

目录

1、 实验内容概要.....	1
1) 实验目的.....	1
2) 实验要求.....	1
3) 实验设备.....	1
4) 实验原理.....	2
5) 实验步骤.....	2
2、 系统需求分析.....	2
3、 系统概要设计.....	5
4、 数据库设计.....	21
5、 系统详细设计.....	23
7、 总结.....	42

1、实验内容概要

1) 实验目的

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、配置设备、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。

2) 实验要求

基于课上内容以及自己的理解构建 SNS 网络的架构视图，包括：

逻辑视图，主要为类图

进程视图，包括活动图，时序图

部署视图

开发视图，主要为架构示意图

用例图

以及对视图进行必要的描述。

3) 实验设备

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)

IDE : Eclipse

系统 : Win10 64-bit

JDK版本 : 1.8

Web服务器 : Tomcat 7.0.47

4) 实验原理

利用Socket客户端模拟多个主机，同时通过随机数模拟客户端的状态改变。

5) 实验步骤

1.系统需求分析

2.系统概要设计

3.系统详细设计

4.编码实现

5.测试

2、 系统需求分析

•简介

—分布式监控管理系统，方便用户随时了解分布式集群的情况（机器的状态）以及及时进行各种机器的操作。

—Google 云计算、Hadoop 等平台的重要组成部分。

•功能

—PC 端

实现用户登录，实时监控各设备节点状态、配置设备、接收节点状态变更推送等功能。

—分布式机器端

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。

•非功能需求

—性能需求

响应时间迅速，数据传输快速，用户能在 1s 内查看到所有设备的当前状态，同时设备实时状态的变更也需要在 1s 内传输到前端页面显示完成。快速响应保障了发出的管理指令能够被尽快执行，即便执行指令需要较长的时间，也能较准确地把当前状态告知管理员，例如数据备份时需要显示备份的进度。

—可扩展性需求

链接的站点能够扩展到几千个，能满足多用户并发访问，在网络应用系统功能配置上一方面要全面满足当前及可预见的未来一段时间内的应用需求，另一方面要能方便地进行功能扩展，可灵活地增、减功能模块。

-灵活性需求

提供外部访问接口，能够实现后期如手机应用、外部应用的调用。程序升级简单，使用接口沟通前后端，WebSocket 服务器端采用 CopyOnWriteArraySet <WebSocket> 保存每个客户端对应的 WebSocket 对象，这是一种线程安全的集合类型，能较为容易的扩展客户端的类型以及数量。同时，采用 socket 模拟设备，采用 I/O 多路复用或多线程的 Socket 服务器端能轻易支持多个客户端的同时连接，并且只修改监听的端口地址即可扩展出新的 socket 服务器，轻易达到分布式的要求。通过预先定义的状态码与颜色对应关系，使设备状态以不同的颜色展示出来，可以轻易修改这种颜色映射。

-稳定性需求

多个 PC 客户端或移动客户端能同时连接并查看设备状态，同时设备掉电或主动离线时需要实时显示在监控页面上。

-可管理性需求

全面深入地了解系统的运行状况、定期做系统维护以降低系统故障率、发现故障或系统瓶颈并及时修复、根据业务需求调整系统运行方式、根据业务负载增减资源，以及保证系统关键数据的安全等。大多数系统管理任务由系统管理员通过使用一系列管理工具来完成，少数管理任务需要领域专家的参与，另外一些任务可由管理系统自动完成。

-安全性需求

禁止非法登录或未经登录就查看设备状态，请求需要过滤并验证。不使用 cookie 保存用户名密码功能。

3、系统概要设计

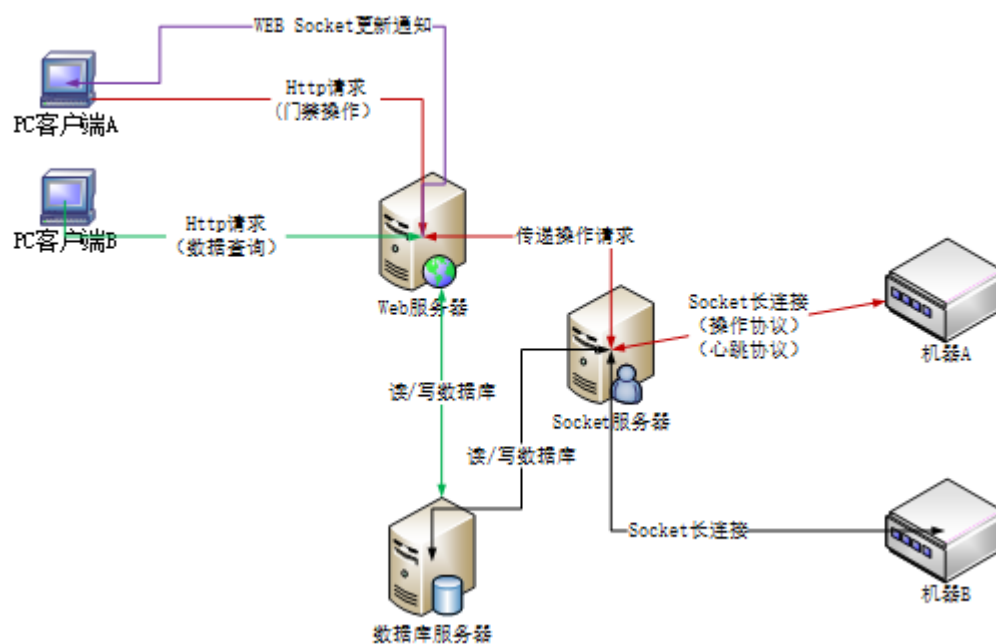
1) 分布式机器端不断向 socket 服务器发送自身设备信息 (类似心跳)。(心跳也是数据通信中的一种数据,特殊点在于定时发送,形似心跳而得名。一般来说,当客户端连接到服务端之后,为了确保了解到连接的状态真实性,或者为了防止某些网络在长时间没有数据传输时自动断开,服务端会定时发送一条数据(一般数据内容为空)给客户端。如果在一定时间内(一般选择发送3次心跳的间隔)都没有收到客户端的回复,那么就认为该客户端已经断开了,此时应该踢掉它。)

2) 由于分布式机器端是不断地向 socket 服务器发送设备信息的,socket 服务器要不断的访问数据库,这里为了减轻数据库的开销,我们可以在 socket 服务器与数据库之间分配一个缓冲池。(缓冲池是数据库连接池允许应用程序重用已存在于池中的数据库连接,以避免反复的建立新的数据库连接。这种技术能有效提高应用程序的伸缩性,因为有限的数据库连接能够给大量的客户提供服务。提高了系统性能,避免了大量建立新连接的开销。当打开一个数据库连接时,一个数据库连接池也就创建了。数据库连接池的创建与数据库连接字符串精确的相关(包括空格、大小写)。所有的连接池是根据连接字符串来区分的。在创建一个新的数据库连接时,如果连接字符串不完全相同,将创建不同的连接池。一旦数据库连接池被创建,它将一直存在直到该进程结束。维护一个非活动状态的连接池几乎不需要什么系统开销。)

3) 客户端向 web 服务器发送登录请求并发起数据查询的 WebSocket 请求。

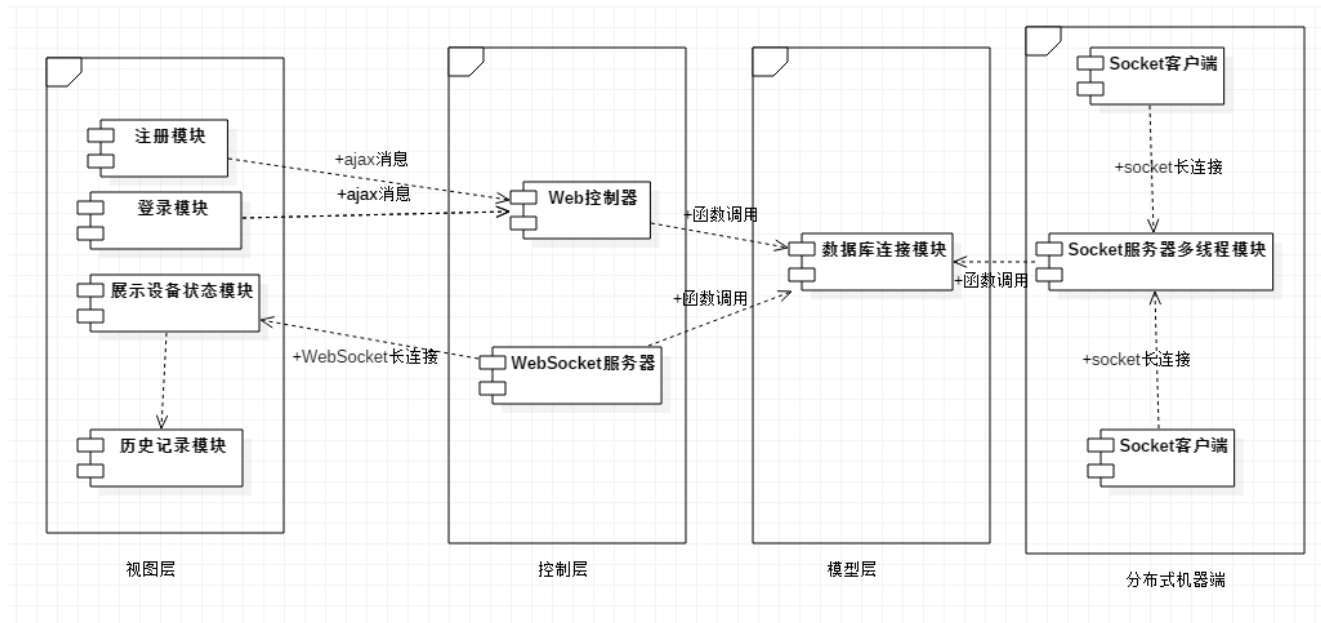
4) WebSocket 服务器直接访问数据库。同时像 PC 客户端实时推送设备数据。

5) PC 客户端从 WebSocket 服务器端获取数据后用于展示,同时将历史数据信息保存在界面的 History 栏中,为了及快速传输,我们选择只保存本客户端请求的连接后开始的历史记录(既得到的历史信息以连接开始的时间为起点)。



物理视图

此图为项目的物理视图，展示了每种服务器或设备在物理上的部署情况。



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 服务器和客户端采用分布式的方式。Socket 服务器的实现不止一种，分别是多线程和 I/O 多路复用。而 Socket 本身就不依赖于特定的平台和语言，更是天生的分布式应用。

1) MVC 概念

MVC(Model View Controller) ,分别表示 M 模型(model) - V 视图(view) - C 控制

器(controller) ,

MVC 是一种软件框架模式,用一种业务逻辑、数据、界面显示分离的方法组织代码,将业务逻辑聚集到一个部件里面,在改进和个性化定制界面及用户交互的同时,不需要重新编写业务逻辑。MVC 被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。

2) MVC 中 Controller 的作用与实现

控制层在服务器端,作为连接视图层(比如用户交互的界面)和模型层(处理业务、提供服务)的纽带,对客户端 request 进行过滤和转发,决定由哪个类来处理请求,或者决定给客户端返回哪个视图,即确定服务器的 response 相对应的视图。

一个 Controller 其实本质上也是一个 servlet,它的实现需要符合 servlet 的标准,即继承 HttpServlet 类,覆写 doGet 或 doPost 等方法,同时可以定义过滤器。

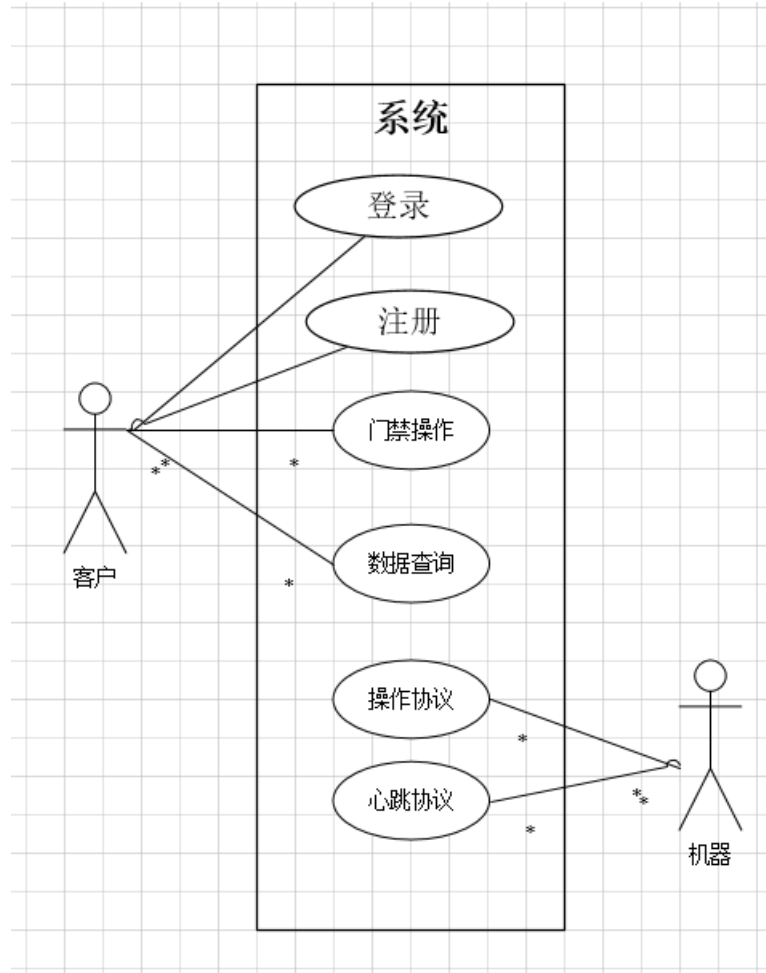
3) MVC 中 View 的作用与实现

视图层是真正与客户端交互的界面,一般为 jsp 动态页面或 html 静态网页,视图层的数据可以来自用户(用户输入数据通过请求发送到服务器端)和模型层(服务器返回请求的数据),但是如何展示这些数据却是由视图层自己控制和安排的,所以可以根据用户的需求来改变展示的方式和形式。

4) MVC 中 Model 的作用与实现

模型层才是真正处理用户请求的地方,所以模型层会很庞大(因为业务的种类很多,需要处理和计算的任务也会有很多),所以通常还会对模型层进行再分层。比如与

数据库的连接和对数据进行处理等。



用例图

客户的用例场景有以下 4 种：

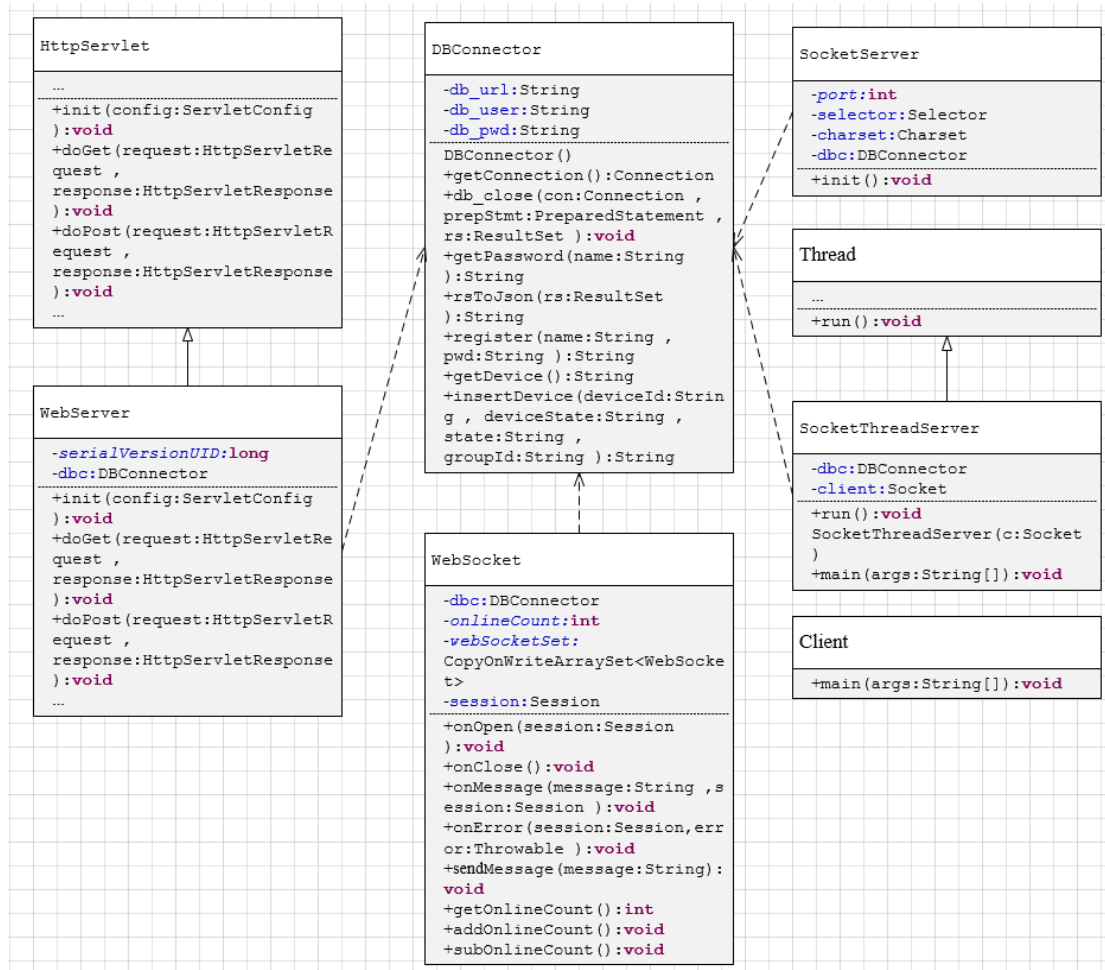
- 1). 登录，需要正确输入用户名和密码，然后跳转至监控设备页面（即主页），此时转到数据查询场景（即查看设备的状态）。

-
- 2). 注册，新用户能通过注册查看设备状态，需要输入用户名（未被占用），密码，并二次确认密码。注册成功则跳转至主页。
 - 3). 门禁操作，即通过输入固定格式的命令，可以控制某个设备的状态。
 - 4). 数据查询，当跳转主页后，自动发起 WebSocket 连接，此时服务器将自动将设备的最新状态推送给用户。

机器的用例场景有 2 种：

心跳协议和操作协议，其中心跳协议能自动将当前设备的状态发送至 socket 服务器端。

操作协议能使用户对本设备进行配置和更改状态。



类图

其中，DBConnector 类是模型层的实现，是整个项目的关键存取，

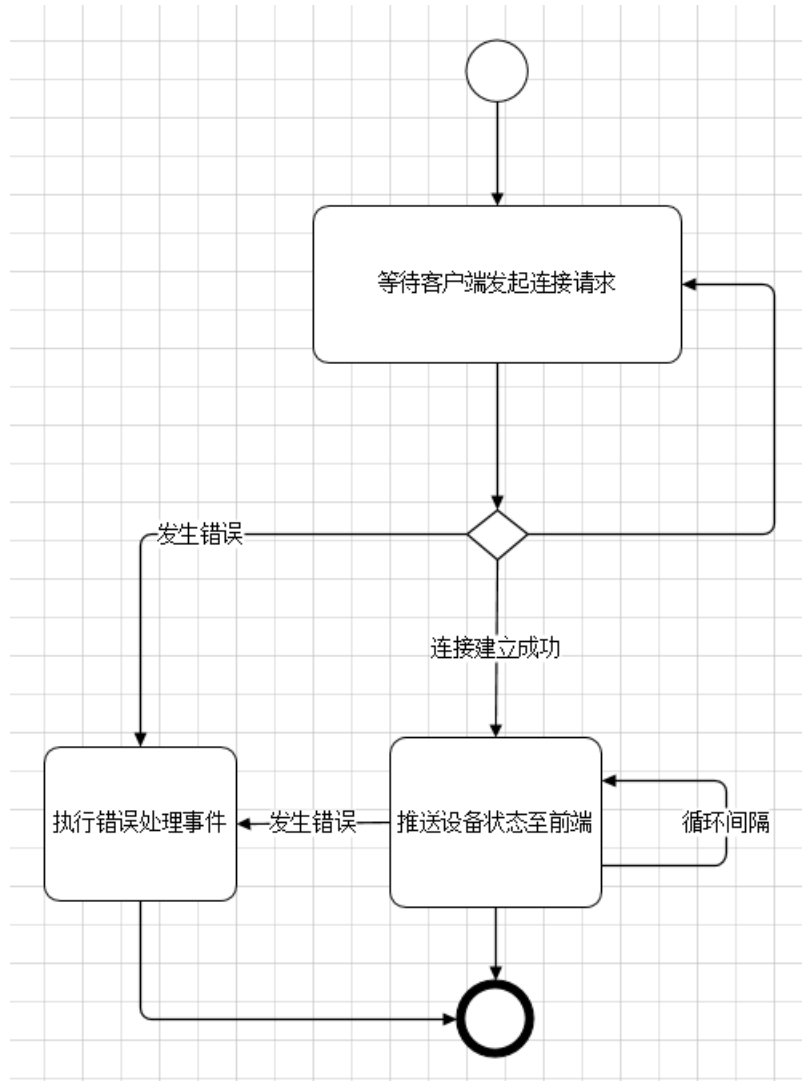
SocketServer 类是用 I/O 多路复用方式的 Socket 服务器端的实现，SocketThreadServer

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

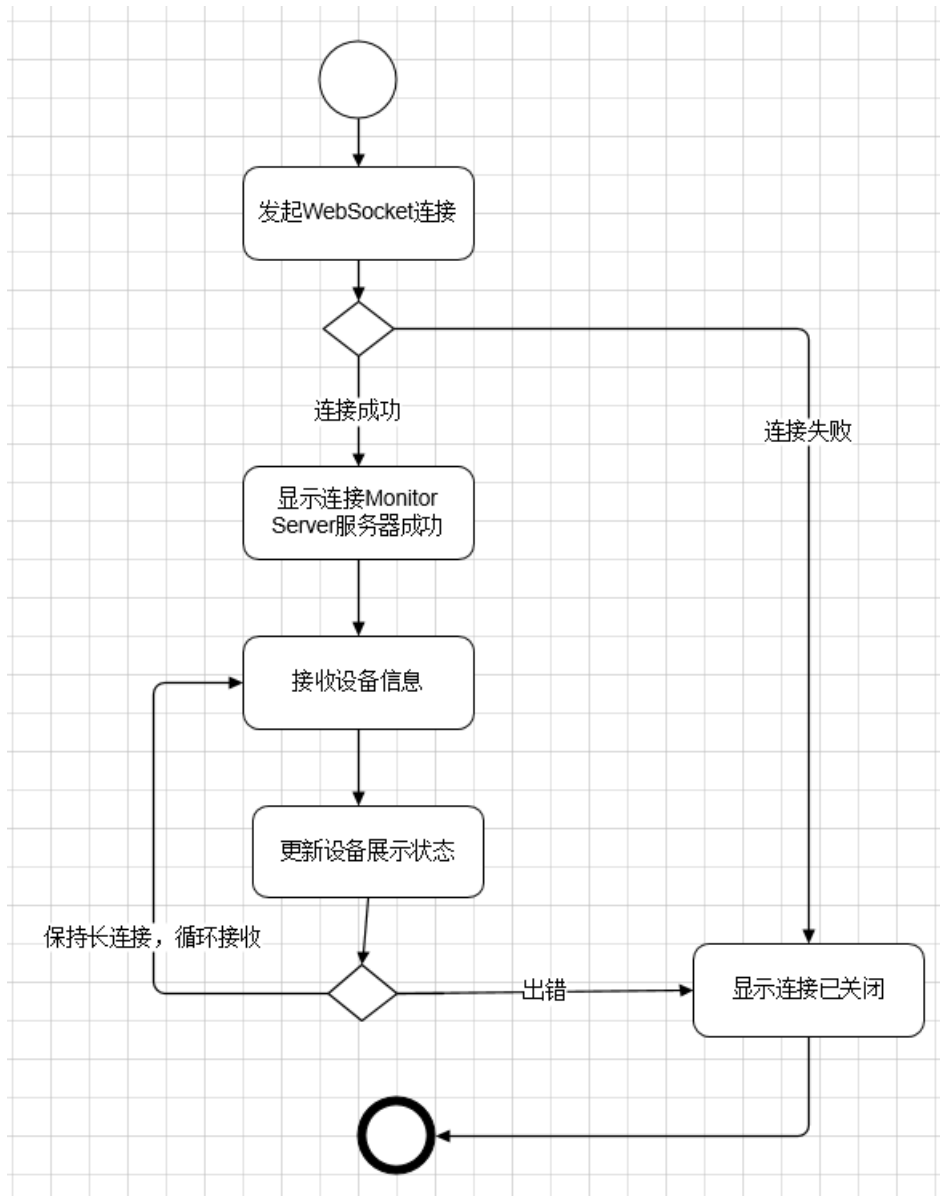
滤，WebSocket 类是 WebSocket 的服务器，是控制层的一部分。



WebSocket 服务器流程图

WebSocket 流程图：

- (1) 首先等待客户端发起连接请求
- (2) 开始建立连接，若连接失败则回到(1)重新进行等待连接并且执行错误处理事件
- (3) 若连接成功后将推送设备状态到前端页面并且在一定时间间隔内进行循环实现实时推送。若发生错误则执行错误处理事件



监控设备状态页面流程图

监控设备状态页面流程图：

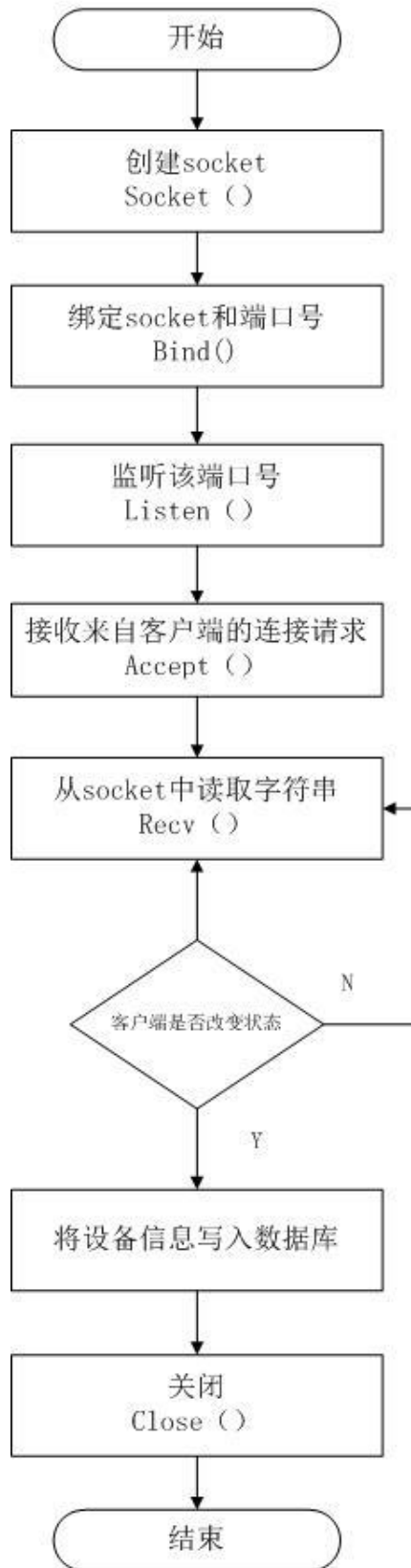
(1) 发起 WebSocket 连接

(2) 如果连接失败则显示连接已关闭并结束监控设备，若连接成功则显示连接 Monitor Server 服务器成功并转向 (3)

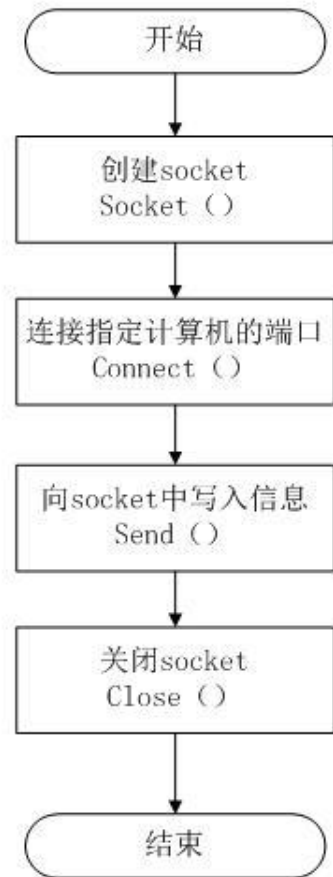
(3) 接受 socket 客户端的信息

(4) 进行更新设备的信息，并重复(3)使其保持长连接，进行循环接收设备信息，实时展示设备状态。若出现错误则示连接已关闭并结束监控设备

Socket服务器流程图



Socket客户端流程图



socket 服务器流程：

- (1) 服务器根据地址类型，socket 类型、协议创建 socket
- (2) 服务器为 socket 绑定 ip 地址和端口号
- (3) 服务器 socket 监听端口号请求，随时准备接收客户端发来的连接，此时服务器的 socket 并没有被打开
- (4) 服务器 socket 接收到客户端 socket 请求，被动打开，开始接收客户端请求，直到客户端返回连接信息。此时 socket 进入阻塞状态，所谓阻塞即 accept()方法一直到客户端返回连接信息后才返回，开始接收下一个客户端的请求
- (5) 服务器 accept 方法返回，连接成功
- (6) 服务器读取字符信息
- (7) 服务器判断客户端是否改变状态，若改变则将设备信息写入数据库，否则转向(6)
- (8) 服务器端关闭

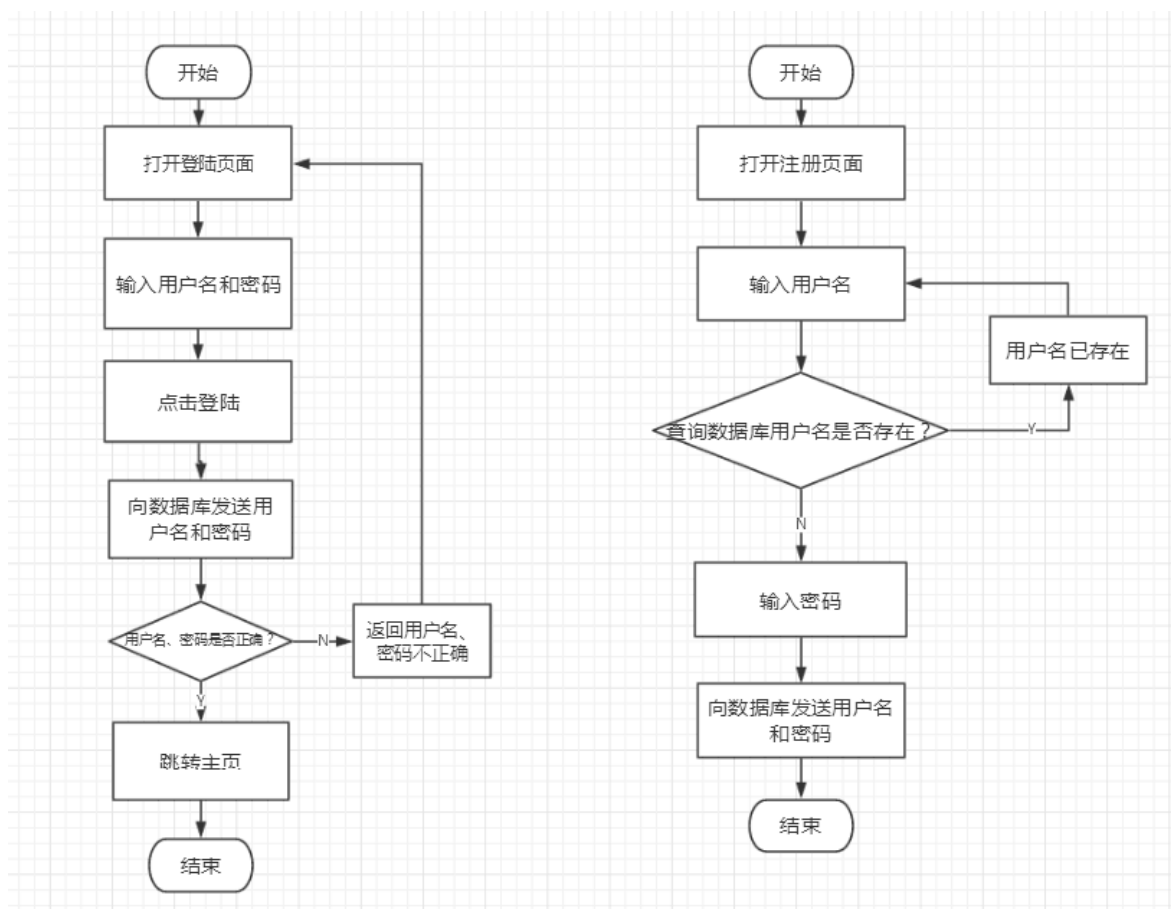
socket 客户端流程：

- (1) 客户端创建 socket
- (2) 客户端打开 socket，根据服务器 ip 地址和端口号试图连接服务器 socket

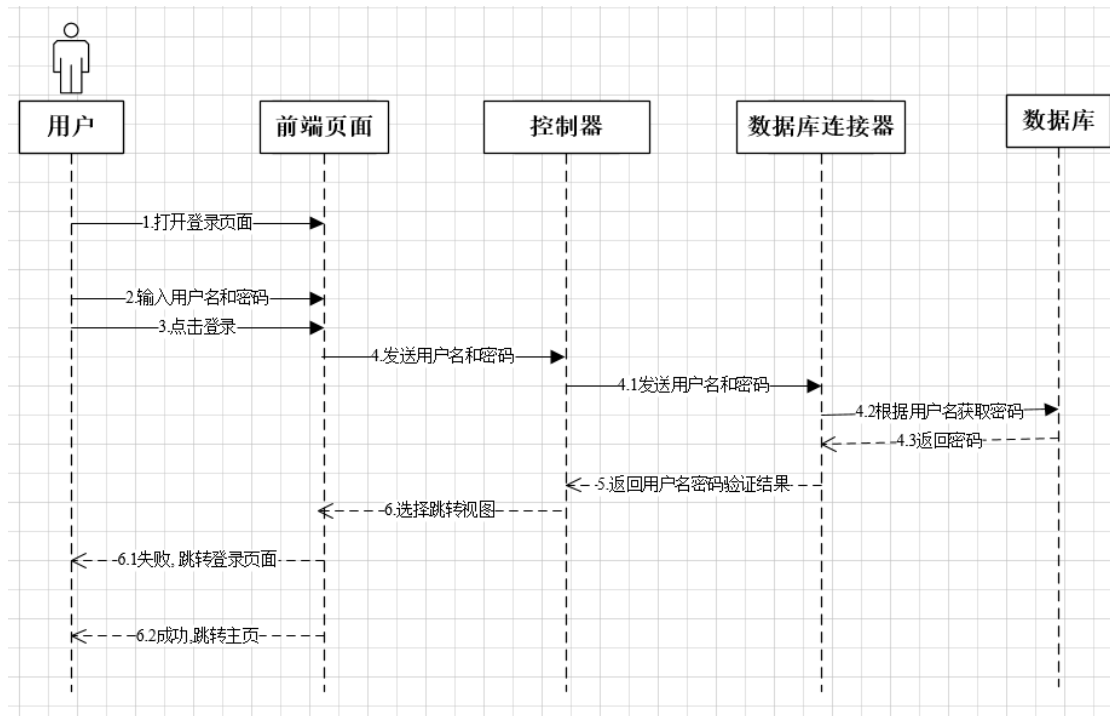
(3) 客户端连接成功，向服务器发送连接状态信息

(4) 客户端向 socket 写入信息

(5) 客户端关闭（服务器自动将其转为离线状态）



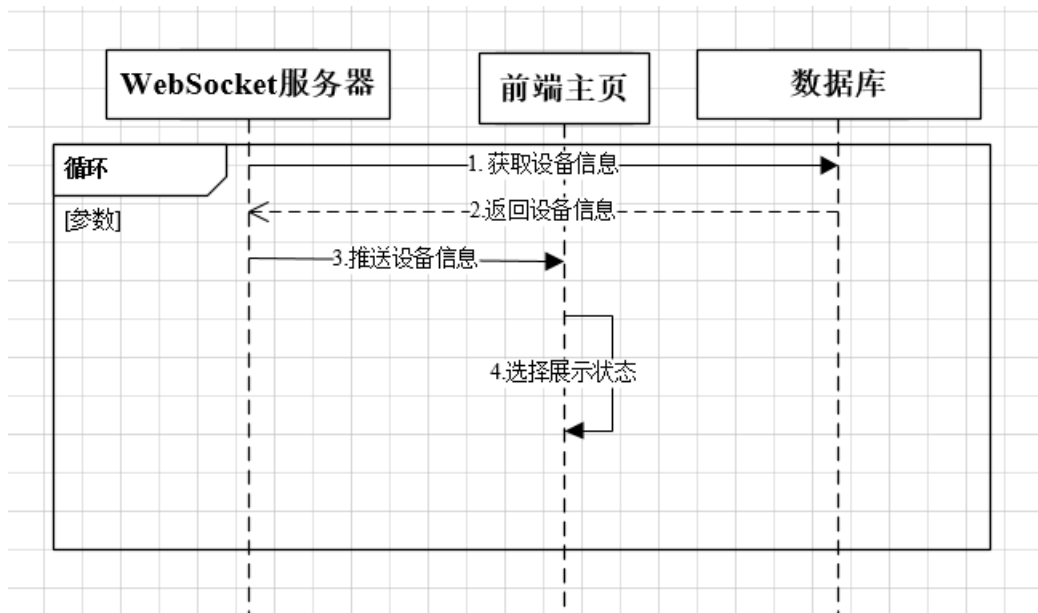
登录和注册流程图



登录时序图

登录时序图：

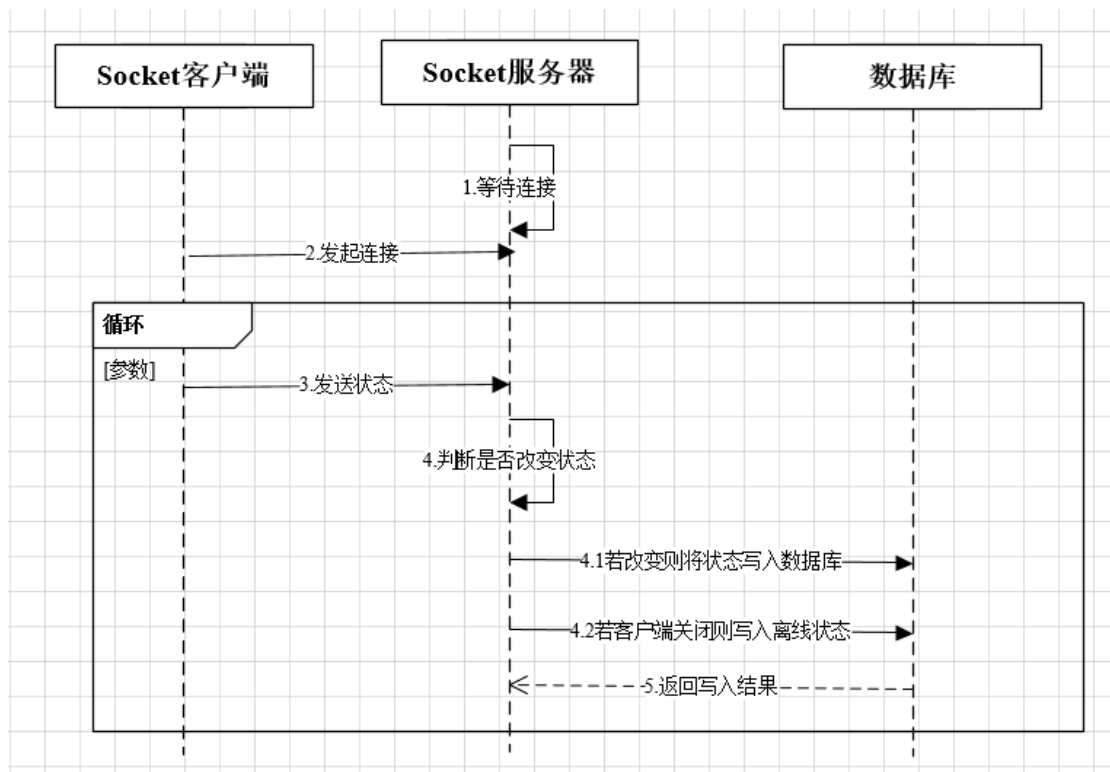
- (1) 用户打开登录界面
- (2) 用户输入用户名和密码并点击登录
- (3) 前端提交用户名和密码给控制器，并通过控制器发送给数据库连接器
- (4) 数据库连接器根据用户名获取数据库中返回的密码
- (5) 数据库连接器返回用户名验证结果给控制器
- (6) 如果用户名和密码配对则成功，跳转到主页面；如果失败，则跳转到登录界面



前后端交互时序图

前后端交互时序图：

- (1) WebSocket 服务器通过数据库获取设备信息
- (2) 被访问的数据库将所需要的设备信息返回到 WebSocket 服务器
- (3) 收到设备信息的服务器将数据推送到前端页面
- (4) 前端页面收到数据之后进行筛选并展示出当前状态，体现出实时性



Socket 服务器客户端交互时序图

socket 服务器客户端交互图：

- (1) 服务器等待客户端发送连接
- (2) 客户端给服务器发起连接
- (3) 客户端发送自身状态给服务器
- (4) 服务器判断客户端是否改变状态，如果改变则将状态写入数据库，如果客户端关闭则写入离线状态到数据库

(5) 数据库返回写入结果给服务器

重复操作 (3) (4) (5)

4、数据库设计

数据字典可以理解为集合，是对数据流图中所有元素的描述。它对数据库中的每一个数据定义一个字段，包括对一切动态数据、静态数据的数据结构和相互关系等内容的说明，以保持数据在系统中的一致性，它相当于字典的作用。数据字典是分析阶段的得力工具。

1)、数据流词条

数据流名称：用户信息
来源：系统管理员
去向：登录页面
包含的数据项：用户名、密码

表 1 用户信息数据流表

数据流名称：设备信息
来源：机器
去向：Socket 服务器、Web 服务器、客户端、数据库
包含的数据项：设备号、设备状态、门磁状态、小组号、时间戳

表 2 设备信息数据流表

2)、数据元素词条

列名	数据类型	是否为 NULL 值
用户名 user_Name	char(15)	否
密码 user_Password	char(15)	否

表 3 用户表 monitor_user 表

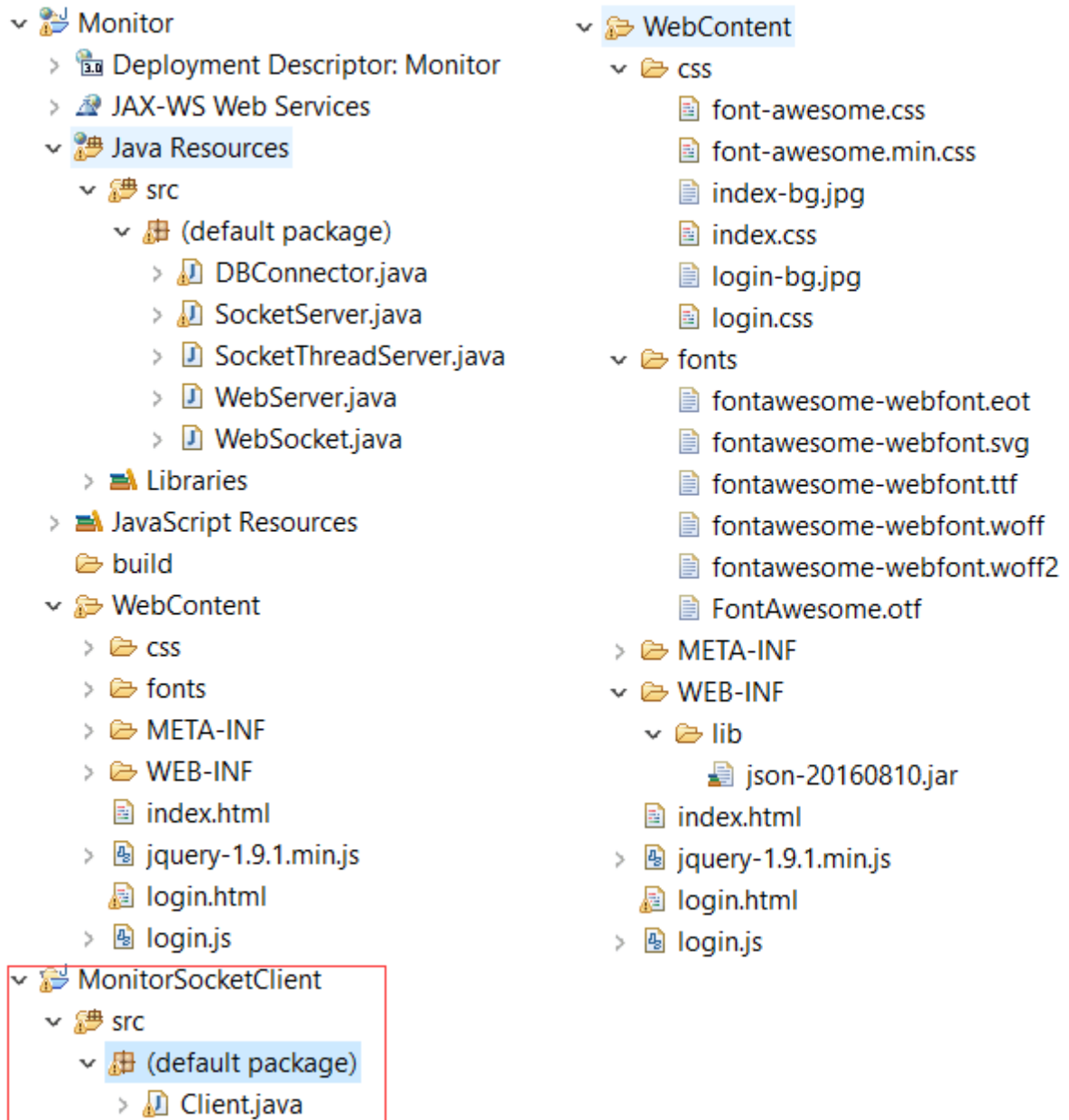
列名	数据类型	是否为 NULL 值
设备号 device_Id	char(10)	否
设备状态 device_State	char(1)	否
门磁状态 state	char(1)	否
小组号 group_Id	char(6)	否
时间戳 time_Stamp	TIMESTAMP	否

表 4 设备表 monitor_device 表

5、系统详细设计

本详细设计章节的目的就是希望读者可以通过阅读本详细设计，能够了解如何开发一个简单的 SMART Monitor 项目，同时也对其中的一些关键代码进行说明。

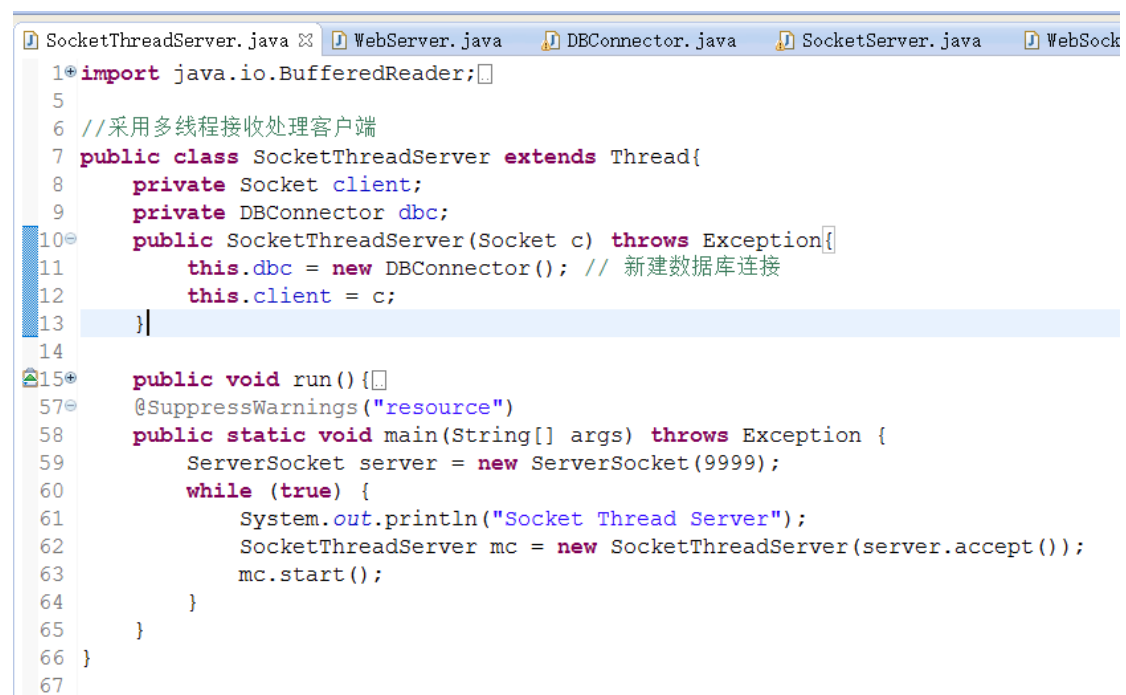
为了节约篇幅，并不会对每个代码截图都解释，变量和函数的命名风格和许多注释能够做到一定程度上的说明。即使前端的 js 和 html 文件，我们也给出了核心代码截图和注释。



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。



```
SocketThreadServer.java WebServer.java DBConnector.java SocketServer.java WebSock
1 import java.io.BufferedReader;
2
3 //采用多线程接收处理客户端
4
5
6 //采用多线程接收处理客户端
7 public class SocketThreadServer extends Thread{
8     private Socket client;
9     private DBConnector dbc;
10    public SocketThreadServer(Socket c) throws Exception{
11        this.dbc = new DBConnector(); // 新建数据库连接
12        this.client = c;
13    }
14
15    public void run() {}
16
17    @SuppressWarnings("resource")
18    public static void main(String[] args) throws Exception {
19        ServerSocket server = new ServerSocket(9999);
20        while (true) {
21            System.out.println("Socket Thread Server");
22            SocketThreadServer mc = new SocketThreadServer(server.accept());
23            mc.start();
24        }
25    }
26 }
27 }
```

SocketThreadServer 类代码概览

采用多线程方式，对每一个 Socket 客户端的请求单独处理，这样能监听到以后客户端的单独离线和特殊处理。

```
SocketThreadServer.java WebServer.java DBConnector.java SocketServer.java WebSocket.java Client.java
15 public void run() {
16     String backup = "0", content = "", deviceId = "", deviceState = "", state, groupId = "000001";
17     try {
18         BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));
19         // Mutil User but can't parallel
20         while (true) {
21             char[] s = new char[27];
22             int rt = in.read(s);
23             if (rt == -1) break;
24             content = new String(s);
25             System.out.println("Server: "+content);
26             state = content.substring(16, 17);
27             if ( backup.equals(state))//若本次状态与上次有变化则写入数据库
28                 continue; //否则跳过继续下一次循环
29
30             backup = state;//将当前的状态保存起来|
31             // 将信息存入数据库
32             deviceId = content.substring(3, 13);
33             deviceState = content.substring(14, 15);
34             dbc.insertDevice(deviceId, deviceState, state, groupId);
35         }
36     } catch (Exception e) {
37         System.out.println("出错啦~");
38     } finally {
39         deviceState = "0"; // 更新为离线状态
40         state = "0";
41         try {
42             dbc.insertDevice(deviceId, deviceState, state, groupId);
43             System.out.println("客户端主动关闭, 写入数据库为离线状态完成"+content);
44             client.close();
45         } catch (Exception e) {
46             // TODO Auto-generated catch block
47             e.printStackTrace();
48         }
49     }
50 }
```

SocketThreadServer 类 run 方法代码截图

而在 run 方法中，会对每一个客户端上一次的状态进行保存，若本次状态没有改变（每一个状态码都是唯一的）则不进行数据库的更新，这样能减少数据库的压力。同时检测若客户端离线，则自动更新数据库的状态为离线（既状态码为 0）。

```
SocketThreadServer.java WebServer.java DBConnector.java SocketServer.java WebSocket.java Client.java
1 *import java.io.IOException;
9
11 * Servlet implementation class server
13 @WebServlet("/server")
14 public class WebServer extends HttpServlet {
15     private static final long serialVersionUID = 1L;
16     //保存数据库连接
17     DBConnector dbc = null;
18
20 * @see HttpServlet#HttpServlet()
22 public WebServer() {}
26
28 * @see Servlet#init(ServletConfig)
30 public void init(ServletConfig config) throws ServletException {
31     // TODO Auto-generated method stub
32     try {
33         this.dbc = new DBConnector();
34     } catch (Exception e) {
35         // TODO Auto-generated catch block
36         e.printStackTrace();
37     }
38 }
39
41 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
43 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
44     // TODO Auto-generated method stub
45     doPost(request, response);
46 }
47
49 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
51 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
74 }
```

WebServer 类代码概览

```
51 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
52     // TODO Auto-generated method stub
53     try {
54         //System.out.println("doPost");
55         String usnm = request.getParameter("username");
56         String uspwd = request.getParameter("userpwd");
57         if(request.getParameter("type").equals("login")){ //登录
58             if(dbc.getPassword(usnm) == null)
59                 response.getWriter().write("User is not exist");
60             else if( dbc.getPassword(usnm).equals(uspwd)) //若用户名密码正确，则跳转主页
61                 response.getWriter().write("success");
62             else response.getWriter().write("wrong password");
63         }else{//注册
64             if(dbc.getPassword(usnm) != null)
65                 response.getWriter().write("User has existed");
66             else
67                 response.getWriter().write(dbc.register(usnm, uspwd));
68         }
69     } catch (Exception e) {
70         // TODO Auto-generated catch block
71         e.printStackTrace();
72     }
73 }
74 }
```

WebServer 类 doPost 方法代码截图

```

SocketThreadServer.java WebServer.java *DBConnector.java SocketServer.java WebSocket.java Client.java
1 *import java.sql.ResultSet;
10
11 public class DBConnector {
12     String db_url = "jdbc:mysql://localhost:3306/monitor";
13     String db_user = "root";
14     String db_pwd = "hello";
15
16 public DBConnector() throws ClassNotFoundException{
17     Class.forName("com.mysql.jdbc.Driver");
18 }
19
20 public Connection getConnection() throws SQLException{
21     return (Connection) java.sql.DriverManager.getConnection(db_url, db_user, db_pwd);
22 }
23
24 public void db_close(Connection con, PreparedStatement prepStmt, ResultSet rs) throws Exception{
25     if(rs != null) rs.close();
26     if(prepStmt != null) prepStmt.close();
27     if(con != null) con.close();
28 }
29
30 public String rsToJson(ResultSet rs) throws Exception{
47
48 public String getPassword(String name) throws Exception{
59
60 public String getDevice() throws Exception{
84
85 public String register(String name, String pwd) throws Exception{
100
101 public String insertDevice(String deviceId, String deviceState, String state, String groupId) throws Exception{
127 }
...

```

DBConnector 类代码概览

```

30 public String rsToJson(ResultSet rs) throws Exception{
31     JSONArray array = new JSONArray();
32     ResultSetMetaData metaData = (ResultSetMetaData) rs.getMetaData();
33     int colCount = metaData.getColumnCount(); //获取列数
34
35     while(rs.next()){ //遍历result set中的每条数据
36         JSONObject jsonObj = new JSONObject();
37         for(int i = 1; i <= colCount; ++i){ //遍历每一列
38             String colName = metaData.getColumnLabel(i);
39             String val = rs.getString(colName);
40             jsonObj.put(colName, val);
41         }
42         array.put(jsonObj);
43     }
44     //db_close(con, prepStmt, rs); //关闭连接
45     return array.toString();
46 }
47
48 public String getPassword(String name) throws Exception{
49     Connection con = getConnection();
50     String sql = "select * from monitor_user where user_Name=?";
51     PreparedStatement prepStmt = (PreparedStatement) con.prepareStatement(sql);
52     prepStmt.setString(1, name); //用name参数代替第一个?
53     ResultSet rs = prepStmt.executeQuery(); //执行sql语句
54     String pwd = null;
55     if(rs.next()) pwd = rs.getString(2); //获取第二个字段, 即密码
56     db_close(con, prepStmt, rs); //关闭连接
57     return pwd;
58 }

```

DBConnector 类 rsToJson 和 getPassword 方法代码概览

```

60 public String getDevice() throws Exception{
61     Connection con = getConnection();
62     String sql = "select * from monitor_device";
63     PreparedStatement prepStmt = (PreparedStatement) con.prepareStatement(sql);
64     ResultSet rs = prepStmt.executeQuery(); //执行sql语句
65
66     //转化为json输出
67     JSONArray array = new JSONArray();
68     ResultSetMetaData metaData = (ResultSetMetaData) rs.getMetaData();
69     int colCount = metaData.getColumnCount(); //获取列数
70
71     while(rs.next()){//遍历result set中的每条数据
72         JSONObject jsObj = new JSONObject();
73         for(int i = 1; i <= colCount; ++i){//遍历每一列
74             String colName = metaData.getColumnLabel(i);
75             String val = rs.getString(colName);
76             jsObj.put(colName, val);
77         }
78         array.put(jsObj);
79     }
80
81     db_close(con, prepStmt, rs); //关闭连接
82     return array.toString();
83 }

```

DBConnector 类 getDevice 方法代码概览

```

85 public String register(String name, String pwd) throws Exception{
86     Connection con = getConnection();
87     String sql = "insert into monitor_user(user_name,user_password) values(?,?)";
88     PreparedStatement prepStmt = (PreparedStatement) con.prepareStatement(sql);
89     prepStmt.setString(1, name);//用name参数代替第一个?
90     prepStmt.setString(2, pwd);//用pwd参数代替第二个?
91
92     int rs = prepStmt.executeUpdate(); //执行sql语句
93     String rt = "fail";
94     if(rs > 0) rt = "success";
95     con.close(); //关闭连接
96     prepStmt.close();
97     //db_close(con, prepStmt, rs); //关闭连接
98     return rt;
99 }

```

DBConnector 类 register 方法代码概览

```

101= public String insertDevice(String deviceId, String deviceState, String state, String groupId) throws Exception{
102     //System.out.println(deviceId + "," + deviceState + "," + state + "," + groupId);
103
104     Connection con = getConnection();
105     String sql = "select * from monitor_device where device_id="+deviceId;
106     PreparedStatement prepStmt = (PreparedStatement) con.prepareStatement(sql);
107     ResultSet rs = prepStmt.executeQuery(); //执行sql语句
108     if(rs.next()){ //说明存在该设备, 则进行更新操作
109         sql = "UPDATE monitor_device set device_state="+deviceState+", state="+state+" where device_id="+deviceId;
110         prepStmt = (PreparedStatement) con.prepareStatement(sql);
111     }else{
112         sql = "insert into monitor_device(device_id,device_state,state ,group_id) values(?,?,?,?)";
113         prepStmt = (PreparedStatement) con.prepareStatement(sql);
114         prepStmt.setString(1, deviceId); //用name参数代替第一个?
115         prepStmt.setString(2, deviceState); //用pwd参数代替第一个?
116         prepStmt.setString(3, state); //用pwd参数代替第一个?
117         prepStmt.setString(4, groupId); //用pwd参数代替第一个?
118     }
119
120     String rt = "fail";
121     if(prepStmt.executeUpdate() > 0) rt = "success"; //执行插入sql语句 返回一个int类型的值
122     con.close(); //关闭连接
123     prepStmt.close();
124     //db_close(con, prepStmt, rs); //关闭连接
125     return rt;
126 }
127 }

```

DBConnector 类 insertDevice 方法代码概览

```

SocketThreadServer.java WebServer.java DBConnector.java SocketServer.java WebSocket.java Client.java
1 *import java.io.IOException;[]
19
20 //该注解用来指定一个URI, 客户端可以通过这个URI来连接到WebSocket. 类似Servlet的注解mapping, 无需在web.xml中配置.
21 @ServerEndpoint("/websocket")
22 public class WebSocket {
23     //静态变量, 用来记录当前在线连接数. 应该把它设计成线程安全的.
24     private static int onlineCount = 0;
25     DBConnector dbc = null; //数据库连接对象
26
27     //concurrent包的线程安全Set, 用来存放每个客户端对应的MyWebSocket对象. 若要实现服务端与单一客户端通信的话, 可以使用Map来存放, 其中Key可以为用户标识
28     private static CopyOnWriteArraySet<WebSocket> websocketSet = new CopyOnWriteArraySet<WebSocket>();
29
30     //与某个客户端的连接会话, 需要通过它来给客户端发送数据
31     private Session session;
32
33     * 连接成功调用的方法[]
34 *
35 public void onOpen(Session session) throws Exception{[]
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71 *
72 * 连接关闭调用的方法[]
73 *
74 public void onClose(){[]
75
76
77
78
79
80
81 *
82 * 收到客户端消息后调用的方法[]
83 *
84 public void onMessage(String message, Session session) {[]
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101 *
102 * 发生错误时调用[]
103 *
104 public void onError(Session session, Throwable error){[]
105
106
107
108
109
110
111
112 *
113 * 这个方法与上面几个方法不一样. 没有用注解, 是根据自己需要添加的方法. []
114 *
115 public void sendMessage(String message) throws IOException{[]
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132 }

```

WebSocket 类代码概览

```

SocketThreadServer.java  WebServer.java  DBConnector.java  SocketServer.java  WebSocket.java  Cli
33  /**
34  * 连接建立成功调用的方法
35  * @param session 可选的参数。session为与某个客户端的连接会话，需要通过它来给客户端发送数据
36  * @throws Exception
37  */
38  @OnOpen
39  public void onOpen(Session session) throws Exception{
40      this.session = session;
41      websocketSet.add(this);    //加入set中
42      addOnlineCount();        //在线数加1
43      //System.out.println("有新连接加入! 当前在线人数为" + getOnlineCount());
44
45      //创建数据库连接
46      try {
47          this.dbc = new DBConnector();
48      } catch (ClassNotFoundException e) {
49          // TODO Auto-generated catch block
50          e.printStackTrace();
51      }
52
53      Timer timer = new Timer();
54      //延迟0s, 同时每1秒调用一次, 此处必须使用new TimerTask(), 同时重写run方法
55      timer.schedule(new TimerTask(){
56          public void run(){
57              for(WebSocket item: websocketSet){
58                  try {
59                      item.sendMessage(dbc.getDevice());
60                  } catch (Exception e) {
61                      e.printStackTrace();
62                      continue;
63                  }
64              }
65          }
66      }, 0, 1000);
67
68  }

```

WebSocket 类 onOpen 方法代码概览


```

71*   * 连接关闭调用的方法□
73   @OnClose
74   public void onClose(){
75       websocketSet.remove(this); //从set中删除
76       subOnlineCount(); //在线数减1
77       //System.out.println("有一连接关闭!当前在线人数为" + getOnlineCount());
78   }
79
81*   * 收到客户端消息后调用的方法□
85   @OnMessage
86   public void onMessage(String message, Session session) {
87       //System.out.println("来自客户端的消息:" + message);
88
89       //群发消息
90       for(WebSocket item: websocketSet){
91           try {
92               item.sendMessage(message);
93           } catch (IOException e) {
94               e.printStackTrace();
95               continue;
96           }
97       }
98   }

```

WebSocket 类 onClose 和 onMessage 方法代码概览

由于采用了集合的形式，所以每一次从数据库读取消息都会循环群发消息到每一个客户端，在 onOpen 方法中，采用 timer 定时器，间隔 1s 后立即推送设备信息至所有的客户端。

```

99*   * 发生错误时调用□
103  @OnError
104  public void onError(Session session, Throwable error){
105      System.out.println("发生错误");
106      error.printStackTrace();
107  }
108
110*   * 这个方法与上面几个方法不一样。没有用注解，是根据自己需要添加的方法。□
114  public void sendMessage(String message) throws IOException{
115      this.session.getBasicRemote().sendText(message);
116      //this.session.getAsyncRemote().sendText(message);
117  }
118
119  public static synchronized int getOnlineCount() {
120      return onlineCount;
121  }
122
123  public static synchronized void addOnlineCount() {
124      WebSocket.onlineCount++;
125  }
126
127  public static synchronized void subOnlineCount() {
128      WebSocket.onlineCount--;
129  }
130 }

```

WebSocket 类 onError , sendMessage , getOnlineCount , addOnlineCount 和

subOnlineCount 方法代码截图

```
SocketThreadServer.java WebServer.java DBConnector.java SocketServer.java WebSocket.java Client.java
1 *import java.io.IOException;
8 //http://blog.csdn.net/zhangyiacm/article/details/49488721
9 public class Client {
10
11 public static void main(String[] args) throws Exception, IOException {
12
13     String host = "127.0.0.1";
14     int port = 9999; // 端口号
15     int count = 0; //记录发送了多少次
16     Socket client = new Socket(host, port); // 定义socket;
17
18     while (true) {
19         Writer wt = new OutputStreamWriter(client.getOutputStream()); // 定义输出流
20         int z1 = (new Random()).nextInt(2); // 1为启动, 0为关机,使用随机生成
21         int z2 = 0; // 0表示正常, 其他表示不正常
22         if (z1 == 1)
23             z2 = (new Random()).nextInt(4) + 1; // 生成1-4之内的随机数
24         else
25             z2 = 0;
26         String str = "#mj" + args[0] + "," + z1 + "," + z2 + ",000001,u";
27         count++;
28         System.out.println(count+" : "+str);
29
30         wt.write(str); // #mj2011102701,1,0,000000,u
31         wt.flush();
32
33         try {
34             Thread.sleep(1000 * 3);
35         } catch (InterruptedException e) {
36             e.printStackTrace();
37             break; //若定时器发生问题则跳出循环
38         }
39     }
40     client.close();
41 }
42 }
```

Client 类代码截图

```
13 <body style="zoom: 1;">
14 <h1>SMART-Monitor<sup>V0.0.1</sup></h1>
15
16 <div class="login" style="margin-top:50px;">
17
18 <div class="header">...
24 </div>
25
26 <div class="web_qr_login" id="web_qr_login" style="display: block; height: 235px;">
27
28 <!--登录-->
29 <div class="web_login" id="web_login">...
57 </div>
58 <!--登录end-->
59 </div>
60
61 <!--注册-->
62 <div class="qlogin" id="qlogin" style="display: none;">...
109 </div>
110 <div class="copyright jianyi">© SMART Monitor by PMOL</div>
111 <div class="jianyi">潘韵如(SA16225225) 苗苗(SA16225213) 欧勇(SA16225221) 林鑫(SA16225168)</div>
112
```

登录, 注册 html 页面概览

```
index.html
14 <body>
15 <div id="history"><b>History:</b><br><div></div></div>
16 <div class="wrap common-icon-show">
17 <div class="message" id="message"><span class="rw-words"></span></div>
18 <ul id="devices"></ul>
19 <!-- 取消注释能向WebSocket服务器主动发送信息 -->
20 <!-- <input id="text" type="text" /><input type="button" style="width:30%" onclick="send()" value="Send">
21 <input type="button" style="width:30%" value="Close" onclick="closeWebSocket()"-->
22 </div>
23
24 <div class="copyright">SMART Monitor design by PMOL</div><br/>
25 <script src="jquery-1.9.1.min.js"></script>
26 <script>
27
28 var websocket = new WebSocket('ws://localhost:8080/Monitor/websocket'); //创建WebSocket对象
29 //alert(websocket.readyState);//查看websocket当前状态
30
31 //连接成功建立的回调方法
32 websocket.onopen = function(event){
33     $('#message > span.rw-words').html("连接 SMART Monitor Socket服务器成功");
34 }
35
36 //接收到消息的回调方法
37 websocket.onmessage = function(event){ ...
38 }
39
40 //连接发生错误的回调方法
41 websocket.onerror = function(){
42     $('#message > span.rw-words').html("error");
43 };
44
45 //连接关闭的回调方法
46 websocket.onclose = function(){
47     $('#message > span.rw-words').html("close");
48 }
49
50 //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。
51 window.onbeforeunload = function(){
52     websocket.close();
53 }
54
55 //关闭连接
56 function closeWebSocket(){ ...
57 }
58
59 //发送消息
60 function send(){ ...
61 }
62 }
```

监视设备状态页面（主页）js 代码（既 websocket 客户端）概览

在 js 脚本中已经为将来的扩展留下了很多的回调方法，比如 send 方法，可以发送消息至 WebSocket 服务器，这样就能实现控制或配置更多的设备，而不仅仅是被动接收消息。本示例主要利用了 onMessage 方法监听从服务器收到消息。

```

36 //接收到消息的回调方法
37 websocket.onmessage = function(event){
38     var data = JSON.parse(event.data);
39     var on = 0, count = 0, off = 0; //用于统计出错设备和离线设备数量
40
41     $.each(data, function(i, item){
42         var crt = $('#device_id'+item.device_id); //先试着寻找该ID的设备图标
43         if(crt.length <= 0) //若不存在该节点则先添加该节点
44             var li = $('<li id="" data-state="0"><a href="javascript:void(0)" title="正常运行"><i class="fa fa-laptop fa-5x" aria-hidden="true"></i><br><span>&nbsp; </span></a></li>');
45             li.attr('id', 'device_id'+item.device_id).attr('data-state', item.state).find('span').text(item.device_id);
46             $('#devices').append(li);
47
48         //将数据记录到历史栏中
49         if(item.state != crt.attr('data-state')){ //只有当状态不一致的时候才会对dom做修改，状态码唯一，无论是离线还是在线
50             $('#history > div').prepend('<br><span style="font-weight:bold;">ID: '+item.device_id+', State: '+ item.state +'</span>, Time: '+ item.time_stamp +'<br></div>');
51         }
52         //存在该ID的设备图标，则直接修改状态
53         crt.find('i').attr('style',''); //修改颜色为默认灰色
54         if(item.device_state == '1'){
55             ++count;
56             crt.find('a').attr('title', '状态码: '+item.state);
57             if(item.state == '1'){
58                 crt.find('i').attr('style','color: black;'); //表示正常运行
59                 crt.find('span').attr('style','color: black;'); //表示正常运行
60                 ++on;
61                 --count;
62             }else if(item.state == '2'){
63                 crt.find('i').attr('style','color: #cc0000;');
64                 crt.find('span').attr('style','color: #cc0000;');
65             }else if(item.state == '3'){
66                 crt.find('i').attr('style','color: #0000cc;');
67                 crt.find('span').attr('style','color: #0000cc;');
68             }else if(item.state == '4'){
69                 crt.find('i').attr('style','color: #cc6600;');
70                 crt.find('span').attr('style','color: #cc6600;');
71             }
72         }else if(item.device_state == '0'){ //device_state为 0表示对方已关机，或无法连接
73             ++off;
74             crt.attr('data-state','0'); //设置状态为空
75             crt.find('i').attr('style','');
76             crt.find('span').attr('style','');
77             crt.find('a').attr('title', '离线');
78         }
79         //console.log(item);
80     });
81
82     //设定消息栏
83     $('#message > span.rw-words').html('<b style="color:black;">' + on + '</b>个设备正常运行, <b style="color:red;">' + count + '</b>个设备出错, <b style="color:#999;">' + off + '</b>个设备离线');
84 }

```

监视设备状态页面（主页）onMessage回调方法代码截图

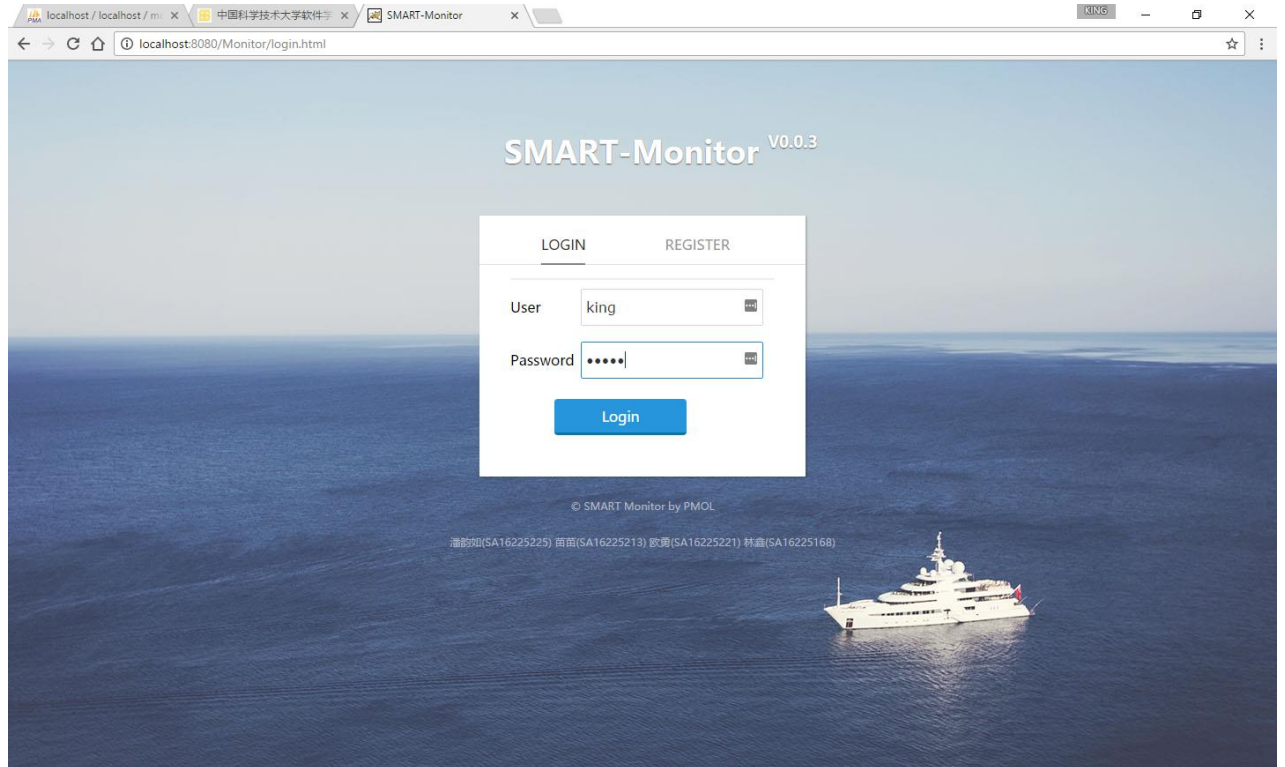
```

53 $(document).ready(function() {
54     //为点击事件，发起post请求，将登录用户名和密码发送到服务器端
55     $('#monitor_login').click(function() {
56         $.post('./server', {
60             }, function(data, status) {
64             });
65     });
66
67     //验证注册
68     $('#reg').click(function() {
69         //用户名不能为空
70         if ($('#user').val() == "") {
77         }
78         //用户名在4-16个字符之间
79         if ($('#user').val().length < 4 || ($('#user').val().length > 16) {
87         }
88         //验证密码
89         if ($('#passwd').val().length < 6) {
93         }
94         if ($('#passwd2').val() != $('#passwd').val()) {
98         }
99         //发起post请求，将新注册的用户名和密码发送到服务器端
100         $.post('./server', {
104             }, function(data, status) {
116             });
117     });
118 }

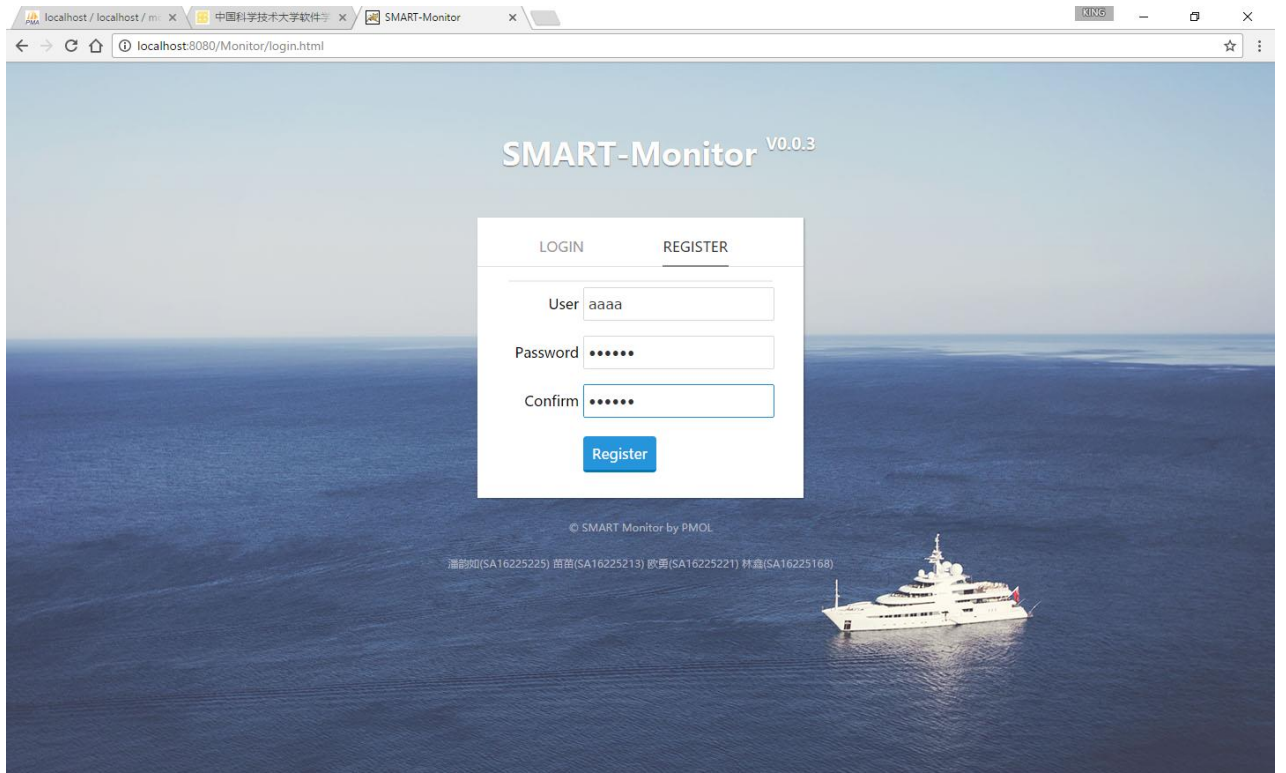
```

登录，注册界面js 代码概览

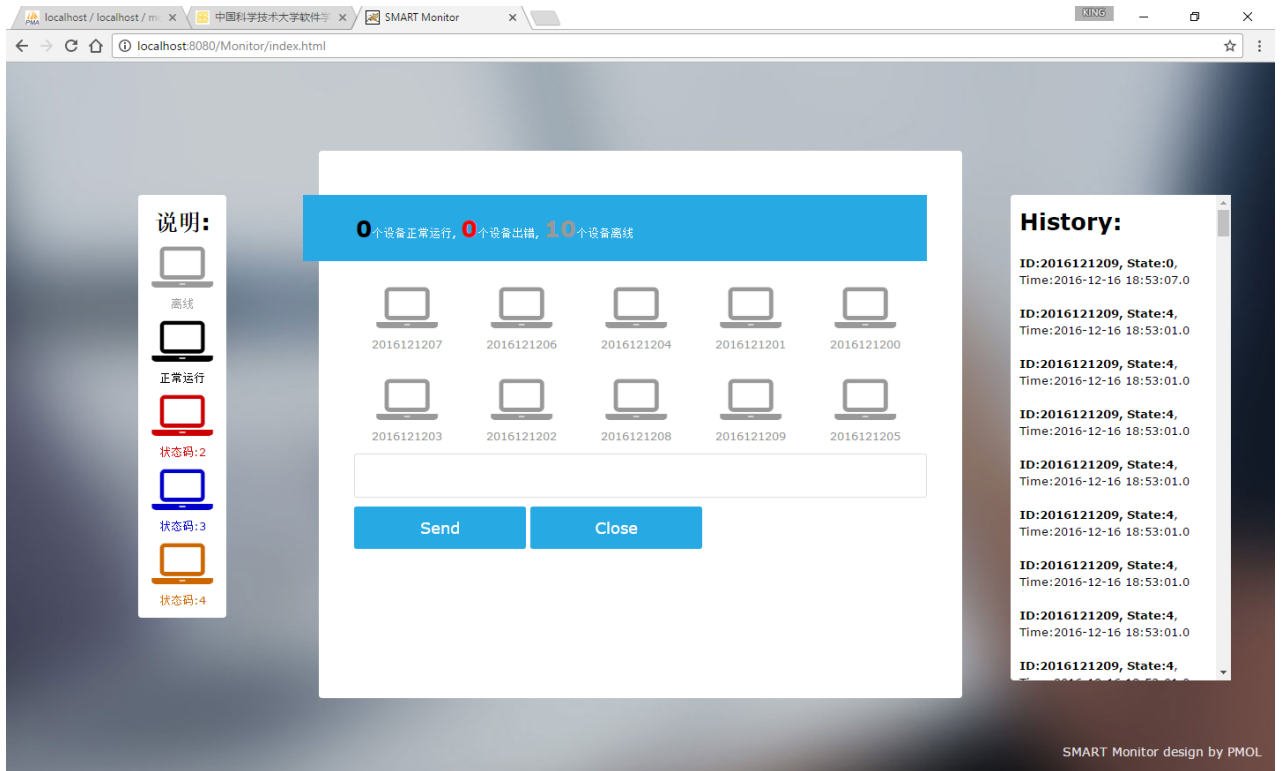
6、程序调试截图



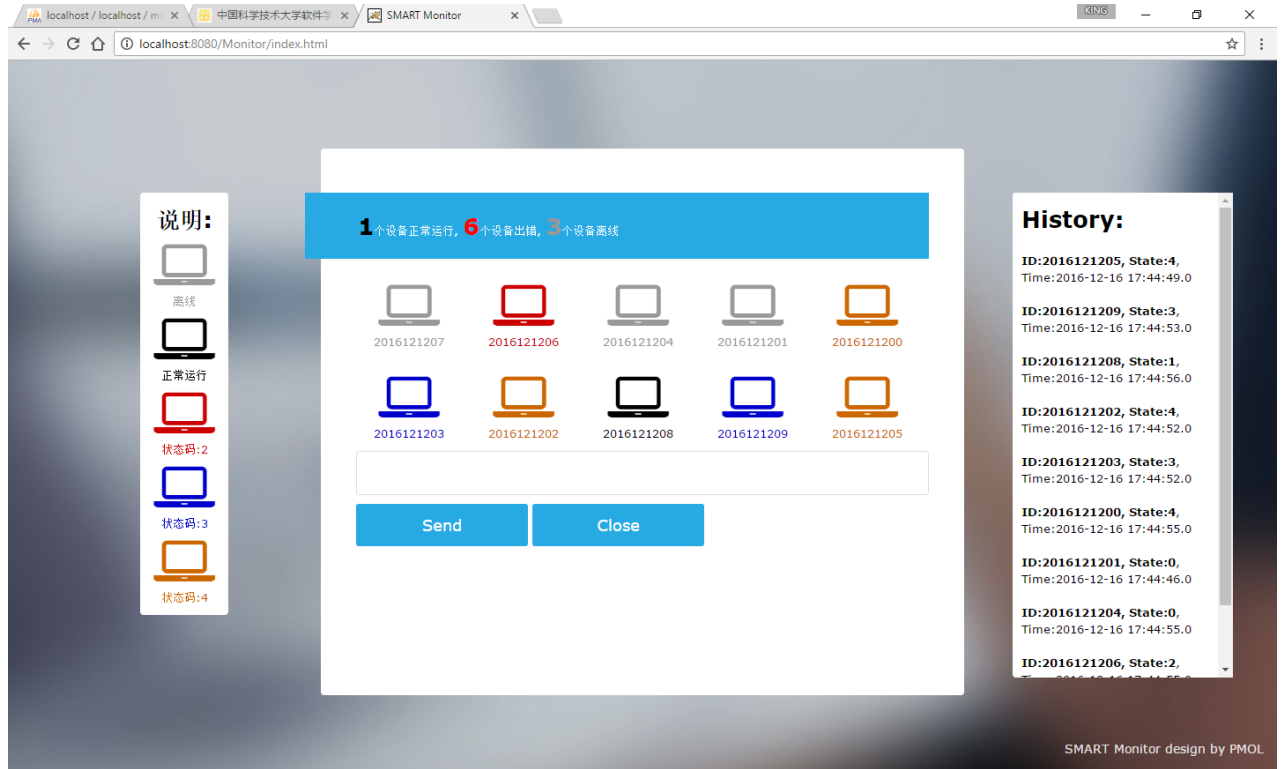
登录界面



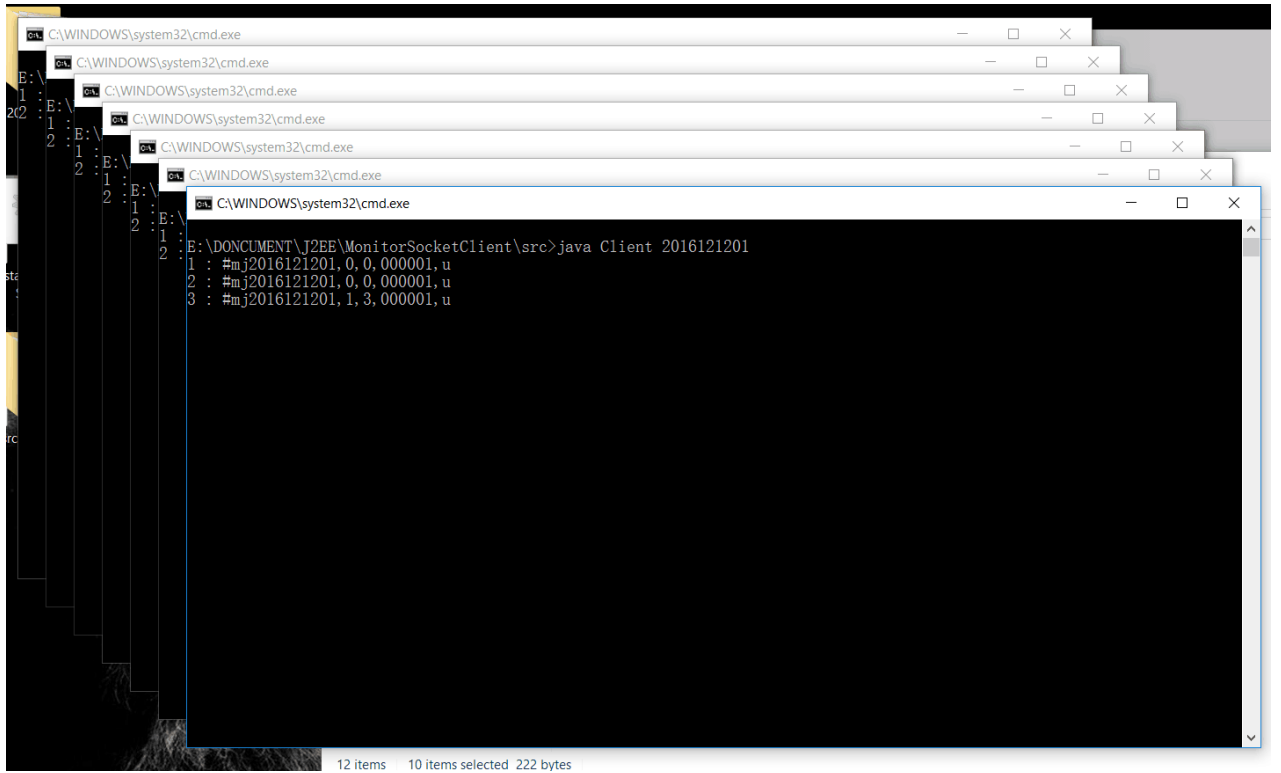
注册界面



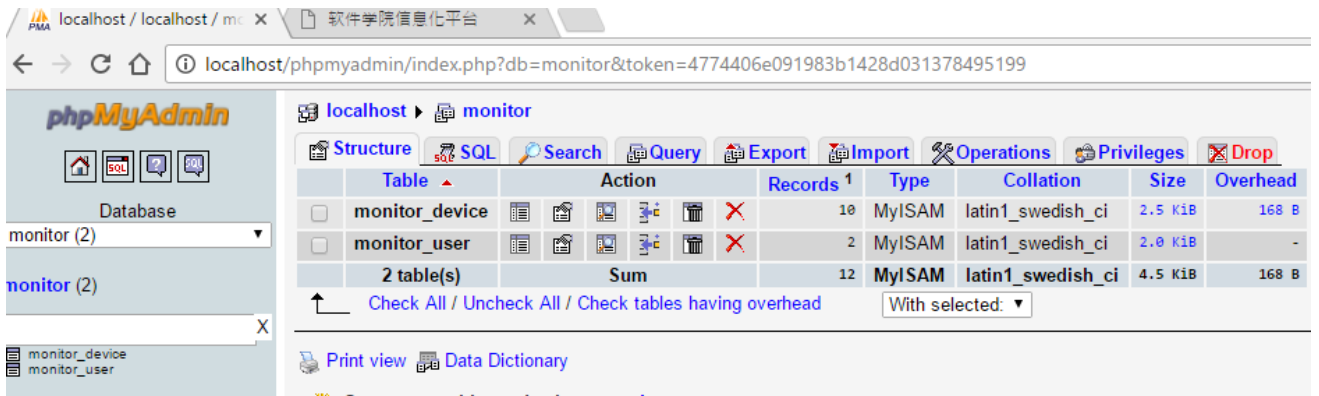
成功登录主页界面 (设备都未开启)



设备开启后的主页界面



模拟的设备界面（暂定 10 个）



（通过 WAMP Server 界面管理工具查看的）数据库概览图

localhost ▶ monitor ▶ monitor_device

Showing rows 0 - 9 (10 total, Query took 0.0004 sec)

```
SELECT *
FROM 'monitor_device'
LIMIT 0, 30
```

Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
Sort by key: None

+ Options

	device_Id	device_State	state	group_Id	time_Statmp
<input type="checkbox"/>	2016121207	0	0	1	2016-12-14 18:04:31
<input type="checkbox"/>	2016121206	0	0	1	2016-12-14 18:04:21
<input type="checkbox"/>	2016121204	0	0	1	2016-12-14 18:04:31
<input type="checkbox"/>	2016121201	0	0	1	2016-12-14 18:04:33
<input type="checkbox"/>	2016121200	0	0	1	2016-12-14 18:04:32
<input type="checkbox"/>	2016121203	0	0	1	2016-12-14 18:04:33
<input type="checkbox"/>	2016121202	0	0	1	2016-12-14 18:04:27
<input type="checkbox"/>	2016121208	0	0	1	2016-12-14 18:04:25
<input type="checkbox"/>	2016121209	0	0	1	2016-12-14 18:04:33
<input type="checkbox"/>	2016121205	0	0	1	2016-12-14 18:04:22

Check All / Uncheck All With selected:

数据库 monitor_device 表内数据展示

localhost ▶ monitor ▶ monitor_user "保存用户信息"

Showing rows 0 - 1 (2 total, Query took 0.0004 sec)

```
SELECT *
FROM 'monitor_user'
LIMIT 0, 30
```

Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
Sort by key: None

+ Options

	user_Name 用户名	user_Password 密码
<input type="checkbox"/>	king	hello
<input type="checkbox"/>	aaaa	aaaaaa

Check All / Uncheck All With selected:

数据库 monitor_user 表内数据展示

7、总结

我们选择将其中一些功能舍弃，只实现其核心的一些功能，比如 WebSocket 连接前后端。用这种 WebSocket 长连接实时刷新的方式，比使用 ajax 或间隔刷新页面等获取数据的方式要好很多，保持长连接的同时也使传输量大大减少，效率提高，且页面仅仅部分刷新保证了用户使用体验。

在整个设计中，我们的设备的状态码是唯一的，且由于我们使用的状态码为字符格式，所以可以允许不止 4 种（z1 和 z2 两个不同的状态组合）的状态。可以多达 6 万种（ $256*256 = 65536$ ）

这样一个简单的 Demo 还有许多非功能性需求还远远无法满足，比如多达上千设备的并发连接的问题，或数据的安全传输问题。

在 Socket 服务器端，我们采用了两种方式，一个是测试中截图给出的多线程方式，这种方式开发较为简单，而且逻辑比较清晰。另外一种是 I/O 多路复用的方式，这种方式较为复杂，而且 java 实现方式的资料不多，查找相关资料的时候比较困难。