.setAttribute("userName", name);
Cookie ck = new Cookie("JSESSIONID",s.getId());
ck.setMaxAge(600);
response.addCookie(ck);
```

Figure 3.5: tomcat 的 Session 默认生存周期

3.3.2 Cookie 和 Session 配合实现登陆

使用 Session 登陆后，其有效期范围为一次会话过程，如果关闭浏览器之后，Session 就会消失了（只是服务器和客户端对应不起来了）。当客户端再打开网站时，服务器以为又是一个新的用户来访问了，就又开启一个 Session，用户需要重新登陆。怎么让服务器能够知道是某个用户再次访问本网站而不是一个新用户来访问的呢？答案是可以结合 Cookie 技术来实现。

默认情况下，tomcat 会将 Session 的生存周期设置为 30 分钟，如若想修改，请参照 Figure 3.4 的代码。

在 Figure 3.3 中，我们将登陆后的用户名保存在了 Session 里，为了确保这个用户在一定的时间范围内即使关闭浏览器再打开浏览器来访问本网站也不需要重新登陆，我们需要做的就是把这个 Session id 写到 Cookie 里面发送给客户端。客户端下次再访问本网站的时候，自动带上 Cookie，tomcat 服务器端会自动检查是否有一个叫 JSESSIONID 的 Cookie，如果有，就和服务器上的 JSESSIONID 进行比较，如果查找到了匹配的，那服务器就知道是某个特定的用户再次来访问了，服务器就会自动的取出那个特定 Session 相关的数据。这里还需要把 Cookie 的生命周期设置一下，以免关闭浏览器后 Cookie 失效。具体代码如 Figure 3.5 所示。