

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | B/S体系软件设计 |
| 姓 名： | 童芳慧 |
| 学 院： | 计算机科学与技术学院 |
| 系： | 计算机科学与技术 |
| 专 业： | 计算机科学与技术2105 |
| 学 号： | 3210102034 |
| 指导教师： | 胡晓军 |

2021年 1月 5 日

**浙江大学实验报告**

课程名称： B/S软件设计 实验类型： 综合

实验项目名称： 物联网设备管理平台

学生姓名： 童芳慧 专业： 计算机科学与技术 学号： 3210102034

同组学生姓名： 指导老师： 胡晓军

实验地点： 实验日期： 2023 年 1 月 5 日

目录

[1.项目介绍 3](#_Toc155382825)

[1.1背景与目的 3](#_Toc155382826)

[1.1.1 背景 3](#_Toc155382827)

[1.1.2 目的 3](#_Toc155382828)

[1.2技术选型 3](#_Toc155382829)

[1.2.1 Spring Boot 3](#_Toc155382830)

[1.2.2 Vue.js 4](#_Toc155382831)

[1.3 功能概述 4](#_Toc155382832)

[1.3.1 用户登录注册 4](#_Toc155382833)

[1.3.2 设备统计管理 4](#_Toc155382834)

[1.3.3 消息查看统计 4](#_Toc155382835)

[1.3.4 安全与权限控制 5](#_Toc155382836)

[1.3.5 界面友好性 5](#_Toc155382837)

[2. 系统架构 5](#_Toc155382838)

[2.1 前端架构 5](#_Toc155382839)

[2.2 后端架构 6](#_Toc155382840)

[2.3 数据库设计 9](#_Toc155382841)

[3. 技术实现细节 11](#_Toc155382842)

[3.1 前端实现 11](#_Toc155382843)

[3.1.1登录界面 11](#_Toc155382844)

[3.1.2注册界面： 14](#_Toc155382845)

[3.1.3首页： 17](#_Toc155382846)

[3.1.4设备配置界面： 22](#_Toc155382847)

[3.1.5数据查询界面： 28](#_Toc155382848)

[3.1.6地图显示： 30](#_Toc155382849)

[3.2 后端实现 32](#_Toc155382850)

[3.2.1User： 33](#_Toc155382851)

[3.2.2 Device 34](#_Toc155382852)

[3.2.3MessageForm 37](#_Toc155382853)

[3.2.4 Menu 39](#_Toc155382854)

[3.2.5 Result 39](#_Toc155382855)

[3.2.6 Message 40](#_Toc155382856)

[3.2.7其他 40](#_Toc155382857)

[4. 部署与维护 41](#_Toc155382858)

[4.1 部署流程 41](#_Toc155382859)

[5. 项目总结 41](#_Toc155382860)

# 1.项目介绍

## 1.1背景与目的

### 1.1.1 背景

物联网（IoT）技术的发展日益迅猛，各类智能设备大量涌现。为了更有效地管理和监控这些物联网设备，我设计并开发了一款物联网设备管理平台。该平台旨在提供统一的管理界面，使用户能够方便地注册、登录，查看设备的实时状态，以及处理相关的消息和统计信息。

### 1.1.2 目的

项目的目的是构建一个高效、可扩展的物联网设备管理平台，通过该平台，用户可以轻松地管理大量设备，监测设备状态，以及获取与设备相关的消息。通过统一的平台，用户能够更加便捷地实现对物联网设备的全面管理，提高设备的利用率和系统的整体效率。

## 1.2技术选型

### 1.2.1 Spring Boot

Spring Boot是一个基于Spring框架的开发框架，它简化了Spring应用的初始化和开发过程。我们选择Spring Boot的主要原因包括：

* **快速开发：** Spring Boot提供了自动配置和约定优于配置的原则，使得开发者能够更快速地搭建项目。
* **强大的生态系统：** Spring Boot结合了Spring框架的强大功能，拥有丰富的社区和资源，提供了大量的插件和工具，使得开发更加便捷。
* **微服务支持：** Spring Boot天然支持微服务架构，通过Spring Cloud等相关组件，可以轻松搭建分布式系统。

### 1.2.2 Vue.js

Vue.js是一款轻量级、高性能的JavaScript框架，用于构建用户界面。我们选择Vue.js的原因包括：

* **响应式数据绑定：** Vue.js提供了强大的数据绑定机制，能够实时响应数据的变化，使得页面的更新更加高效。
* **组件化开发：** Vue.js采用组件化的开发方式，使得代码更易维护、扩展和重用。
* **灵活性：** Vue.js可以作为独立库使用，也可以通过Vue CLI等工具搭建复杂的单页面应用，满足不同项目的需求。

通过整合Spring Boot和Vue.js，我们搭建了一套强大的B/S体系架构，既能够处理后端逻辑，又能够提供流畅、友好的前端用户界面，从而实现了物联网设备管理平台的设计目标。

## 1.3 功能概述

### 1.3.1 用户登录注册

用户注册： 提供用户注册功能，新用户可以通过填写必要信息注册账户，系统将对用户信息进行验证和存储。

用户登录： 已注册用户可以通过输入用户名和密码进行登录，系统验证登录信息并允许合法用户进入系统。

### 1.3.2 设备统计管理

设备列表展示： 在系统中展示所有注册的物联网设备，以清晰的列表形式呈现设备的基本信息，如设备名称、状态等。

设备状态管理： 用户能够查看设备的实时状态，包括在线、离线等信息。

### 1.3.3 消息查看统计

消息列表展示：将与设备相关的消息以列表的形式展示，包括消息内容、发送时间、发送者等信息。

消息详情查看：用户可以查看消息的详细内容，以更全面地了解与设备相关的信息。

消息统计：系统提供对消息进行统计的功能，用户能够查看不同类型消息的数量、趋势等统计信息。

### 1.3.4 安全与权限控制

用户权限控制： 设计不同用户角色，如普通用户、管理员等，每个角色拥有不同的操作权限，确保系统安全可控。

安全机制： 使用加密算法对用户密码进行加密存储，采用安全的用户认证方式，如JWT（JSON Web Token）等，保障用户信息的安全性。

### 1.3.5 界面友好性

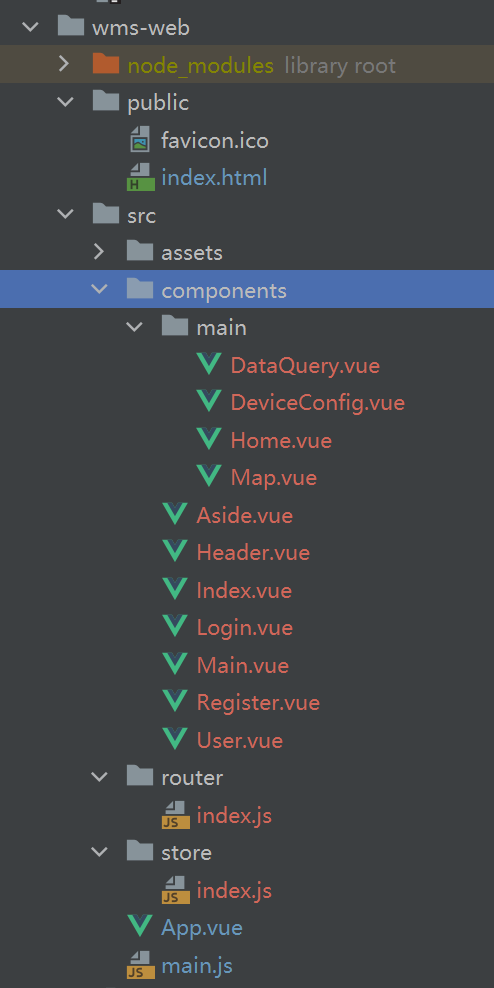
Responsiveness： 界面能够自适应不同设备和屏幕尺寸，提供良好的用户体验。

交互设计： 通过合理的交互设计，用户能够轻松地完成各种操作，提高系统的易用性。

# 2. 系统架构

## 2.1 前端架构

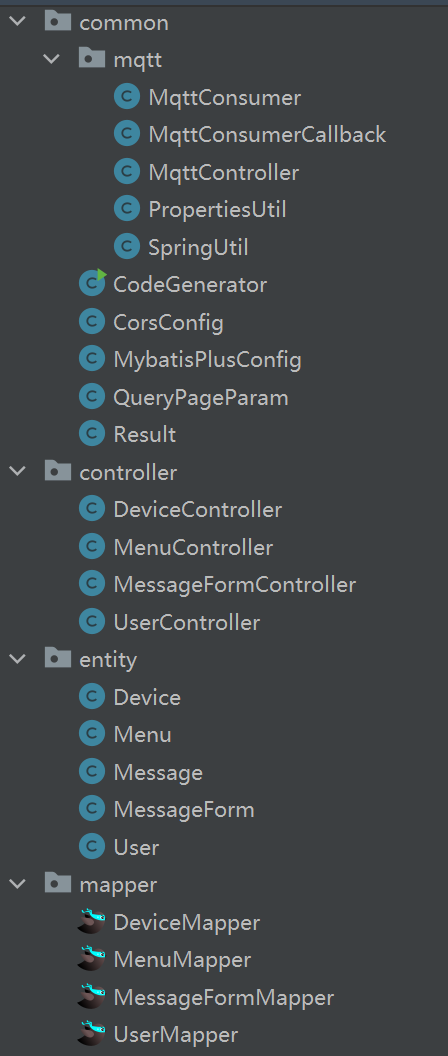
前端的文件组织形式：

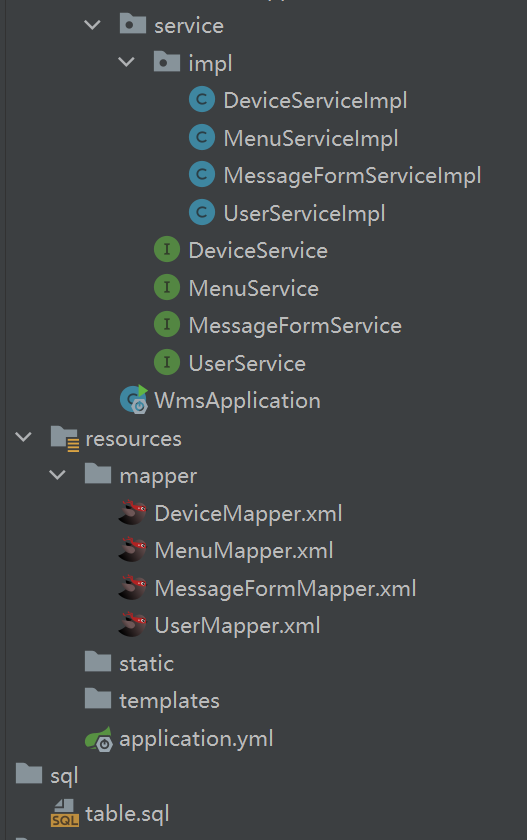


前端使用axios调用后端接口实现数据传递，后端统一传回Result格式的数据。

## 2.2 后端架构

后端框架组织形式：





数据库中的每一个表都对应了一个entity和对应的controller，mapper，service，前端调用controller暴露出的接口后，controller执行对应函数的代码，并相应调用service函数进行实现，每个抽象service函数下会有一个service实现类，service再调用mapper中的函数进行操作获得结果。

## 2.3 数据库设计

User：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 描述 | 备注 |
| id | int | 用户id | 主键，非空，自增 |
| name | varchar(40) | 用户名 | 非空 |
| password | varchar(64) | 账号密码 | 非空 |
| email | varchar((64) | 用户邮箱 | 非空，unique |
| roleId | Integer | 角色权限 0超级管理员，1管理员（默认），2普通账号 | 在本实验中，没有进行区分，权限管理功能可以进一步扩展。 |
| isValid | String | 是否有效，Y有效（默认） |  |

Device：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 描述 | 备注 |
| id | int | 设备id | 主键，非空，自增 |
| name | varchar(40) | 设备名 | 非空 |
| description | varchar(256) | 设备描述信息 | 非空 |
| userid | int | 设备的用户的 id | 非空，作为外键 |
| type | int | 设备的类型 | 非空，默认为其他设备（4）  车载设备（1）、智能家居（2）、可穿戴设备（3） |
| active | Integer | 是否活跃 | 非空，默认为1 |

MessageForm

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 描述 | 备注 |
| id | int | id | 主键，非空，自增 |
| device\_id | int | 设备id | 非空，外键 |
| alert | int | 是否触发警报 | 0 表示正常，1 表示警报 |
| info | varchar(128) | 设备发送的消息 | 默认是null |
| lat | double | 设备发送的经度 | 默认是 0 |
| lng | numeric | 设备发送的纬度 | 默认是 0 |
| time\_stamp | varchar(128) | 发送时候的时间戳 | 非空 |
| value | int | 设备发送的值 | 默认是 0 |

Menu：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 描述 | 备注 |
| id | int | id | 主键，非空，自增 |
| menuCode | String | 菜单编码 | 非空，外键 |
| menuName | String | 菜单名字 | 0 表示正常，1 表示警报 |
| menuLevel | String | 菜单级别 | 非空 |
| menuParentCode | String | 菜单的父编码 |  |
| menuClick | String | 点击触发的函数（页面地址） | 非空 |
| menuRight | String | 权限 | 0超级管理员，1表示管理员，2表示普通用户，可以用逗号组合使用 |
| menuComponent | String | 对应文件地址 | 非空 |
| menuIcon | String | 菜单图标 |  |

# 3. 技术实现细节

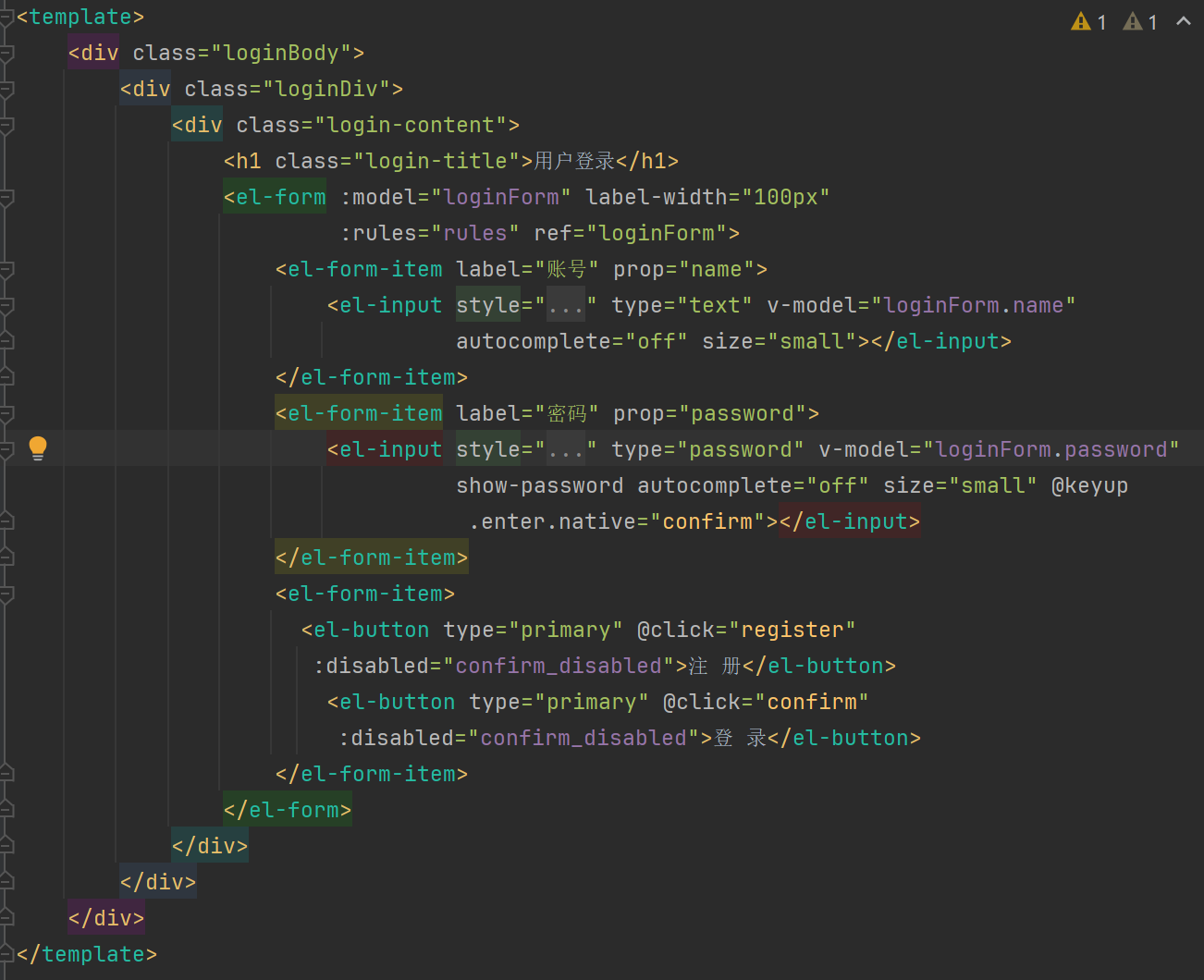
## 3.1 前端实现

在前端的实现中，我将页面分成三个部分：Aside，Header，Main进行分别实现，并在页面切换时修改中间的main部分。

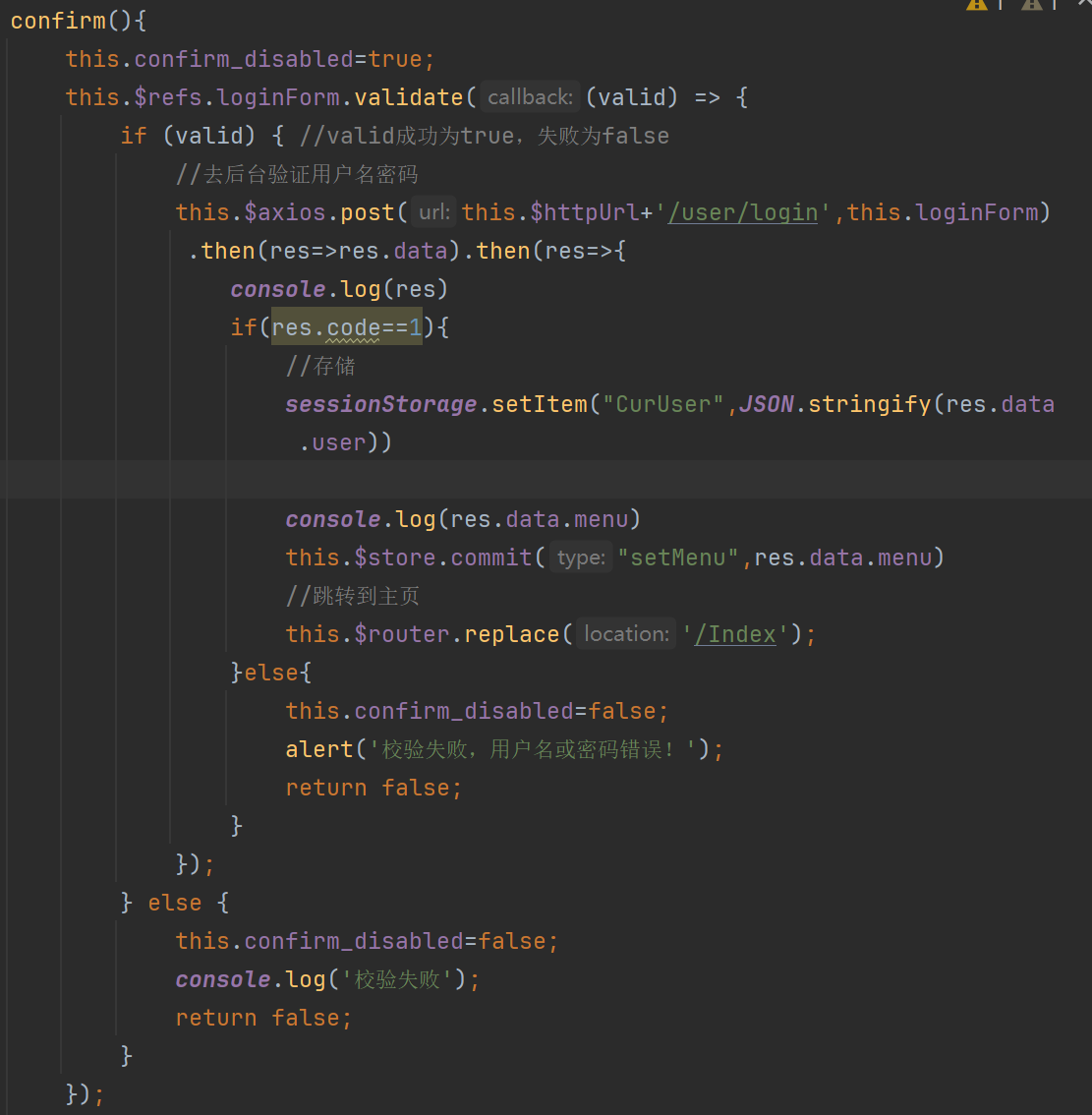
### 3.1.1登录界面



在登录界面，用户在表单中输入账号和密码进行登录，前端调用后端接口/user/login将窗体传回进行验证，如果账号密码匹配，则将信息储存在sessionStorage中进行后续使用，调用设置目录，按照角色权限设置目录，并将网址推送到/Index进入首页。



confirm后的数据处理：



### 3.1.2注册界面：



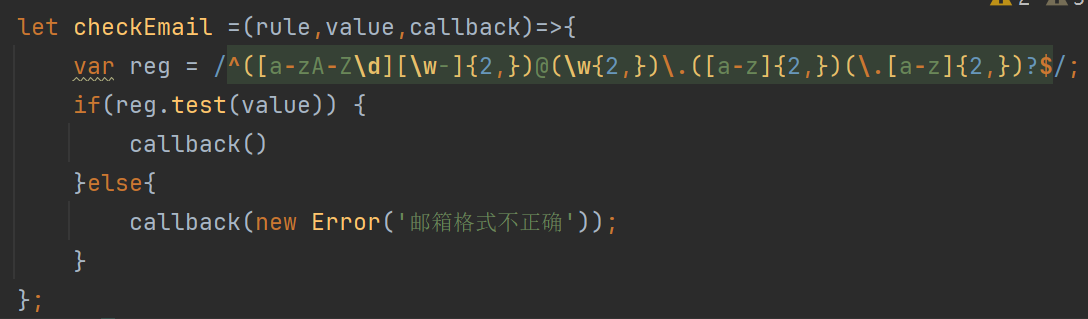
用户注册界面输入账号，密码，邮箱进行注册，其中账号在输入时会自动检查数据库中是否有重复的，并进行提示，密码要求长度在6-20个字符之间，邮箱要求符合格式，不然也会报错。



账号重复检查：



邮箱格式检查：



格式检查调用：



注册函数：

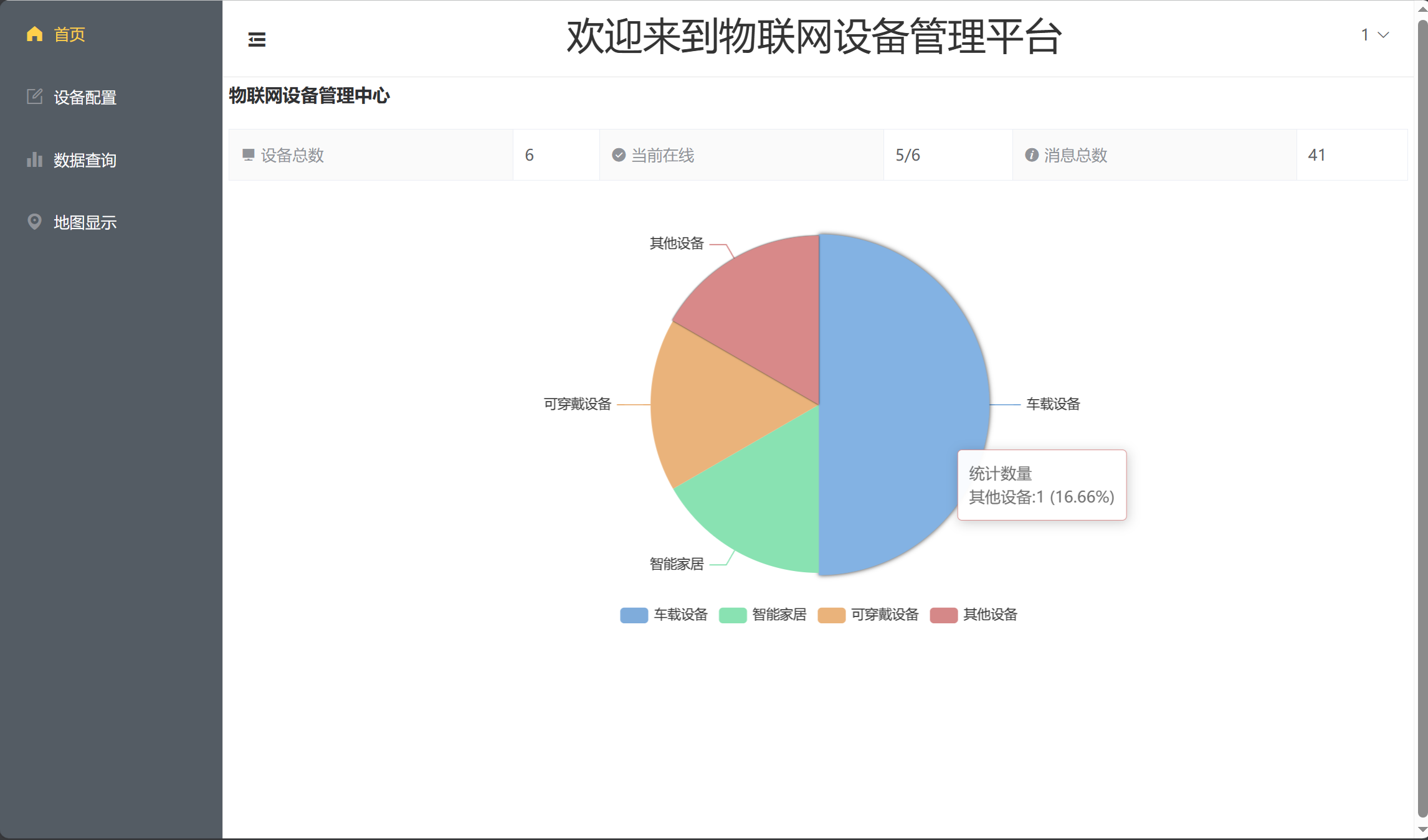
其中设置了变量控制按钮在得到响应前只能点击一次，保护了程序安全。



### 3.1.3首页：

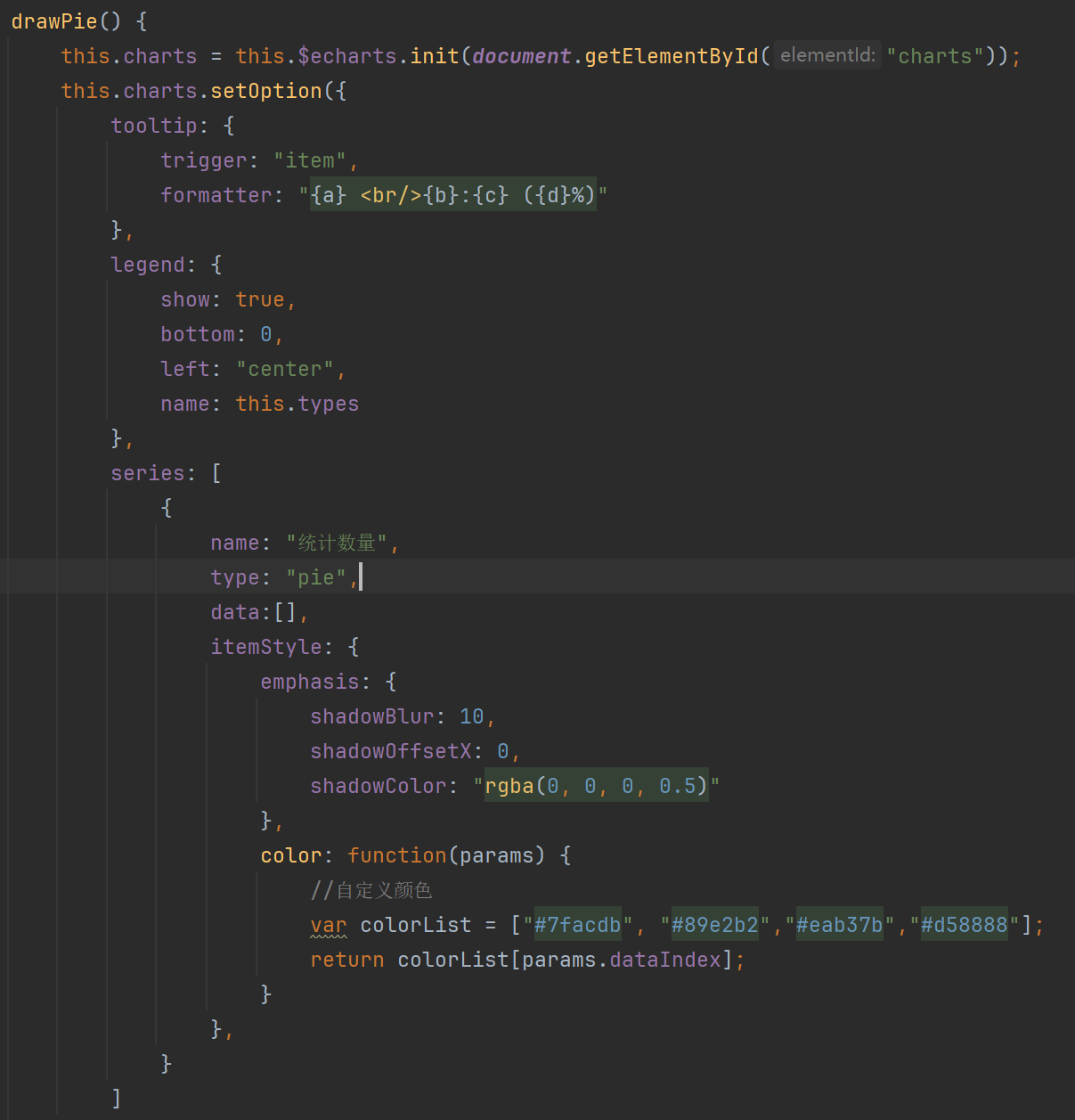
首页会列出设备总数，当前在线数和消息总数。并统计设备中各个分类的数目，并绘制饼图。

右上角显示当前用户名，并可以出现下拉框退出登录。





使用echarts绘制饼图：



获取设备数据：



获取消息数：



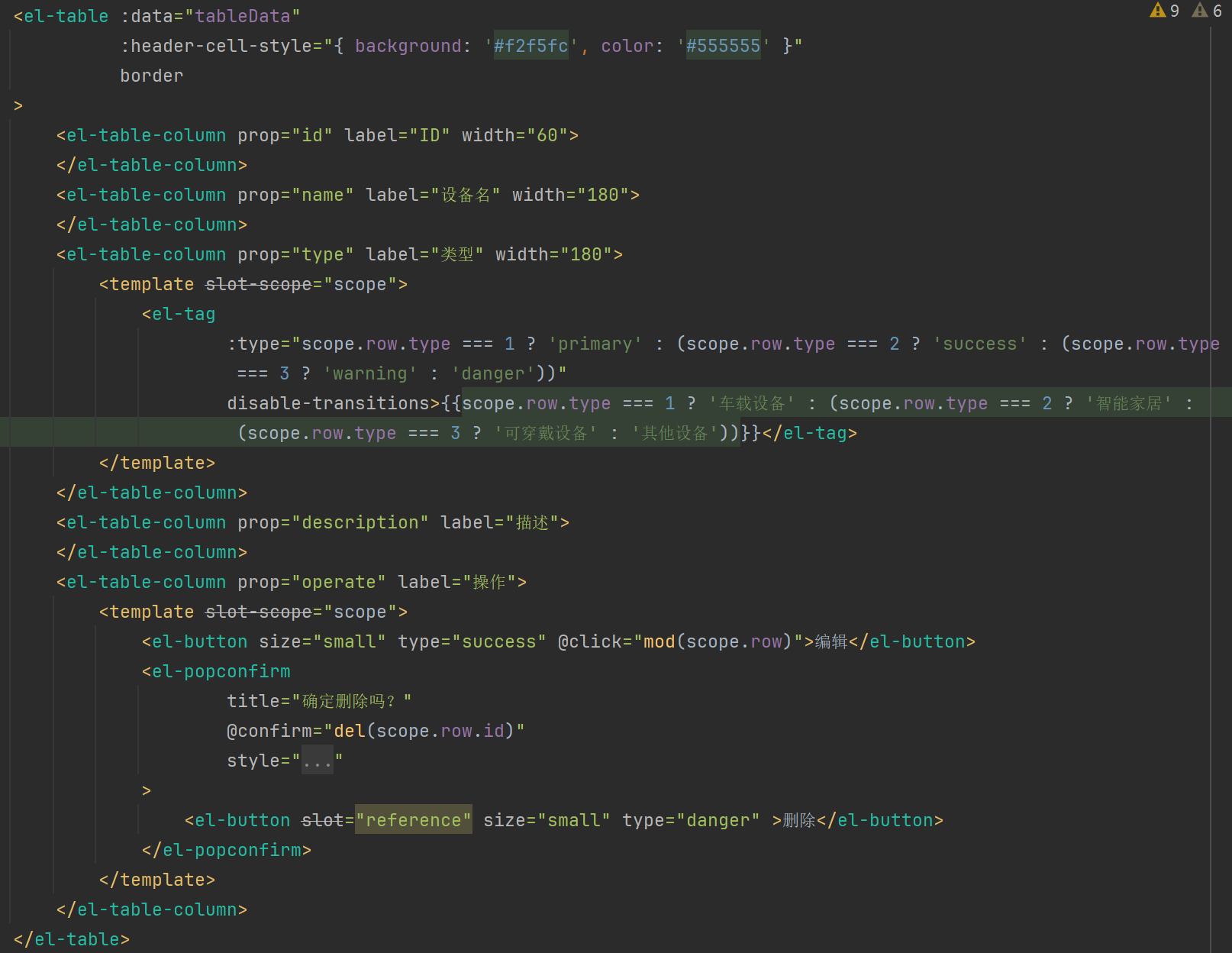
### 3.1.4设备配置界面：



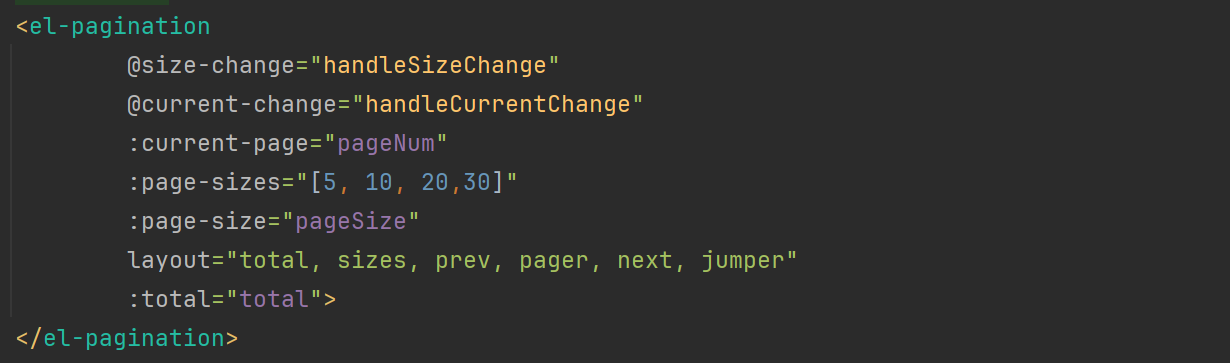
输入框和按钮：



设备信息展示



分页功能：



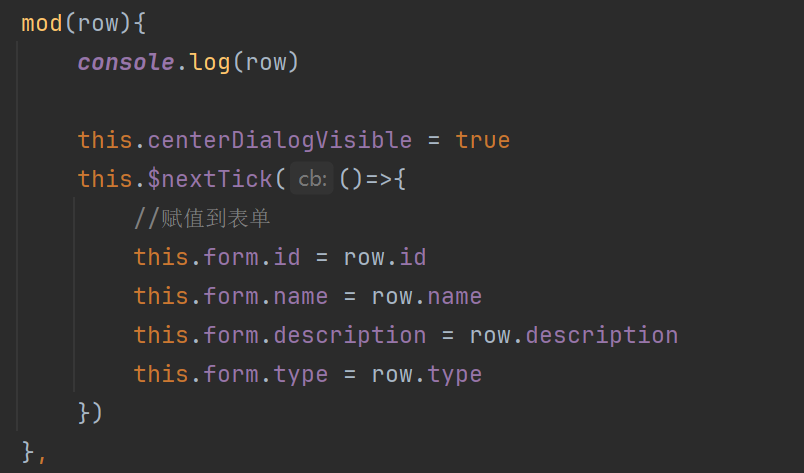
修改设备信息弹窗：



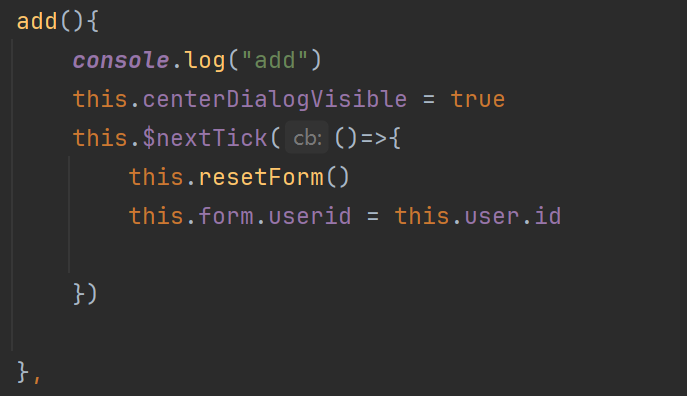
删除设备功能实现：



修改设备：

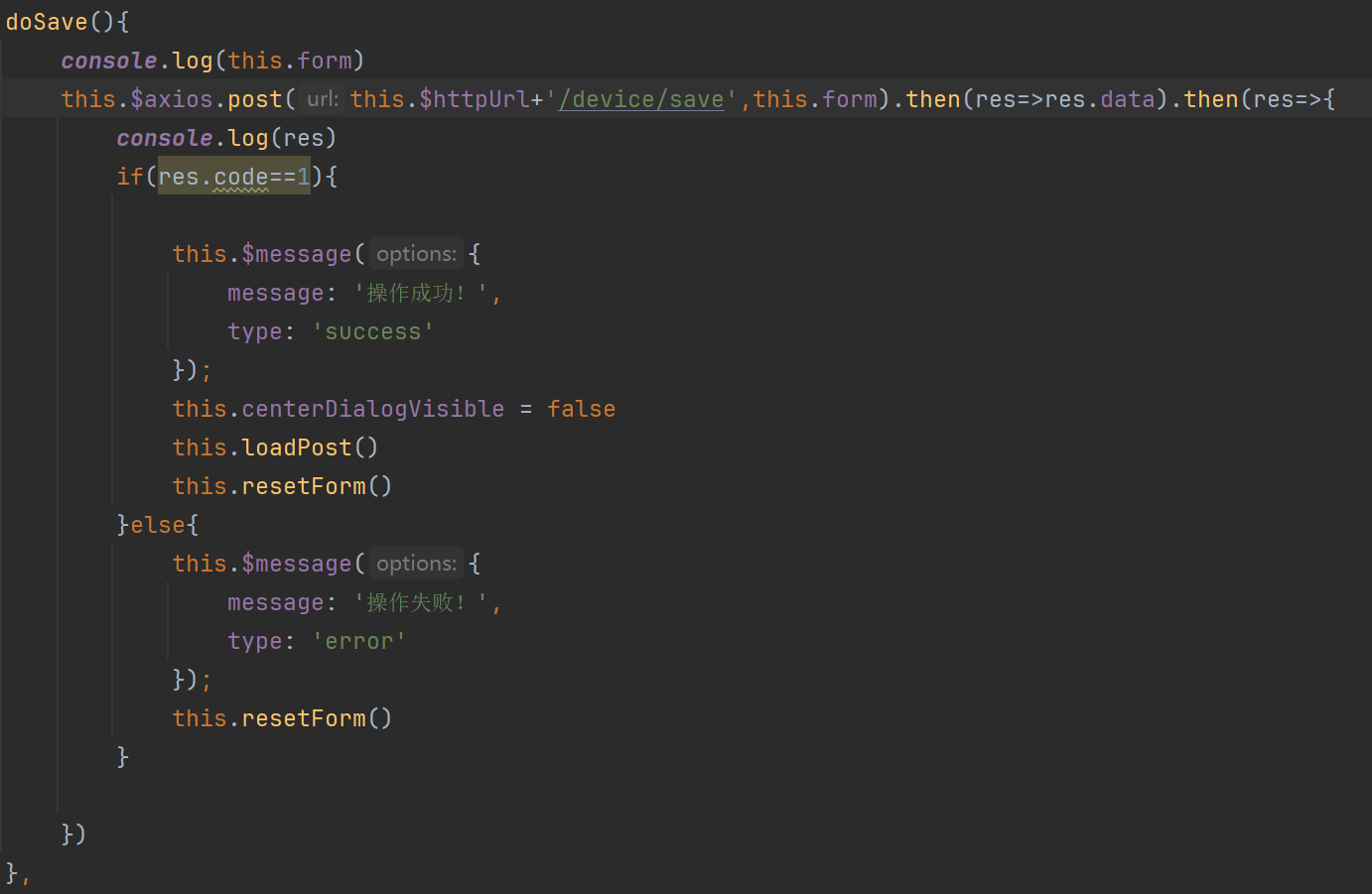


新增设备：

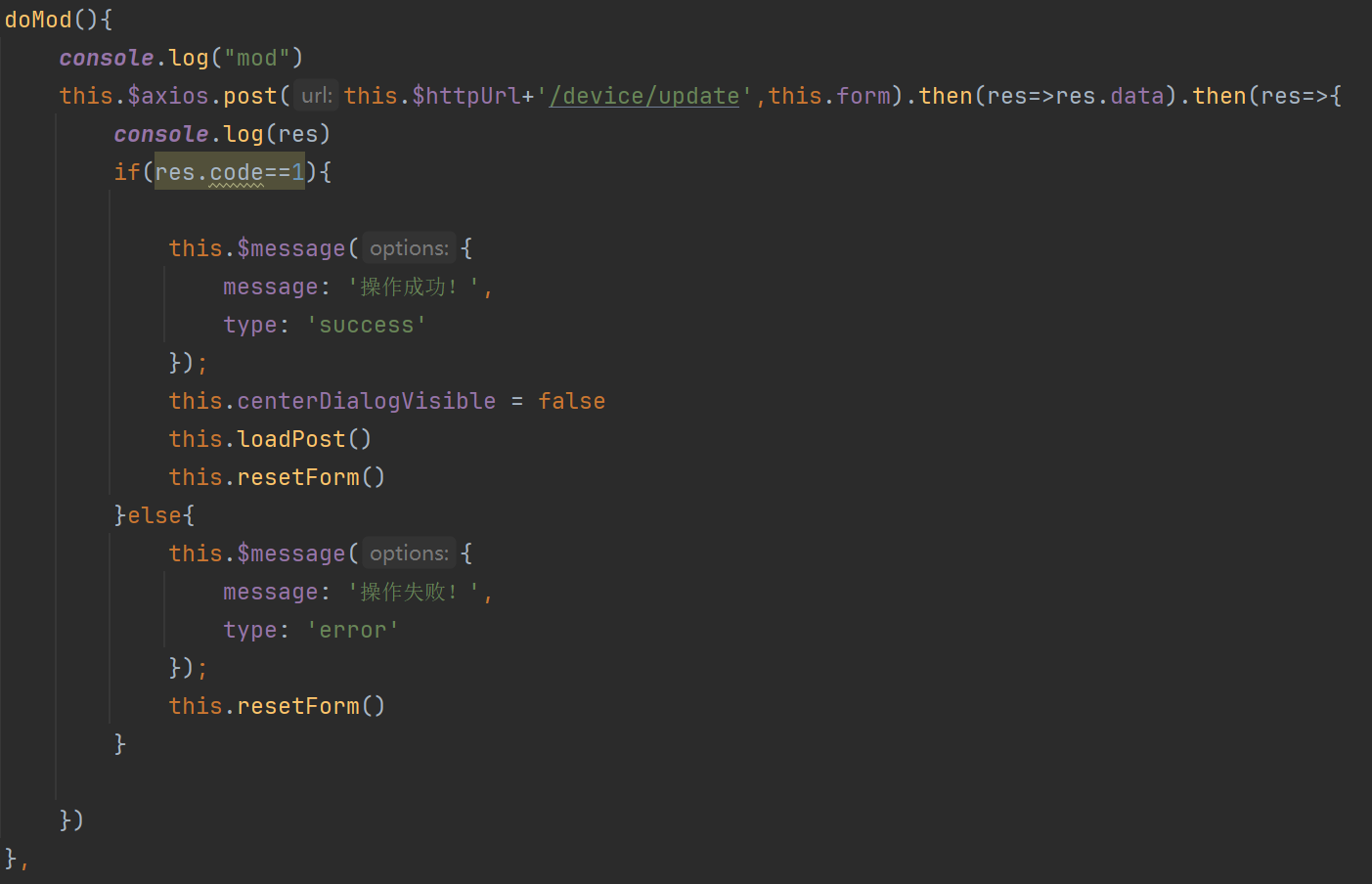


以上两个mod和add功能是针对弹出表单的内容设置。

储存操作：

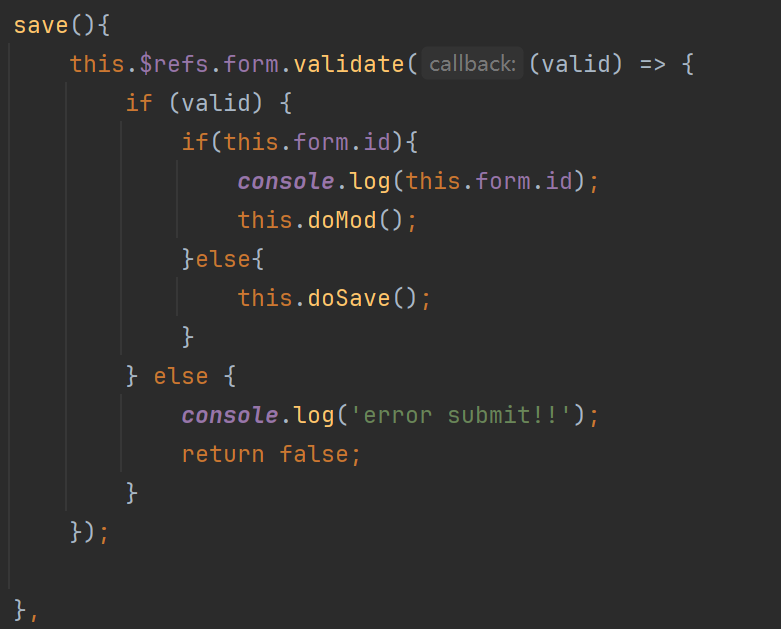


修改操作：



按钮确认对应的操作：

根据form.id是否存在判断是新增还是编辑，然后调用对应的处理函数



从后台加载数据：



### 3.1.5数据查询界面：

展示消息



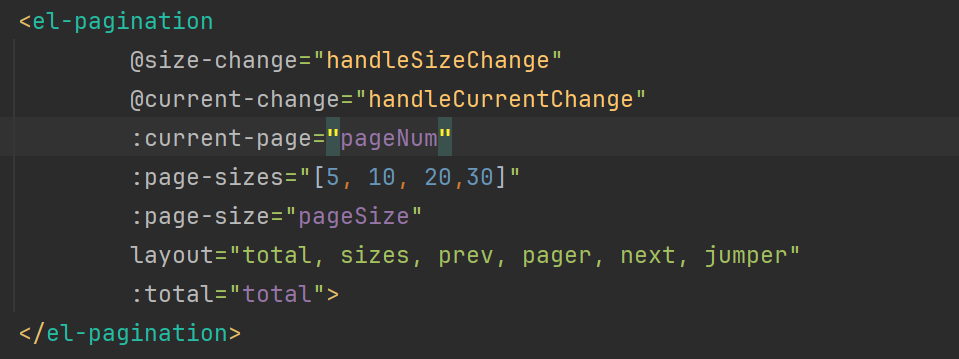
查询输入框和按钮界面：



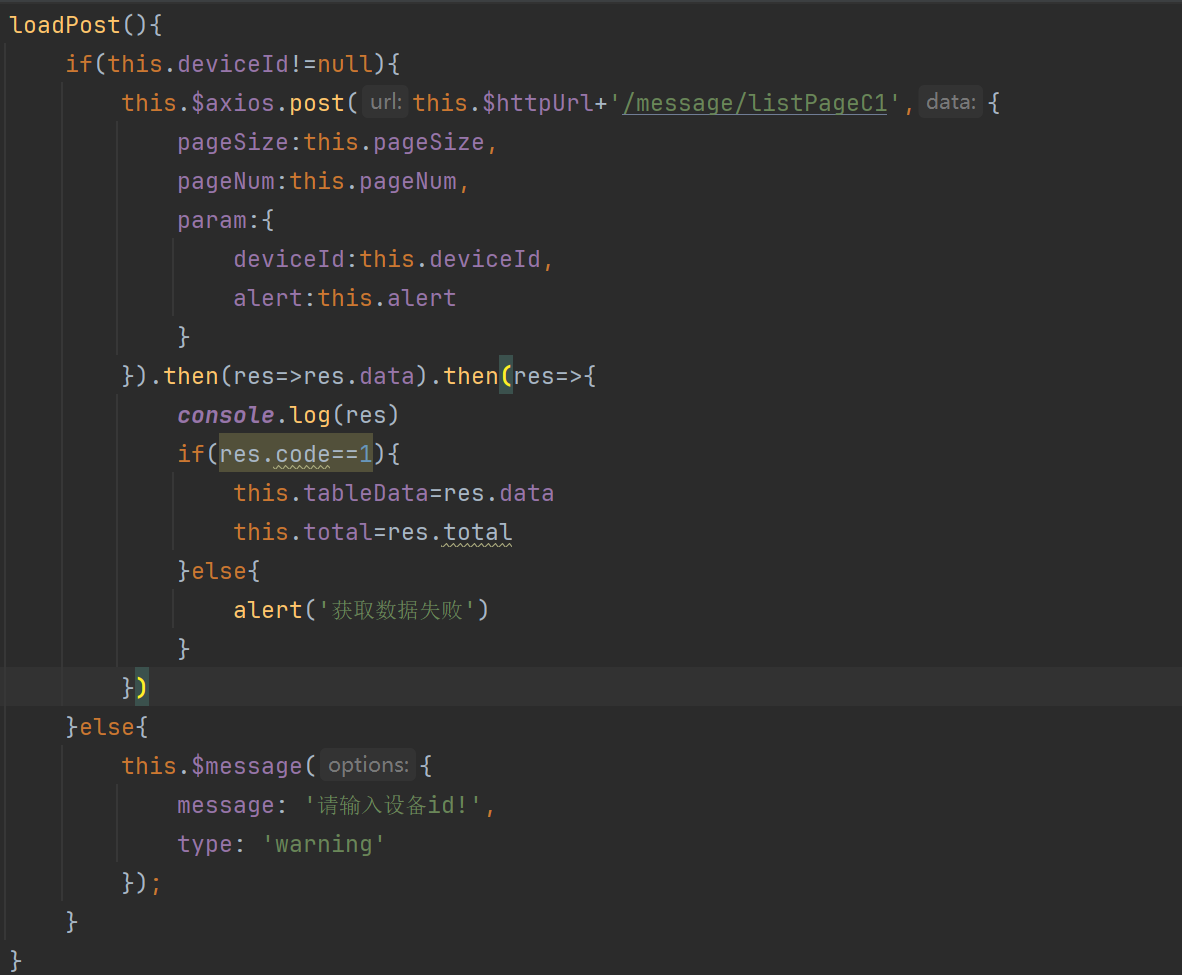
消息框：



分页：



数据查询：

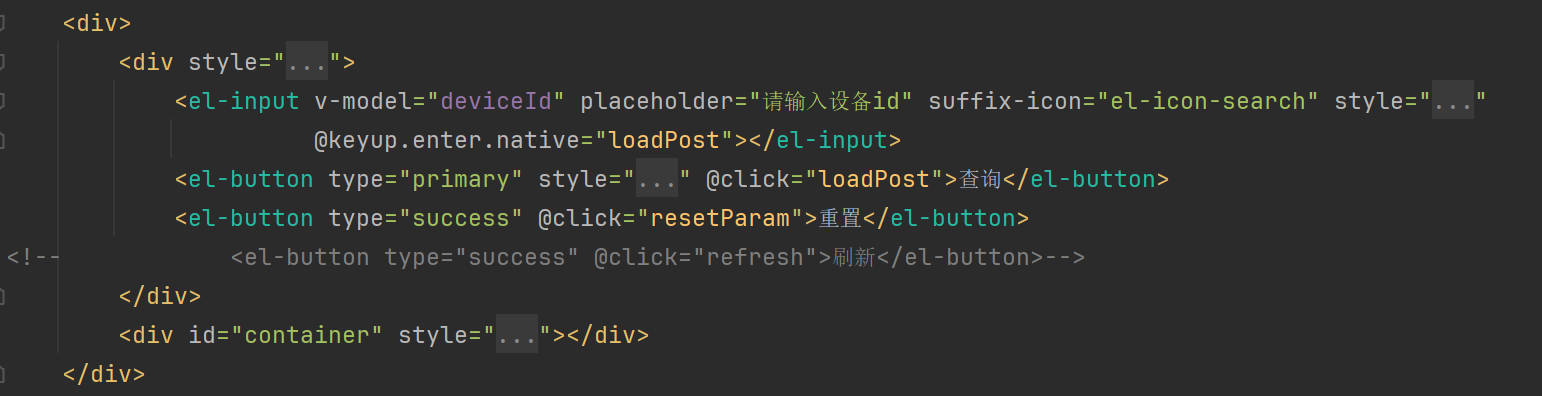


### 3.1.6地图显示：

根据选中的设备号，查询对应设备的信息中包含的经纬度，展示变化路径，标记点

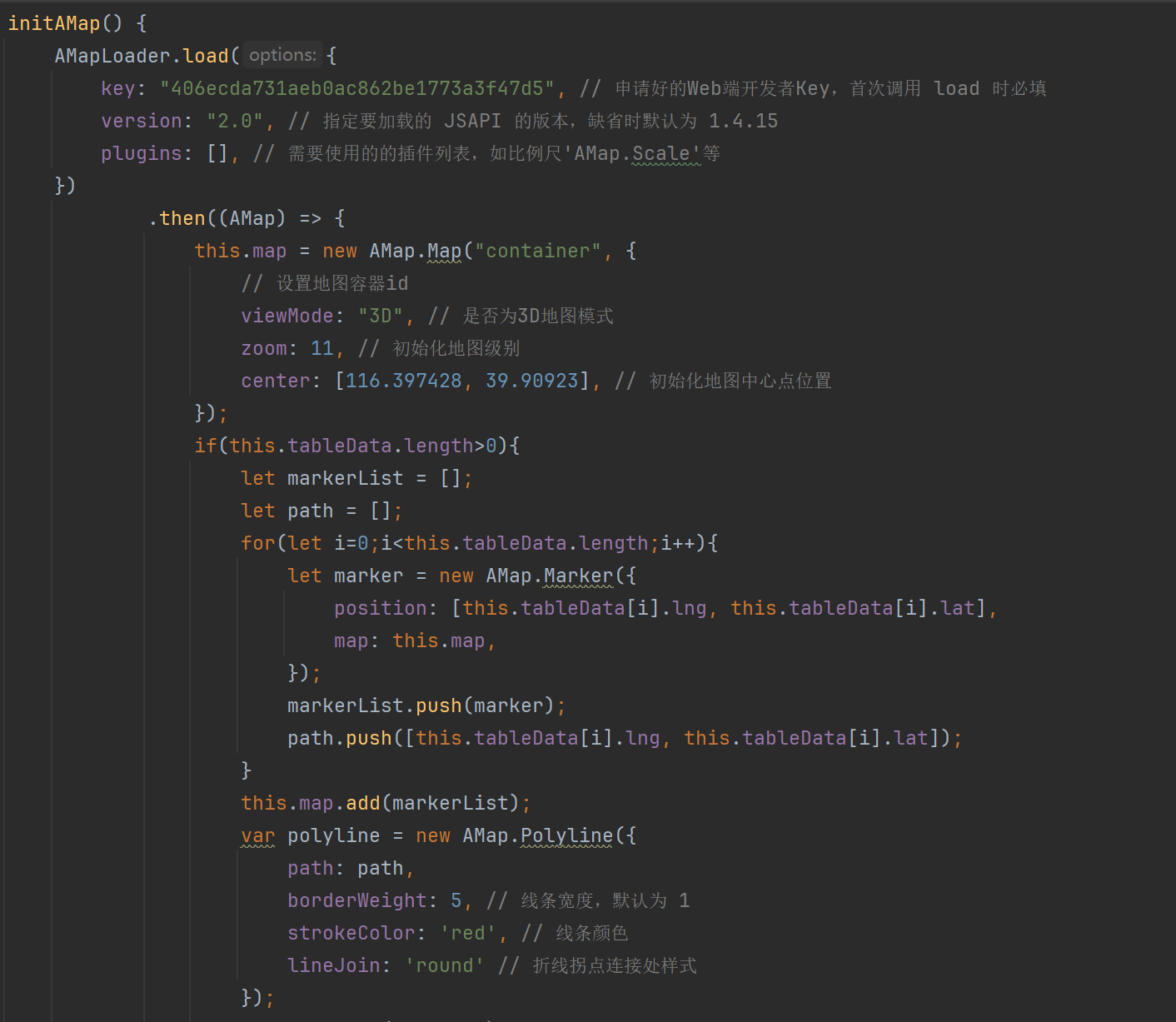


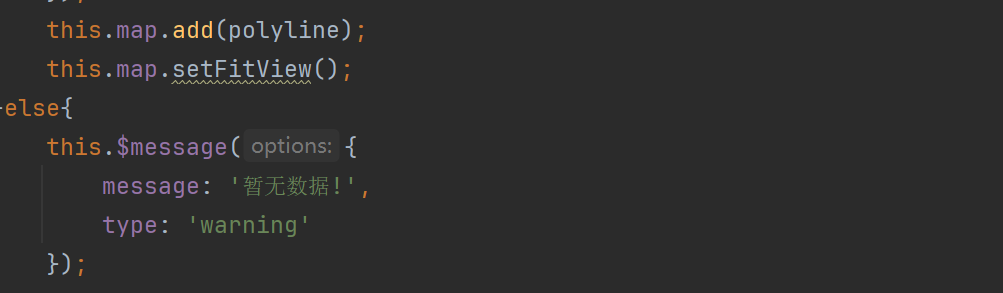
查询框和地图组件：



创建地图：

每次点击查询时都创建地图，并将查询到的点标记，连线，并使用setFitView()函数调整窗口，合理缩放拖拽地图看到所有点的标记。





## 3.2 后端实现

后端采用springboot框架，同时使用Mybatis提供数据库支持。项目采用common，controller，entity，mapper，service文件夹进行规范管理。

主要功能实现：

### 3.2.1User：

controller：

使用GetMapping注解向前端暴露接口进行调用，并实现用户的增删改查功能。

@GetMapping("/list")  
public List<User> list(){  
 return userService.list();  
}  
@GetMapping("/findByName")  
public Result findByNo(@RequestParam String name){  
 List list = userService.lambdaQuery().eq(User::getName,name).list();  
 return list.size()>0?Result.*res*(1,list):Result.*res*(0);  
}  
  
//更新  
@PostMapping("/update")  
public Result update(@RequestBody User user){  
 return userService.updateById(user)?Result.*res*(1):Result.*res*(0);  
}  
//删除  
@GetMapping("/del")  
public Result del(@RequestParam String id){  
 return userService.removeById(id)?Result.*res*(1):Result.*res*(0);  
}  
  
//登录  
@PostMapping("/login")  
public Result login(@RequestBody User user){  
 List list = userService.lambdaQuery()  
 .eq(User::getName,user.getName())  
 .eq(User::getPassword,user.getPassword()).list();  
  
  
 if(list.size()>0){  
 User user1 = (User)list.get(0);  
 List menuList = menuService.lambdaQuery().like(Menu::getMenuright,user1.getRoleId()).list();  
 HashMap res = new HashMap();  
 res.put("user",user1);  
 res.put("menu",menuList);  
 return Result.*res*(1,res);  
 }  
 return Result.*res*(0);  
}  
  
//注册  
@PostMapping("/register")  
public Result register(@RequestBody User user){  
 return userService.save(user)?Result.*res*(1):Result.*res*(0);  
}

### 3.2.2 Device

Controller：

实现对设备基本的增删改查功能

//新增  
@PostMapping("/save")  
public Result register(@RequestBody Device device){  
 System.*out*.println("device=="+device);  
 return deviceService.save(device)?Result.*res*(1):Result.*res*(0);  
}  
  
//删除  
@GetMapping("/del")  
public Result del(@RequestParam String id){  
 return deviceService.removeById(id)?Result.*res*(1):Result.*res*(0);  
}  
  
//更新  
@PostMapping("/update")  
public Result update(@RequestBody Device device){  
 return deviceService.updateById(device)?Result.*res*(1):Result.*res*(0);  
}  
  
@GetMapping("/findByName")  
public Result findByNo(@RequestParam String name){  
 List list = deviceService.lambdaQuery().eq(Device::getName,name).list();  
 return list.size()>0?Result.*res*(1,list):Result.*res*(0);  
}

列出查询的设备信息，动态根据前端传入的信息进行查询。

@PostMapping("/listPageC1")  
public Result listPageC1(@RequestBody QueryPageParam query){  
 HashMap param = query.getParam();  
 Integer userId = (Integer) param.get("userId");  
 System.*out*.println("userId=="+userId);  
 String name = (String)param.get("name");  
 Integer type = param.get("type") == null?0:(Integer)param.get("type");  
  
 Page<Device> page = new Page();  
 page.setCurrent(query.getPageNum());  
 page.setSize(query.getPageSize());  
  
 LambdaQueryWrapper<Device> lambdaQueryWrapper = new LambdaQueryWrapper();  
   
 lambdaQueryWrapper.eq(Device::getUserid,userId);  
   
 if(StringUtils.*isNotBlank*(name) && !"null".equals(name)){  
 lambdaQueryWrapper.like(Device::getName,name);  
 }  
 if(type != 0){  
 lambdaQueryWrapper.eq(Device::getType,type);  
 }  
 IPage result = deviceService.pageCC(page,lambdaQueryWrapper);  
 System.*out*.println("total=="+result.getTotal());  
  
 return Result.*res*(1, result.getRecords(),result.getTotal());  
}

查询当前设备中各种类的数量和活跃设备数，并传给前端。

@PostMapping("/countType")  
public Result countType(@RequestBody QueryPageParam query){  
 HashMap param = query.getParam();  
 Integer userId = (Integer) param.get("userId");  
 System.*out*.println("userId=="+userId);  
 List<Integer> l = new ArrayList<>();  
 for(int i = 1; i <= 4; i++){  
 LambdaQueryWrapper<Device> lambdaQueryWrapper = new LambdaQueryWrapper();  
 lambdaQueryWrapper.eq(Device::getUserid,userId);  
 lambdaQueryWrapper.eq(Device::getType,i);  
 int count = deviceService.count(lambdaQueryWrapper);  
 l.add(count);  
 }  
   
 //查询总数  
 LambdaQueryWrapper<Device> lambdaQueryWrapper = new LambdaQueryWrapper();  
 lambdaQueryWrapper.eq(Device::getUserid,userId);  
 l.add(deviceService.count(lambdaQueryWrapper));  
   
 //查询活跃数  
 lambdaQueryWrapper.eq(Device::getActive,1);  
 l.add(deviceService.count(lambdaQueryWrapper));  
  
 return Result.*res*(1, l);  
}

其中调用的pageCC函数在DeviceMapper.xml中进行实现。

<select id="pageCC" resultType="com.wms.entity.Device">  
 select \* from device ${ew.customSqlSegment}  
</select>

DeviceServiceImpl:

在后端处理完iotclient的信息后，对device中的信息进行检查，如果发现活跃状态（activate）需要更新，则修改。

public void updateAlert(Integer deviceid, Integer alert) {  
 Device device = deviceMapper.selectById(deviceid);  
 if(device == null)return;  
 if(device.getActive() == 0 && alert == 0){  
 device.setActive(1);  
 deviceMapper.updateById(device);  
 }else if(device.getActive() == 1 && alert == 1){  
 device.setActive(0);  
 deviceMapper.updateById(device);  
 }  
}

### 3.2.3MessageForm

Controller：

根据前端传入的设备id在页面列出message的信息。

@PostMapping("/listPageC1")  
public Result listPageC1(@RequestBody QueryPageParam query){  
 HashMap param = query.getParam();  
 Integer deviceId =Integer.*valueOf*((String) param.get("deviceId"));  
 Integer alert = param.get("alert") == null?-1:Integer.*parseInt*((String) param.get("alert"));  
 System.*out*.println("alert=="+alert);  
 Page<MessageForm> page = new Page();  
 page.setCurrent(query.getPageNum());  
 page.setSize(query.getPageSize());  
   
 LambdaQueryWrapper<MessageForm> lambdaQueryWrapper = new LambdaQueryWrapper();  
   
 lambdaQueryWrapper.eq(MessageForm::getDeviceid,deviceId);  
   
 if(alert != -1){  
 lambdaQueryWrapper.eq(MessageForm::getAlert,alert);  
 }  
 System.*out*.println(("准备查询"));  
 IPage result = messageFormService.pageCC(page,lambdaQueryWrapper);  
 System.*out*.println("total=="+result.getTotal());  
   
 return Result.*res*(1, result.getRecords(),result.getTotal());  
}

根据前端传入的userId，先遍历Device表找到该用户拥有的device，然后根据每个deviceId查找对应的消息，并将消息数加起来。

@PostMapping("/messageNum")  
public Result countType(@RequestBody QueryPageParam query) {  
 HashMap param = query.getParam();  
 Integer userId = (Integer) param.get("userId");  
 System.*out*.println("userId==" + userId);  
 //查询设备  
 List list = deviceService.lambdaQuery().eq(Device::getUserid,userId).list();  
 Integer messageNum = 0;  
 if(list.size() != 0){  
 for(int i = 0;i<list.size();i++){  
 Device device = (Device) list.get(i);  
 Integer deviceId = device.getId();  
 System.*out*.println("deviceId=="+deviceId);  
 //获得消息数  
 messageNum += messageFormService.lambdaQuery().eq(MessageForm::getDeviceid,deviceId).count();  
 }  
 }  
 return Result.*res*(1, messageNum);  
   
}

MassageFormServiceImpl：

解析监听到的消息，并进行解析储存。先把json格式消息变成中间类message，然后将message中的clientId中的编号提取出来作为deviceId，并将message赋值给messageForm，最后让deviceService中的函数更新警报值，并将messageForm插入数据库中。

@Override  
 public boolean msgHandle(String msg) {  
 ObjectMapper objectMapper = new ObjectMapper();  
 MessageForm messageForm = new MessageForm();  
// DeviceServiceImpl deviceService = new DeviceServiceImpl();  
 try {  
 if(!copy(objectMapper.readValue(msg,Message.class),messageForm))return false;  
 deviceService.updateAlert(messageForm.getDeviceid(),messageForm.getAlert());  
 messageFormMapper.insert(messageForm);  
 return true;  
 }catch (Exception e){  
 e.printStackTrace();  
 return false;  
 }  
 }  
   
   
 public boolean copy(Message message, MessageForm messageForm) {  
 try {  
 messageForm.setAlert(message.getAlert());  
 String name = message.getClientId();  
   
 Pattern pattern = Pattern.*compile*("\\d+");  
 Matcher matcher = pattern.matcher(name);  
 if(!matcher.find())return false;  
 Integer deviceId = Integer.*parseInt*(matcher.group());  
 messageForm.setDeviceid(deviceId);  
   
 messageForm.setInfo(message.getInfo());  
 messageForm.setLat(message.getLat());  
 messageForm.setLng(message.getLng());  
 messageForm.setTimestamp(message.getTimestamp());  
 messageForm.setValue(message.getValue());  
 return true;  
 }catch (Exception e){  
 e.printStackTrace();  
 return false;  
 }  
 }

### 3.2.4 Menu

对前端的菜单页进行动态管理，通过读取后端menu数据库中的信息，展示菜单页，方便页面修改和用户权限管理。

### 3.2.5 Result

返回统一格式。

private int code;//编码 1/0  
private String msg;//成功/失败  
private Long total;//总记录数  
private Object data;//数据

### 3.2.6 Message

消息转化为MessageForm的中间格式。

private Integer alert;  
private String clientId;  
private String info;  
private Double lat;  
private Double lng;  
private String timestamp;  
private Integer value;

### 3.2.7其他

还有一些其他的支持，如Mqtt消息的客户端编写，其中broker使用的是mosquitto，所以不在这里说明，

mqtt消息处理回调函数：

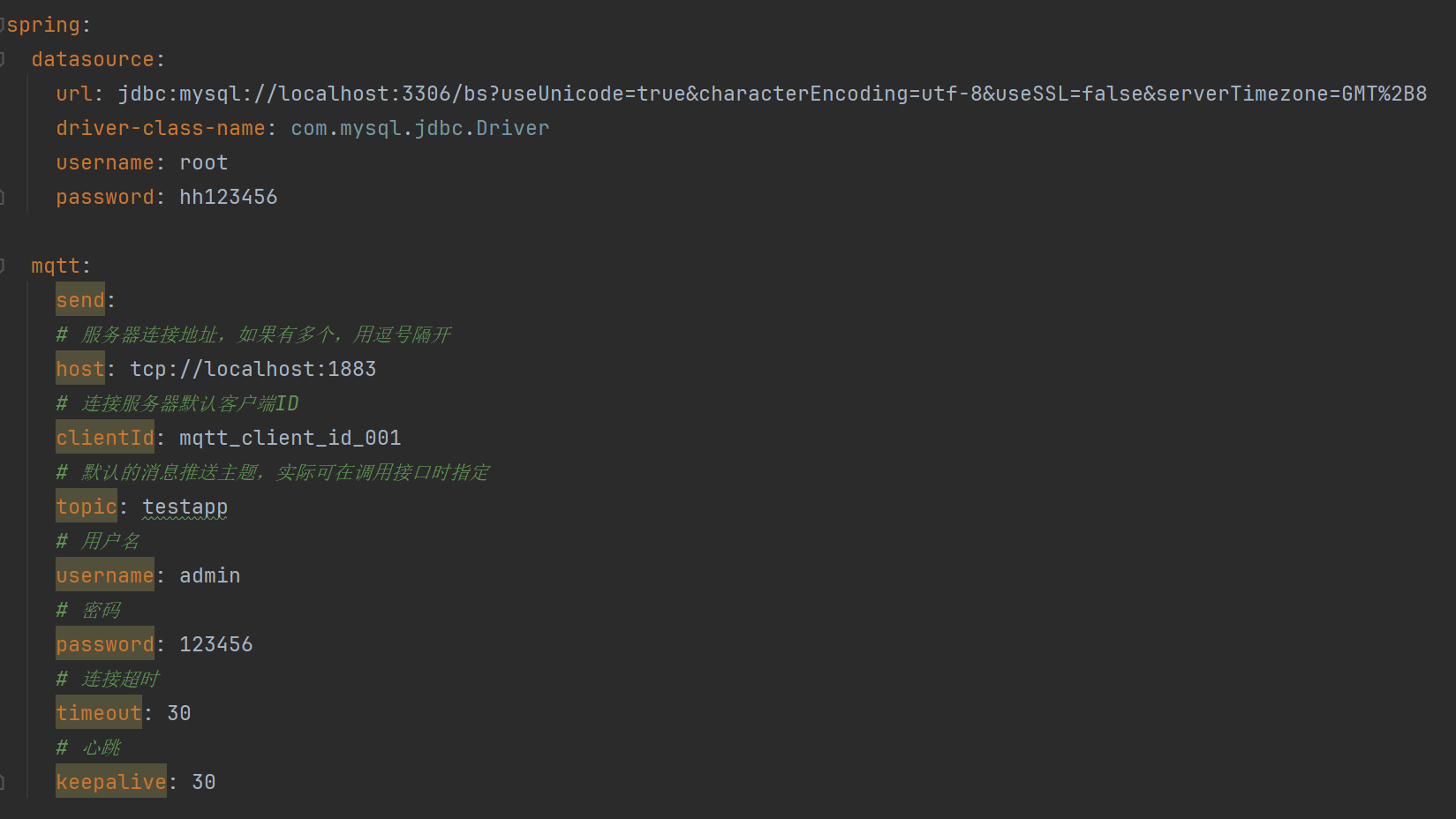
@Override  
public void messageArrived(String topic, MqttMessage message) {  
 try {  
 String msg = new String(message.getPayload());  
 System.*out*.println("收到topic:" + topic + " 消息：" + msg);  
   
 MessageFormService messageFormService = new MessageFormServiceImpl();  
 if(messageFormService.msgHandle(msg)){  
 System.*out*.println("消息处理成功");  
 }else{  
 System.*out*.println("消息处理失败");  
 }  
 } catch (Exception e) {  
 System.*out*.println("处理mqtt消息异常:" + e);  
 }  
}

# 4. 部署与维护

## 4.1 部署流程

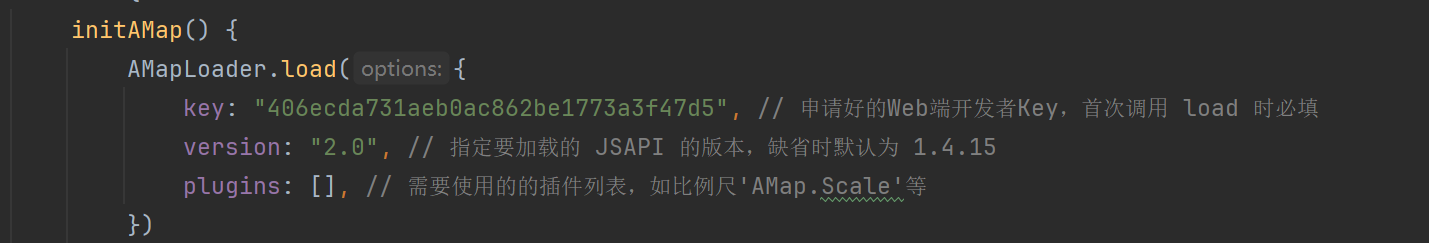
配置文件修改：

src/main/resources/application.yml

修改对应的mysql库名用户名密码，mqtt用户名密码。

src/components/main/Map.vue

修改高德API的key



运行table.sql，建表。

# 5. 项目总结

在物联网设备管理平台的设计与开发过程中，我们成功地实现了用户登录注册、设备统计管理、消息查看统计等核心功能。通过采用Spring Boot和Vue.js的架构，我们建立了一个高效、可扩展的B/S体系软件，同时在安全性和用户友好性上取得了显著的成果。

亮点与成果：

采用前后端分离技术，便于代码的后期维护。

成功整合前后端技术，实现了流畅的用户界面和高效的后端逻辑。

设计了安全且灵活的用户权限控制机制，保障系统的安全性。

通过良好的交互设计，提高了系统的易用性，使用户能够轻松完成各种操作。

心得体会：

通过项目实践，我深入理解了Spring Boot和Vue的使用和整合，对前后端分离的开发模式有了更清晰的认识。我也学到了如何设计和管理数据库，调用Mybatis进行数据的增删改查，以及数据库与后端的良好结合。

在项目开发过程中，遇到了各种技术和逻辑上的问题。通过查阅文档、参考示例代码，我学到了如何高效地解决问题和调试代码。并且，我也学会查看网页控制台的报错来查看问题。

通过项目，我学到了如何进行系统设计，包括模块划分、技术选型、数据库设计等方面。这让我对一个完整系统的构建有了更全面的认识。

在整个实验过程中，我深感实际项目开发远比理论课程更为真实和具有挑战性。通过不断的努力，我成功地完成了一个功能完善的物联网设备管理平台，这为我今后在软件开发领域的学习和实践奠定了坚实的基础。同时，我也认识到了学习是一个不断迭代和提升的过程，希望能在未来的项目中继续积累经验，不断提升自己的技能水平。