

J2EE 轻量级框架

实验 P2

学号：SA16225221

姓名：欧勇

报告撰写时间：2016/12/25

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. Talk about Interceptors in your project. The following diagrams are helpful to express your idea:

- A. UML sequence diagram**
- B. Program control flows diagram**

2. Implement Validations for your project. The following diagrams are helpful to express your idea:

- A. UML sequence diagram**
- B. Program control flows diagram**

3. viewer tags' usage demo

4. 实验过程

本次实验二在实验一的基础上改善而来, 项目介绍部分未作修改。可直接从第 9 页的 [struts2](#) 节看起。

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:

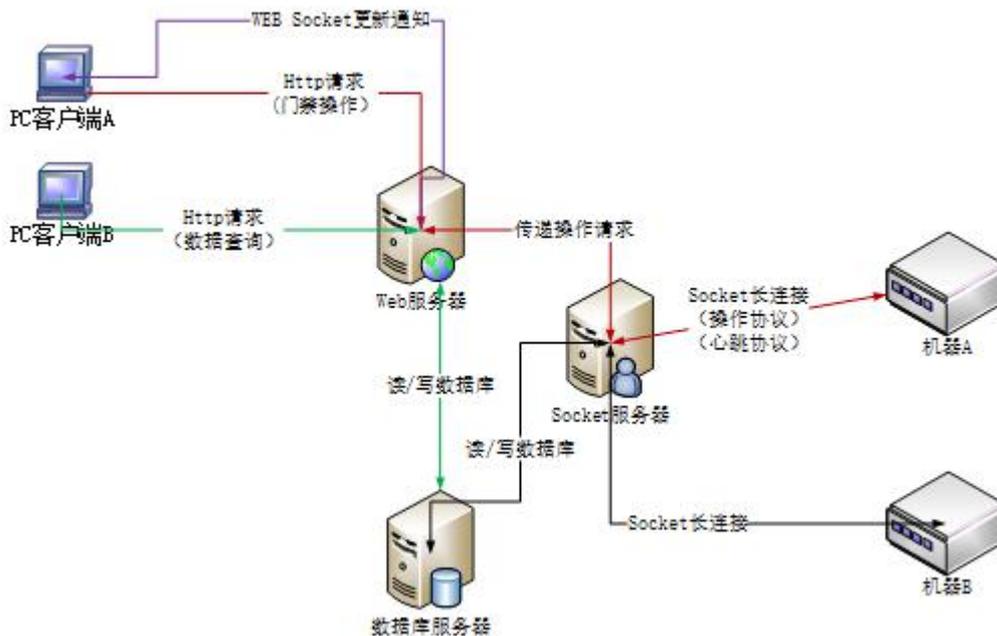


图 1.1. SMART Monitor 体系架构图

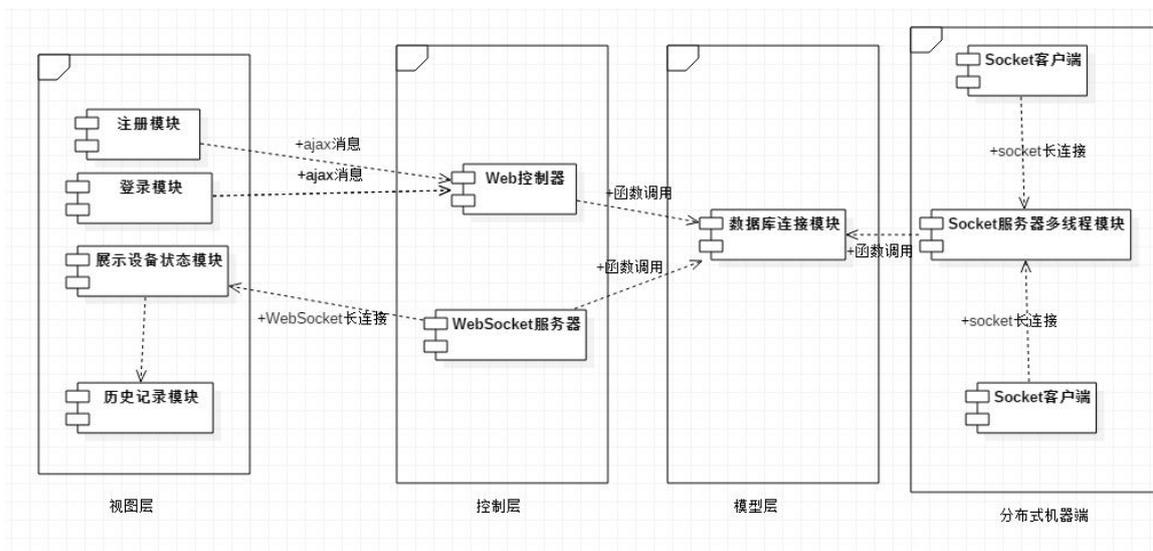


图 1.2. SMART Monitor 开发视图

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。

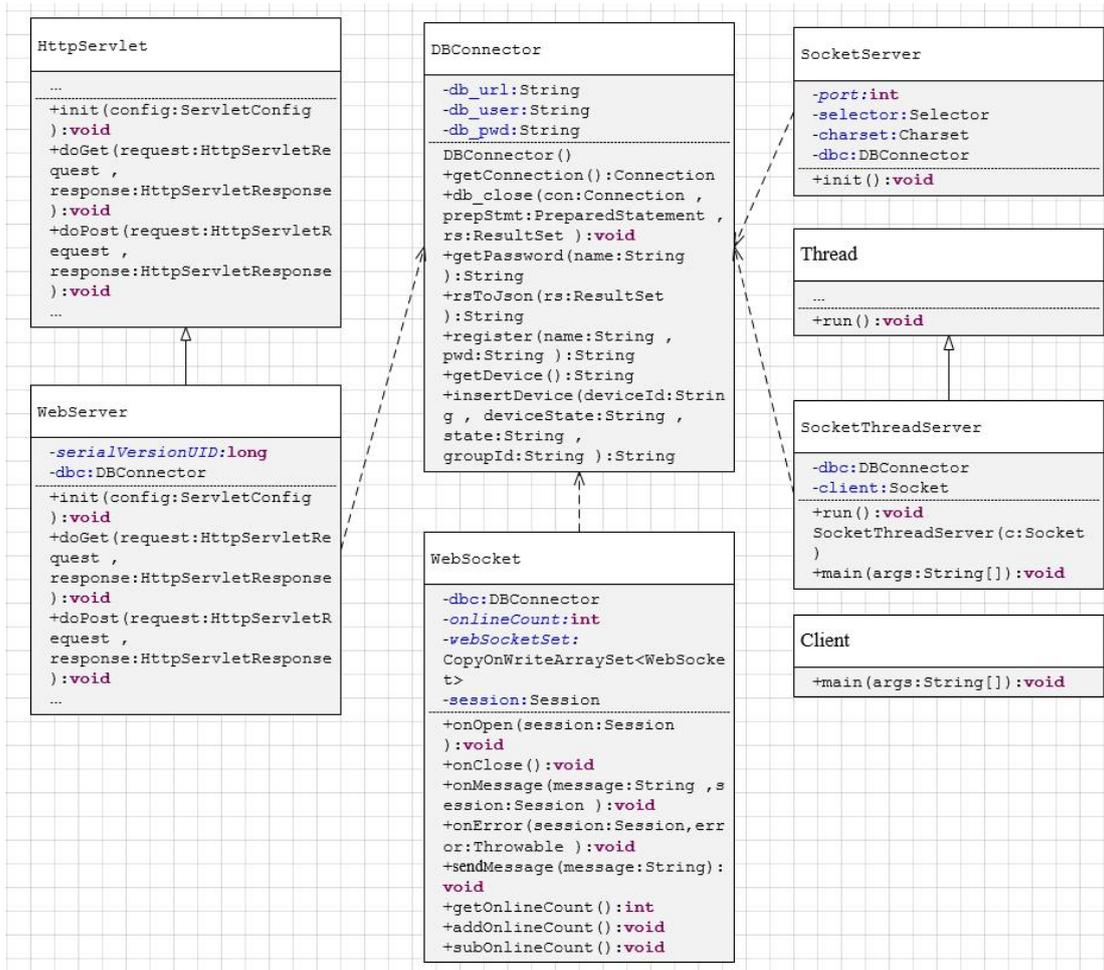


图 1.3. 类图

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。

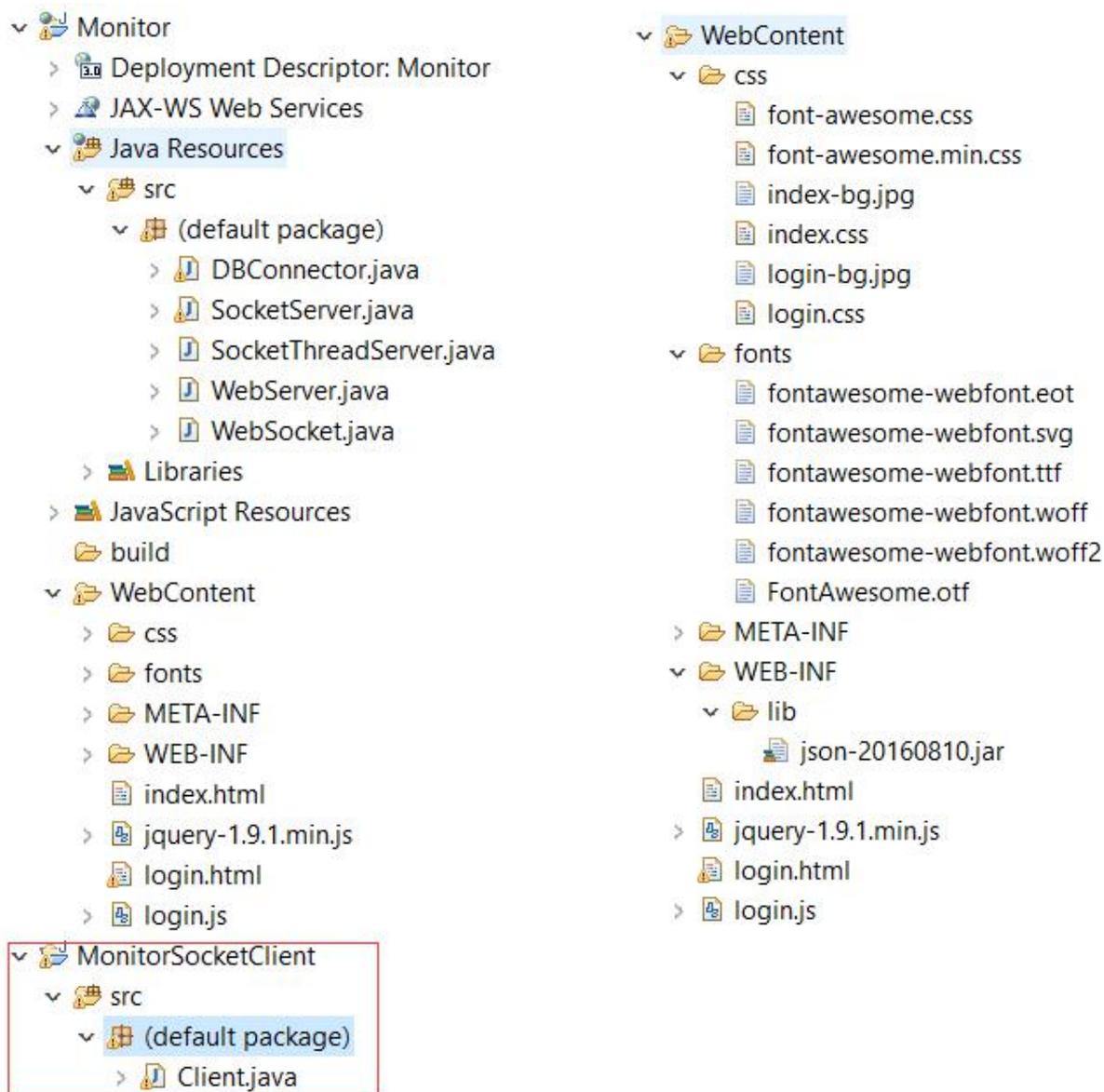


图 1.4. SMART Monitor 文件组织图

这个是 Web 服务器和 Scket 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。

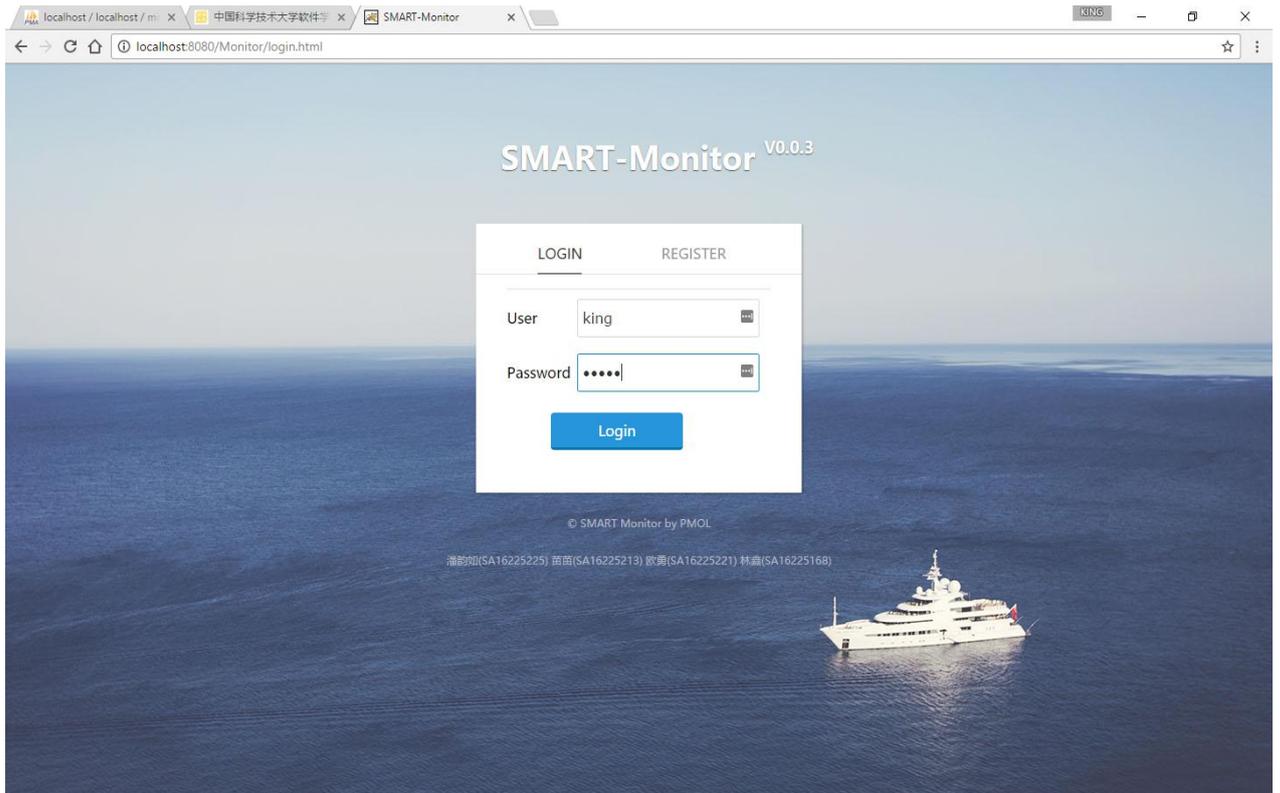


图 1.5.1. 登录界面

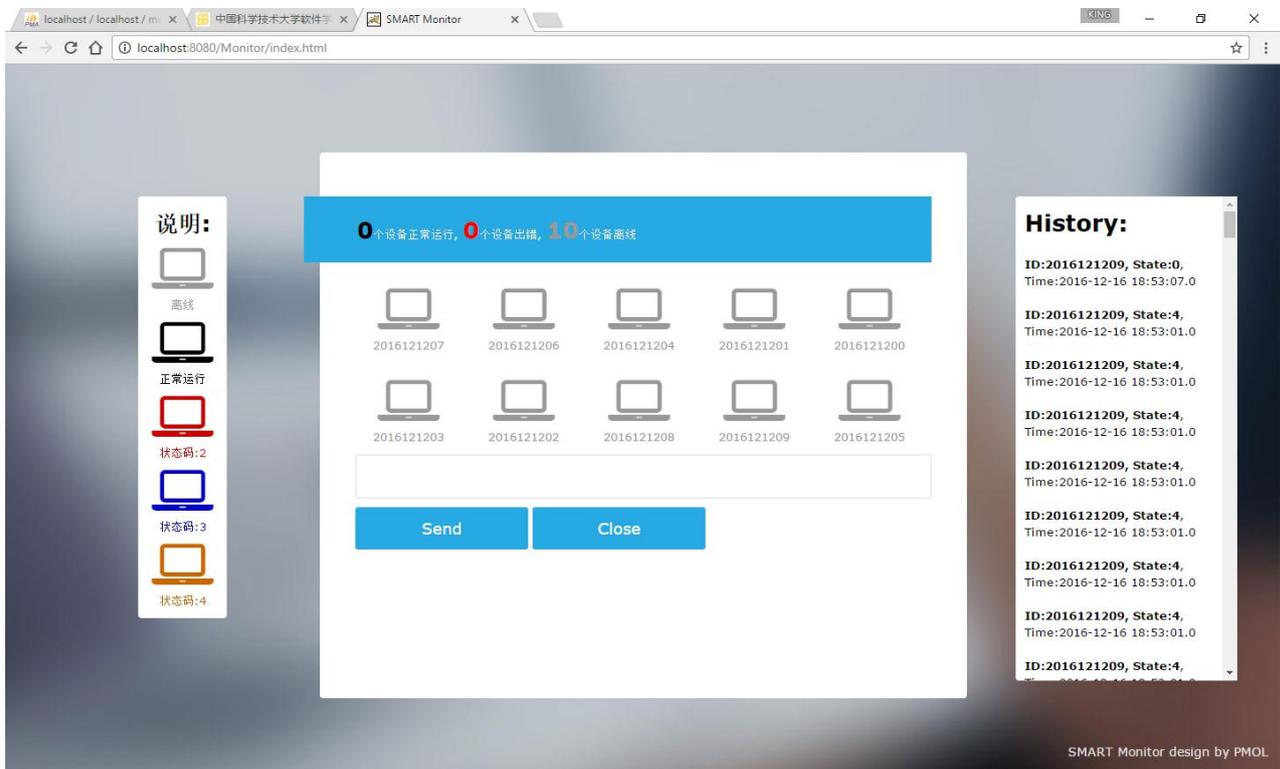


图 1.5.2. 成功登录主页界面（设备都未开启）

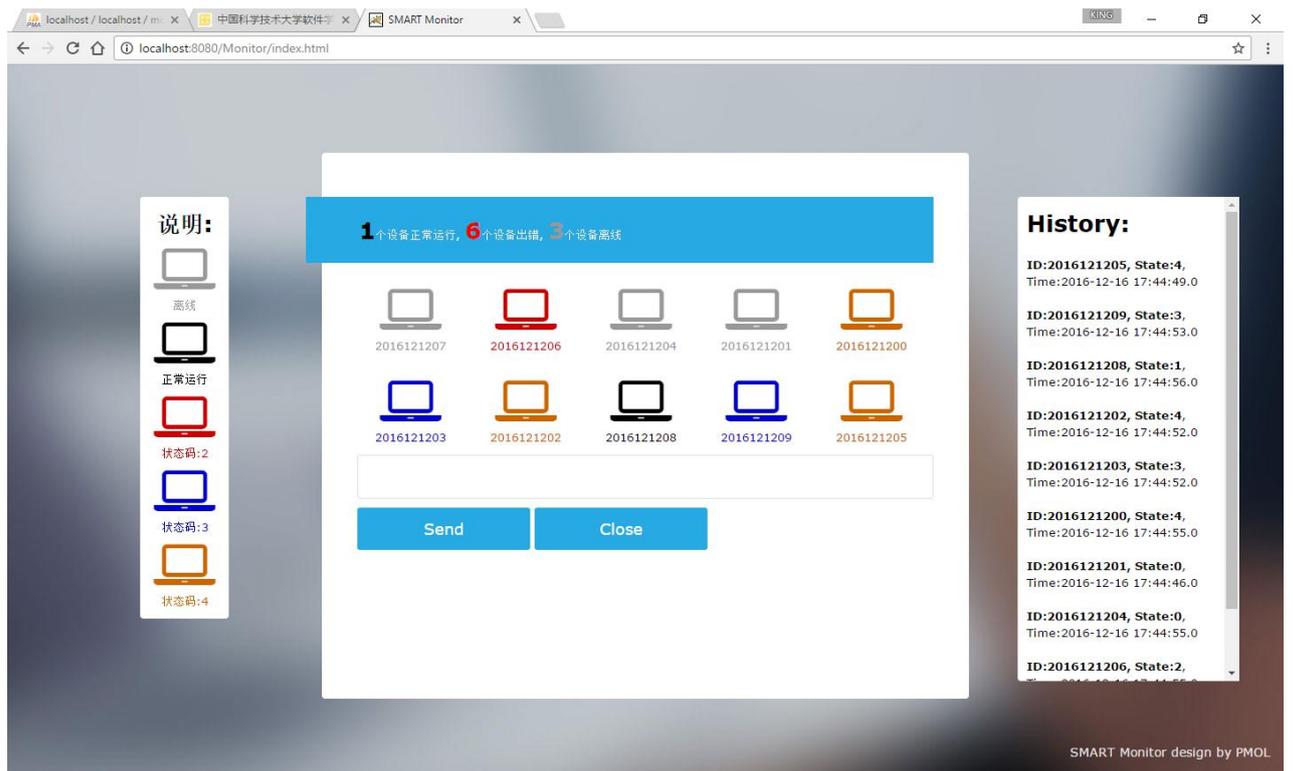


图 1.5.3. 设备开启后的主页界面

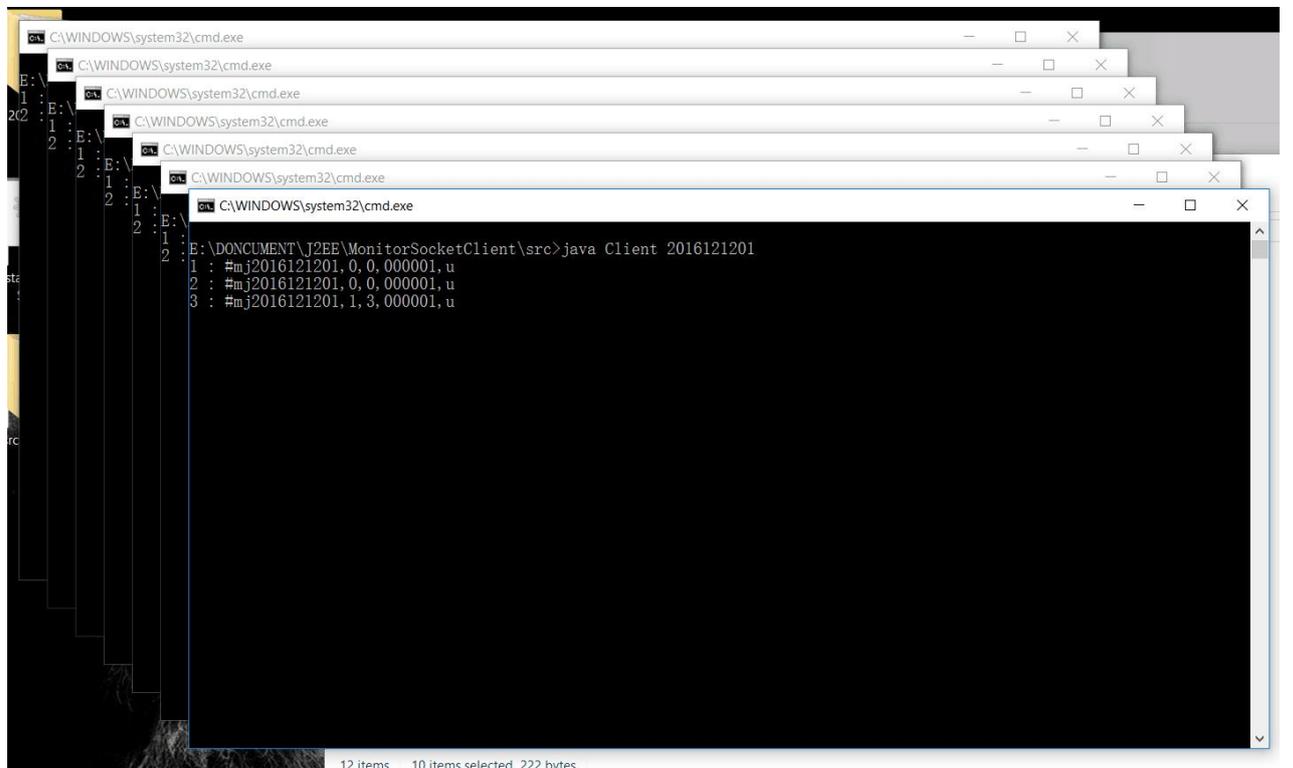


图 1.6. 模拟的设备界面（暂定 10 个）

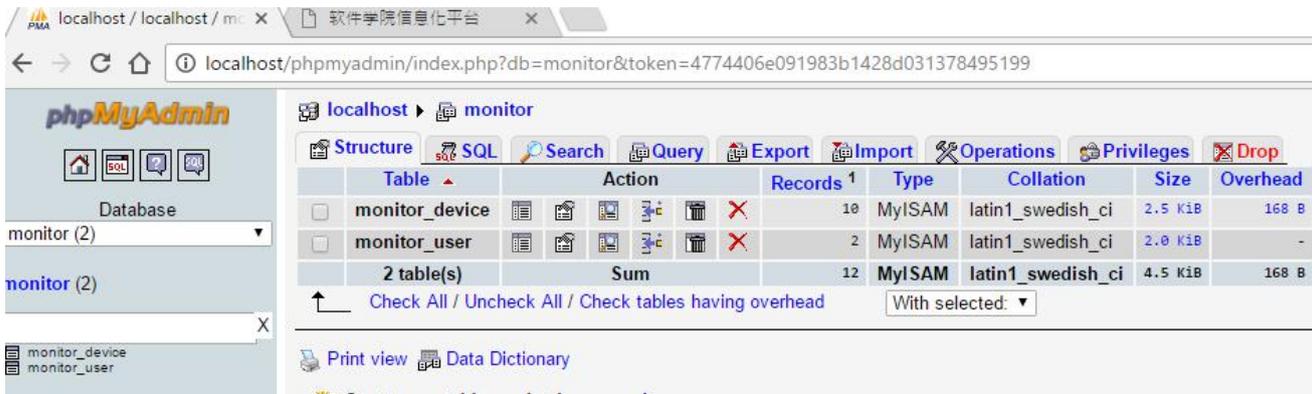


图 1.7.1. (通过 WAMP Server 界面管理工具查看的) 数据库概览图

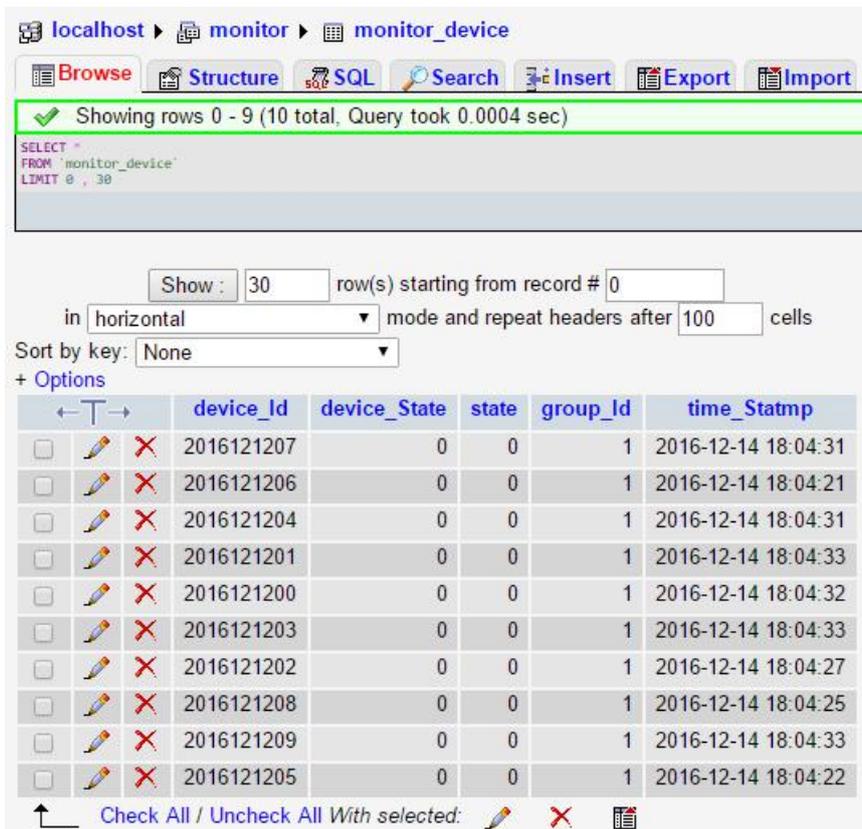


图 1.7.2. 数据库 monitor_device 表内数据展示

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，最后的时间戳 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。

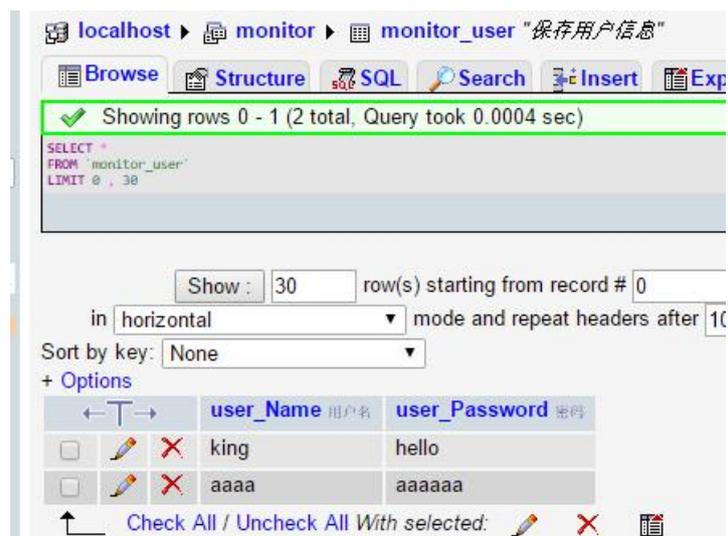


图 1.7.3. 数据库 monitor_user 表内数据展示

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。

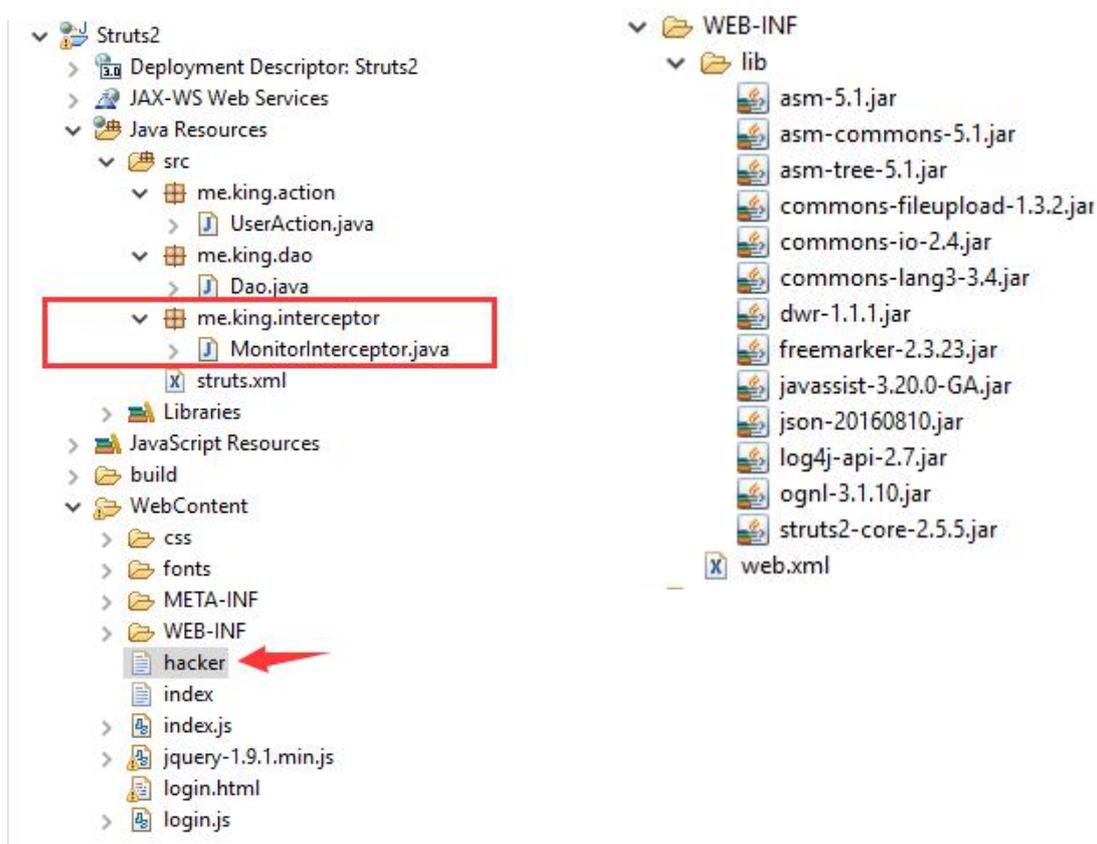


图 2.1.1. SMART Monitor 采用 Struts2 作为控制器后的文件组织结构图

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 目录下新建了三个包，一个专门用于保存 action，一个则用于保存 DAO，最后一个为拦截器包，自定义了一个 MoniorInterceptor 类作为拦截器。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，hacker 文件用于测试拦截器时返回的信息。

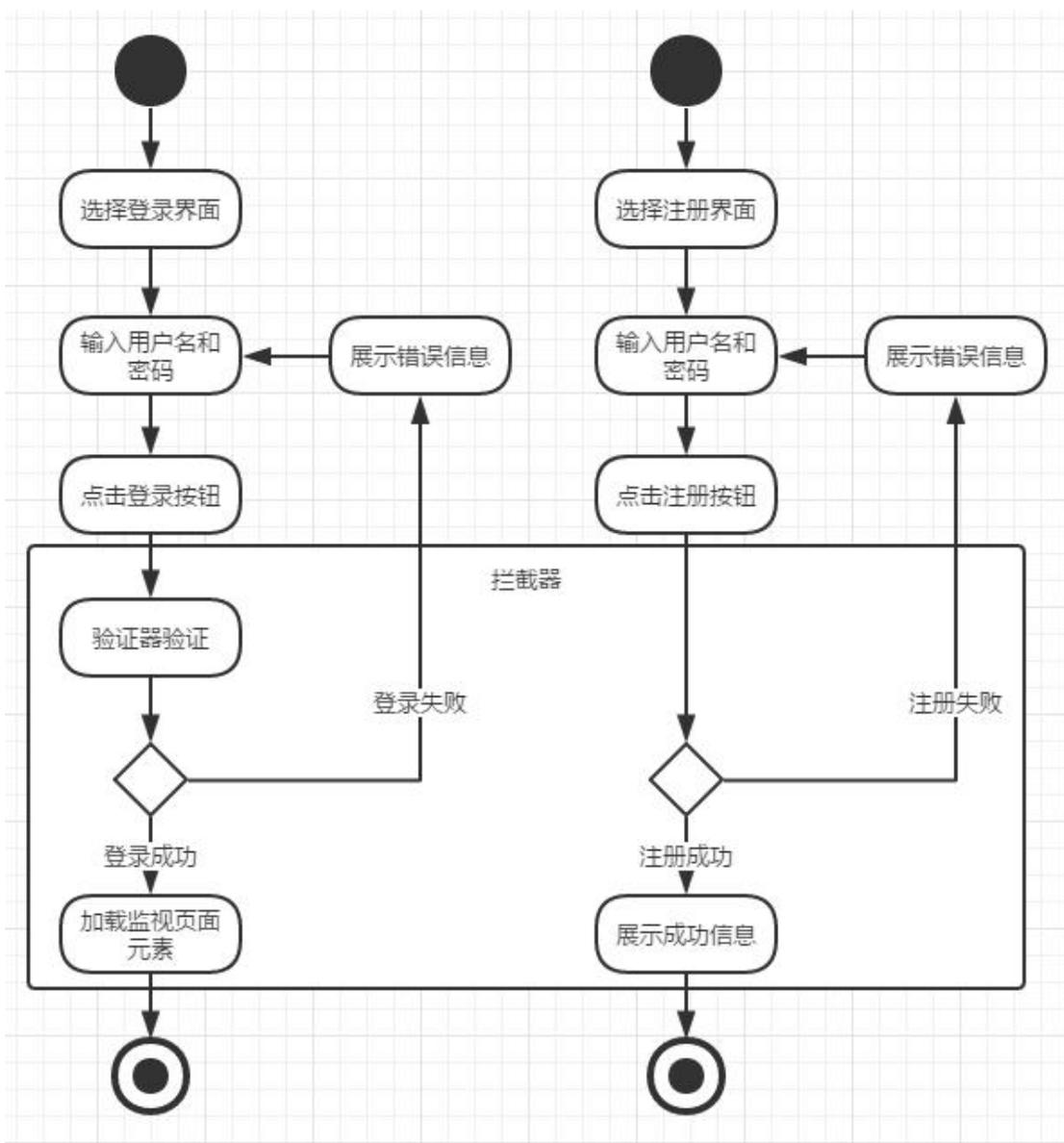


图 2.1.2. 登录和注册流程图

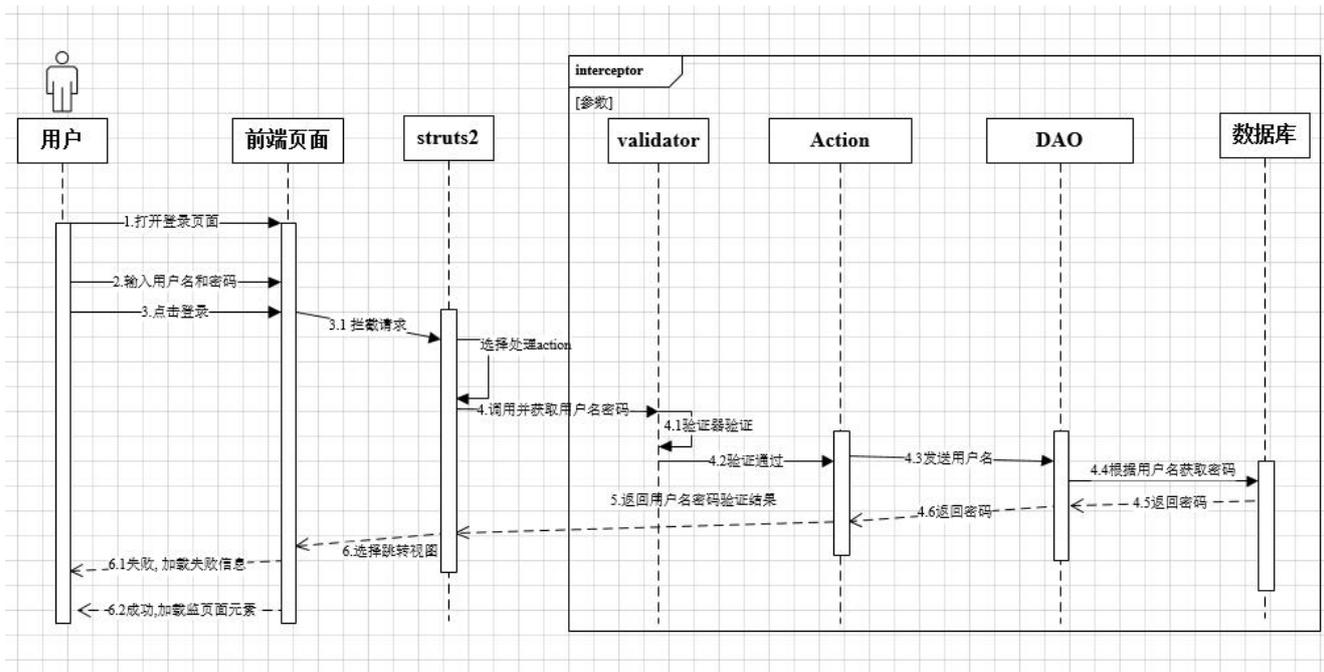


图 2.1.3. 登录时序图

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，即 struts2，剩下流程由控制器管理
- (4) struts2 框架将请求先由拦截器拦截，然后由验证器验证，通过后将请求分发到处理的 action 中。
- (5) action 通过 DAO 返回用户名对应密码并处理验证，将结果进行拦截，最后给控制器
- (6) struts2 框架根据 action 的输出类型字符串选择视图，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

```

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

如图 2.2.1 在 WebContent/WEB_INF/Web.xml 中配置着使用 struts2 拦截所有请求。

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN" "http://struts.
3 <struts>
4   <constant name="struts.i18n.encoding" value="UTF-8" />
5   <package name="SMART" extends="struts-default">
6
7     <interceptors>
8       <interceptor name="nameInterceptor" class="me.king.interceptor.MonitorInterceptor" />
9       <interceptor-stack name="myInterceptor">
10        <interceptor-ref name="nameInterceptor" />
11        <interceptor-ref name="defaultStack" />
12      </interceptor-stack>
13    </interceptors>
14    <default-interceptor-ref name="myInterceptor" /><!-- 所有action都需要先通过默认拦截器 -->
15
16    <action name="login" class="me.king.action.UserAction" method="login">
17
18      <result name="hacker"/></result> <!-- 用于测试拦截器的result -->
19      <result name="input"/></result> <!-- 用于测试验证器的result -->
20
21      <result type="plainText" name="success">
22        <param name="charSet">UTF-8</param>
23        <param name="location">/index</param>
24      </result>
25      <result type="stream" name="error">
26        <param name="charSet">UTF-8</param>
27        <param name="inputName">inputStream</param>
28        <param name="contentType">text/html</param>
29      </result>
30    </action>
31
32    <action name="register" class="me.king.action.UserAction"
33      method="regist">
34      <result type="stream">
35        <!-- 由getResult()返回输出结果的InputStream -->
36        <param name="charSet">UTF-8</param>
37        <param name="success">inputStream</param>
38        <param name="contentType">text/html</param>
39      </result>
40    </action>
41  </package>
42 </struts>

```

图 2.2.2 struts.xml 配置文件内容

在 src 目录下存放则 struts 的配置文件，配置信息如图 2.2.2，先配置自定义 interceptor，然后定义拦截器栈，最后定义默认拦截器。

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 4 个 result，默认 dispatcher 类型的 hacker 和 input，当返回字符串为“hacker”时，将/hacker 文件分发到前端。当为 input 的时候，将 validateLogin.jsp 分发到前端。

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 `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。（此处使用 `hacker` 文件测试）

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。

```
UserAction.java struts.xml login.js validateLogin.jsp
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 //执行login方法之前，会先执行此validateLogin方法
44 public void validateLogin() {}
51
52 //处理登录请求的方法
53 public String login() throws Exception {}
68 //处理注册请求的方法
69 public String regist() throws Exception {}
87 }
```

图 2.3.1 UserAction 类概览

```
43 //执行login方法之前, 会先执行此validateLogin方法
44 public void validateLogin(){
45     System.out.println("validateLogin");
46     if(getUsername().equals("error")) { //当用户名为error时, 表示错误则将错误添加到fieldErrors域中
47         addFieldError("error", "wrong info");
48         System.out.println("username == error");
49     }
50 }
51
52 //处理登录请求的方法
53 public String login() throws Exception {
54     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。
55     String sql = "select * from monitor_user where user_Name='"
56         + getUsername() + "'";
57     ResultSet rS = dao.executeQuery(sql); //定义sql语句并执行
58
59     if (rS.next()) { // 判断是否找到该用户
60         if (getPassword().equals(rS.getString(2)))
61             return SUCCESS; //找到用户并且密码正确, SUCCESS = success
62         else rt = "Error: Wrong Password"; //密码错误
63     } else rt = "Error: User Is Not Exist"; //用户不存在
64     //将返回的ajax信息保存在inputStream中
65     inputStream = new ByteArrayInputStream(rt.getBytes());
66     return ERROR; //只有登录不成功就采用ERROR类型的result, ERROR = error
67 }
68 //处理注册请求的方法
69 public String regist() throws Exception {
70     String rt;
71     String sql = "select * from monitor_user where user_Name='"
72         + getUsername() + "'";
73     ResultSet rS = dao.executeQuery(sql);
74
75     if (!rS.next()) { //首先判断用户是否存在
76         sql = "insert into monitor_user(user_Name,user_Password) values('"
77             + getUsername() + "','" + getPassword() + "')";
78         //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
79         int rs = dao.executeUpdate(sql);
80         if (rs > 0) rt = "Register Success"; //插入新用户成功
81         else rt = "Fail"; //数据库插入新数据失败
82     }
83     rt = "User Existed";
84     inputStream = new ByteArrayInputStream(rt.getBytes());
85     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
86 }
87 }
```

图 2.3.2 UserAction 类 validateLogin、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。

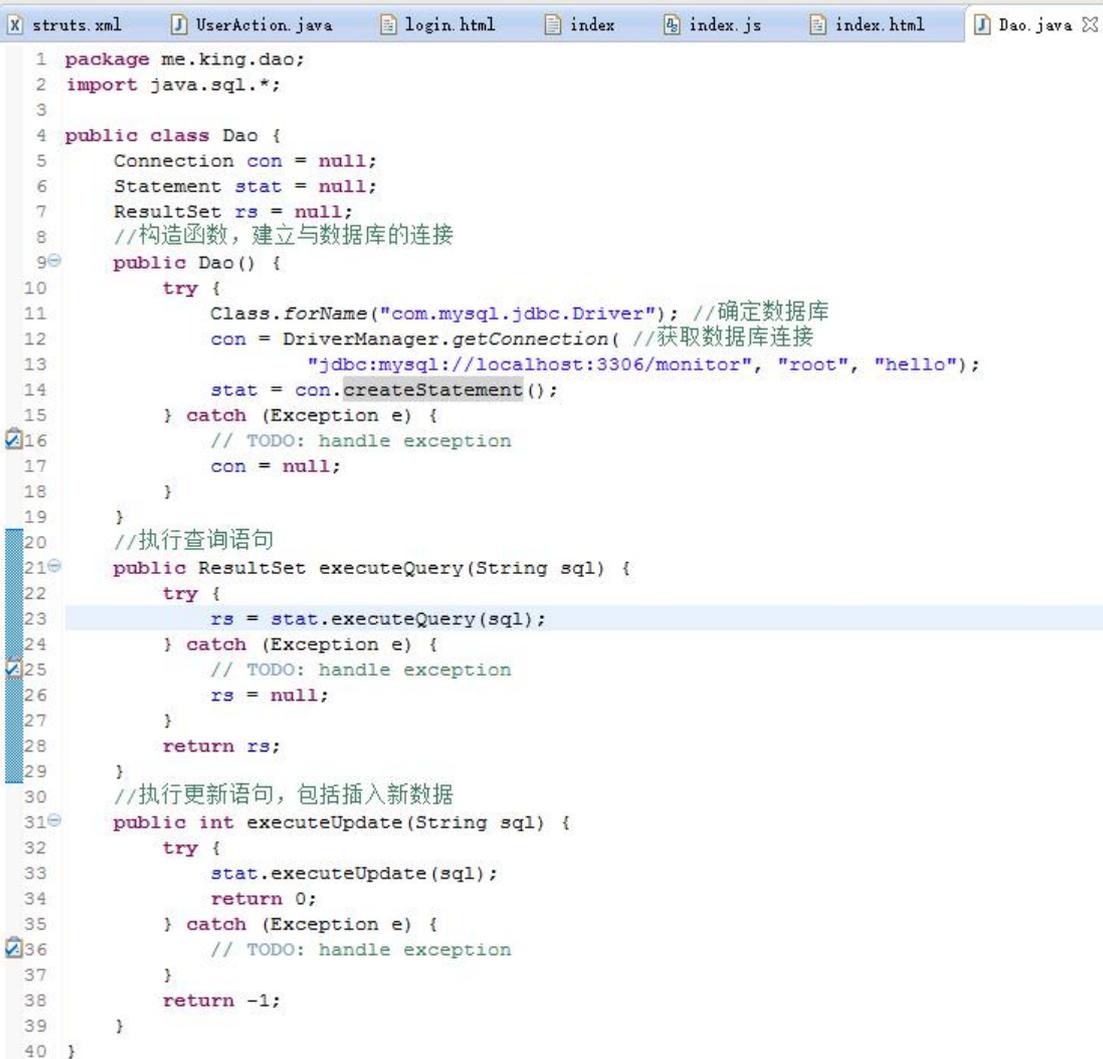
```
1 package me.king.interceptor;
2
3 import javax.servlet.http.HttpServletRequest;
4
5
6
7
8
9
10
11 public class MonitorInterceptor extends AbstractInterceptor{
12     private static final long serialVersionUID = 3874974286589880583L;
13
14     @Override
15     public String intercept(ActionInvocation invocation) throws Exception {
16         // TODO Auto-generated method stub
17         System.out.println("before invoke()");
18         // 取得请求相关的ActionContext实例
19         ActionContext ctx = invocation.getInvocationContext();
20         HttpServletRequest request= (HttpServletRequest) ctx.get(StrutsStatics.HTTP_REQUEST);
21         if(request.getParameter("username").equals("hacker")){ //成功拦截hacker
22             return "hacker";
23         }
24
25         String rt = invocation.invoke(); //执行action 或后续的拦截器,将结果保存至rt中
26         System.out.println("after invoke() " + rt);
27         return rt;
28     }
29 }
```

图 2.3.3 MonitorInterceptor 类截图

图2.3.3 MonitorInterceptor 类,本次实验自定义的 MonitorInterceptor 拦截器的流程为:

1. 输出辅助信息 before invoke()
2. 获取 ActionContext 实例 并从中获取 request 对象
3. 从 request 对象中获取登录用户, 并验证
4. 当发现登录用户名为 hacker (简单模拟无法通过拦截器的用户) 则直接返回字符串 "hacker"
5. 若为授权用户, 则执行 action 或 后续的拦截器,将结果保存至 rt 中 (rt 为标识 action 或其他拦截器的返回类型字符串)
6. 输出辅助信息 after invoke()以及 rt 中的结果字符串
7. 最后将 rt 返回。

具体测试截图请看图 2.7.4 测试登录失败/成功后服务器端控制台输出的信息



```
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

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。

```

hacker  UserAction.java  struts.xml  MonitorInterceptor.java  index  login.js
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 style="font-size:1.3em;">Example:</b><br>
13     <ul>
14       <li><a href="javascript:void(0)" title="Offline"><i class="fa fa-lapto
15       <li><a href="javascript:void(0)" title="Running"><i class="fa fa-lapto
16       <li><a href="javascript:void(0)" title="State:2"><i class="fa fa-lapto
17       <li><a href="javascript:void(0)" title="State:3"><i class="fa fa-lapto
18       <li><a href="javascript:void(0)" title="State:4"><i class="fa fa-lapto
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

```

hacker  UserAction.java  struts.xml
1 Error: You are a hacker!

```

图 2.5.2 hacker 文件内容

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 2.5.3。

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。所以已经在 index.html 中注释掉 index.js 脚本的加载标签了。

```
struts.xml  UserAction.java  login.html  index  index.js  index.html  Dao.java  web.xml
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 2.5.3 index.js 部分脚本内容概览

```
UserAction.java  struts.xml  login.js  validateLogin.jsp
1  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2  pageEncoding="ISO-8859-1"%>
3  <%@ taglib prefix="s" uri="/struts-tags"%>
4  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
5  <html>
6  <head>
7  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8  <title>Insert title here</title>
9  </head>
10 <body>
11 <s:fielderror/>
12 </body>
13 </html>
```

图 2.5.4 validateLogin.jsp 内容概览

validateLogin.jsp 主要是为了测试验证器而使用的输出页面，与 hacker 和 index 不同，由于验证器使用到了 addFieldError 方法，所以需将结果页面指定为 jsp，而且也需 struts2 的标签库配合使用。

```

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

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。

```

56 $(document).ready(function() {
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
64             if(data.indexOf('Error') === 0) // 若信息以Error开头
65                 $('#login_tips').text(data);
66             else if(data.indexOf('errorMessage') !== -1) // 当返回的信息中由errorMessage时表示未通过验证器; 获取wrong info
67                 $('#login_tips').text( data.substring(data.indexOf('<span>')+6, data.indexOf('</span>')) );
68             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
69                 $('#head').html('');
70                 for(var i = 0; i<15; ++i)
71                     $('#head').append($(data)[i]);
72
73                 $('#body').html('');
74                 for(var i = 15; i< $(data).length; ++i)
75                     $('#body').append($(data)[i]);
76             }
77         });
78     });
79 }

```

图 2.6.2. login.js 中登录 ajax 代码

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。当响应信息是以 Error 开头的字符串时，则表示返回的是出错信息，则直接将信息更新到前端页面。否则加载监视页面元素标签。

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。

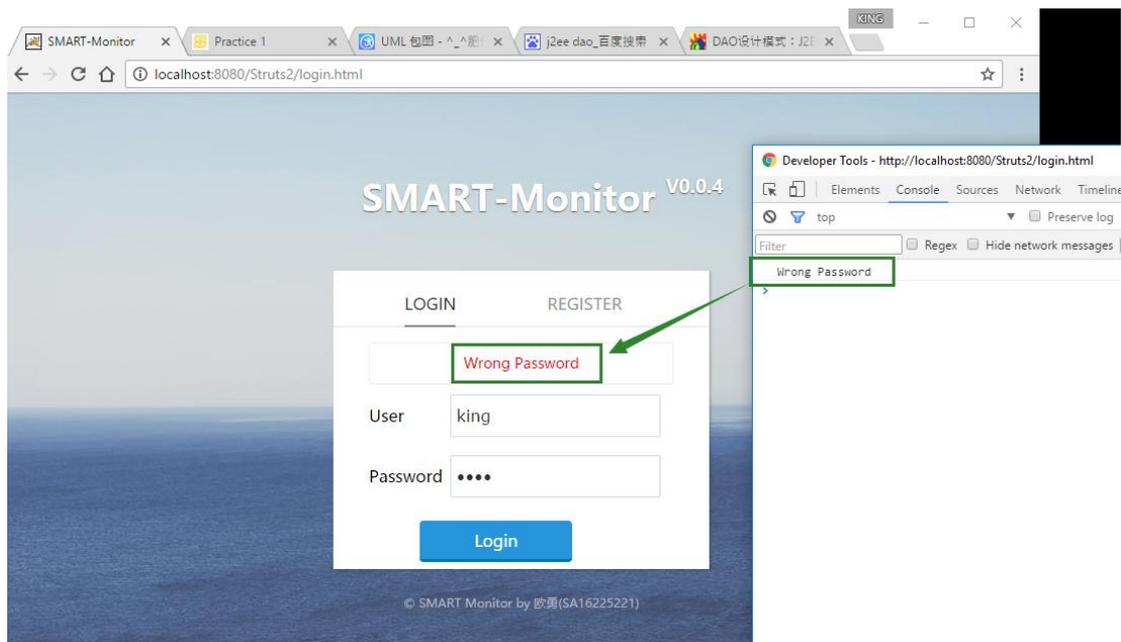


图 2.7.1 测试登录失败的提示

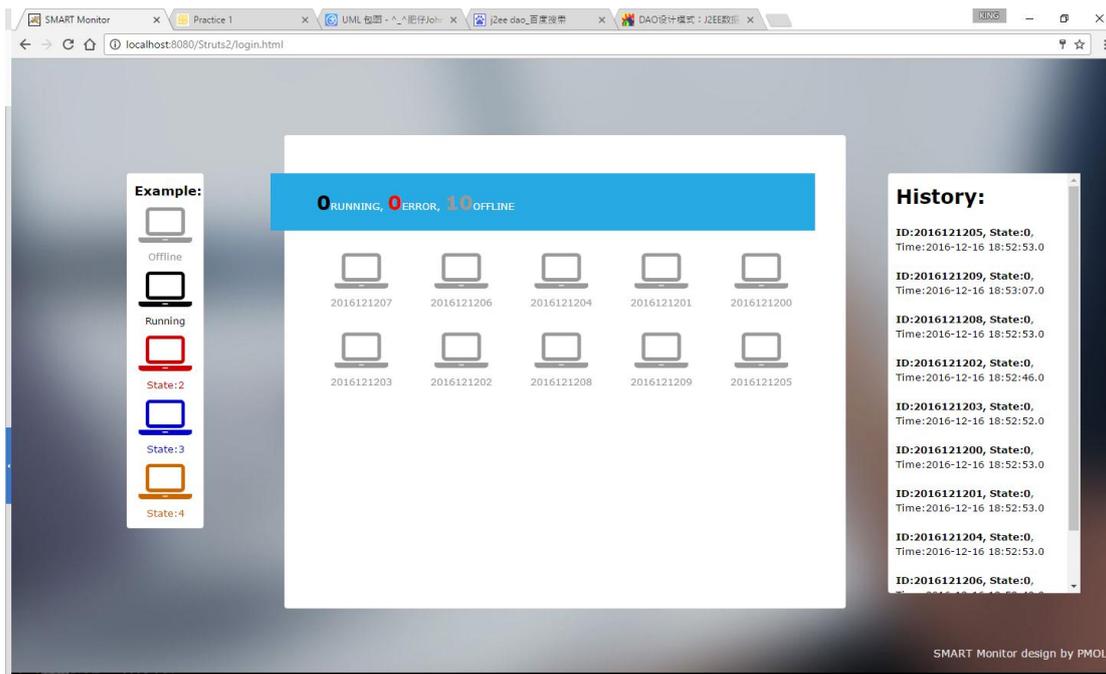


图 2.7.2 测试登录成功后跳转

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

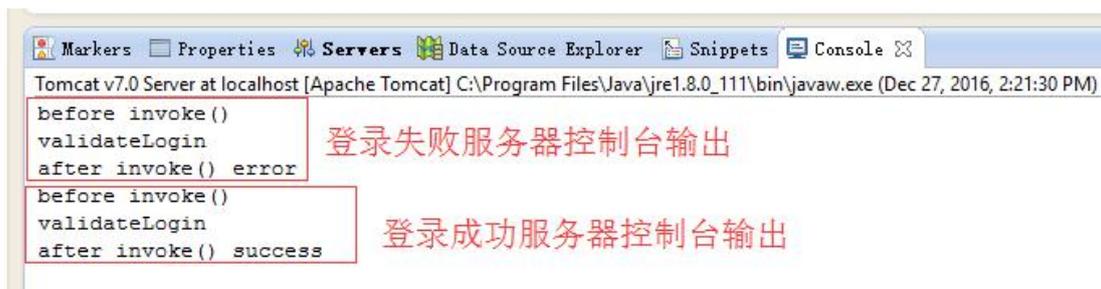


图 2.7.4 测试登录失败/成功后服务器端控制台输出的信息

可以看出当登录成功/失败的时候，是正确调用拦截器、验证器的，关于拦截器具体代码请看图 2.3.3 MonitorInterceptor 类截图。

关于验证器具体请看图 2.3.2 UserAction 类 validateLogin、login 和 regist 方法截图

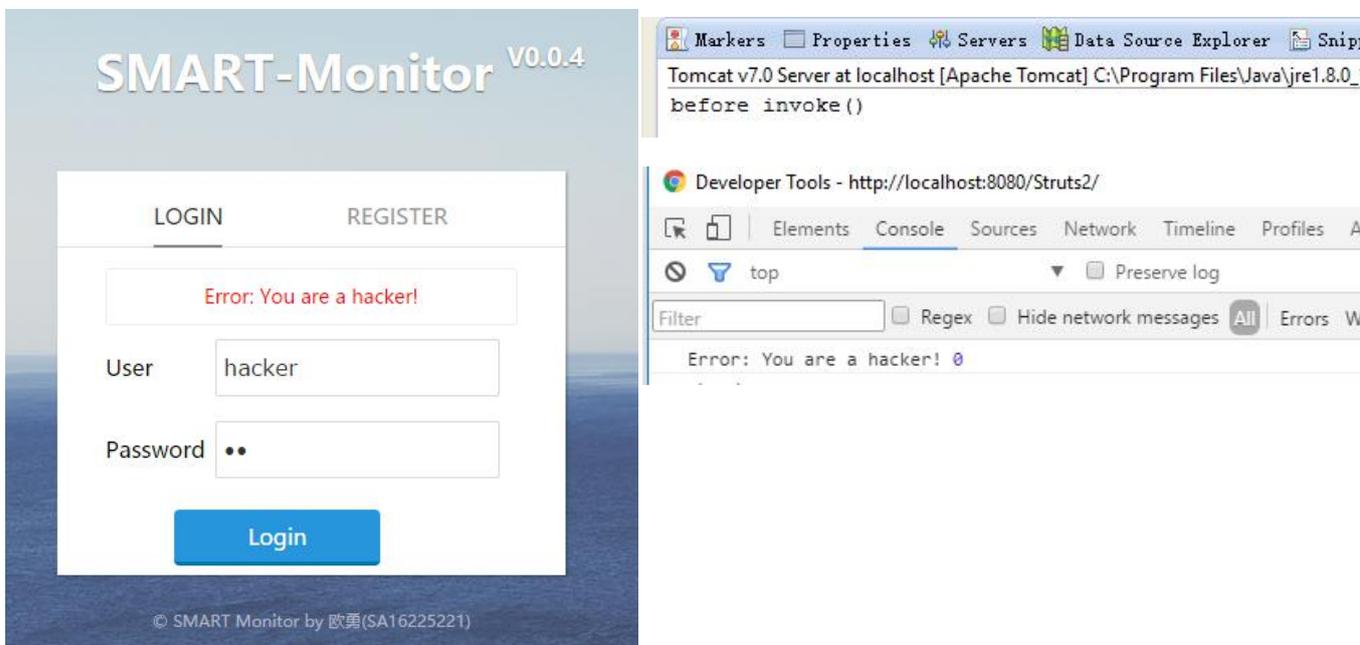


图 2.7.5 使用 hacker 作为用户名登录测试拦截器

由图 2.7.5 可以看出，当使用 hacker 登录时，前端页面收到返回响应信息为 Error: You are a hacker!即 hacker 文件中的内容，同时服务器端控制台仅仅只输出 before invoke()就返回了，而没有进入执行 UserAction 类的登录 login 方法。

