

# J2EE 轻量级框架

## 实验一 P1

学号：SA16225221

姓名：欧勇

报告撰写时间：2016/12/19

## 1. 实验环境/器材

操作系统: Windows 10  
IDE: Eclipse Kepler  
SDK: JDK 1.8  
Web Server: tomcat  
数据库: MySQL 5.1.53  
数据库可视化管理软件: Wamp Server  
浏览器: Chrome 54.0.2840.87 m (64-bit)

## 2. 实验目的

搭建 SSH 开发环境, 理解 SSH 程序开发基本概念和调试方法。

## 3. 实验内容

1. MVC pattern in your project. The following diagrams are helpful to express your idea:
  - A. UML package diagram
  - B. Module structure chart
2. Use struts actions and results demo. The following diagrams are good to express your idea:
  - A. UML sequence diagram
  - B. Program control flows diagram
3. Debug ways used in java programming

## 4. 实验过程

### 1) 项目介绍

项目为体系结构的实验, 名字 SMART Monitor, 项目内容是构建一个物联网智能监控系统, 使其 PC 端能实现用户登录, 实时监控各设备节点状态、接收节点状态变更推送等功能; 其分布式机器端能够接收服务器端命令, 发送心跳包, 发送异常信号, 发送设备状态变化命令等。项目的体系架构图如 图 1.1, 开发视图如 图 1.2:

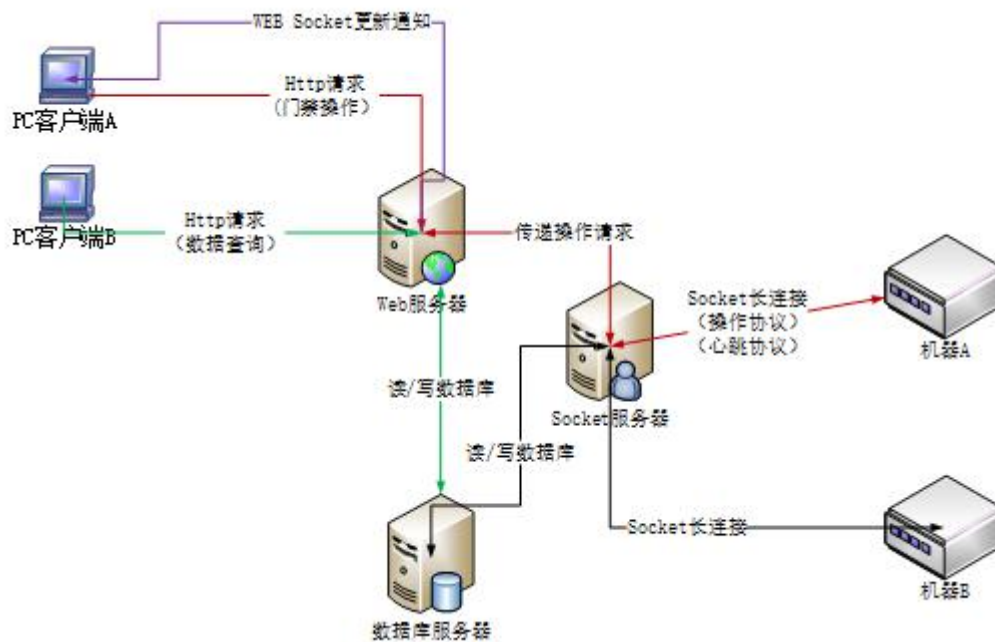


图 1.1. SMART Monitor 体系架构图

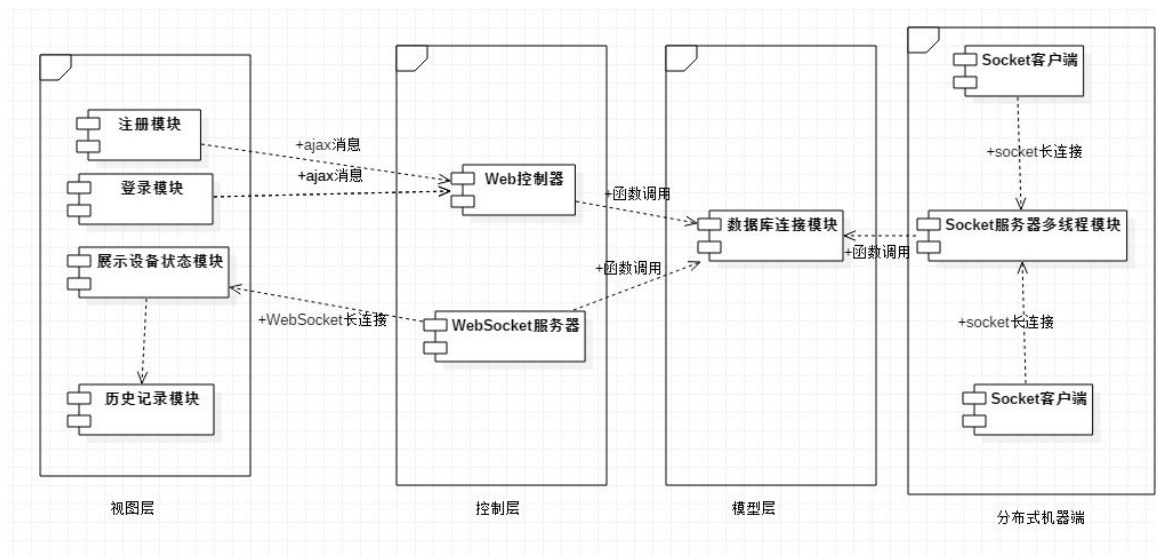


图 1.2. SMART Monitor 开发视图

项目的架构采用经典的 MVC 架构，其中视图层即前端的几个 html 页面，控制层为，使用 Servlet 实现控制层逻辑以及转发和过滤。

WebSocket 与前端页面直接建立联系，将模型层数据直接推送至 WebSocket 客户端中，之后使用 js 脚本动态展示出来。

整个项目中数据库是关键点，所有的服务和业务都是依赖数据模型的设计而实现的，这也符合了 MVC 中，M 模型才是业务核心的原理，其他两层都依赖于模型层。

最后，Socket 客户端采用分布式的方式。Socket 服务器的实现方式为多线程。

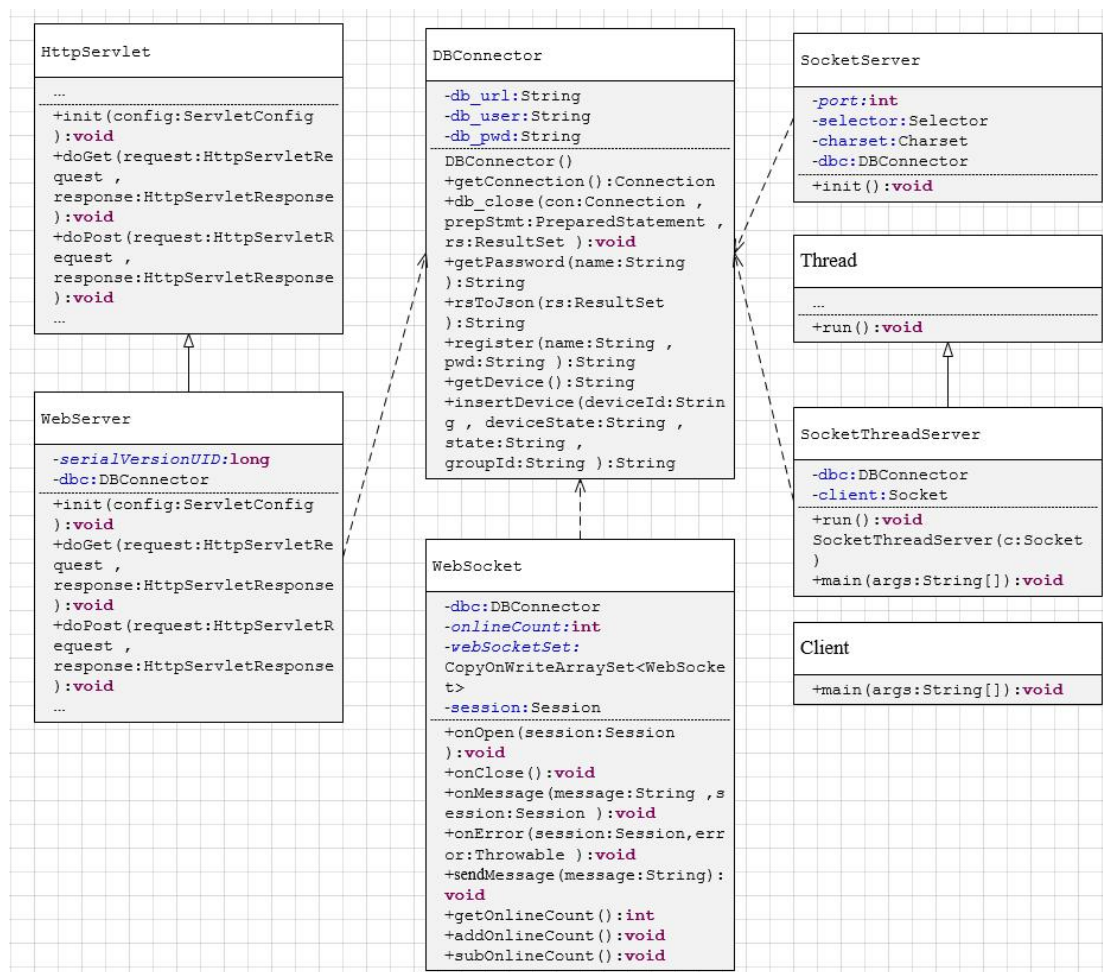


图 1.3. 类图

其中，DBConnector 类是模型层的实现，是整个项目的关键存取，SocketServer 类是用 I/O 多路复用方式的 Socket 服务器端的实现，SocketThreadServer 类是用多线程方式的 Socket 服务器端实现，Client 类是 Socket 客户端的实现，模拟了设备，继承自 HttpServlet 的 WebServer 类是一个 Servlet，作为控制层，对请求进行分发和过滤，WebSocket 类是 WebSocket 的服务器，是控制层的一部分。

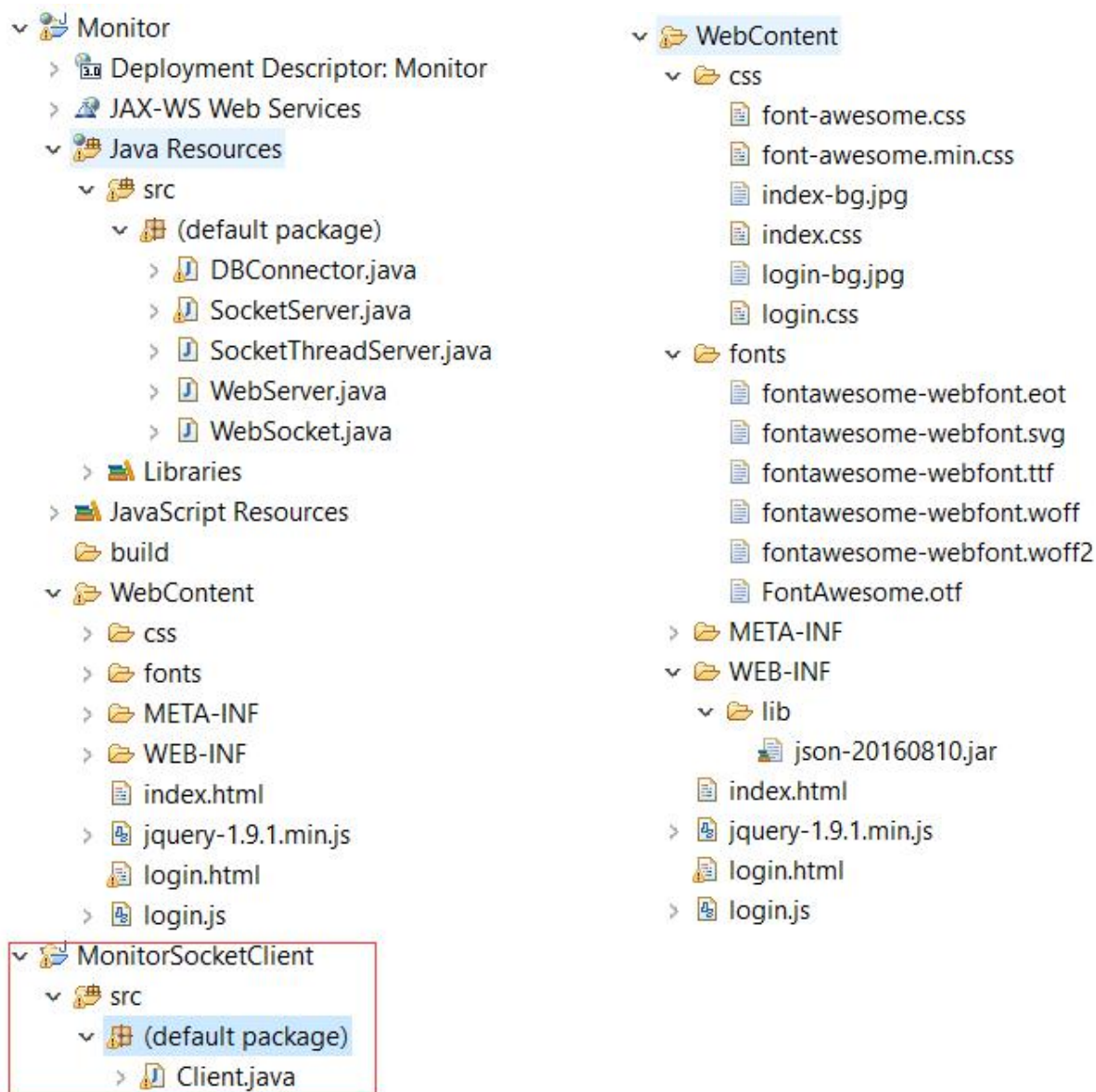


图 1.4. SMART Monitor 文件组织图

这个是 Web 服务器和 Sockt 客户端的文件组织截图，其中 DBConnector 是数据库连接类，WebServer 是 Web 服务器端用于登录，注册以及请求验证，转发等功能，WebSocket 是 WebSocket 类，用于与前端保持 socket 长连接的并定时推送所有的设备状态到客户端。

SocketThreadServer 是用多线程实现的一个 Socket 服务器端，主要工作是利用多线程接收多个客户端的心跳信息，然后判断其状态是否改变，若改变则将改变数据保存到数据库，若不变则不保存数据。

前端展示设备的图标采用了 font-awesome 的 css 图标库。

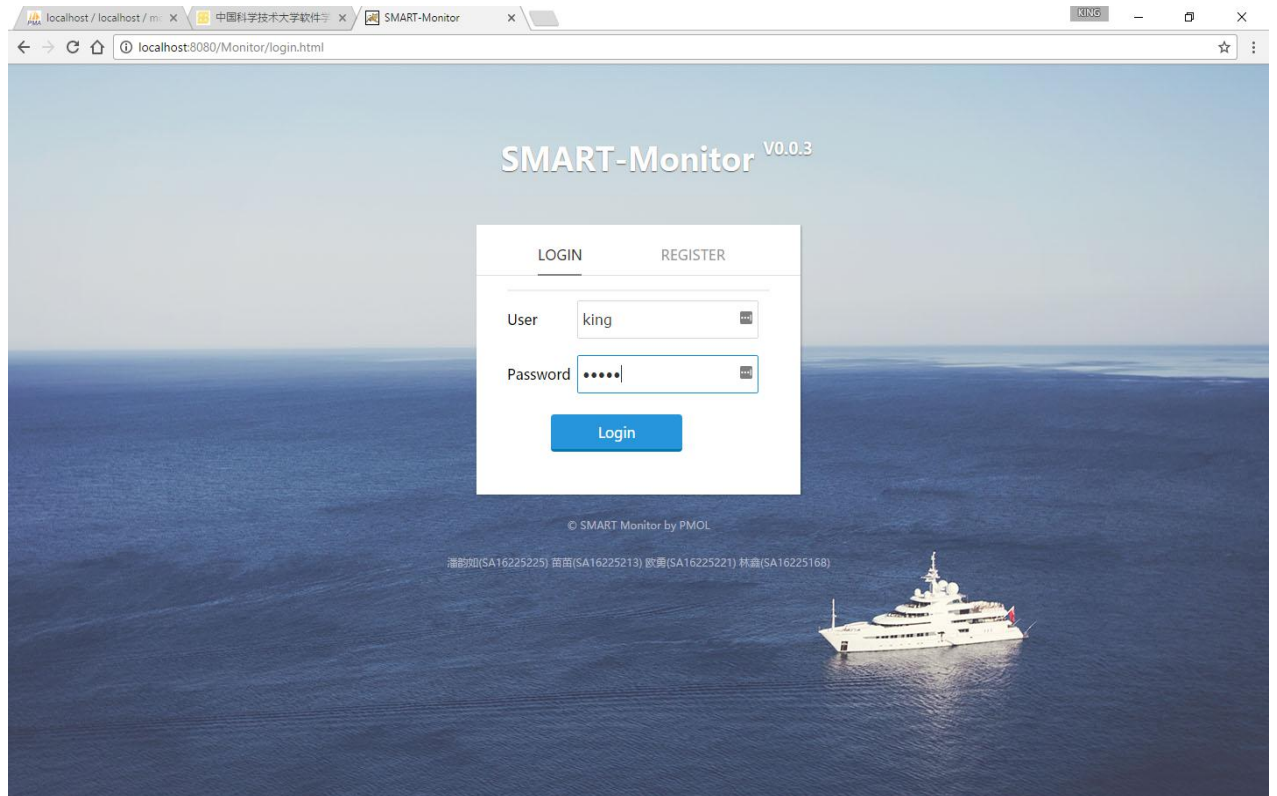


图 1.5.1. 登录界面

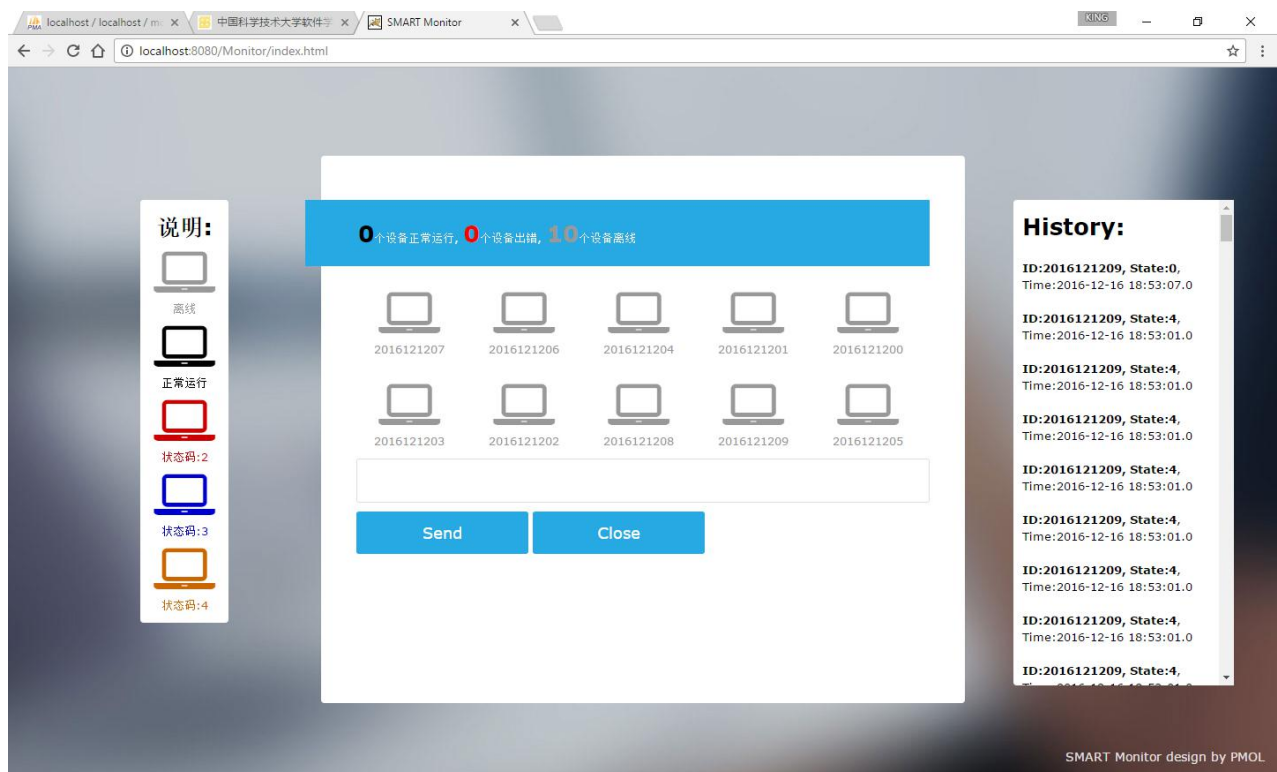


图 1.5.2. 成功登录主页界面（设备都未开启）

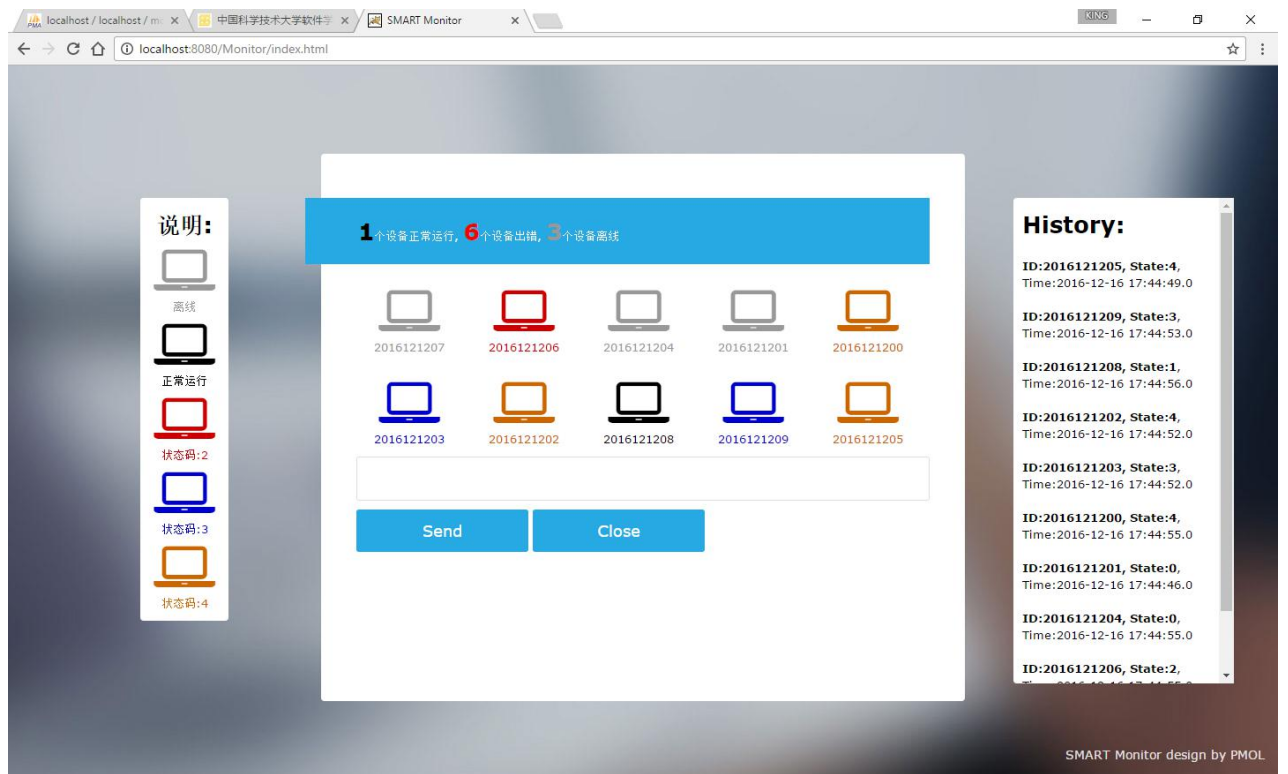


图 1.5.3. 设备开启后的主页界面

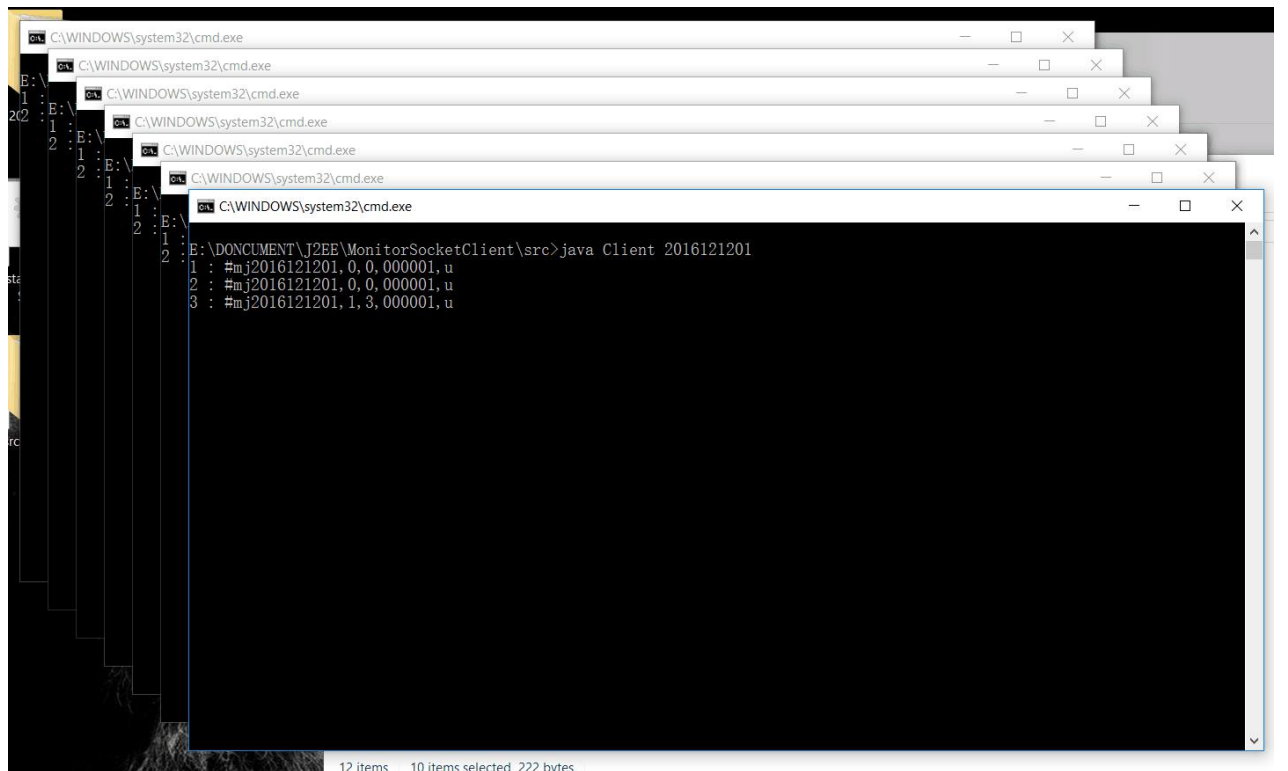


图 1.6. 模拟的设备界面（暂定 10 个）

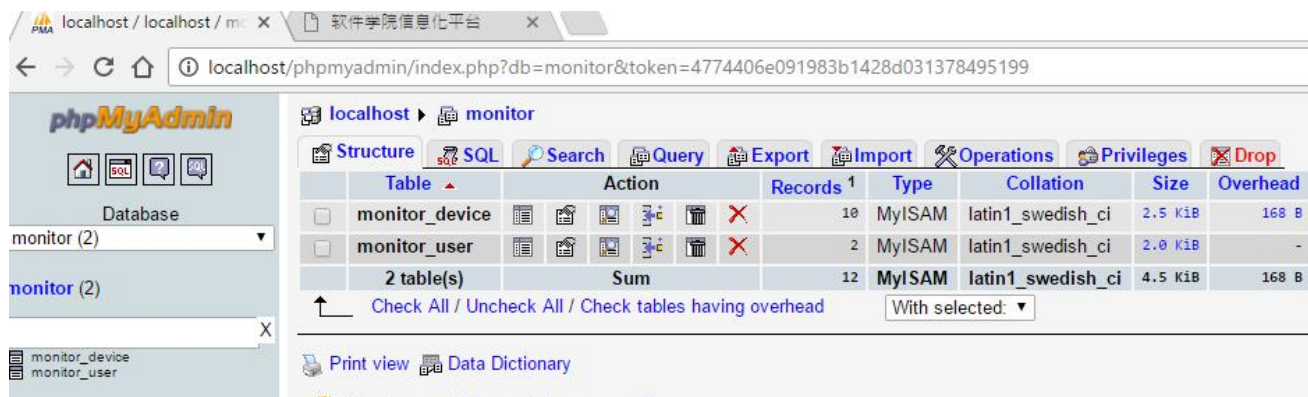


图 1.7.1. ( 通过 WAMP Server 界面管理工具查看的 ) 数据库概览图

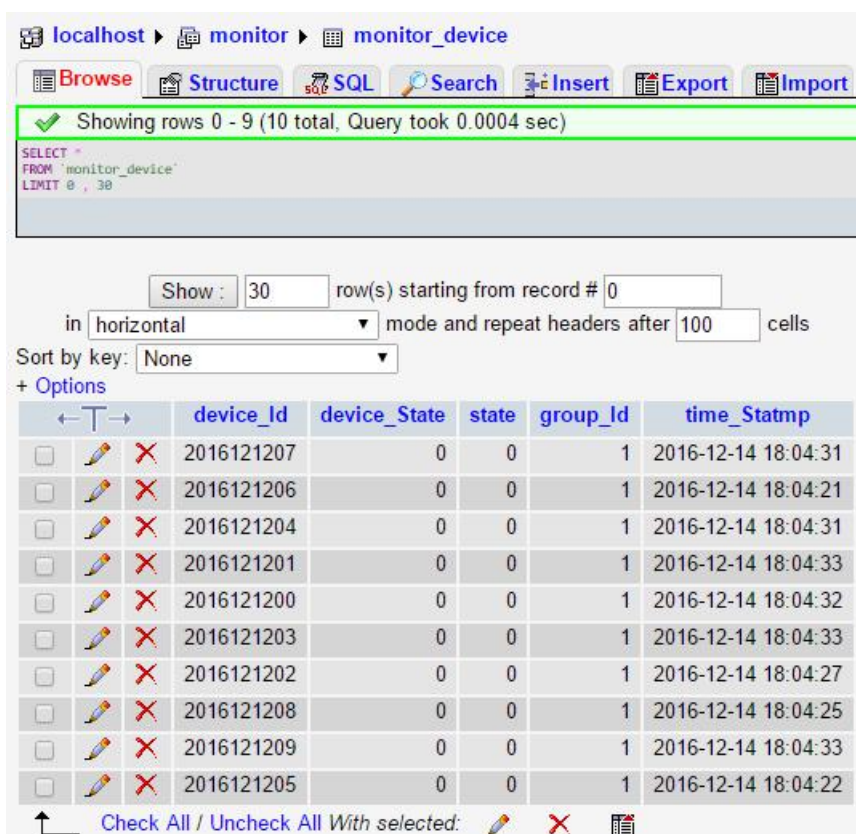


图 1.7.2. 数据库 monitor\_device 表内数据展示

图 1.7.2. 展示的数据库表中 device\_Id 表示设备号共 10 位，device\_State 表示设备本身的状态 1 位，state 表示设备的门磁状态 1 位，然后是 group\_Id 用来表示小组号 6 位，最后的 time\_Statmp 为时间戳，表示设备上上次状态变更的时间。

图 1.7.3 为用户表，user\_Name 表示用户名，user\_Password 表示密码，暂时只有 king 和 aaaa 两位用户，为了测试方便，密码采用明文存储。



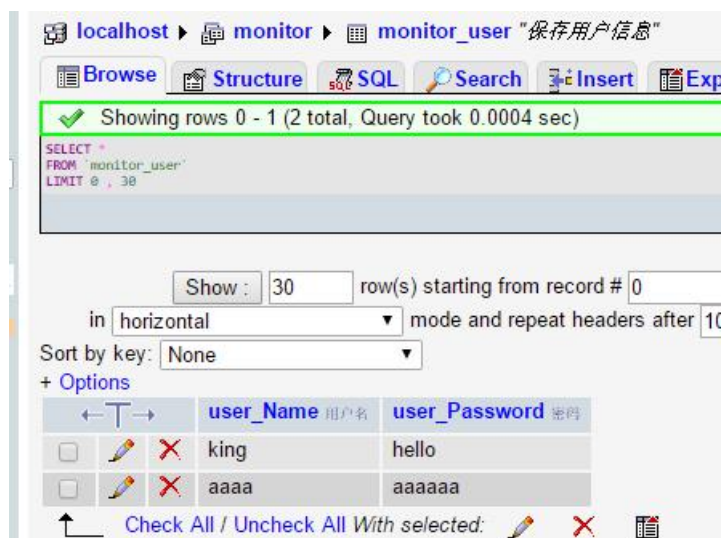


图 1.7.3. 数据库 monitor\_user 表内数据展示

## 2) 使用 struts2 后的项目

本次实验使用 struts2 框架替代 servlet 成为控制器，并使用 DAO 类作为访问数据库的中间对象，同时为了更简单的说明实验问题，只展示登录和注册功能。

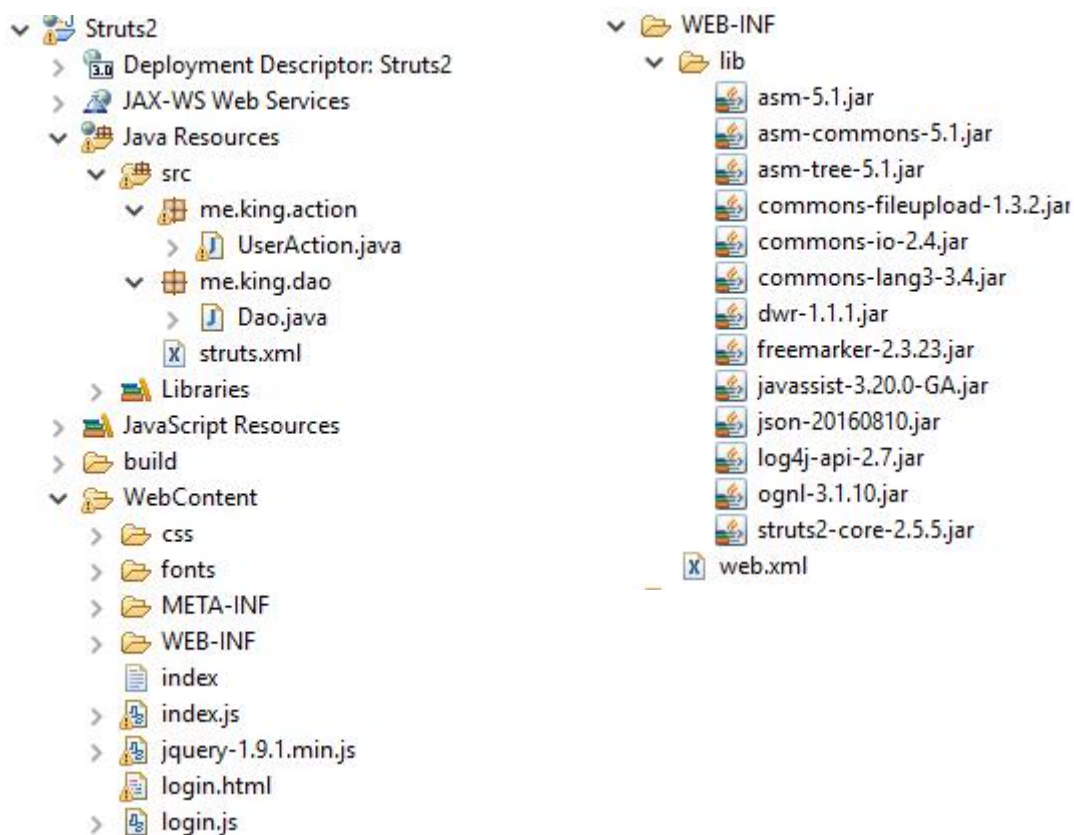


图 2.1.1. SMART Monitor 采用 Struts2 作为控制器后的文件组织结构图

在上图 2.1.1 中，右边的导入的 struts2 的依赖包以及 json 格式包（本次实验未用到），为了更合理的组织文件，分别在 src 目录下新建了两个包，一个专门用于保存 action，一个则用于保存 DAO，本次实验修改登录和注册功能，所以只有 UserAction 类。在 WebContent 目录下，login.html 为登录/注册页面，index 文件中保存着展示设备状态的 html 标签文本。

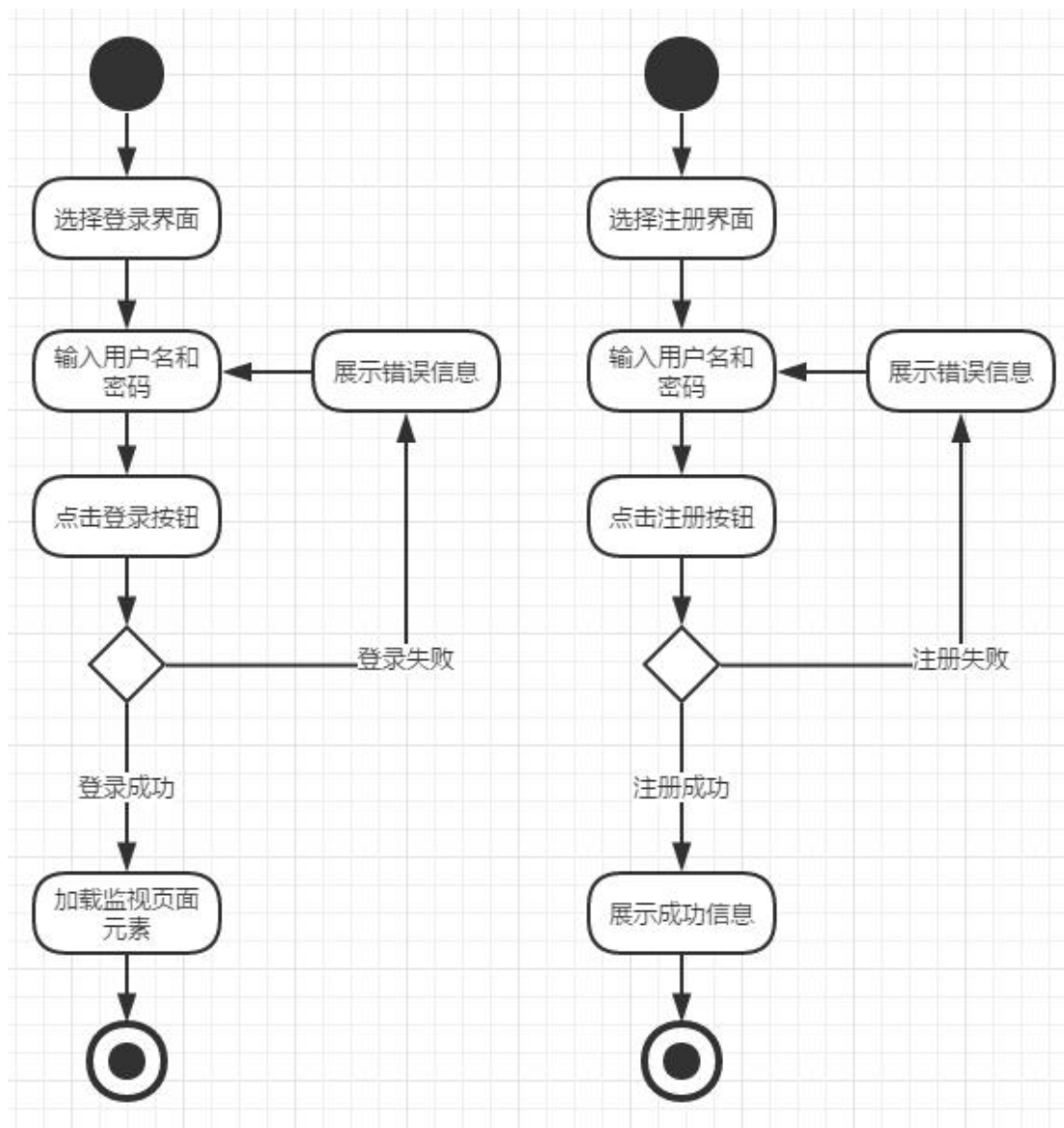


图 2.1.2. 登录和注册流程图

如图 2.1.3 所示登录时序图，控制器为 struts2，数据库连接器为 DAO 类。

- (1) 用户打开登录界面
- (2) 用户输入用户名和密码并点击登录
- (3) 前端提交用户名和密码给控制器，并通过控制器发送给数据库连接器

- (4) 数据库连接器根据用户名获取数据库中返回的密码
- (5) 数据库连接器返回用户名验证结果给控制器
- (6) 如果用户名和密码配对则成功，则加载监视界面元素；如果失败，则提示错误信息

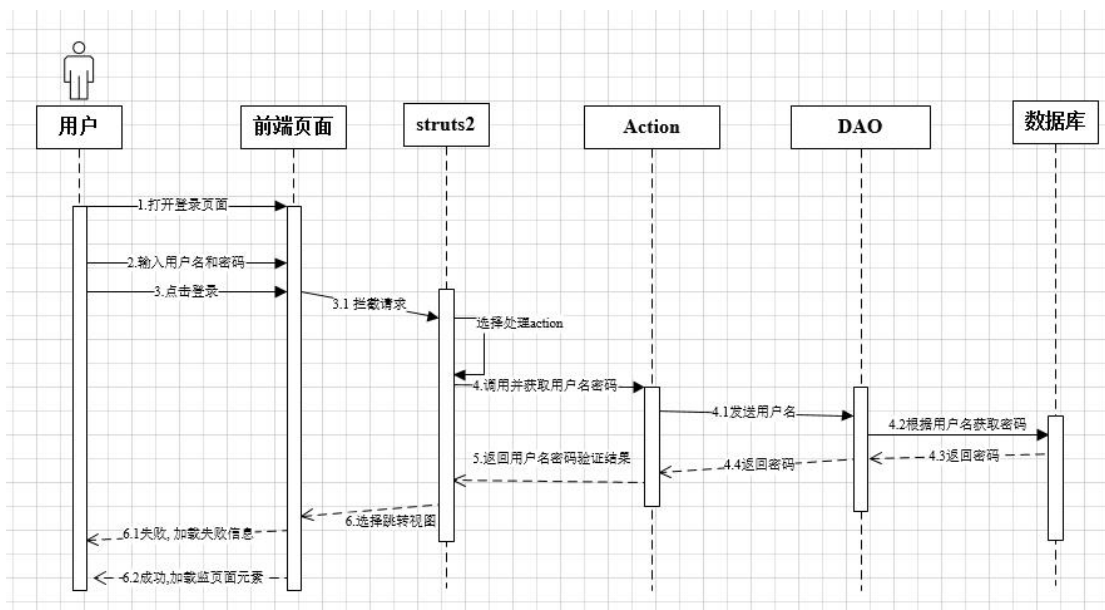


图 2.1.3. 登录时序图

在 WebContent/WEB\_INF/Web.xml 中配置着使用 struts2 拦截所有请求。如下图 2.2.1。

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
5   id="WebApp_ID" version="3.0">
6   <display-name>Monitor</display-name>
7   <welcome-file-list>
8     <welcome-file>login.html</welcome-file>
9   </welcome-file-list>
10
11   <!-- Struts2配置 -->
12   <filter>
13     <filter-name>struts2</filter-name>
14     <filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>
15   </filter>
16   <filter-mapping>
17     <filter-name>struts2</filter-name>
18     <url-pattern>/*</url-pattern>
19   </filter-mapping>
20 </web-app>
21

```

图 2.2.1 Web.xml 配置文件内容

在 src 目录下存放则 struts 的配置文件，配置信息如下图 2.2.2，继承自默认的 struts-default 的包下配置两个 action，一个为 login，一个为 register，都使用 me.king.action.UserAction 类，但是使用不同的 method。

login 表示处理登录请求的 action，register 表示处理注册请求的 action。

因为所有的 ajax 请求在网络中传输的时候都是采用 UTF-8 编码，所以需要所有的字符集设置为 UTF-8。

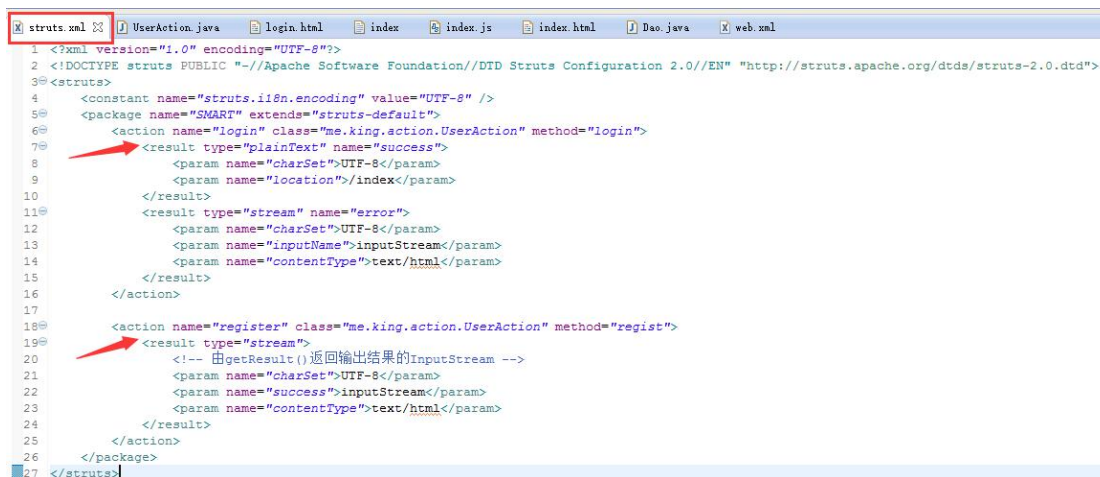
在登录 action 中，配置了两个 result，一个类型为 plainText，当返回字符串为 success 的时候使用此 result，将 index 文件中的内容以普通文本发送发送到客户端。（用此方法实现的 ajax 的返回信息需要保存在一个文件中）

另外一个为 stream 类型，result 的名称为 error，当登录出错的时候使用此 result。并将 inputStream 中的信息返回给客户端，此方法实现的 ajax 需要特别定义一个 inputStream 对象保存信息。

可以看出，这两种方式都是为了支持 ajax 而设置的的 result 类型，而没有使用默认的 dispatcher 类型。

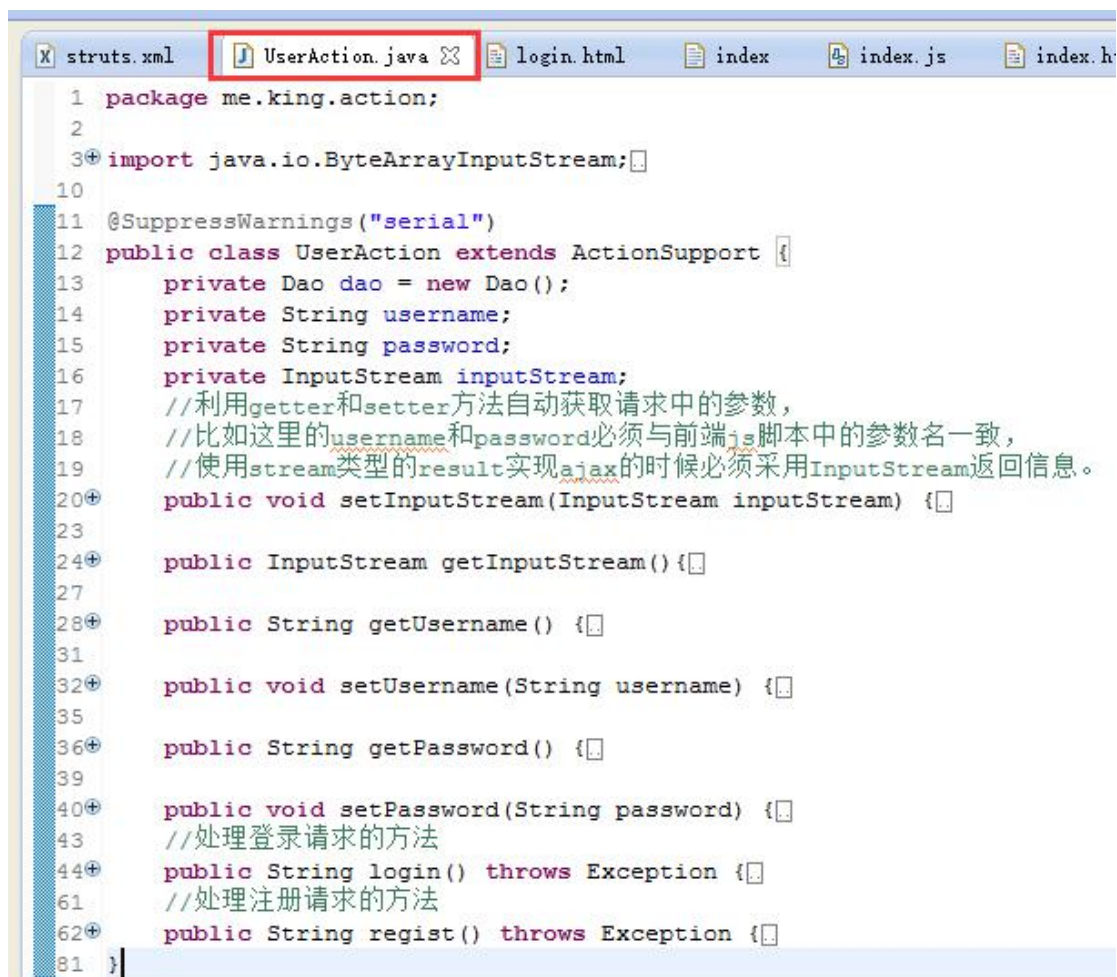
当客户端发起的是一个 ajax 请求的时候，当服务器端向 ajax 客户端返回信息的时候，默认的 dispatcher 并不会直接内部转发到指定的视图处，而是将该视图作为文本返回。

当 result 的 name 和 Action 的返回字符串不是默认的几种类型的时候，客户端会报 404 错误，这个错误我测试了很久，但是没有找到原因。



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd">
3 <struts>
4   <constant name="struts.i18n.encoding" value="UTF-8" />
5   <package name="SMART" extends="struts-default">
6     <action name="login" class="me.king.action.UserAction" method="login">
7       <result type="plainText" name="success">
8         <param name="charset">UTF-8</param>
9         <param name="location">/index</param>
10      </result>
11     <result type="stream" name="error">
12       <param name="charset">UTF-8</param>
13       <param name="inputName">inputStream</param>
14       <param name="contentType">text/html</param>
15     </result>
16   </action>
17
18   <action name="register" class="me.king.action.UserAction" method="regist">
19     <result type="stream">
20       <!-- 由getResult() 返回输出结果的InputStream -->
21       <param name="charset">UTF-8</param>
22       <param name="success">inputStream</param>
23       <param name="contentType">text/html</param>
24     </result>
25   </action>
26 </package>
27 </struts>
```

图 2.2.2 struts.xml 配置文件内容



```
1 package me.king.action;
2
3 import java.io.ByteArrayInputStream;
10
11 @SuppressWarnings("serial")
12 public class UserAction extends ActionSupport {
13     private Dao dao = new Dao();
14     private String username;
15     private String password;
16     private InputStream inputStream;
17     //利用getter和setter方法自动获取请求中的参数,
18     //比如这里的username和password必须与前端js脚本中的参数名一致,
19     //使用stream类型的result实现ajax的时候必须采用InputStream返回信息。
20 public void setInputStream(InputStream inputStream) {}
23
24 public InputStream getInputStream() {}
27
28 public String getUsername() {}
31
32 public void setUsername(String username) {}
35
36 public String getPassword() {}
39
40 public void setPassword(String password) {}
43     //处理登录请求的方法
44 public String login() throws Exception {}
61     //处理注册请求的方法
62 public String regist() throws Exception {}
81 }
```

图 2.3.1 UserAction 类概览

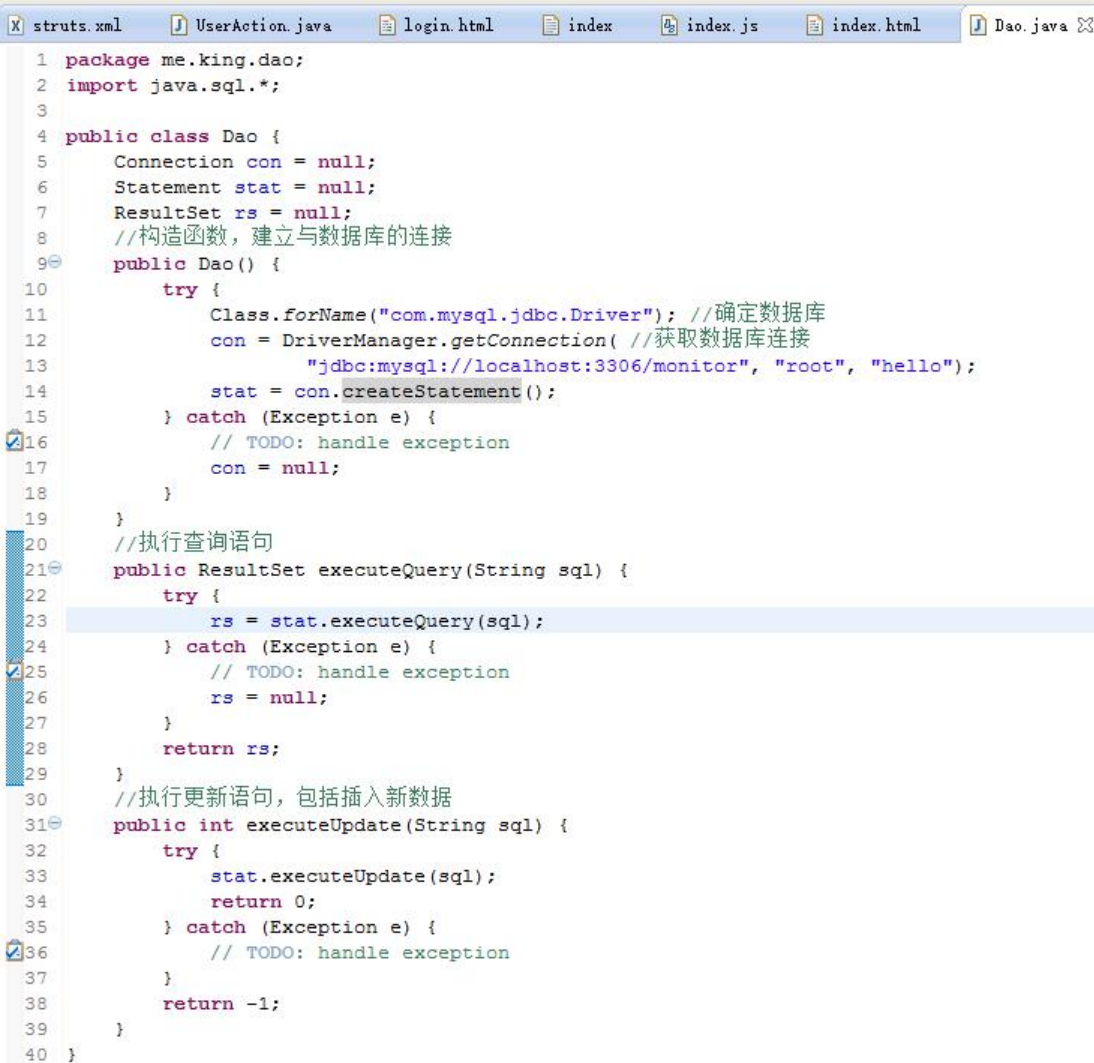
```
40 public void setPassword(String password) {}
43 //处理登录请求的方法
44 public String login() throws Exception {
45     String rt; //同时返回信息将返回给前端ajax的信息保存在rt字符串中。
46     String sql = "select * from monitor_user where user_Name="
47         + getUsername() + "'";
48     ResultSet rS = dao.executeQuery(sql); //定义sql语句并执行
49
50     if (rS.next()) { // 判断是否找到该用户
51         if (getPassword().equals(rS.getString(2)))
52             return SUCCESS; //找到用户并且密码正确, SUCCESS = success
53         else rt = "Wrong Password"; //密码错误
54     } else rt = "User Is Not Exist"; //用户不存在
55     //将返回的ajax信息保存在inputStream中
56     inputStream = new ByteArrayInputStream(rt.getBytes());
57     return ERROR; //只有登录不成功就采用ERROR类型的result, ERROR = error
58 }
59 //处理注册请求的方法
60 public String regist() throws Exception {
61     String rt;
62     String sql = "select * from monitor_user where user_Name="
63         + getUsername() + "'";
64     ResultSet rS = dao.executeQuery(sql);
65
66     if (!rS.next()) { //首先判断用户是否存在
67         sql = "insert into monitor_user(user_Name,user_Password) values('"
68             + getUsername() + "','" + getPassword() + "')";
69         //当用户不存在的时候, 采用插入sql语句将新用户的用户名和密码插入数据库
70         int rs = dao.executeUpdate(sql);
71         if (rs > 0) rt = "Register Success"; //插入新用户成功
72         else rt = "Fail"; //数据库插入新数据失败
73     }
74     rt = "User Existed";
75     inputStream = new ByteArrayInputStream(rt.getBytes());
76     return SUCCESS; //不管新用户创建成功还是失败都需要以ajax的形式返回信息给前端
77 }
78 }
```

图 2.3.2 UserAction 类 login 和 regist 方法截图

具体的代码解释已经写在注释中, 注意处理登录的 login 方法和处理注册的 regist 方法的策略不太一样。即新用户注册成功后不会直接跳转而是提示用户注册成功, 但是需要登录才能进入监控设备的页面。

在图 2.4 中, 由于本次实验的重点不是对 DAO 的实验, 所以没有完成的按照 DAO 的设计模式开发, 没有数据库连接类、VO、DAO 接口、DAO 实现类和 DAO 工厂类等复杂的类。

直接实现了建立连接和执行 sql 语句的简单函数。



```
1 package me.king.dao;
2 import java.sql.*;
3
4 public class Dao {
5     Connection con = null;
6     Statement stat = null;
7     ResultSet rs = null;
8     //构造函数，建立与数据库的连接
9     public Dao() {
10         try {
11             Class.forName("com.mysql.jdbc.Driver"); //确定数据库
12             con = DriverManager.getConnection( //获取数据库连接
13                 "jdbc:mysql://localhost:3306/monitor", "root", "hello");
14             stat = con.createStatement();
15         } catch (Exception e) {
16             // TODO: handle exception
17             con = null;
18         }
19     }
20     //执行查询语句
21     public ResultSet executeQuery(String sql) {
22         try {
23             rs = stat.executeQuery(sql);
24         } catch (Exception e) {
25             // TODO: handle exception
26             rs = null;
27         }
28         return rs;
29     }
30     //执行更新语句，包括插入新数据
31     public int executeUpdate(String sql) {
32         try {
33             stat.executeUpdate(sql);
34             return 0;
35         } catch (Exception e) {
36             // TODO: handle exception
37         }
38         return -1;
39     }
40 }
```

图 2.4. DAO 类代码截图

图 2.5.1 展示 index 文件的内容为一个 html 的片段，没有 html 标签，只有 head 和 body 标签，仅仅只是把监视设备页面的框架预先定义好，具体的样式由 css/index.css 定义，控制页面行为的脚本存放在 index.js 中，主要写的是 WebSocket 的客户端脚本，比如发起 WebSocket 连接，监听事件的回调函数等，如下图 2.5.2。

由于仅仅实验一主要实现采用 struts2 框架的是注册和登录功能，所以在本次实验中并没有用到 WebSocket。

```

1 <head>
2   <meta charset="UTF-8">
3   <meta http-equiv="Pragma" content="no-cache">
4   <meta http-equiv="Cache-Control" content="no-cache">
5   <meta http-equiv="Expires" content="0">
6   <link href="/css/font-awesome.min.css" type="text/css" rel="stylesheet">
7   <link href="/css/index.css" type="text/css" rel="stylesheet">
8   <title>SMART Monitor</title>
9 </head>
10
11 <body>
12   <div id="display"><b>Example:</b><br>
13     <ul>
14       <li><a href="javascript:void(0)" title="Offline"><i class="fa fa-laptop fi
15       <li><a href="javascript:void(0)" title="Running"><i class="fa fa-laptop fi
16       <li><a href="javascript:void(0)" title="State:2"><i class="fa fa-laptop fi
17       <li><a href="javascript:void(0)" title="State:3"><i class="fa fa-laptop fi
18       <li><a href="javascript:void(0)" title="State:4"><i class="fa fa-laptop fi
19     </ul>
20   </div>
21   <div id="history"><b>History:</b><br>
22     <div></div>
23   </div>
24   <div class="warp common-icon-show">
25     <div class="message" id="message"><span class="rw-words"></span></div>
26     <ul id="devices"></ul>
27   </div>
28
29   <div class="copyright">SMART Monitor design by PMOL</div><br>
30   <script src="jquery-1.9.1.min.js"></script>
31   <script src="index.js"></script>
32 </body>

```

图 2.5.1 Index 文件内容

```

1   var websocket = new WebSocket('ws://localhost:8080/Monitor/websocket'); //创建WebSocket对象
2   //alert(websocket.readyState); //查看websocket当前状态
3   //连接成功建立的回调方法
4   websocket.onopen = function(event) {
5     $('#message > span.rw-words').html("Link SMART Monitor Socket Server Success");
6   }
7   //接收到消息的回调方法
8   websocket.onmessage = function(event) {}
9
10  //连接发生错误的回调方法
11  websocket.onerror = function() {}
12  //连接关闭的回调方法
13  websocket.onclose = function() {}
14
15  //监听窗口关闭事件，当窗口关闭时，主动去关闭websocket连接，防止连接还没断开就关闭窗口，server端会抛异常。
16  window.onbeforeunload = function() {}

```

图 2.5.2 index.js 部分脚本内容概览

图 2.6.1 为登录/注册的前端 html 源代码，登录和注册都放在不同的 div 内。可以自由切换。样式由 css/login.css 控制，脚本在 login.js 中，主要实现了为切换 div 的动画，以及登录/注册的前端验证（比如用户名不能小于 4 位，密码不能小于 6 位等）和 ajax 请求代码。



```

struts.xml  UserAction.java  *login.html  index  index.js  index.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5 <title>SMART-Monitor</title>
6 <script type="text/javascript" src="./jquery-1.9.1.min.js"></script>
7 <script type="text/javascript" src="./login.js"></script>
8 <link href="./css/login.css" rel="stylesheet" type="text/css">
9 </head>
10 <body style="zoom: 1;">
11 <h1>
12     SMART-Monitor<sup>V0.0.4</sup>
13 </h1>
14
15 <div class="login" style="margin-top: 50px;">
16     <!-- 表头 -->
17 <div class="header">
27 <!-- 登录 -->
28 <div class="web_qr_login" id="web_qr_login">
64 <!-- 登录end -->
65
66 <!-- 注册 -->
67 <div class="qlogin" id="qlogin" style="display: none;">
107 <!-- 注册end -->
108 </div>
109 <div class="copyright jianyi">© SMART Monitor by 欧勇 (SA16225221)</div>
110 <div class="jianyi"></div>
111 </body>
112 </html>

```

图 2.6.1 login.html 代码概览

在图 2.6.2 中，展示了登录的时候发起的 ajax 请求源代码，其中需要填入 action 配置的名字 login，以及 post 请求附带 username 和 password 信息（需要于后台的 getter 和 setter 一致），以及成功请求后获取返回信息的回掉函数。

```

57 $('#monitor_login').click(function(){
58     $.post('login',{
59         username: $("#u").val(),
60         password: $("#p").val(),
61     },function(data, status){
62         console.log(data); //输出收到的信息
63         if(data === 'Wrong Password' || data === 'User Is Not Exist')
64             $('#login_tips').text(data);
65         else{ //当收到的data为html标签组成的字符串的时候，对其解析并插入
66             $('#head').html('');
67             for(var i = 0; i<15; ++i)
68                 $('#head').append($(data)[i]);
69
70             $('#body').html('');
71             for(var i = 15; i< $(data).length; ++i)
72                 $('#body').append($(data)[i]);
73         }
74     });
75 });

```

图 2.6.2. login.js 中登录 ajax 代码

### 3) 调试程序

在 Eclipse 中开启服务器后，直接在浏览器中输入登录页面的网址：

<http://localhost:8080/Struts2/login.html>,

先测试输入密码错误的情况（用户名不存在的情况类似，就不贴图了），在网页并没刷新的情况下客户端收到服务器发送回来密码错误的 Wrong Password 提示。

截图右边为浏览器控制台打印收到的信息。

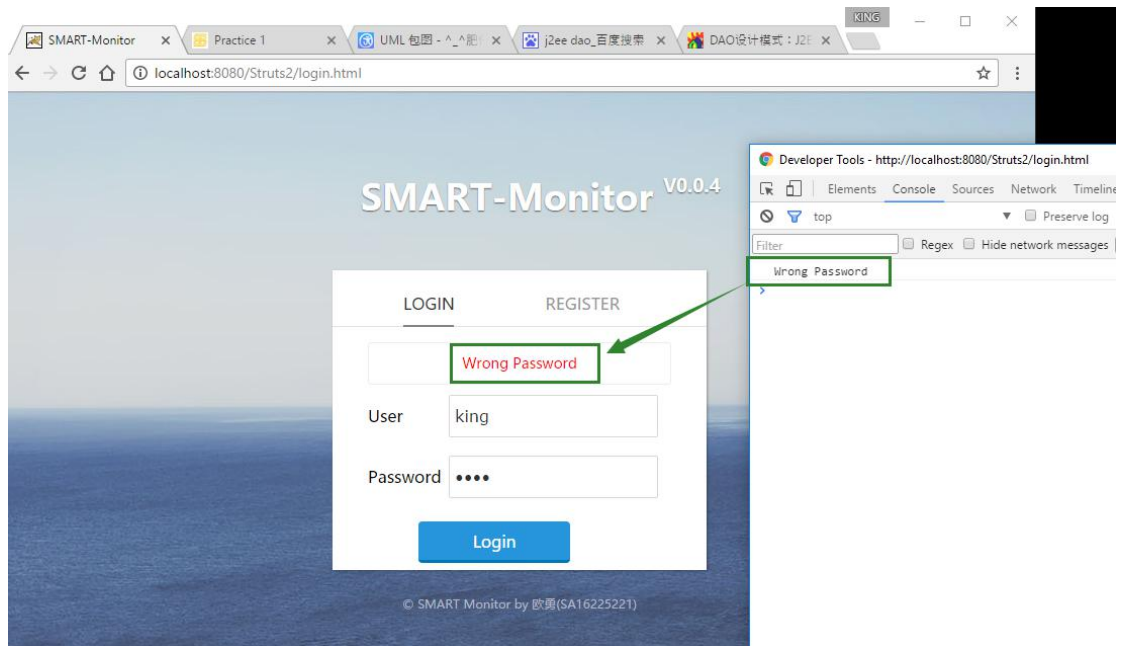


图 2.7.1 测试登录失败的提示

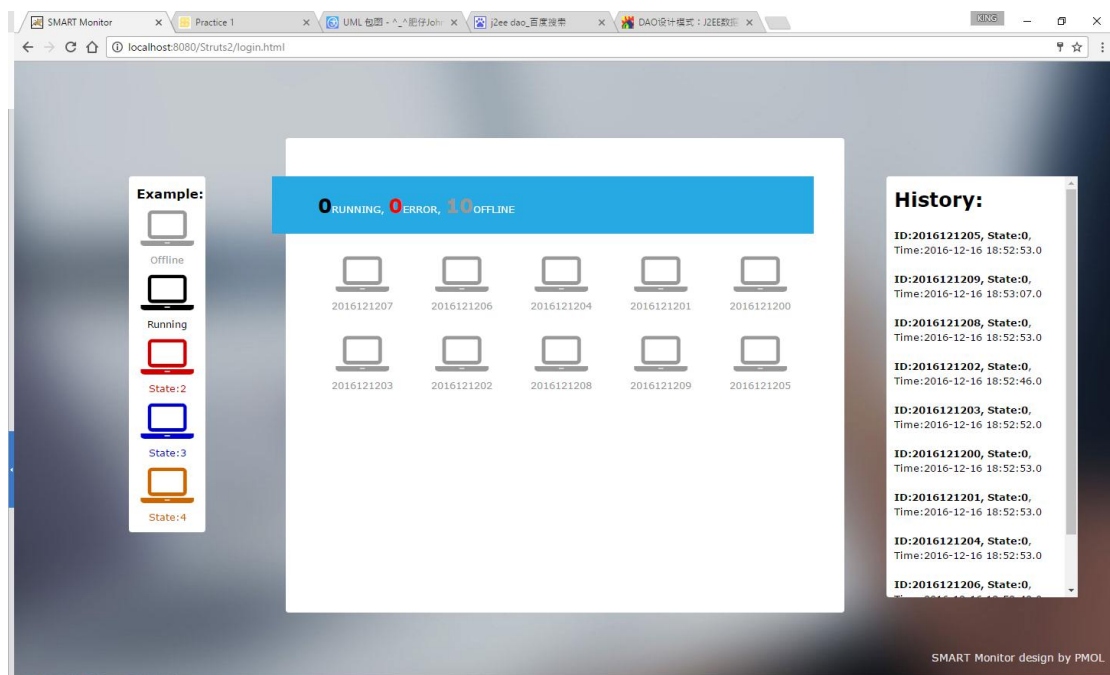


图 2.7.2 测试登录成功后跳转

然后测试登录成功的情况，输入正确的用户名和密码后浏览器展示的页面如图 2.7.2 所示，由于没有开启 Socket 服务器和 Socket 客户端，现在所有的设备都显示为下线。同时可以看到浏览器地址并没有改变，而 action 的配置中也没有使用 dispatcher 类型处理返回视图。

而打开浏览器控制台可以看到如图 2.7.3 可以看到 2 个红色箭头处，登录成功后浏览器接受到的 response 返回信息，其内容与图 2.5.1 Index 文件的内容一致。



图 2.7.3 测试登录成功后浏览器接受到的 response 返回信息

## 5. 实验总结

对概念/方法的理解与总结，实验碰到的问题及解决方法.....

为了将 struts2 融入原本以及写好的体系结构实验中，在配置上吃了很多苦头，尤其是为了在 action 中使用多个方法的配置的时候，发现 struts2 对 ajax 的支持非常不好，同时对前后端分离其实做的也不够简洁如意。

在测试中，我尝试过，result 不使用 stream 类型，而全部使用 plainText 或 dispatcher 的方式，这种情况下，需要预先定义很多文本，文本的内容为希望返回给前端 ajax 的响应信息，这种方式非常的不灵活，后来查了下可以加载一个 struts2 的 json-plugin 库用于自动将 action 中的属性构造为 json 格式字符串返回给前端，但是这种方式也不灵活，虽然能自动构造 json 字符串，但是却非常不好，因为每次都会将所有的属性都自动发过去，而且需要定义很多的 getter 和 setter 方法。

在实验过程中遇到的错误大都是 404 错误，但是也遇到过 500 错误，当数据库服务器没有开启就登录的时候，网页会报 500 错误，tomcat 控制台会报空指针异常。

总而言之，个人感觉 struts2 框架对需要大量 ajax 请求的项目不适合。但是对过滤和拦截请求还是非常好用的，通过这种方式，恶意的访问者无法直接通过输入 url 的方式跳过登录环节，起到了很好的保护作用。