

浙江大学

本科实验报告

课程名称: B/S 体系软件设计

姓 名:

学 院: 计算机科学与技术学院

系: 计算机科学与技术

专 业:

学 号:

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： 专业： 计算机科学与技术 学号：

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
1.1 背景与目的.....	3
1.1.1 背景.....	3
1.1.2 目的.....	3
1.2 技术选型.....	3
1.2.1 Spring Boot.....	3
1.2.2 Vue.js	4
1.3 功能概述.....	4
1.3.1 用户登录注册.....	4
1.3.2 设备统计管理.....	4
1.3.3 消息查看统计.....	4
1.3.4 安全与权限控制.....	5
1.3.5 界面友好性.....	5
2. 系统架构.....	5
2.1 前端架构.....	5
2.2 后端架构.....	6
2.3 数据库设计.....	9
3. 技术实现细节.....	11
3.1 前端实现.....	11
3.1.1 登录界面.....	11
3.1.2 注册界面：	14
3.1.3 首页：	17
3.1.4 设备配置界面：	22
3.1.5 数据查询界面：	28
3.1.6 地图显示：	30
3.2 后端实现.....	32
3.2.1User:	33
3.2.2 Device	34

3.2.3 MessageForm.....	37
3.2.4 Menu	39
3.2.5 Result	39
3.2.6 Message.....	40
3.2.7 其他.....	40
4. 部署与维护.....	41
4.1 部署流程.....	41
5. 项目总结.....	41

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

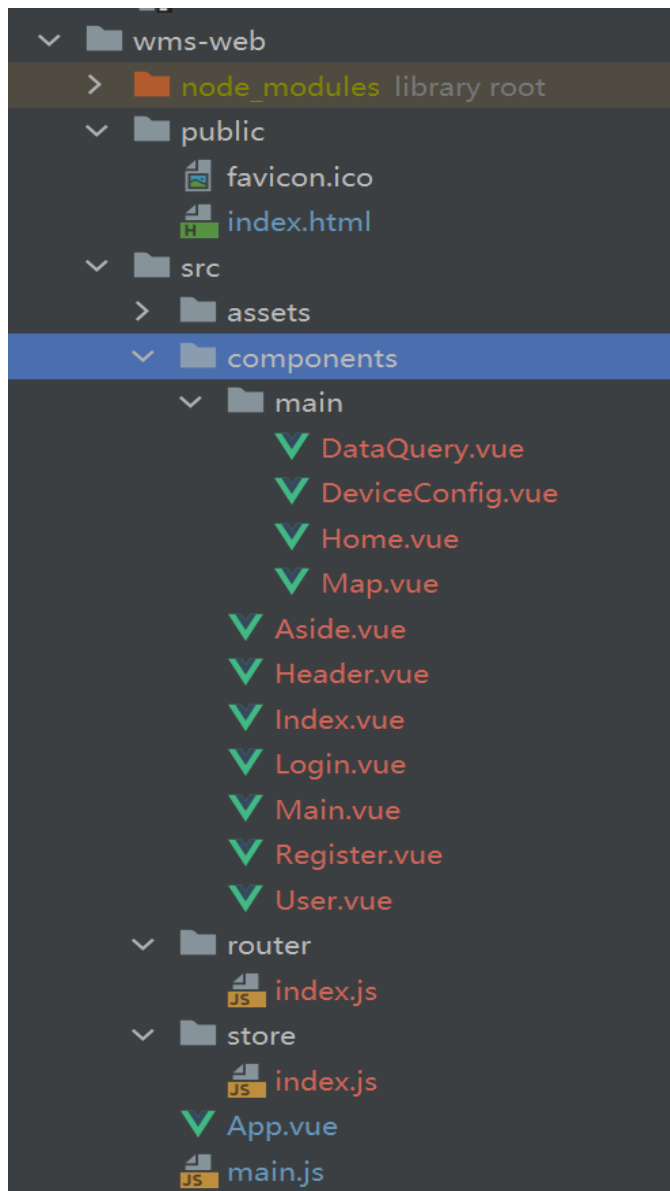
Responsiveness：界面能够自适应不同设备和屏幕尺寸，提供良好的用户体验。

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：

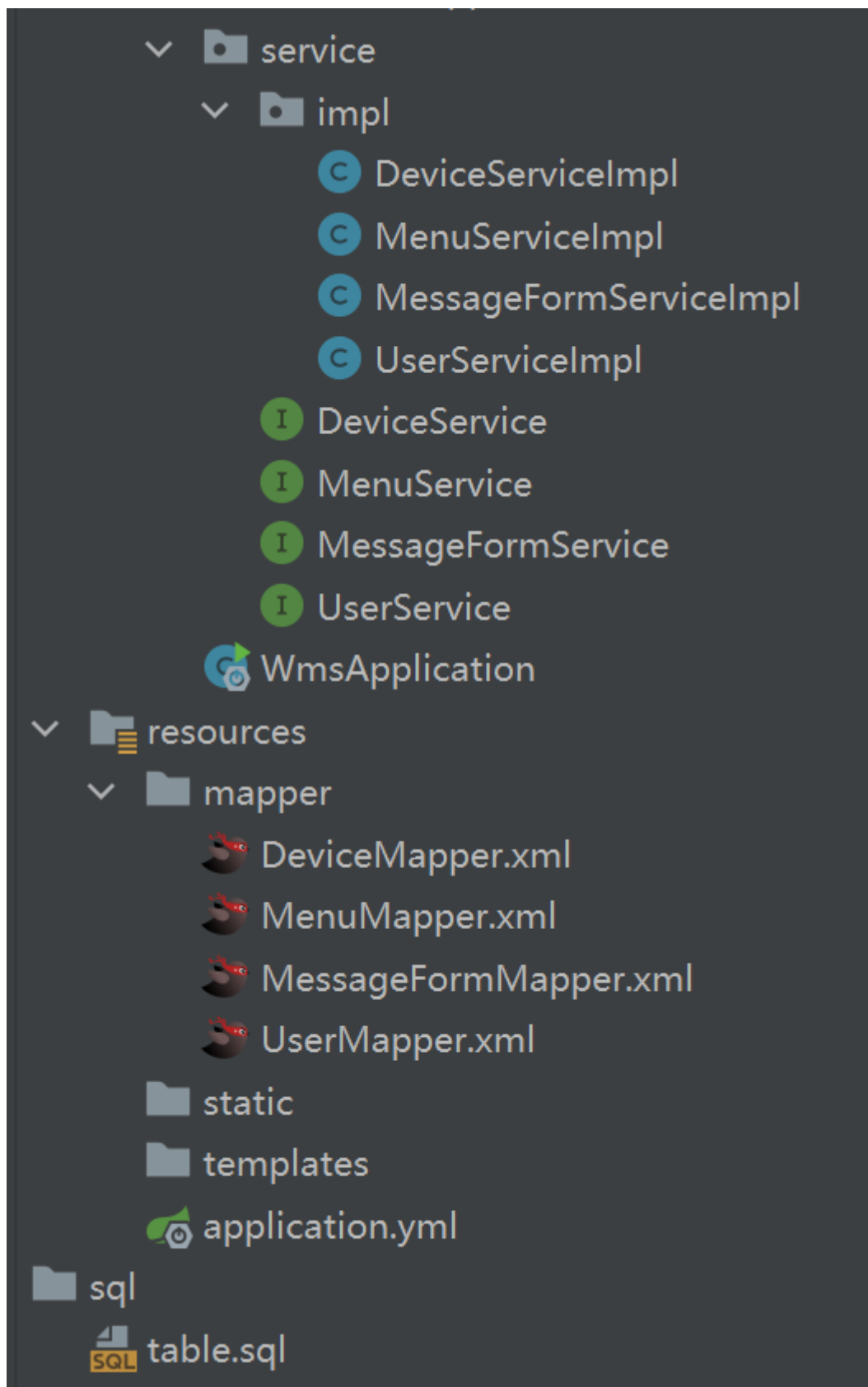


前端使用 axios 调用后端接口实现数据传递，后端统一传回 Result 格式的数据。

2.2 后端架构

后端框架组织形式：

- ▼  common
 - ▼  mqtt
 -  MqttConsumer
 -  MqttConsumerCallback
 -  MqttController
 -  PropertiesUtil
 -  SpringUtil
 -   CodeGenerator
 -  CorsConfig
 -  MybatisPlusConfig
 -  QueryPageParam
 -  Result
 - ▼  controller
 -  DeviceController
 -  MenuController
 -  MessageFormController
 -  UserController
 - ▼  entity
 -  Device
 -  Menu
 -  Message
 -  MessageForm
 -  User
 - ▼  mapper
 -   DeviceMapper
 -   MenuMapper
 -   MessageFormMapper
 -   UserMapper



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

A user login form titled "用户登录" (User Login) is centered on a light blue background. The form is white with rounded corners and contains two input fields: "账号" (Account) and "密码" (Password), both marked with a red asterisk. Below the fields are two blue buttons: "注册" (Register) and "登录" (Login).

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。

```

<template>
  <div class="loginBody">
    <div class="loginDiv">
      <div class="login-content">
        <h1 class="login-title">用户登录</h1>
        <el-form :model="loginForm" label-width="100px"
          :rules="rules" ref="loginForm">
          <el-form-item label="账号" prop="name">
            <el-input style="..." type="text" v-model="loginForm.name"
              autocomplete="off" size="small"></el-input>
          </el-form-item>
          <el-form-item label="密码" prop="password">
            <el-input style="..." type="password" v-model="loginForm.password"
              show-password autocomplete="off" size="small" @keyup
              .enter.native="confirm"></el-input>
          </el-form-item>
          <el-form-item>
            <el-button type="primary" @click="register"
              :disabled="confirm_disabled">注 册</el-button>
            <el-button type="primary" @click="confirm"
              :disabled="confirm_disabled">登 录</el-button>
          </el-form-item>
        </el-form>
      </div>
    </div>
  </div>
</template>

```

confirm 后的数据处理:

```
confirm(){
  this.confirm_disabled=true;
  this.$refs.loginForm.validate( callback: (valid) => {
    if (valid) { //valid成功为true, 失败为false
      //去后台验证用户名密码
      this.$axios.post( url: this.$httpUrl+'/user/login',this.loginForm)
        .then(res=>res.data).then(res=>{
          console.log(res)
          if(res.code==1){
            //存储
            sessionStorage.setItem("CurUser",JSON.stringify(res.data
              .user))

            console.log(res.data.menu)
            this.$store.commit( type: "setMenu",res.data.menu)
            //跳转到主页
            this.$router.replace( location: '/Index');
          }else{
            this.confirm_disabled=false;
            alert(' 校验失败, 用户名或密码错误! ');
            return false;
          }
        });
    } else {
      this.confirm_disabled=false;
      console.log(' 校验失败');
      return false;
    }
  });
}
```

3.1.2 注册界面：



The image shows a user registration form titled "用户注册" (User Registration) centered on a light blue background. The form is a white rounded rectangle containing three input fields, each preceded by a red asterisk indicating a required field. The fields are labeled "账号" (Username), "密码" (Password), and "邮箱" (Email). Below the input fields are two blue buttons: "取消" (Cancel) on the left and "注册" (Register) on the right.

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。

```

<template>
  <div class="loginBody">
    <div class="loginDiv">
      <div class="login-content">
        <h1 class="login-title">用户注册</h1>
        <el-form :model="registerForm" label-width="100px"
          :rules="rules" ref="registerForm">
          <el-form-item label="账号" prop="name">
            <el-input style="..." type="text" v-model="registerForm.name"
              autocomplete="off" size="small"></el-input>
          </el-form-item>
          <el-form-item label="密码" prop="password">
            <el-input style="..." type="password" v-model="registerForm
              .password"
              show-password autocomplete="off" size="small" ></el-input>
          </el-form-item>
          <el-form-item label="邮箱" prop="email">
            <el-input style="..." type="text" v-model="registerForm.email"
              autocomplete="off" size="small"></el-input>
          </el-form-item>
          <el-form-item>
            <el-button type="primary" @click="cancel"
              :disabled="confirm_disabled">取 消</el-button>
            <el-button type="primary" @click="register"
              :disabled="confirm_disabled">注 册</el-button>
          </el-form-item>
        </el-form>
      </div>
    </div>
  </div>
</template>

```

账号重复检查:

```

let checkDuplicate =(rule,value,callback)=>{
  this.$axios.get( url: this.$httpUrl+"/user/findByName?name="+this
    .registerForm.name).then(res=>res.data).then(res=>{
    if(res.code!=1){
      callback()
    }else{
      callback(new Error('账号已经存在'));
    }
  })
};

```

邮箱格式检查:

```
let checkEmail =(rule,value,callback)=>{
  var reg = /^[a-zA-Z\d][\w-]{2,}@\w{2,}\.([a-z]{2,})(\.[a-z]{2,})?$/;
  if(reg.test(value)) {
    callback()
  }else{
    callback(new Error('邮箱格式不正确'));
  }
};
```

格式检查调用：

```
rules:{
  name: [
    { required: true, message: '请输入账号', trigger: 'blur' },
    { validator: checkDuplicate, trigger: 'blur' }
  ],
  password: [
    { required: true, message: '请输入密码', trigger: 'blur' },
    { min: 6, max: 20, message: '长度在 6 到 20 个字符', trigger: 'blur' }
  ],
  email: [
    { required: true, message: '请输入邮箱', trigger: 'blur' },
    { validator: checkEmail, message: '请输入正确的邮箱地址', trigger: 'blur' }
  ]
}
},
```

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。


```

register(){
  this.confirm_disabled=true;
  this.$refs.registerForm.validate( callback: (valid) => {
    if (valid) { //valid成功为true, 失败为false
      //去后台验证用户名密码
      this.$axios.post( url: this.$httpUrl+ '/user/register',this
        .registerForm).then(res=>res.data).then(res=>{
        console.log(res)
        if(res.code==1){
          //跳转到登录
          this.$message( options: {message: '注册成功! ', type:
            'success'});
          this.$router.replace( location: '/');
        }else{
          this.confirm_disabled=false;
          alert('注册失败! 请重试');
          return false;
        }
      });
    } else {
      this.confirm_disabled=false;
      console.log('注册失败');
      return false;
    }
  });
},

```

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。



```

<template>
  <div>
    <el-descriptions title="物联网设备管理中心" :column="3" size="40" border>
      <el-descriptions-item>
        <template slot="label">
          <i class="el-icon-s-platform"></i>
          设备总数
        </template>
        {{deviceNum}}
      </el-descriptions-item>
      <el-descriptions-item>
        <template slot="label">
          <i class="el-icon-success"></i>
          当前在线
        </template>
        {{activeNum}}/{{deviceNum}}
      </el-descriptions-item>
      <el-descriptions-item>
        <template slot="label">
          <i class="el-icon-info"></i>
          消息总数
        </template>
        {{dataReceive}}
      </el-descriptions-item>
    </el-descriptions>
    <div class="chart-container">
      <!--
      <div id="charts" style="width: 400px;height: 300px;"></div>-->
      <div id="charts" style="..." ></div>
    </div>
  </div>
</template>

```

使用 echarts 绘制饼图:

```

drawPie() {
  this.charts = this.$echarts.init(document.getElementById( elementId: "charts"));
  this.charts.setOption({
    tooltip: {
      trigger: "item",
      formatter: "{a} <br/> {b}:{c} ({d}%)"
    },
    legend: {
      show: true,
      bottom: 0,
      left: "center",
      name: this.types
    },
    series: [
      {
        name: "统计数量",
        type: "pie",
        data:[],
        itemStyle: {
          emphasis: {
            shadowBlur: 10,
            shadowOffsetX: 0,
            shadowColor: "rgba(0, 0, 0, 0.5)"
          },
          color: function(params) {
            //自定义颜色
            var colorList = ["#7facdb", "#89e2b2", "#eab37b", "#d58888"];
            return colorList[params.dataIndex];
          }
        },
      },
    ]
  })
}

```

获取设备数据：

```

//动态获取数据
async initData() {
  this.$axios.post( url: this.$httpUrl+' /device/countType', data: {
    param:{
      userId:this.user.id
    }
  }).then(res=>res.data).then(res=>{
    console.log(res)
    var typeCount = [];
    if(res.code==1){
      console.log("获取成功! ")
      for(let i = 0; i < 4; i++){
        var obj = new Object();
        obj.name = this.types[i];
        obj.value = res.data[i];
        typeCount[i] = obj;
      }
      this.deviceNum = res.data[4];
      this.activeNum = res.data[5];
    }else{
      alert('获取数据失败')
    }
    this.charts.setOption({
      series:[{
        data: typeCount,
      }]
    });
  });
}
}

```

获取消息数:

```
// 获取消息数
this.$axios.post( url: this.$httpUrl+' /message/messageNum', data: {
  param:{
    userId:this.user.id
  }
}).then(res=>res.data).then(res=> {
  console.log(res)
  if (res.code == 1) {
    console.log("获取成功! ")
    this.dataReceive = res.data;
  } else {
    alert(' 获取数据失败')
  }
})
},
```

3.1.4 设备配置界面：

首页

设备配置

数据查询

地图显示

欢迎来到物联网设备管理平台

1 v

请输入名字

请选择类型

查询

重置

新增

ID	设备名	类型	描述	操作
1	device0001	车载设备	设备1描述	<div>编辑</div> <div>删除</div>
2	device0002	智能家居	设备2描述	<div>编辑</div> <div>删除</div>
3	device0003	可穿戴设备	设备3描述	<div>编辑</div> <div>删除</div>
4	device0004	其他设备	设备4描述	<div>编辑</div> <div>删除</div>
5	device0005	车载设备	设备5描述	<div>编辑</div> <div>删除</div>
6	device0006	车载设备	jdH666	<div>编辑</div> <div>删除</div>

共 6 条

10条/页

< 1 >

前往 1 页

输入框和按钮：

```

<div style="...">
  <el-input v-model="name" placeholder="请输入名字" suffix-icon="el-icon-search" style="..."
    @keyup.enter.native="loadPost"></el-input>
  <el-select v-model="type" placeholder="请选择类型" style="...">
    <el-option
      v-for="item in types"
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>
  <el-button type="primary" style="..." @click="loadPost">查询</el-button>
  <el-button type="success" @click="resetParam">重置</el-button>

  <el-button type="primary" style="..." @click="add">新增</el-button>
</div>

```

设备信息展示

```

<el-table :data="tableData"
  :header-cell-style="{ background: '#f2f5fc', color: '#555555' }"
  border
>
  <el-table-column prop="id" label="ID" width="60">
  </el-table-column>
  <el-table-column prop="name" label="设备名" width="180">
  </el-table-column>
  <el-table-column prop="type" label="类型" width="180">
    <template slot-scope="scope">
      <el-tag
        :type="scope.row.type === 1 ? 'primary' : (scope.row.type === 2 ? 'success' : (scope.row.type
          === 3 ? 'warning' : 'danger'))"
        disable-transitions>{{scope.row.type === 1 ? '车载设备' : (scope.row.type === 2 ? '智能家居' :
          (scope.row.type === 3 ? '可穿戴设备' : '其他设备'))}}</el-tag>
    </template>
  </el-table-column>
  <el-table-column prop="description" label="描述">
  </el-table-column>
  <el-table-column prop="operate" label="操作">
    <template slot-scope="scope">
      <el-button size="small" type="success" @click="mod(scope.row)">编辑</el-button>
      <el-popconfirm
        title="确定删除吗? "
        @confirm="del(scope.row.id)"
        style="..."
      >
        <el-button slot="reference" size="small" type="danger" >删除</el-button>
      </el-popconfirm>
    </template>
  </el-table-column>
</el-table>

```

分页功能:

```

<el-pagination
  @size-change="handleSizeChange"
  @current-change="handleCurrentChange"
  :current-page="pageNum"
  :page-sizes="[5, 10, 20, 30]"
  :page-size="pageSize"
  layout="total, sizes, prev, pager, next, jumper"
  :total="total">
</el-pagination>

```

修改设备信息弹窗：

```

<el-dialog
  title="提示"
  :visible.sync="centerDialogVisible"
  width="30%"
  center>

  <el-form ref="form" :rules="rules" :model="form" label-width="80px">
    <el-form-item label="设备名" prop="name">
      <el-col :span="20">
        <el-input v-model="form.name"></el-input>
      </el-col>
    </el-form-item>
    <el-form-item label="设备描述" prop="description">
      <el-col :span="20">
        <el-input v-model="form.description"></el-input>
      </el-col>
    </el-form-item>
    <el-form-item label="类型">
      <el-select v-model="form.type" placeholder="请选择">
        <el-option
          v-for="item in types"
          :key="item.value"
          :label="item.label"
          :value="item.value">
        </el-option>
      </el-select>
    </el-form-item>
  </el-form>
  <span slot="footer" class="dialog-footer">
    <el-button @click="centerDialogVisible = false">取 消</el-button>
    <el-button type="primary" @click="save">确 定</el-button>
  </span>

```

删除设备功能实现：


```

del(id){
  console.log(id)

  this.$axios.get( url: this.$httpUrl+' /device/del?id='+id).then(res=>res.data).then(res=>{
    console.log(res)
    if(res.code==1){
      this.$message( options: {
        message: '操作成功!',
        type: 'success'
      });
      this.loadPost()
    }else{
      this.$message( options: {
        message: '操作失败!',
        type: 'error'
      });
    }
  })
},

```

修改设备:

```

mod(row){
  console.log(row)

  this.centerDialogVisible = true
  this.$nextTick( cb: ()=>{
    //赋值到表单
    this.form.id = row.id
    this.form.name = row.name
    this.form.description = row.description
    this.form.type = row.type
  })
},

```

新增设备:

```

add(){
    console.log("add")
    this.centerDialogVisible = true
    this.$nextTick( cb: ()=>{
        this.resetForm()
        this.form.userid = this.user.id
    })
},

```

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：

```

doSave(){
    console.log(this.form)
    this.$axios.post( url: this.$httpUrl+ '/device/save',this.form).then(res=>res.data).then(res=>{
        console.log(res)
        if(res.code==1){
            this.$message( options: {
                message: '操作成功! ',
                type: 'success'
            });
            this.centerDialogVisible = false
            this.loadPost()
            this.resetForm()
        }else{
            this.$message( options: {
                message: '操作失败! ',
                type: 'error'
            });
            this.resetForm()
        }
    })
},

```

修改操作：

```

doMod(){
  console.log("mod")
  this.$axios.post( url: this.$httpUrl+' /device/update',this.form).then(res=>res.data).then(res=>{
    console.log(res)
    if(res.code==1){

      this.$message( options: {
        message: '操作成功! ',
        type: 'success'
      });
      this.centerDialogVisible = false
      this.loadPost()
      this.resetForm()
    }else{
      this.$message( options: {
        message: '操作失败! ',
        type: 'error'
      });
      this.resetForm()
    }
  })
},

```

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

```

save(){
  this.$refs.form.validate( callback: (valid) => {
    if (valid) {
      if(this.form.id){
        console.log(this.form.id);
        this.doMod();
      }else{
        this.doSave();
      }
    } else {
      console.log('error submit!!');
      return false;
    }
  });
},

```

从后台加载数据：

```
loadPost(){
  this.$axios.post( url: this.$httpUrl+ '/device/listPageC1', data: {
    pageSize:this.pageSize,
    pageNum:this.pageNum,
    param:{
      userId:this.user.id,
      name:this.name,
      type:this.type
    }
  }).then(res=>res.data).then(res=>{
    console.log(res)
    if(res.code==1){
      this.tableData=res.data
      this.total=res.total
    }else{
      alert('获取数据失败')
    }
  })
}
```

3.1.5 数据查询界面：

展示消息

首页

设备配置

数据查询

地图显示

欢迎来到物联网设备管理平台

1

3

查询重置

ID	设备id	消息类型	消息内容	纬度	经度	时间戳	设备发送值
12	3	正常	Device Data 2024/01/04 23:53:30	30.47327892780304	119.94717222452164	1704383610795	74
16	3	警报	Device Data 2024/01/04 23:53:33	30.265341567993165	120.08264516592027	1704383613873	92
17	3	正常	Device Data 2024/01/04 23:53:33	30.21236741542816	120.02562851905823	1704383613874	63
18	3	正常	Device Data 2024/01/04 23:53:33	30.28905715942383	120.06952562332154	1704383613875	31
28	3	正常	Device Data 2024/01/05 13:12:36	30.196308565139773	120.26999045610428	1704431556156	27
30	3	正常	Device Data 2024/01/05 13:12:37	30.434327483177185	120.40334892272949	1704431557169	6
36	3	正常	Device Data 2024/01/05 13:12:43	30.438784050941468	120.28424693346024	1704431563180	7
45	3	正常	Device Data 2024/01/05 13:12:51	30.449037170410158	120.32410438060761	1704431571182	13

共 8 条10条/页11前往1页

查询输入框和按钮界面：

```

<div style="...">
  <el-input v-model="deviceId" placeholder="请输入设备id" suffix-icon="el-icon-search" style="..."
    @keyup.enter.native="loadPost"></el-input>
  <el-select v-model="alert" placeholder="请选择消息类型" style="margin-left: 5px;">
    <el-option
      v-for="item in alerts"
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>
  <el-button type="primary" style="..." @click="loadPost">查询</el-button>
  <el-button type="success" @click="resetParam">重置</el-button>
</div>

```

消息框:

```

<el-table :data="tableData"
  :header-cell-style="{ background: '#f2f5fc', color: '#555555' }"
  border
>
  <el-table-column prop="id" label="ID" width="60">
  </el-table-column>
  <el-table-column prop="deviceId" label="设备id" width="180">
  </el-table-column>
  <el-table-column prop="alert" label="消息类型" width="180">
    <template slot-scope="scope">
      <el-tag
        :type="scope.row.alert === 0 ? 'success' : 'danger'"
        disable-transitions>{{scope.row.alert === 0 ? '正常' : '警报'}}</el-tag>
    </template>
  </el-table-column>
  <el-table-column prop="info" label="消息内容">
  </el-table-column>
  <el-table-column prop="lat" label="纬度">
  </el-table-column>
  <el-table-column prop="lng" label="经度">
  </el-table-column>
  <el-table-column prop="timestamp" label="时间戳">
  </el-table-column>
  <el-table-column prop="value" label="设备发送值">
  </el-table-column>
</el-table>

```

分页:

```

<el-pagination
  @size-change="handleSizeChange"
  @current-change="handleCurrentChange"
  :current-page="pageNum"
  :page-sizes="[5, 10, 20, 30]"
  :page-size="pageSize"
  layout="total, sizes, prev, pager, next, jumper"
  :total="total">
</el-pagination>

```

数据查询：

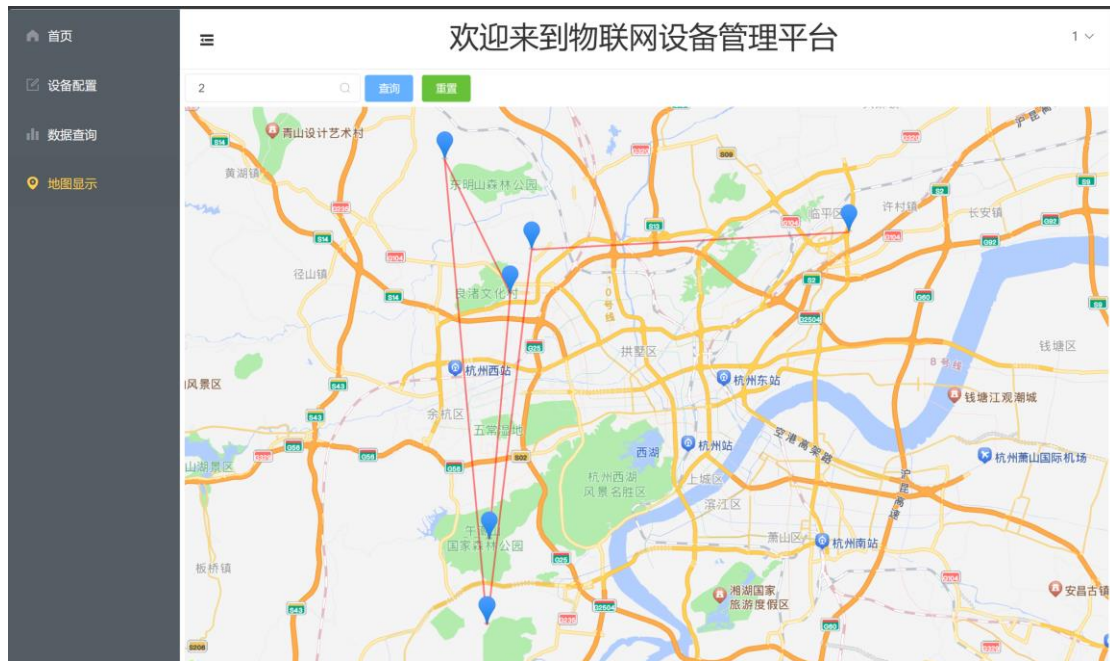
```

loadPost(){
  if(this.deviceId!=null){
    this.$axios.post( url: this.$httpUrl+ '/message/listPageC1', data: {
      pageSize:this.pageSize,
      pageNum:this.pageNum,
      param:{
        deviceId:this.deviceId,
        alert:this.alert
      }
    }).then(res=>res.data).then(res=>{
      console.log(res)
      if(res.code==1){
        this.tableData=res.data
        this.total=res.total
      }else{
        alert('获取数据失败!')
      }
    })
  }else{
    this.$message( options: {
      message: '请输入设备id!',
      type: 'warning'
    });
  }
}
}

```

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：

```
<div>
  <div style="...">
    <el-input v-model="deviceId" placeholder="请输入设备id" suffix-icon="el-icon-search" style="..."
      @keyup.enter.native="loadPost"></el-input>
    <el-button type="primary" style="..." @click="loadPost">查询</el-button>
    <el-button type="success" @click="resetParam">重置</el-button>
    <!-- <el-button type="success" @click="refresh">刷新</el-button> -->
  </div>
  <div id="container" style="..."></div>
</div>
```

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。

```

initAMap() {
  AMapLoader.load( options: {
    key: "406ecda731aeb0ac862be1773a3f47d5", // 申请好的Web端开发者Key, 首次调用 load 时必填
    version: "2.0", // 指定要加载的 JSAPI 的版本, 缺省时默认为 1.4.15
    plugins: [], // 需要使用的的插件列表, 如比例尺'AMap.Scale'等
  })

  .then((AMap) => {
    this.map = new AMap.Map("container", {
      // 设置地图容器id
      viewMode: "3D", // 是否为3D地图模式
      zoom: 11, // 初始化地图级别
      center: [116.397428, 39.90923], // 初始化地图中心点位置
    });
    if(this.tableData.length>0){
      let markerList = [];
      let path = [];
      for(let i=0;i<this.tableData.length;i++){
        let marker = new AMap.Marker({
          position: [this.tableData[i].lng, this.tableData[i].lat],
          map: this.map,
        });
        markerList.push(marker);
        path.push([this.tableData[i].lng, this.tableData[i].lat]);
      }
      this.map.add(markerList);
      var polyline = new AMap.Polyline({
        path: path,
        borderWidth: 5, // 线条宽度, 默认为 1
        strokeColor: 'red', // 线条颜色
        lineJoin: 'round' // 折线拐点连接处样式
      });

```

```

      this.map.add(polyline);
      this.map.setFitView();
    }else{
      this.$message( options: {
        message: '暂无数据!',
        type: 'warning'
      });
    }
  });

```

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 findByName(@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 =
userService.lambdaQuery().like(Menu::getMenuRight,user1.getR
oleId()).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 findByName(@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);
    }
}

```

3.2.3 MessageForm

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);
}

```

MessageFormServiceImpl:

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),messageFo
rm))return false;

            deviceService.updateAlert(messageForm.getDeviceid(),messageF
orm.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

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/bs?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT%2B8
    driver-class-name: com.mysql.jdbc.Driver
    username: root
    password: hh123456

  mqtt:
    send:
      # 服务器连接地址，如果有多个，用逗号隔开
      host: tcp://localhost:1883
      # 连接服务器默认客户端ID
      clientId: mqtt_client_id_001
      # 默认的消息推送主题，实际可在调用接口时指定
      topic: testapp
      # 用户名
      username: admin
      # 密码
      password: 123456
      # 连接超时
      timeout: 30
      # 心跳
      keepalive: 30
```

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

```
initAMap() {
  AMapLoader.load({ options: {
    key: "406ecda731aeb0ac862be1773a3f47d5", // 申请好的Web端开发者Key，首次调用 load 时必须填
    version: "2.0", // 指定要加载的 JSAPI 的版本，缺省时默认为 1.4.15
    plugins: [], // 需要使用的的插件列表，如比例尺 'AMap.Scale'等
  }})
```

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。