java web

张海宁[1]

May 16, 2018

[1]hnzhang1@gzu.edu.cn

Contents

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

Cookie[3]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`
[3]`https://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)。

Session[4]是保存在服务器端的，是一个容器，其中保存的是一对一对的数据（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 seesion 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)

Method	Description
int getMaxAge()	Returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
void setMaxAge(int expiry)	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);
   注意是一对一对的写，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
long getCreationTime()	Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
long getLastAccessedTime()	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.
int getMaxInactiveInterval()	Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
void setMaxInactiveInterval(int interval)	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 可以理解成介于客房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，有若干个方法，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

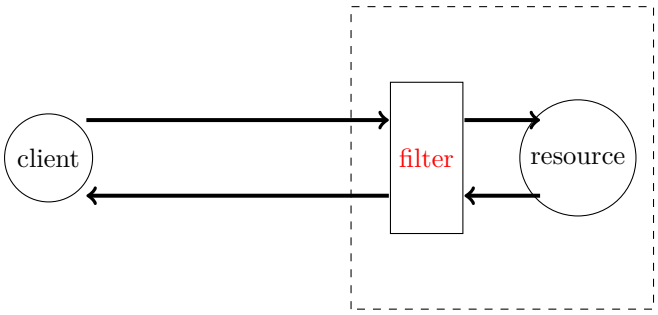

Figure 2.3: the position and function of a filter

“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. ServletContex
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
ServletContexAttributeListener	ServletContextAttributeEvent
HttpSessionListener HttpSessionActivationListener	HttpSeesionEvent
HttpSessionAttributeListener HttpSessionBindingListener	HttpSessionBindingEvent
ServletRequestListener	ServletRequestEvent
ServletRequestAttributeListener	ServletRequestAttributeEvent

Table 2.1: Listener 接口与 Event 类

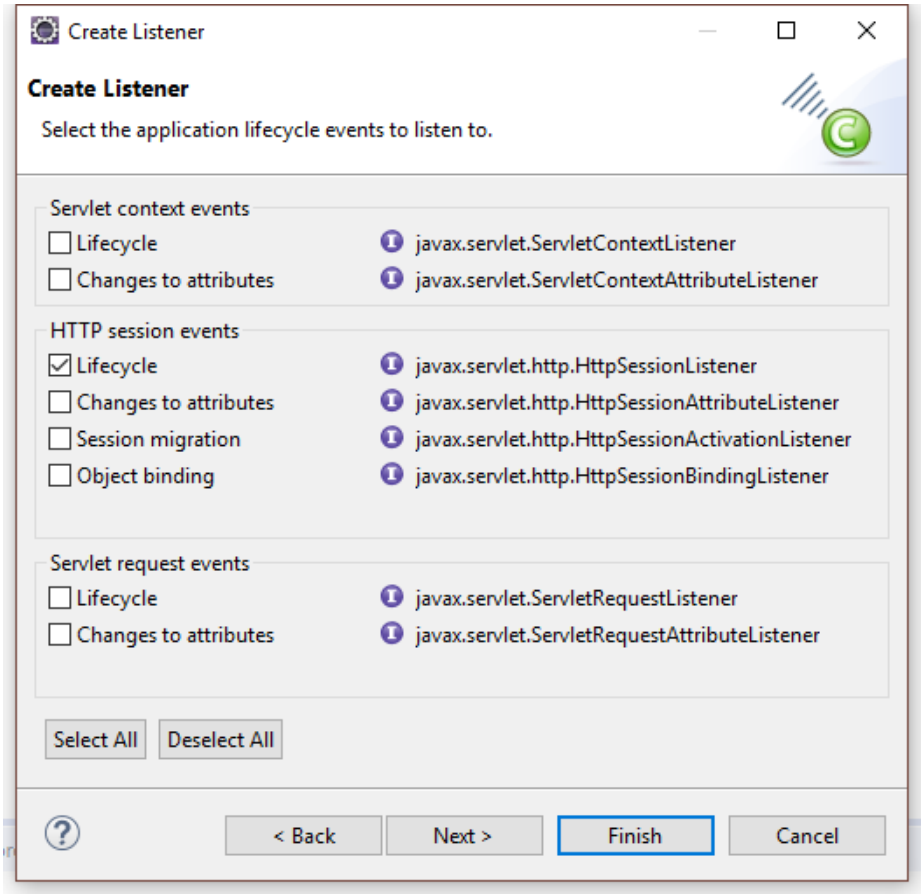

Figure 2.7: 选择 listener 监听事件

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 sessionDestoryed 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

   用户登陆之后要给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。

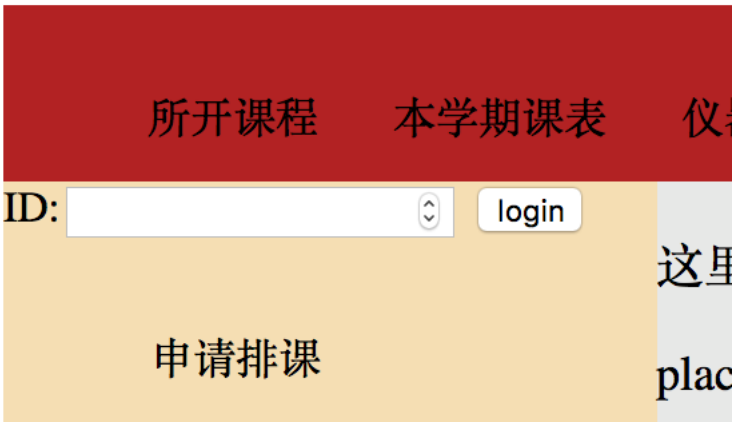

Figure 3.1: Login 界面

```
<!--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。

Chapter 4

Log4j and Database

4.1 目的

1. 练习 Log4j 的使用
2. 掌握数据库增删改查操作以及分页，加深对它们的理解。

4.2 目标

1. 在项目中使用日志记录重要信息
2. 在项目中使用数据库分页技术来显示查询结果

4.3 过程

4.3.1 Log4j

Log4j 是 Apache 的一个开源项目，是用来为 java 项目记录日志的。其有三个组件组成：

1. Logger

   可以定义多个 Logger，每个 Logger 实现不同的功能

2. Appender

   指定 log 信息存储的目标地址

3. Layout

   格式化 log 信息的输出样式

name	type	desc	level
DEBUG	Object	输出调试级别的日志信息	1
INFO	Object	输出消息日志	2
WARN	Object	输出警告级别的日志信息	3
ERROR	Object	输出错误级别的日志信息	4
FATAL	Object	输出致使错误级别的日志信息	5

Table 4.1: Log4j 的日志级别

Appender 接口的实现类	Description
org.apache.log4j.ConsoleAppender	输出到控制台
org.apache.log4j.FileAppender	输出到日志文件
org.apache.log4j.RollingFileAppender	当文件大小超出限制时，重新生成新的日志文件
org.apache.log4j.DailyRollingAppender	每天只生成一个对应的日志文件

Table 4.2: Appender 的几种常用目的地

Logger

Logger 是 Log4j 的日志记录器，是 Log4j 的核心组件。Log4j 将输出的日志信息定义了 5 种级别，如表4.1。

Appender

Appender 可以指定日志文件的目的地，几种常用的目的地如表4.2所示。

Layout

Layout 决定了日志的输出格式，其必须和 Appender 配合使用，它可以根据用户的个人习惯格式化日志的输出格式，例如：文本文件、HTML 文件等，常用的格式如表4.3所示。

layout 的子类	description
org.apache.log4j.HTMLLayout	将日志以 HTML 格式输出
org.apache.log4j.PatternLayout	将日志以指定的转换模式格式化输出
org.apache.log4j.SimpleLayout	将日志以简单的格式输出
org.apache.log4j.TTCCLayout	这种日志包含线程等信息

Table 4.3: Layout 的种类

```
#Logger
#设置rootLogger的记录级别为info,输出目的地为file
log4j.rootLogger=info,file

#Appender
#将file的属性设置为一个大小固定，可循环覆盖的文件
#指定日志文件的位置为/User\dots log.html
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=/Users/hainingzhang/Documents/
  GitHub/javaweb/lab-java/WebContent/login_log.html
log4j.appender.file.MaxFileSize=10kb
log4j.appender.file.MaxBackupIndex=3

#Layout
#指定日志格式为html
log4j.appender.file.layout=org.apache.log4j.HTMLLayout
```

Figure 4.1: log4j.properties

Log4j 的使用

1. 创建配置文件

   要使用 Log4j，首先要将下载的 Log4j-xxx.jar 文件放到项目文件夹内，并将其添加到 build path 中；然后再创建一个配置文件。完成这两个操作之后中就可以在项目中使用 Log4j 来记录日志了。一个典型的 Log4j 配置文件如 Figure 4.1所示.

2. 在 servlet 中使用 Log4j

   在 servlet 中需要首先获取 Lot4j 配置文件的路径并加载，然后就可以获取日志记录器进行写日志了，部分代码如 Figure 4.2所示。

4.3.2 数据库的基本查操作

使用 jsp 操作数据库的流程如下：

1. 加载数据库驱动类，并创建连接
2. 通过连接创建 Statement 或 PreparedStatement（推荐使用这种，不需要进行拼接 sql 语句）
3. 通过 Statement 或 PreparedStatement 对象进行数据库查询操作

一个数据库操作的示例 (查询和更改) 见 Figure 4.3。

```
//specify the property file of log4j and load it
//需要指定log4j.properties文件的具体位置
String path = getServletContext().getRealPath("WEB-INF/
  log4j.properties");
PropertyConfigurator.configure(path);
//get the rootlogger
Logger lg = LogManager.getLogger(Login.class);
//write the log at any position
lg.info("do login.");
...
lg.info(name+" is loging.");
```

Figure 4.2: 在 servlet 中使用 Log4j

4.3.3 数据库的分页

在数据量大的情况下，将所有的数据显示在同一个页面中，给查看带来不便的同时，也给服务器带来了压力，这种情况下就需要对数据进行分页查询。一般来说，有两种数据库分页查询的机制：

1. 一次查出所有数据，然后再根据用户的选择进行显示（通过用户点击第几页）
2. 通过数据库自身的分页机制进行分次查询，用户点击第几页就去数据库中查询第几页

在数据量很大的时候，采用第一种机制是不明智的。 数据库自身的分页机制是与数据库紧密关联的，每种数据库实现的方式都不一样。此处以 MySQL 为例。

MySQL 数据库通过 *limit* 关键字来控制 *select* 语句返回的记录数。limit 可以接收一或两个整数参数，它们的区别为：

1. With one argument, the value specifies the number of rows to return from the beginning of the result set.

   SELECT * FROM tbl LIMIT 5;

2. With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1).

   SELECT * FROM tbl LIMIT 5,10;

分页实例

接下来通过查询 teachers 表中的教师信息为例来展示如何实现分页查询。分页查询的流程如 Figure 4.4所示。

```
public class Db {
public Db(){
try {
Class.forName("com.mysql.jdbc.Driver");
conn = DriverManager.getConnection(
     "jdbc:mysql://localhost/lab?useUnicode=true&characterEncoding=UTF-8",
 "root","xxx");
} catch (Exception e) {
e.printStackTrace();
}
}
public ArrayList<Teacher> find(int page,int countPerPage){
ArrayList<Teacher> teacherList = new ArrayList<Teacher>();
String sql = "select id,name,logDate from teachers limit ?,?";
PreparedStatement ps;
try {
ps = conn.prepareStatement(sql);
ps.setInt(1, page);
ps.setInt(2, countPerPage);
ResultSet rs=ps.executeQuery();
while(rs.next()) {
Teacher t = new Teacher();
t.setStaffid(rs.getInt("id"));
t.setNm(rs.getString("name"));
t.setLogDate(rs.getString("logDate"));
teacherList.add(t);
}
} catch (SQLException e) {
e.printStackTrace();
}
return teacherList;
}
public int modifyTeacher(String id,String name) {
int rz=0;
String sql = "update teachers set name=? where id=?";
try {
PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(2, Integer.parseInt(id));
ps.setString(1, name);
rz=ps.executeUpdate();
} catch (SQLException e) {
e.printStackTrace();
}
return rz;
}
}
```

Figure 4.3: Db.java 文件中的部分代码

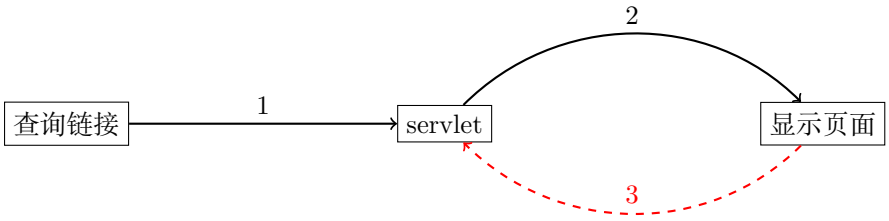

Figure 4.4: 分页查询的流程

```
public int getTotalPages(int countPerPage) {
 int page=0;
 int totalCount=0;
 String sql="select count(*) from teachers";
 try {
  Statement st = conn.createStatement();
  ResultSet rs = st.executeQuery(sql);
  rs.next();
  totalCount =rs.getInt(1);
 } catch (SQLException e) {
  e.printStackTrace();
 }
 page = (totalCount/countPerPage)+1;
 return page;
}
```

Figure 4.5: Db.java 文件中获取总页数的代码

分页查询的数据库操作代码涉及到以下两点：

1. 要知道数据到底有多少条，然后就可以计算出总共的页数，代码如 Figure 4.5所示。

2. 指定查询哪一页，每页显示多少条数据。代码如 Figure 4.6所示。

查询链接

在 teacher.jsp 页面的合适位置，放置一个查询链接，如 Figrue 4.7所示，通过本链接去调用负责查询的 servlet。

Servlet

负责查询数据的 servlet 要完成以下工作：

```
public ArrayList<Teacher> find(int page,int countPerPage){
 ArrayList<Teacher> teacherList = new ArrayList<Teacher>();
 String sql="select id,name,logDate from teachers limit ?,?";
 PreparedStatement ps = conn.prepareStatement(sql);
 ps.setInt(1, page);  ps.setInt(2, countPerPage);
 ResultSet rs=ps.executeQuery();
 while(rs.next()) {
  Teacher t = new Teacher();
  t.setStaffid(rs.getInt("id"));
  t.setNm(rs.getString("name"));
  t.setLogDate(rs.getString("logDate"));
  teacherList.add(t);
 }
 return teacherList;
}
```

Figure 4.6: Db.java 文件中分页查询的代码

```
<form action="QueryTeacher" method="post">
<input type="submit" value="query teacher info"/>
</form>
```

Figure 4.7: 查询链接

```
ArrayList<Teacher> tLst=null;
int curPage = 1;
int TotalPage=0;
int countPerPage = 5;
Db d = new Db();
TotalPage=d.getTotalPages(countPerPage);
d=new Db();
if(request.getParameter("page")==null) {
 tLst=d.find(1, countPerPage);
}else {
  curPage = Integer.parseInt(
    request.getParameter("page"));
  tLst=d.find(curPage, countPerPage);
 }
request.setAttribute("teacherList", tLst);
```

Figure 4.8: servlet-QueryTeacher.java 中查询指定记录的代码

1. 查询指定的记录

   offset and how many

2. 生成页数相关的信息

   current page and total page

3. 将结果传递给显示页面

在 servlet 中查询指定页数据的部分代码如 Figure 4.8所示，注意其中的变量 page 是由显示页面传递过来的参数，是为了获取用户在显示页面点击的第几页（比如用户点击了第 10 页，那么就把这个提交给 servlet，servlet 查询了第 10 页后，把结果再返回给显示页面）。

servlet 不光要查询数据，还要生成有关页码的导航条，如代码 Figure 4.9所示。

servlet 将查询到的数据和所生成的页码导航条封装到 request 中传递给显示页面，代码如 Figure 4.10所示，其中代码的意思是：将本 servlet 处理之后的 request 封装起来，传递给 *teacher_list.jsp* 页面。

显示页面中可以显示 servlet 查询到的数据，又可以从显示页面选择第几页，从而查询那一页的数据，其代码如 Figure 4.11所示。

4.3.4 修改数据

修改数据的关键在于定位某条数据，我们可以通过主键来定位数据。在数据显示页面，可以增加一列超链接（“修改”），在这个超链接中传入本条记录的主键，

```
StringBuilder strb = new StringBuilder();
strb.append("<div><form name='qt' action='QueryTeacher'
  method='post'>");
strb.append("第<select name='page'
  onchange='document.qt.submit()'>");
for(int i=1;i<=TotalPage;i++) {
 if(i==curPage) {
  strb.append("<option value='"+i+"' selected='selected'>"
    +i+"</option>");
 }else {
  strb.append("<option value="+i+">"+i+"</option>");
  }  }
strb.append("</select>页，共"+TotalPage+"页");
strb.append("</form></div>");
request.setAttribute("pageBar", strb);
```

Figure 4.9: servlet-QueryTeacher.java 中生成页码导航条的代码

```
request.getRequestDispatcher("teacher_list.jsp")
  .forward(request, response);
```

Figure 4.10: servlet-QueryTeacher.java 将 request 传递给显示页面

```
<div class="content main">
当前系统中教师信息如下：<br>
<table>
<c:forEach items="${teacherList }" var="t" >
<tr><td>${t.staffid}</td><td>${t.nm }</td></tr>
</c:forEach>
</table>
${pageBar}

</div>
```

Figure 4.11: teacher_list.jsp

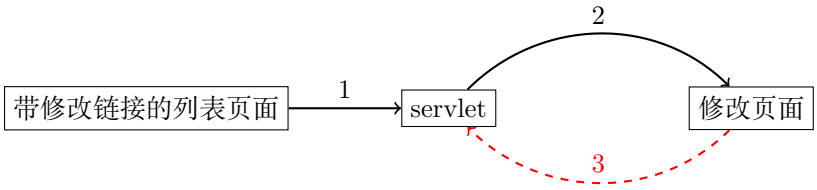

Figure 4.12: 修改数据的流程

```
<c:forEach items="${teacherList }" var="t" >
<tr><td>${t.staffid}</td><td>${t.nm }</td>
<td>
<a href='ModifyTeacher?id=${t.staffid }&name=${t.nm}'>
修改</a></td></tr>
</c:forEach>
```

Figure 4.13: 带修改链接的显示页面 teacher_list.jsp

当点击这个链接的时候，打开一个新的页面，即修改页面，在修改页面中将各字段的值放到一个 input 中，以供修改，然后将本 form 中的数据提交，将数据库表中对应主键的那条记录进行更新。修改数据的流程如 Figure 4.12所示。

这里涉及以下几个内容:

1. 带修改链接的显示页面

   代码如 Figure 4.13所示。

2. 将显示页面数据传递到修改页面的代码

   如 Figure 4.14所示。

3. 修改页面

   代码如 Figure 4.15所示。

4. 处理修改信息

   代码如 Figure 4.16所示。

5. 数据库操作

   代码如 Figure 4.17所示。

```
//ModifyTeacher.java - doGet
request.setCharacterEncoding("utf-8");
request.setAttribute("id", request.getParameter("id"));
request.setAttribute("name", request.getParameter("name"));
request.getRequestDispatcher("teacher_modify.jsp")
  .forward(request, response);
```

Figure 4.14: 显示页面数据传递到修改页面

```
<form action="ModifyTeacher" method="post">
<table>
<tr><td>${id}
<input type="hidden" name="id" value="${id }"/></td>
<td><input type="text" name="name" value="${name }"/></td>
<td><input type="submit" value="提交修改"/></td></tr>
```

Figure 4.15: 修改页面 teacher_modify.jsp

```
//ModifyTeacher.java - doPost
request.setCharacterEncoding("utf-8");
System.out.println(request.getParameter("id")+","+
   request.getParameter("name"));
Db d = new Db();
int rz = d.modifyTeacher(request.getParameter("id"),
   request.getParameter("name"));
if(rz==1) {
 response.getWriter().println("modify successfully.");
}else {
 response.getWriter().println("something wrong.");
}
```

Figure 4.16: servlet 处理修改数据的请求

```
//Db.java - modifyTeacher
public int modifyTeacher(String id,String name) {
 int rz=0;
 String sql = "update teachers set name=? where id=?";
 try {
  PreparedStatement ps = conn.prepareStatement(sql);
  ps.setInt(2, Integer.parseInt(id));
  ps.setString(1, name);
  rz=ps.executeUpdate();
  getClose();
 } catch (SQLException e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
 }
 return rz;
}
```

Figure 4.17: 数据库操作类 Db