城市路灯远程监控系统软件设计与实现

**[摘要]**城市路灯远程监控系统软件，使用B/S模式，一个大型的单页面式应用。后台选择使用springboot，系统中提供监控路灯状态，记录故障信息功能，系统登录权限设置，操作权限设置，并提供维修人员使用的手机端App，维修人员可以接收到监控人员发送的维修信息，提醒，路灯信息统计。监控人员可以在PC端监控城市路灯的情况，发现故障可以向维修人员报告位置信息，维修人员进行修复。统计路灯故障率，维修记录，处理信息记录等信息，使用图表展示。

工作人员可以修改密码，工作状态，系统收集所有工作人员信息，设备信息，设备供应商信息，用户可以查询，获取所需信息。

后台设置了日志记录，按天统计，http请求auth认证。

**关键字：**React、Redux、react-router、node.js、axios、java、spring boot、mysql、Javascript、Webpack、npm、mybatis、logback、aop、material-ui、cookies

The software design and implementation of urban street lamp remote monitoring system

**[abstract]**The software of urban street lamp remote monitoring system，Using B/S mode, a large single page application.The backstage chooses to use the springboot, provides the monitoring street lamp condition, the record breakdown information function in the system.，The system has login authority setting, operation authority setting, and the mobile phone App used by the maintenance personnel, the maintenance personnel can receive the maintenance information sent by the monitoring staff, reminding the traffic lights, and the statistics of the street light information.The monitor can monitor the street lights on the PC side. If the fault is found, the location information can be reported to the maintenance personnel and repaired by the maintenance personnel.The system statistics the street lamp failure rate, maintenance records, processing information records and other information, using charts to show.

Staff can modify the password, work status, system collection of all staff information, equipment information, equipment supplier information, users can query, access to the required information.

The backstage has set up the log record, according to the day statistics, HTTP requests the auth authentication.

KeyWord：React、Redux、react-router、node.js、axios、java、spring boot、mysql、Javascript、Webpack、npm、mybatis、logback、aop、material-ui、cookies

**目录**

前言.......................................................................................................................... 1

1. 绪论.............................................................................................................2

1.1 城市路灯远程监控系统软件研究背景与意义.......................................2

1.2 React介绍................................................................................................2

1.2.1 React的特点.....................................................................................2

1.2.2 React的应用情况 ........................................................................... 2

1.3 Redux........................................................................................................2

1.4 React-router .............................................................................................3

1.5 axios..........................................................................................................3

1.6 universal-cookies.......................................................................................3

1.7 material-ui ................................................................................................3

1.8 gdmap .......................................................................................................3

1.9 rechart.......................................................................................................3

1.10 springboot...................................................................................................3

1.11 mybatis.......................................................................................................3

1.12 aop..............................................................................................................4

1.13 logback.......................................................................................................4

1.14 mysql..........................................................................................................4

1.15 pagehelper..................................................................................................4

1.16 定时任务...................................................................................................4

1.17 druid...........................................................................................................4

1. 系统分析 ..................................................................................................5

2.1 系统可行性分析.......................................................................................5

2.2 系统功能需求分析...................................................................................5

1. 系统设计....................................................................................................7

3.1 系统总体设计...........................................................................................7

3.1.1 系统目标设计..................................................................................7

3.1.2 开发设计思想..................................................................................7

3.1.3 系统功能模块设计..........................................................................7

3.2 数据库设计...............................................................................................9

3.2.1 数据库需求分析..............................................................................9

3.2.2 数据库概念结构设计......................................................................10

3.3 pc 端页面设计...........................................................................................12

3.3.1 Redux设计........................................................................................10

3.3.2 router设计........................................................................................15

3.3.3 axios使用示例..................................................................................20

3.3.4 pc端页面展示...................................................................................21

3.3.5 app页面展示....................................................................................28

3.4 后台设计...................................................................................................31

3.4.1 aop拦截web请求...........................................................................31

3.4.2 controller接口..................................................................................33

3.4.3 配置跨域..........................................................................................35

3.4.4全局异常处理...................................................................................35

3.4.5 mybatis配置......................................................................................36

3.4.6 mysql配置.........................................................................................38

3.4.7 logback配置......................................................................................38

3.4.8 数据分页...........................................................................................36

3.4.9 定时函数...........................................................................................38

3.4.10 分布式架构感想.............................................................................38

1. App测试......................................................................................................40

4.1 测试结果说明............................................................................................40

4.2 测试局限性分析........................................................................................40

结论.............................................................................................................................41

附录.............................................................................................................................42

附录一 主要参考资料.......................................................................................42

**前言**

鉴于路灯遍布在城市的各个角落，人工管理难度较大，建立路灯远程监控系统，对路灯损坏、维修不及时、电缆被盗等情况是十分必要的。传统的人工巡查和群众举报方式工作量大，监管效率低。相对来说，建立远程监控系统不仅可以实现全天候24h不间断的控制，还能够估计到城市的每一个角落，大大节省了人力物力成本，将被动维护转为主动抢修，提高了工作效率。

目前的城市路灯建设现状存在很多问题，建设市场混乱、建设缺乏规划、管理部门之间的沟通配合不到位，都严重影响城市路灯建设，影响城市的发展。开发一套城市路灯在线监控系统软件可以将多部门联合起来，做到发现故障及时处理，在线进行任务分配。故障路灯位置汇报，维修人员维修任务分配，及时上报维修进度。做到多部门线上联合，协同处理。这样极大的提高了工作效率。

通过软件来将路灯信息、员工信息、设备供应商信息进行汇总展示，为不同的职位提供不同的解决方案是本选题所要研究的内容。为管理人员提供pc端访问方式，为需要经常外出的维修人员提供手机端app版。方便用户使用，提高用户的工作效率。

在本选题中我使用了react来开发前台界面，spring boot来开发后台，数据库选用mysql。ORM框架使用mybatis，数据库连接池使用druid，cookie选用universal cookies，ui组件库使用material-ui，状态管理使用redux，路由使用react-router，地图插件使用高德地图，图标使用react封装的echarts——recharts。同时使用logback记录日志，Axios进行前后端的数据交互。本文主要介绍此项目实现的一些功能和技术实现方式。

1. **绪论**

**1.1 城市路灯远程监控系统软件研究背景与意义**

城市的路灯建设质量是城市发展的情况的重要指标。随着城市的快速发展，人们对夜间照明的要求也越来越高，城市路灯的建设可以有效提高人们夜间出行的安全性，减少夜间交通安全事故的发生。在完成城市的路灯建设工作后，后续的维护管理工作也十分重要。随着计算机软件技术的发展，路灯远程监控系统可以极大的方便路灯的维护管理工作。

目前的城市路灯建设现状存在很多问题，建设市场混乱、建设缺乏规划、管理部门之间的沟通配合不到位，都严重影响城市路灯建设，影响城市的发展。开发一套城市路灯在线监控系统软件可以将多部门联合起来，做到发现故障及时处理，在线进行任务分配。故障路灯位置汇报，维修人员维修任务分配，及时上报维修进度。做到多部门线上联合，协同处理。这样可以极大的提高工作效率。

编写程式路灯远程监控系统软件可以将路灯信息，员工信息，设备供应商信息进行汇总，集中展示，方便用户的查询和管理。同时使用地图来展示路灯位置更加直观，方便用户查看路灯周边建筑信息，根据路灯周围建筑或道路的信息规划维修方案。同时现在移动设备使用率越来越高，移动端的路灯管理软件适合维修人员使用，在外出维修是也可以查看维修信息，路灯故障信息，设备供应商信息。即使在外面不方便使用电脑的地方也可以对查看路灯的信息，当维修结束后也可以看到路灯的实时信息，故障是否修复。

**1.2 React介绍**

**1.2.1 React的特点**

React是Facebook公司开发的一款JS库，Facebook公司给出的开发解释是他们认为MVC无法满足他们的拓展请求，由于他们代码库和组织非常巨大，随着功能或特性的增加，系统的复杂度就呈指数级增长，因此之前的代码就变得异常脆弱，同时也难以理解和调试，特别是模型和视图之间可能存在的双向数据流动。未解决这个难题，他们设计了React—一个Javascript框架，用于构建“可预期的”和“声明式的”Web用户界面，他们已经使Facebook更快地开发Web应用。

React特点归结如下：

* + - 1. 声明式设计： React采用声明范式，可以轻松描述应用。
      2. 组件式设计： 可以将ui封装为一个组件，可以多次使用。
      3. 灵活： 独特的JSX格式，js和html可以混合编写。

**1.2.2 React的应用情况**

React在国外的应用较多，facebook、Yahoo、Reddit等，国内像阿里巴巴。React的应用范围是非常广泛的，而且随着技术的成熟，使用范围将会变得更加广泛。同时react也拥有一个庞大的社区，优秀的代码和插件不断产生。

**1.3 Redux**

Redux是javascript状态容器，提供可预测化的状态管理。存储原理：应用中所用的state都以一个对象树的形式存储在一个单一的store中，唯一改变state的办法是触发action，一个描述发生什么的对象。为了描述action如何改变state树，需要编写reducers。在本项目中我把一些在多个页面使用的数据放在了store中，方便各页面的使用。

**1.4 React-router**

React-router是一个针对React而设计的路由解决方案、可以友好的帮你解决React components到URL之间的同步映射关系。在本App中引入了React-router4。

通过hash值的变化来切换页面，项目中引入了history用来控制页面的跳转。

history.push进行跳转到指定路由，history.goBack()返回上一个页面。

**1.5 axios**

axios是一个基于promise的HTTP库，可以用在浏览器和node.js中，使用方便。

**1.6 universal-cookies**

Universal-cookies提供了set和get方法来设置或获取cookie，在本项目中用来实现记住密码的功能。

cookies.set("roadlogname", this.state.logname, {maxAge: 30\*24\*60\*60})

cookies.set("roadpwd", this.state.pwd, {maxAge: 30\*24\*60\*60})

**1.7 Material-ui**

Material-ui是一个使用react编写的ui库，界面美观，本项目使用了material-ui的各种组件。

**1.8 gdmap**

高德地图是国内优秀的地图提供商，通过注册为开发者可以使用高德地图的api服务，本项目中使用高德地图来记录路灯位置信息。后来我尝试了自定义标志，并提供地图点信息筛选功能。虽然没有在本项目中使用，但是实践是可行的，可以将故障路灯使用红色标识展示，维修路灯使用黄色展示，正常路灯使用绿色展示。点击点信息可以显示弹窗或toolstip，也可以自定义。

**1.9 rechart**

Rechart是基于echart封装的react版本，是国内百度开发的图标记录产品。本项目使用rechart来显示设备信息。

**1.10 spring boot**

Spring Boot是Spring社区较新的一个项目。该项目的目的是帮助开发者更容易的创建基于Spring的应用程序和服务，让更多人的人更快的对Spring进行入门体验，让Java开发也能够实现Ruby on Rails那样的生产效率。为Spring生态系统提供了一种固定的、约定优于配置风格的框架。

Spring Boot具有如下特性：

1.为基于Spring的开发提供更快的入门体验

2.开箱即用，没有代码生成，也无需XML配置。同时也可以修改默认值来满足特定的需求。

3.提供了一些大型项目中常见的非功能性特性，如嵌入式服务器、安全、指标，健康检测、外部配置等。

4.Spring Boot并不是不对Spring功能上的增强，而是提供了一种快速使用Spring的方式。

本项目使用了springboot进行后台的开发。

**1.11 mybatis**

 MyBatis 是一个可以自定义SQL、存储过程和高级映射的[持久层](https://baike.sogou.com/v55187093.htm" \t "https://baike.sogou.com/_blank)框架。MyBatis 摒除了大部分的JDBC代码、手工设置参数和[结果集](https://baike.sogou.com/v66020459.htm" \t "https://baike.sogou.com/_blank)重获。MyBatis 只使用简单的XML 和注解来配置和映射基本数据类型、Map 接口和POJO 到数据库记录。

MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以对配置和原生Map使用简单的 XML 或注解，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录。

本项目使用mybatis进行数据库操作的预处理。优化数据操作。

**1.12 aop**

AOP为Aspect Oriented Programming的缩写，意为：[面向切面编程](https://baike.sogou.com/v53402515.htm" \t "https://baike.sogou.com/_blank)（也叫面向方面），可以通过[预编译](https://baike.sogou.com/v8400613.htm" \t "https://baike.sogou.com/_blank)方式和运行期动态代理实现在不修改源代码的情况下给程序动态统一添加功能的一种技术。AOP实际是GoF设计模式的延续，设计模式孜孜不倦追求的是调用者和被调用者之间的解耦，AOP可以说也是这种目标的一种实现。

本项目中队请求的header查询就是通过aop实现。

**1.13 logback**

Logback是spring boot内置的日志记录插件，可以讲应用的运行状况进行记录，并支持输出到文件或数据库。本项目中通过配置logback-spring.xml文件将日志记录到log文件夹，按天保存。

**1.14 mysql**

MySQL是一个[关系型数据库管理系统](https://baike.sogou.com/v400453.htm" \t "https://baike.sogou.com/_blank)，由瑞典MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 最流行的关系型数据库管理系统，在 WEB 应用方面MySQL是最好的 RDBMS (Relational Database Management System，[关系数据库管理系统](https://baike.sogou.com/v400453.htm" \t "https://baike.sogou.com/_blank)) 应用软件之一。

MySQL是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个[大仓库](https://baike.sogou.com/v816184.htm" \t "https://baike.sogou.com/_blank)内，这样就增加了速度并提高了灵活性。

MySQL所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，它分为社区版和商业版，由于其体积小、速度快、[总体拥有成本](https://baike.sogou.com/v60488729.htm" \t "https://baike.sogou.com/_blank)低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为[网站数据库](https://baike.sogou.com/v7927370.htm" \t "https://baike.sogou.com/_blank)。

本项目使用mysql进行数据的存储。

**1.15 pagehelper**

这是我的研究内容，使用pagehelper结合mybatis实现分页效果，pagehelper时国内的一位工程师开发的，是很好用的一款数据库分页插件。

**1.16 定时任务**

在项目中需要每天定时将路灯状态的数量记录到近七日展示信息的数据库中，这里使用了@Scheduled(cron="0 0 23 \* \* ?")注解，并搭配cron表达式实现每天的23点将信息及录入数据库。

**1.17 druid**

Druid是阿里巴巴开源平台上的一个项目，整个项目由数据库连接池、插件框架和SQL解析器组成。该项目主要是为了扩展JDBC的一些限制，可以让程序员实现一些特殊的需求，比如向密钥服务请求凭证、统计SQL信息、SQL性能收集、SQL注入检查、SQL翻译等，程序员可以通过定制来实现自己需要的功能。

1. **系统分析**

**2.1 系统可行性分析**

可行性研究的目的就是用最小的代价在尽可能短的时间内确定问题是否能够解决，确定问题是否值得去解决。通过客观分析，从而判断系统目标和规模是否现实，系统完成后所能带来的收益是否大到值得开发这个系统的程度。因此，可行性研究实质上就是要进行一次已经压缩简化了的系统分析和设计的过程，也就是在较高层次上以较为抽象的方式进行的系统分析和设计的过程。从以上原则来看，使用新兴技术重新开发城市路灯远程监控系统，将多部门联合起来，做到发现故障及时处理，在线进行任务分配。故障路灯位置汇报，维修人员维修任务分配，及时上报维修进度。做到多部门线上联合，协同处理，提高路灯维护工作的工作效率。同时本设计可行性概括如下：

1. 技术上可行。本人对js有一定基础，有React开发经验，可以完成web页面的编写，结合Redux和React-router实现数据存储和页面跳转。对springboot框架有使用经历，可以完成后台代码编写，通过axios连接前后端，处理请求，Mysql存储用户信息。
2. 经济可行性。React、Redux、React-router、springboot等需要使用的工具都是开源的，无需支付任何费用，只需支付数据库方面和人工费用，成本比较低，在软件完成后如果进一步改进后投入市场，可以方便用户管理路灯维护工作，操作简单。用户无需进行复杂的操作便可完成处理过程。无需进行专业的使用教程便可自由的操作本软件。

所以从以上几个方面的分析看，开发本软件是完全可行的。

**2.2 系统功能需求分析**

**城市路灯远程监控系统软件实现的功能：**

1. 用户登录
2. 记住密码
3. 查看设备实时监控
4. 查看设备信息，位置
5. 处理故障设备
6. 选择在职维修人员分配维修任务
7. 查看维修人员信息，查询
8. 查看处理记录，查询
9. 查看维修记录，提醒维修人员尽快进行维修
10. 查看设备供应商信息
11. 记住密码
12. 修改工作状态
13. 退出登录
14. 维修人员接受任务和提醒

**研究内容：**

1. React复用组件的封装过程以及使用方法
2. Redux store使用，状态管理和数据更新过程
3. React-router路由跳转设置，动态加载原理
4. Springboot框架的使用，mybatis配置
5. Pagerhelper的使用方法
6. 日志记录配置
7. Aop实现请求拦截处理方法
8. 高德地图使用方法
9. Recharts使用方法和数据展示
10. Axios的使用，设置请求头信息方法
11. Jsx使用，语法特征
12. Es6和7的一些高级语法尝试使用
13. 日志记录方法，如何实现日志存储，滚动记录

13.消息中间件的探索

1. **系统设计**

**3.1 系统总体分析**

**3.1.1 系统目标设计**

经过系统设计与实验，最终产生城市路灯远程监控系统软件设计与实现。

**3.1.2 开发设计思想**

使用react开发城市路灯远程监控系统软件，使用B/S模式，开发一个大型的单页面式应用。后台选择使用springboot，由于硬件方面难以调试，所以我考虑在软件中操作是发送模拟指令，不涉及到真实路灯控制，假设路灯会定时将运行状态反馈给后台，更新数据信息，系统中提供监控路灯状态，显示路灯的位置信息，生产厂商和维修信息，记录故障信息、故障时间功能，系统登录权限设置，炫耀用户名和密码，操作权限设置，只有管理人员可以查看展示页面并报修路灯信息，软件提供维修人员使用的手机端App，维修人员可以接收到监控人员发送的维修信息，提醒。监控人员可以在PC端监控城市路灯的情况，发现故障可以向维修人员报告位置信息，维修人员进行修复。统计路灯故障率等信息，使用图表展示。

软件提供员工信息的记录，包括员工的代码，姓名，联系方式，在职状态，备注，职位等信息，方便用户查看，当需要联系某个员工时，可以方便的查看到该员工的联系方式，同样软件记录了设备供应商的信息，包括供应商的名称，地址，联系方式等，当遇到维修人员解决不了的问题时，可以快速的找到设备供应商的联系方式地址，及时与对方联系，研究维修方式。软件的操作记录都会被记录下来，并展示出来，当需要查找某条操作记录时可以通过关键字查找，快速准确的定位到所需数据。图表使用折线图展示近七日的路灯状态信息，可以通过走势来看到路灯最近的趋势，有助于管理人员即使对趋势进行评估预测，辅助管理人员对人物的安排。同时使用饼状图展示故障路灯、维修路灯和正常路灯的百分比，简洁直观，当出现严重事故时，比如大面积停电，饼状图更加明确的展示事故的具体情况。帮助管理人员安排维修任务，及时抽派人手。故障路灯信息是用表格展示，重点显示路灯的位置，故障时间等信息，用户可以点击该行进入路灯信息详情页，显示路灯的具体信息，生产厂商信息等，可以通过这个信息查找到生产厂商的联系方式，地址，方便于厂商之间的联系。同时也可以对维修任务进行指派，安排给指定的维修人员，可以选择维修人员，但是只展示在职的维修人员，休假状态的不能指派任务。同时安排维修的操作会被系统记录，存放在数据库中，任务处理记录页面可以展示。维修任务信息表也可以显示维修路灯的信息，维修人员，安排任务的人员，处理时间等信息，点击该行也会进入详情页。在页面中可以提醒维修人员尽快维修，维修人员的app端会接收到提醒消息。维修人员可以在app端登录，查看自己的任务信息，任务卡片上会显示地址，路灯厂商等信息，维修人员也可以直接和厂商联系路灯维修问题。App端方便维修人员外出维修的工作。维修人员同样可以在app端修改登录密码，修改工作状态。为记录每天的路灯信息，用于近七日路等信息展示。后台设置了定时任务，每天的23点会检查路灯的状态，将数据记录到数据库。同时后台对用户的操作进行记录并保存入日志，当出现系统问题是，方便专业人员进行系统排查，找出问题所在，及时进行处理，解决问题。后台对请求进行了验证，防止恶意的sql注入攻击。引入了数据库连接池druid，减轻数据库的访问压力。选用了mybatis orm框架，对数据库查询语句进行优化，提高访问速度。数据库设计简化字段，防止重复的数据。通过外键将多个表连结起来，节省数据库空间。

为了满足不同用户的需求，界面尽量简单，使用且方便，对用户友好。功能通过侧边栏进行切换，首部标题栏显示用户的信息，用户可以点击头像或登录名查看个人的详细信息，修改密码或状态。在每个功能栏上面都有该栏目的名称，直观的搜索按钮，搜索框默认隐藏，点击搜索按钮时会以一段动画划入屏幕。用户可以选择记住密码，下次可以直接登录，不再需要输入密码。

**3.1.3 系统功能模块设计**

页面关系及预计实现功能分布

1. **pc端进入系统**

修改状态

修改密码

登录

设备供应商信息

维修人员信息

管理人员信息

处理信息记录

维修任务记录

故障路灯处理

路灯监控信息

实时监控信息

信息实时反应那个

信息记录

查询

提醒维修

维修记录

分配维修

故障记录

位置信息

故障数量

维修数量

1. **App端进入系统**

修改状态

修改密码

登录

任务

设置

用户

设备

提醒

任务

**3.2 数据库设计**

**3.2.1 数据库需求分析**

根据要实现的功能要求需要

1. 管理人员登录表——保存用户的账号、密码、员工代码
2. 路灯信息表——路灯编号、位置、经纬度信息、生产厂商代码、设备状态、故障时间、维修职员代码
3. 7日故障数量统计
4. 人员信息表——代码、姓名、联系方式、状态
5. 设备供应商信息表——代码、名称、地址、联系方式
6. 处理记录表——路灯信息、处理人员信息、处理时间
7. 维修人员登录表——账号、密码、员工代码
8. 维修任务表——路灯信息

**3.2.2 数据库概念结构设计**

得到上面的数据项和数据结构以后，就可以设计出能够满足用户需求的各种实体结构，以及他们之间的关系，为以后的逻辑结构打下基础。

1. 管理人员登录表

登录名、密码、员工代码



1. 路灯信息表

路灯代码、地址、状态、生产厂商代码、故障时间、维修人员代码、经纬度



1. 7日故障数量统计

故障数、维修数、正常数、日期时间



1. 人员信息表

员工代码、姓名、电话、状态、职位、备注



1. 设备供应商信息表

厂商代码、名称、地址、联系方式



1. 处理记录表

路灯代码、维修人员代码、故障时间、处理时间、操作类型



1. 维修人员登录表

用户名、密码、员工代码



1. 维修任务表

路灯代码、维修人员代码、消息类型、时间



**3.3 pc端页面设计**

**创建项目:**

npx install -g create-react-app manage

cd manage

npm install redux

npm install react-router

npm install react-dom

npm install material-ui

npm install universal-cookie

npm install antd

npm install axios

npm install history

现在已经导入了项目需要的依赖，npm start可以运行项目

**3.3.1 Redux设计**

当用户登录时需要保存用户信息，退出登录事件处理，故障详情页临时数据，维修详情页临时数据。

在src目录下新建actions文件夹，其中新建index.js文件

具体实现如下：

export const SET\_LOGIN = 'SET\_LOGIN';

export const REMOVE\_LOGIN = 'REMOVE\_LOGIN';

export const SET\_ERROR = 'SET\_ERROR';

export const REMOVE\_ERROR = 'REMOVE\_ERROR';

export const SET\_FIX = 'SET\_FIX';

export const REMOVE\_FIX = 'REMOVE\_FIX';

export function login(login) {

return {

type: SET\_LOGIN,

login

}

};

export function loginout() {

return {

type: REMOVE\_LOGIN

}

}

export function error(error) {

return {

type: SET\_ERROR,

error

}

};

export function removeerror() {

return {

type: REMOVE\_ERROR

}

}

export function fix(fix) {

return {

type: SET\_FIX,

fix

}

};

export function removefix() {

return {

type: REMOVE\_FIX

}

}

这些函数分别可以处理上述需求功能，记录登录信息用于展示，退出登录清除store中的用户登陆信息，记录故障路灯信息，维修任务信息，接下来设置reducers来对actions中的函数进行处理。通过export 可以将函数导出，在项目其他地方可以通过import来使用。

其实此处的故障路灯信息和维修信息可以通过动态路由来实现，在路由后面添加路灯的id，在详情信息展示页面中可以取出路由上的id来继续请求后台获取数据，可以说是两种不同的实现方式。

在src目录下新建reducers文件夹，其中新建index.js文件

具体实现：

import { combineReducers } from 'redux';

import { SET\_LOGIN, REMOVE\_LOGIN, SET\_ERROR, REMOVE\_ERROR, SET\_FIX, REMOVE\_FIX } from '../actions';

function login(state = {}, action) {

switch(action.type) {

case SET\_LOGIN:

return action.login;

case REMOVE\_LOGIN:

return {};

default:

return state;

}

}

function error(state = {}, action) {

switch(action.type) {

case SET\_ERROR:

return action.error;

case REMOVE\_ERROR:

return {};

default:

return state;

}

}

function fix(state = {}, action) {

switch(action.type) {

case SET\_FIX:

return action.fix;

case REMOVE\_FIX:

return {};

default:

return state;

}

}

const road = combineReducers({

login,

error,

fix

});

export default road;

通过函数来实现对store中信息的更改，防止直接对store的数据进行操作，当有多个store要存储的数据时，可以使用combineReducers进行组合。

这样在其他页面就可以使用store中的数据了，也可以进行更改。

使用数据：

function select(state) {

return {

login: state.login

}

}

export default connect(select)(user)

通过select函数可以对数据进行筛选，connect将数据绑定到页面中

更改数据：

this.props.dispatch(函数名(payload))

更改括号内的函数名和参数即可分发事件来更改store中的值。

index.js加入store

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import { createStore } from 'redux';

import { Provider } from 'react-redux';

import registerServiceWorker from './registerServiceWorker';

import road from './reducers';

import R from './router';

let store = createStore(road);

ReactDOM.render(

<Provider store={store}>

<R />

</Provider>, document.getElementById('root'));

registerServiceWorker();

通过createStore来创建store，并绑定到根元素上，子元素都可以访问到这些数据，并通过指定的函数来修改数据，只有store中的数据变更时才会触发页面的重新渲染。

**3.3.2 router设计**

history/index.js

/\*导入history包，路由跳转时使用\*/

import createHistory from 'history/createBrowserHistory'

export default createHistory()

Router/index.js

import React from 'react';

import {

Route,

Router,

Switch

} from 'react-router';

import axios from 'axios';

import history from '../history';

import M from '../page/menu/menu';

import L from '../page/login/login';

axios.defaults.headers.common['auth'] = 'road2018';

axios.defaults.timeout = 3000;

const router = () => (

<Router history={history}>

<Switch>

<Route exact path="/" component={L} />

<Route exact path="/control" component={M} />

<Route exact path="/chart" component={M} />

<Route exact path="/errortable" component={M} />

<Route exact path="/fixtable" component={M} />

<Route exact path="/record" component={M} />

<Route exact path="/managep" component={M} />

<Route exact path="/maintainp" component={M} />

<Route exact path="/proserver" component={M} />

<Route exact path="/errormsg" component={M} />

<Route exact path="/fixmsg" component={M} />

<Route exact path="/user" component={M} />

<Route exact path="/changepwd" component={M} />

</Switch>

</Router>

);

export default router;

引入history包来设置router的页面依据，每个router代表一个路由，component代表展示的组件，公共组件可以放在根目录上，这样子页面也会渲染这些公共组件，exact表示精确匹配，不加这个条件时可以设置动态路由。比如：

<Route path="/dtrouter" component={M} />

不仅会匹配/dtrouter，也会匹配/dtrouter/11111

这样也可以设置公共的组件。

上面还配置了axios的通用设置

axios.defaults.headers.common['auth'] = 'road2018';

设置默认请求头包含auth=road2018信息

axios.defaults.timeout = 3000;

设置默认超时时间3秒

其实在项目变得复杂后可以将axios包装成一个函数，通过引入参数来进行请求数据，可以减少通用请求的代码量。

使用上面的配置，可以实现页面的跳转，通过

history.push()或history.goBack()来进行页面间的跳转。

通用菜单栏设置

import React, { Component } from 'react';

import { connect } from 'react-redux'

import {

Route,

Router,

Switch

} from 'react-router';

import history from '../../history';

import MuiThemeProvider from 'material-ui/styles/MuiThemeProvider';

import AppBar from 'material-ui/AppBar';

import Drawer from 'material-ui/Drawer';

import MenuItem from 'material-ui/MenuItem';

import RemoveRedEye from 'material-ui/svg-icons/image/remove-red-eye';

import IconButton from 'material-ui/IconButton';

import IconMenu from 'material-ui/IconMenu';

import MoreVertIcon from 'material-ui/svg-icons/navigation/more-vert';

import PersonAdd from 'material-ui/svg-icons/action/account-circle';

import ContentLink from 'material-ui/svg-icons/action/settings';

import Divider from 'material-ui/Divider';

import ContentCopy from 'material-ui/svg-icons/action/help';

import Delete from 'material-ui/svg-icons/action/exit-to-app';

import Airplay from 'material-ui/svg-icons/av/airplay';

import Chart from 'material-ui/svg-icons/editor/insert-chart';

import Error from 'material-ui/svg-icons/alert/error';

import Build from 'material-ui/svg-icons/action/build';

import Content from 'material-ui/svg-icons/content/content-paste';

import Work from 'material-ui/svg-icons/action/work';

import Perm from 'material-ui/svg-icons/action/perm-identity';

import Account from 'material-ui/svg-icons/action/supervisor-account';

import Control from '../control/control';

import Map from '../map/map';

import ETable from '../maintain/errortable';

import RTable from '../record/record';

import MP from '../managep/managep';

import MAP from '../maintainp/maintainp';

import PS from '../proserver/proserver';

import ERRM from '../maintain/errmessage';

import FT from '../fix/fixtable';

import FIXM from '../fix/fixmessage';

import U from '../user/user';

import CP from '../user/changepwd';

import { loginout, removeerror, removefix } from '../../actions';

class M extends Component {

state = {

open: false

}

componentDidMount() {

if(this.props.login.logname === undefined) {

history.push('/');

}

}

handleToggle = () => this.setState({open: !this.state.open});

handleClose = () => this.setState({open: false});

loginout = () => {

this.props.dispatch(removeerror());

this.props.dispatch(removefix());

this.props.dispatch(loginout());

history.push('/');

}

render() {

const Logged = (props) => (

<IconMenu

{...props}

iconButtonElement={

<IconButton><MoreVertIcon /></IconButton>

}

targetOrigin={{horizontal: 'right', vertical: 'top'}}

anchorOrigin={{horizontal: 'right', vertical: 'top'}}

>

<MenuItem primaryText={"用户" + this.props.login.logname} leftIcon={<PersonAdd />} />

<MenuItem primaryText="详细信息" leftIcon={<RemoveRedEye />} onClick={() => history.push('/user')}/>

<MenuItem primaryText="修改密码" leftIcon={<ContentLink />} onClick={() => history.push('/changepwd')}/>

<Divider />

<MenuItem primaryText="操作帮助" leftIcon={<ContentCopy />} />

<Divider />

<MenuItem primaryText="退出登录" leftIcon={<Delete />} onClick={this.loginout}/>

</IconMenu>

);

Logged.muiName = 'IconMenu';

return (

<Router history={history}>

<MuiThemeProvider>

<div>

<AppBar

title="远程路灯监控管理系统"

style={{width: '100%'}}

onLeftIconButtonClick={() => this.setState({open: true})}

iconElementRight={<Logged />}

/>

<Drawer

docked={false}

width={200}

open={this.state.open}

onRequestChange={(open) => this.setState({open})}

>

<MenuItem onClick={() => {history.push('/control');this.setState({open:false})}} leftIcon={<Airplay />}>实时监控信息</MenuItem>

<MenuItem onClick={() => {history.push('/chart');this.setState({open:false})}} leftIcon={<Chart />}>路灯监控信息</MenuItem>

<MenuItem onClick={() => {history.push('/errortable');this.setState({open:false})}} leftIcon={<Error />}>故障路灯处理</MenuItem>

<MenuItem onClick={() => {history.push('/fixtable');this.setState({open:false})}} leftIcon={<Build />}>维修任务记录</MenuItem>

<MenuItem onClick={() => {history.push('/record');this.setState({open:false})}} leftIcon={<Content />}>处理信息记录</MenuItem>

<MenuItem onClick={() => {history.push('/managep');this.setState({open:false})}} leftIcon={<Work />}>管理人员信息</MenuItem>

<MenuItem onClick={() => {history.push('/maintainp');this.setState({open:false})}} leftIcon={<Perm />}>维修人员信息</MenuItem>

<MenuItem onClick={() => {history.push('/proserver');this.setState({open:false})}} leftIcon={<Account />}>设备供应商信息</MenuItem>

</Drawer>

<Switch>

<Route exact path="/control" component={Control} />

<Route exact path="/chart" component={Map} />

<Route exact path="/errortable" component={ETable} />

<Route exact path="/fixtable" component={FT} />

<Route exact path="/record" component={RTable} />

<Route exact path="/managep" component={MP} />

<Route exact path="/maintainp" component={MAP} />

<Route exact path="/proserver" component={PS} />

<Route exact path="/errormsg" component={ERRM} />

<Route exact path="/fixmsg" component={FIXM} />

<Route exact path="/user" component={U} />

<Route exact path="/changepwd" component={CP} />

</Switch>

</div>

</MuiThemeProvider>

</Router>

);

}

}

function select(state) {

return {

login: state.login

}

}

export default connect(select)(M);

这里也引入了路由，可以实现子组件的嵌套，通过不同的路由来改变子组件的填充。不会影响到父组件的状态。

最后的

function select(state) {

return {

login: state.login

}

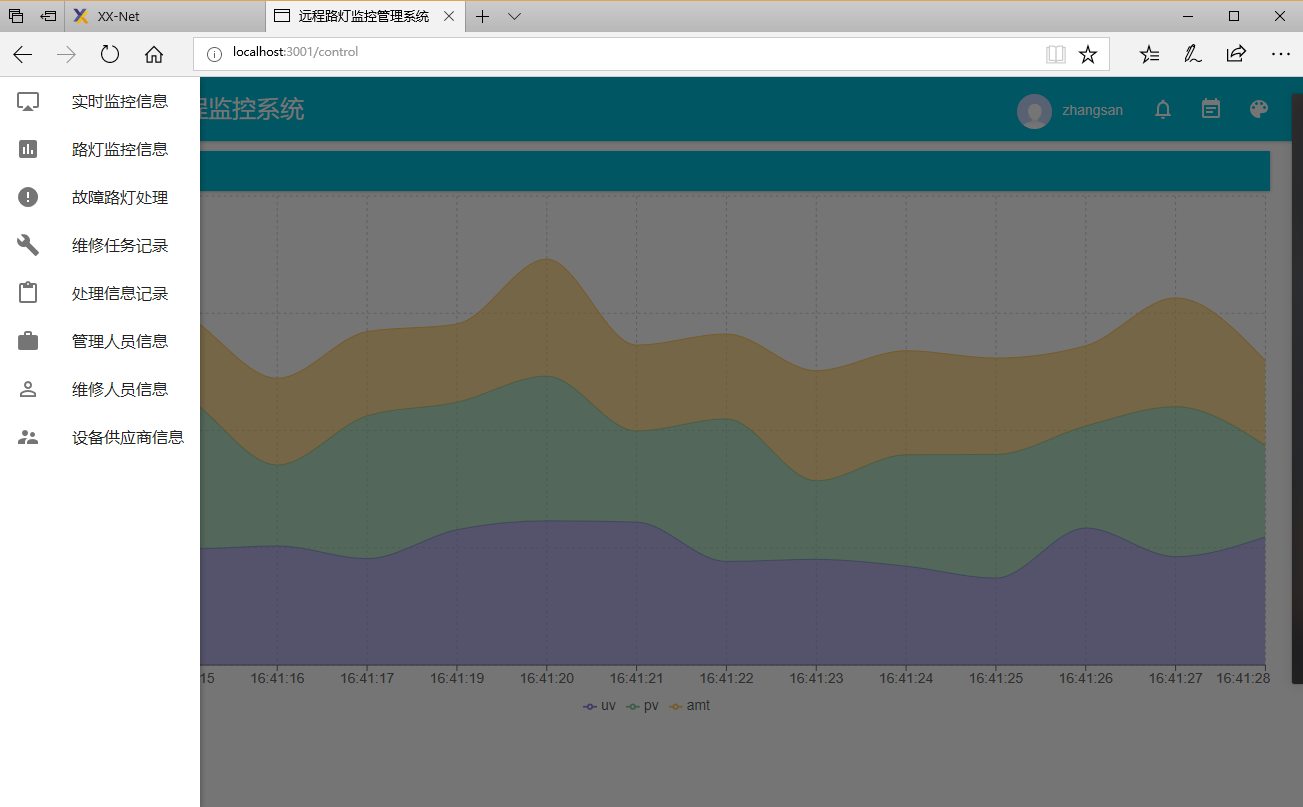
}

可以对state的数据进行引用，可以简化变量名，也可以更改为自定义的名称。

Connect函数绑定页面和store中的数据。

这样菜单栏可以控制页面的跳转，而且也是一个公用组件。通过在内部嵌入其他组件，实现切换页面。页面更新时，只是数据改变的地方更新，不会影响到公共组件。这样可以减轻前端页面刷新的压力，不会因为只改变了一小部分而需要刷新整个页面，同时也对用户的操作更友好。侧边栏我选用的是隐藏式的，可以不占据展示页面位置，让展示页空间更大，展示的数据更清晰。首页的面积图是可以随着时间变化的，可以呈现动态的效果，这里我只是用了模拟数据，实际中可以没隔一段时间重新向后台请求最新的一定数量的数据，通过setInterval函数可以实现订时请求数据。

实现效果：



**3.3.3 axios使用示例**

import axios from 'axios';

import { message } from 'antd';

axios.post('http://localhost:1111/data')

.then((response) => {

if(response.data.success) {

let da = response.data.countrecord;

let d = [];

for(let i = 0; i < da.length; i++) {

let temp = {};

temp.name = da[i].time;

temp.故障 = da[i].error;

temp.维修 = da[i].fix;

d.unshift(temp);

}

let today = {};

today.name = moment().format('YYYY-MM-DD').toString();

today.故障 = response.data.errorcount;

today.维修 = response.data.fixcount;

d.push(today);

this.setState({

data: d

})

} else {

message.error('获取数据失败！');

}

})

.catch(function (error) {

console.log(error);

message.error('网络连接失败！');

});

这样可以请求到后台，并拿到需要数据。在网络不好时，会有网络连接失败的弹窗提示，在数据请求失败时，也会有未获取到数据的提示。拿到数据通过this.setState函数将数据保存在页面的state中，供渲染时使用，axios也可以携带请求body，设置cancel处理函数。通过在url后面添加一个formdata，formdata中以键值对的方式来记录表单数据，后台就可以拿到请求的body中的参数。

axios.defaults.headers.common['auth'] = 'road2018';

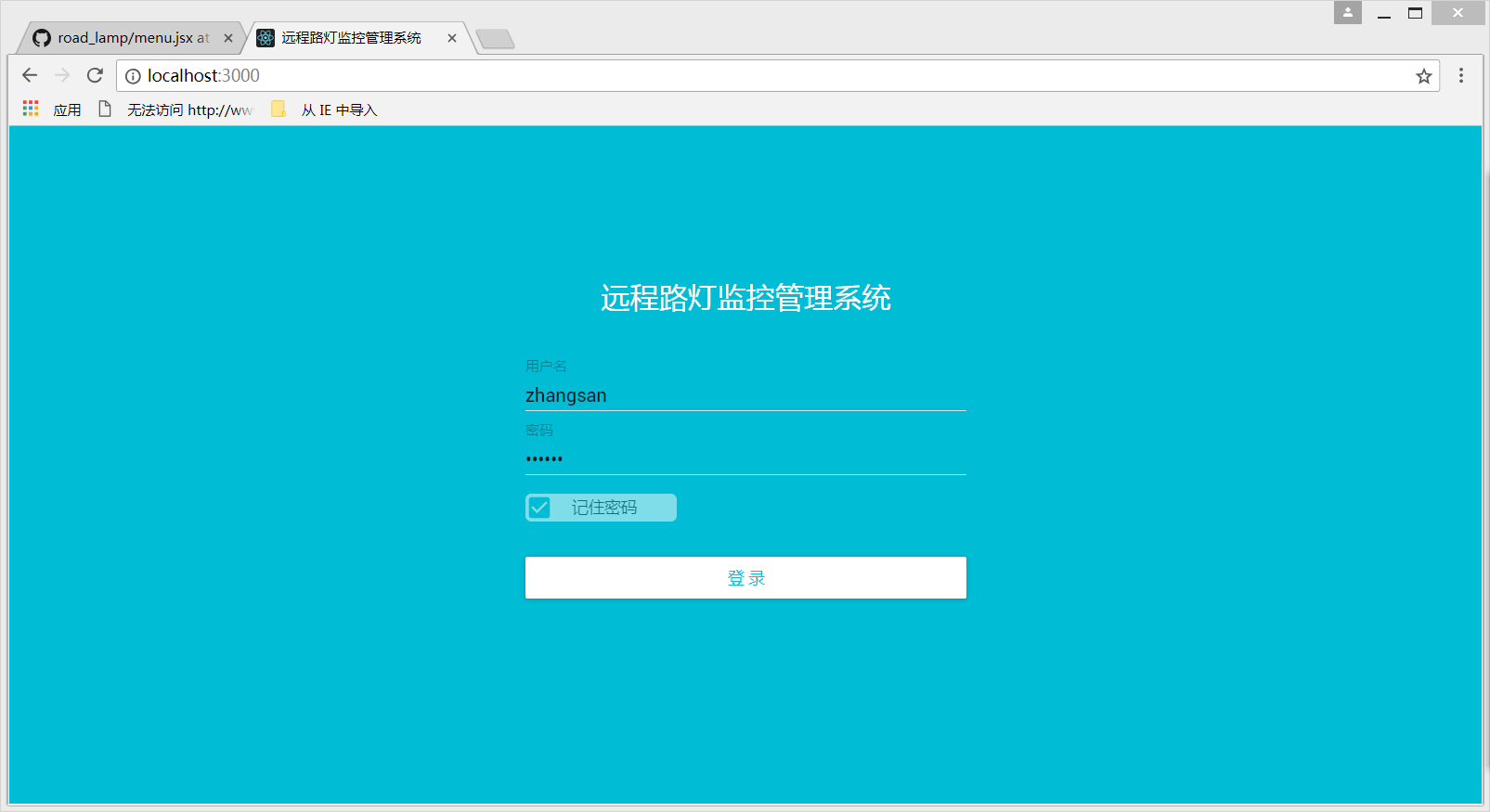
axios.defaults.timeout = 3000;

设置请求头信息和默认超时时间。防止模仿请求进行网络攻击。

**3.3.4 pc端页面展示**

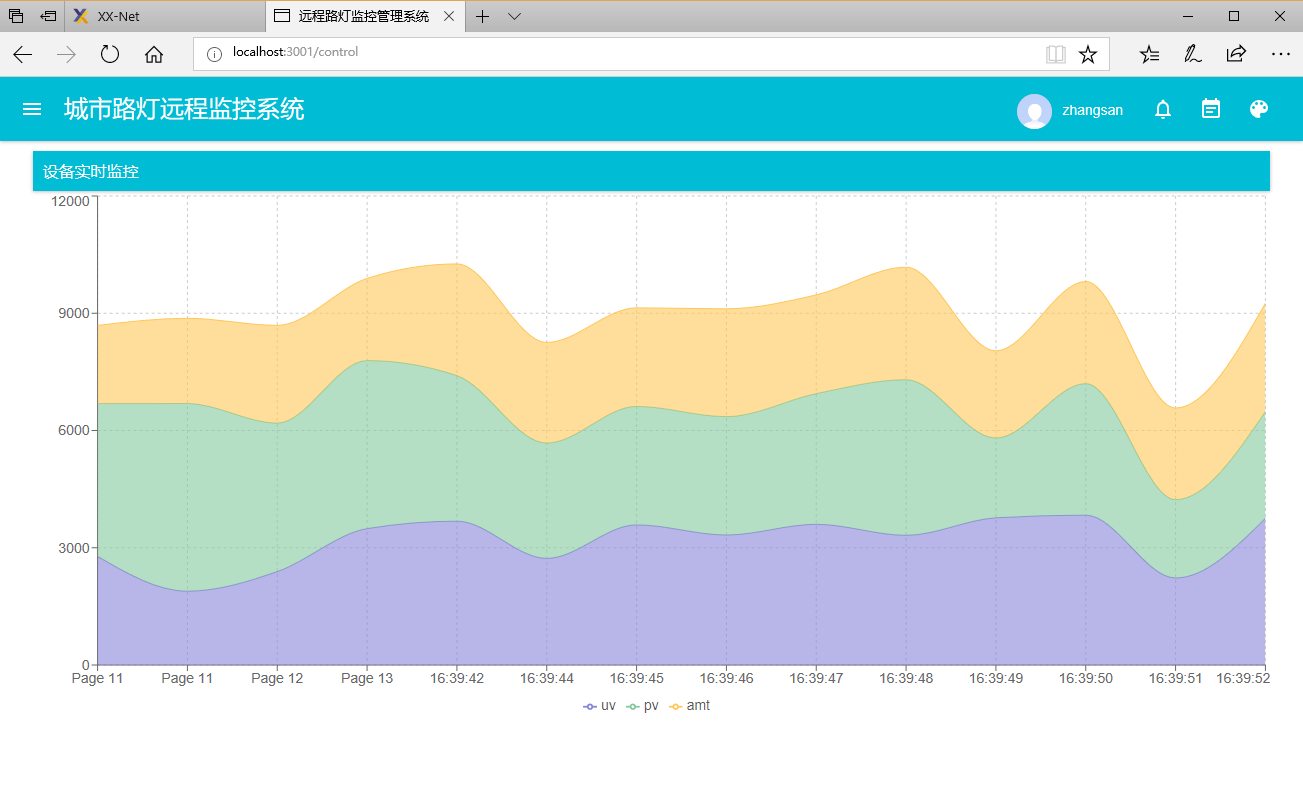
1.登录页面

登陆页面是程序的入口，用户输入正确的账号和密码才可以进入系统，不能通过更改url进入系统的功能页面。页面提供记住密码功能，用户可以选择是否记住密码，通过浏览器端的cookie保存。为安全起见cookie可以选择使用md5加密，通过设置salt，md5（data， salt）可以将cookie进行加密，可以防止非法获取cookie。



1. 实时设备信息页面

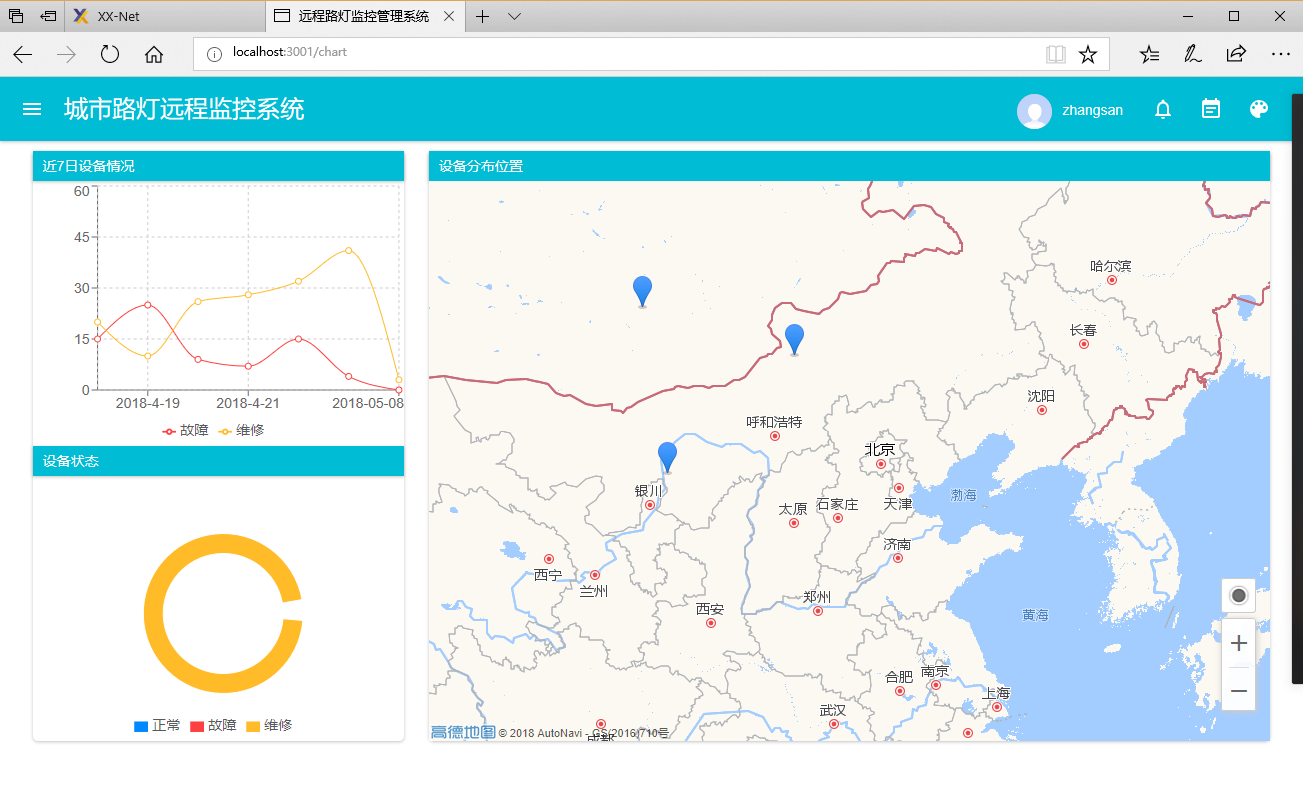
本页面进行设备实时信息的一些反应，这部分需要与硬件相配合，由于开发原因，我保留了接口，数据采用了模拟数据。



1. 设备统计信息页面

此页面展示路灯7天内的故障数量和维修数量，所占百分比，同时也使用地图标志来显示设备的位置信息，方便用户查看设备所在位置。

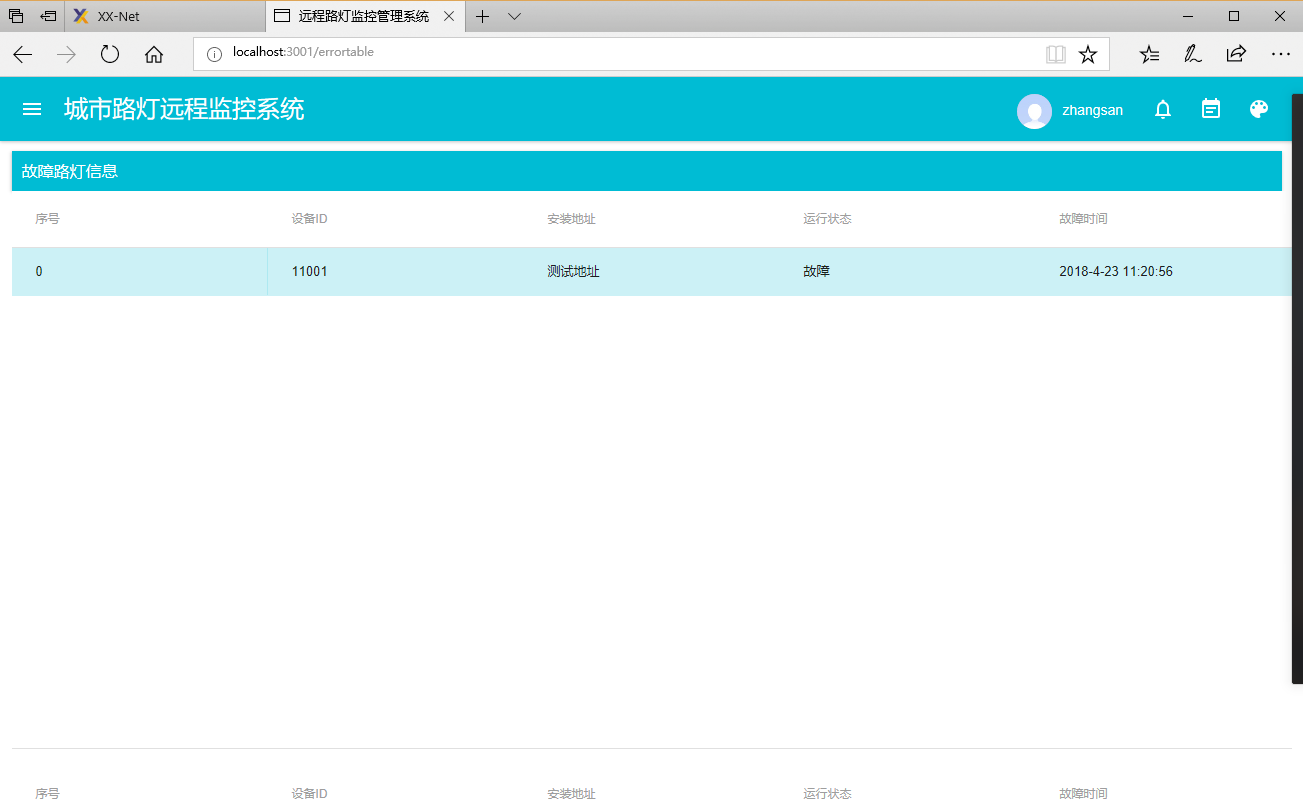
后来我尝试了使用自定义的标志来展示位置信息，可以实现。但由于撰写次论文时项目已经成型，只是实验性的尝试，在后面我会详细介绍使用方法。

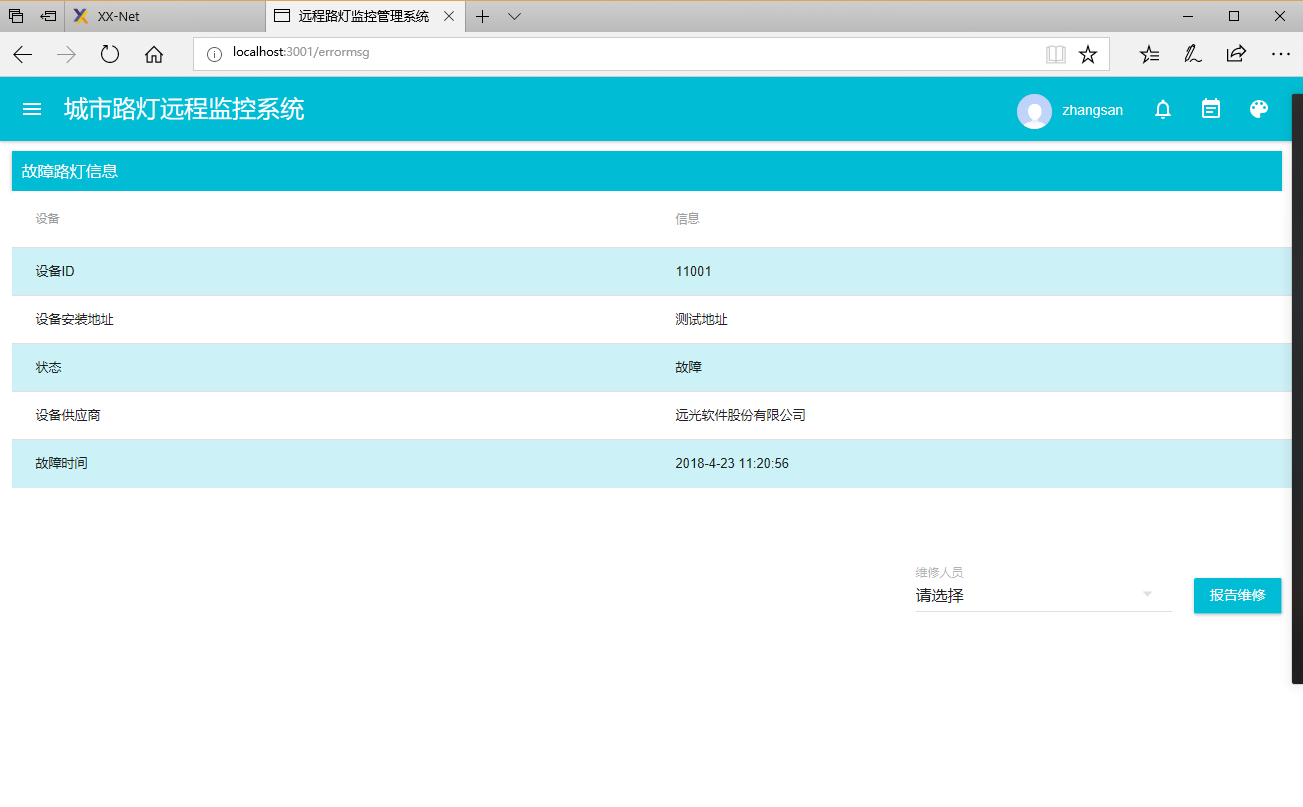


1. 故障路灯信息及处理

管理人员可以在页面内查看故障路灯的详细信息，并安排维修人员进行处理。可以对维修人员进行指派，但是只能选择在职的维修人员，休假的维修人员不能指派任务。

页面中使用表格的形式展示数据信息，以后可以根据需求进行调整，增加行和列都可以。





1. 维修信息记录

这里对正在维修的路灯信息进行统计，可以显示路灯和指派维修人员的信息，也可以对维修人员进行任务提醒。提醒之后，维修人员会在app端收到提醒信息。

昨晚此功能后我翻阅了一些资料查看如何实现通知功能，现有的方式可以使用rabbitmq或kafka消息中间件，都可以实现发布订阅的功能。使用方法有些不同。rabbitmq和kafka都已经集合到了springboot中，通过在pom中引入依赖便可以使用。下面是对rabbitmq的简单使用：

@Configuration

public class RabbitConfig {

@Bean

public Queue helloQueue() {

return new Queue("hello");

}

@Bean

public Queue worldQueue() {

return new Queue("world");

}

}

这是设置了配置类，将两个队列封装为bean。

@Component

public class Sender {

@Autowired

AmqpTemplate amqpTemplate;

public void send() {

String context = "hello" + new Date();

System.out.println("Sender :" + context);

this.amqpTemplate.convertAndSend("hello", context);

}

}

容器类，自动装配了AmqpTemplate，这是rabbitmq为springboot提供的封装pojo，通过调用converAndSend方法来发布消息。

@Component

@RabbitListener(queues = "hello")

public class Receiver {

@RabbitHandler

public void process(String hello) {

System.out.println("Receiver :" + hello);

}

}

监听hello队列的容器类，通过两个注解来对hello队列消息进行处理。

spring.application.name=rabbitmq-hello

spring.rabbitmq.host=localhost

spring.rabbitmq.port=5672

spring.rabbitmq.username=guest

spring.rabbitmq.password=guest

设置rabbitmq的配置信息

Kafka使用方法稍有不同，同样要在pom中引入kafka依赖。

@Component

@Slf4j

public class KafkaSender {

@Autowired

KafkaTemplate<String, String> kafkaTemplate;

Gson gson = new GsonBuilder().create();

/\*\*

\* 发送消息方法

\*/

public void send() {

Message message = new Message();

message.setId(System.currentTimeMillis());

message.setMsg(UUID.randomUUID().toString());

message.setSendTime(new Date());

log.info(".........");

kafkaTemplate.send("zhisheng", gson.toJson(message));

}

}

容器类，自动装配KafkaTemplate，通过send方法发布消息，这里我还尝试了发布json格式的对象，通过gson来实现，国内的fastjson也可以实现，本例中我使用了lombok的@slf4j注解，这是对日志记录的简化。

@Component

@Slf4j

public class KafkaReceiver {

@KafkaListener(topics = {"zhisheng"})

public void listen(ConsumerRecord<?, ?> record) {

Optional<?> kafkaMessage = Optional.ofNullable(record.value());

if(kafkaMessage.isPresent()) {

Object message = kafkaMessage.get();

log.info("...........................");

}

}

}

监听topics消息。

#指定kafka代理地址，可以多个

spring.kafka.bootstrap-servers=192.168.153.135:9092

spring.kafka.producer.retries=0

#每次批量发送消息的数量

spring.kafka.producer.batch-size=16384

spring.kafka.producer.buffer-memory=33554432

#指定消息key和消息体的编解码方式

spring.kafka.producer.key-serializer=org.apache.common.serialization.StringSerializer

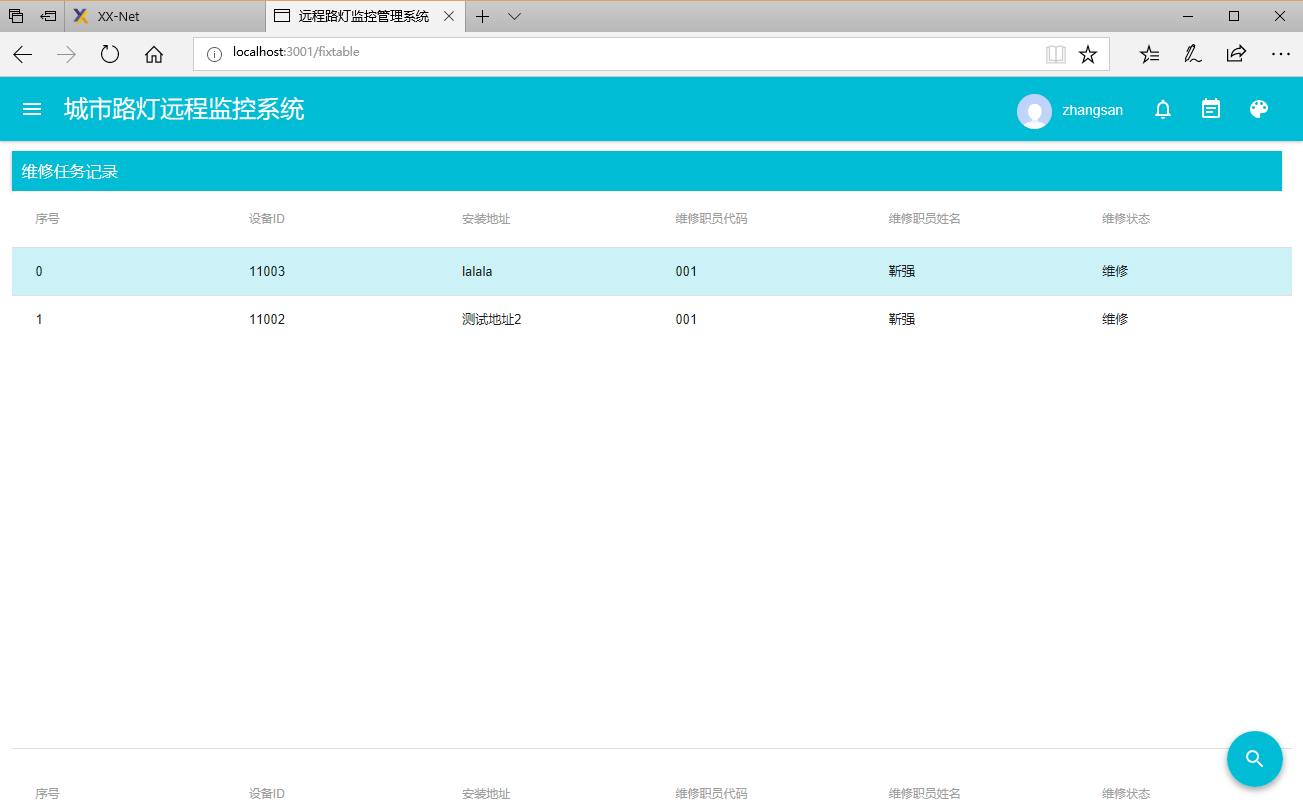
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

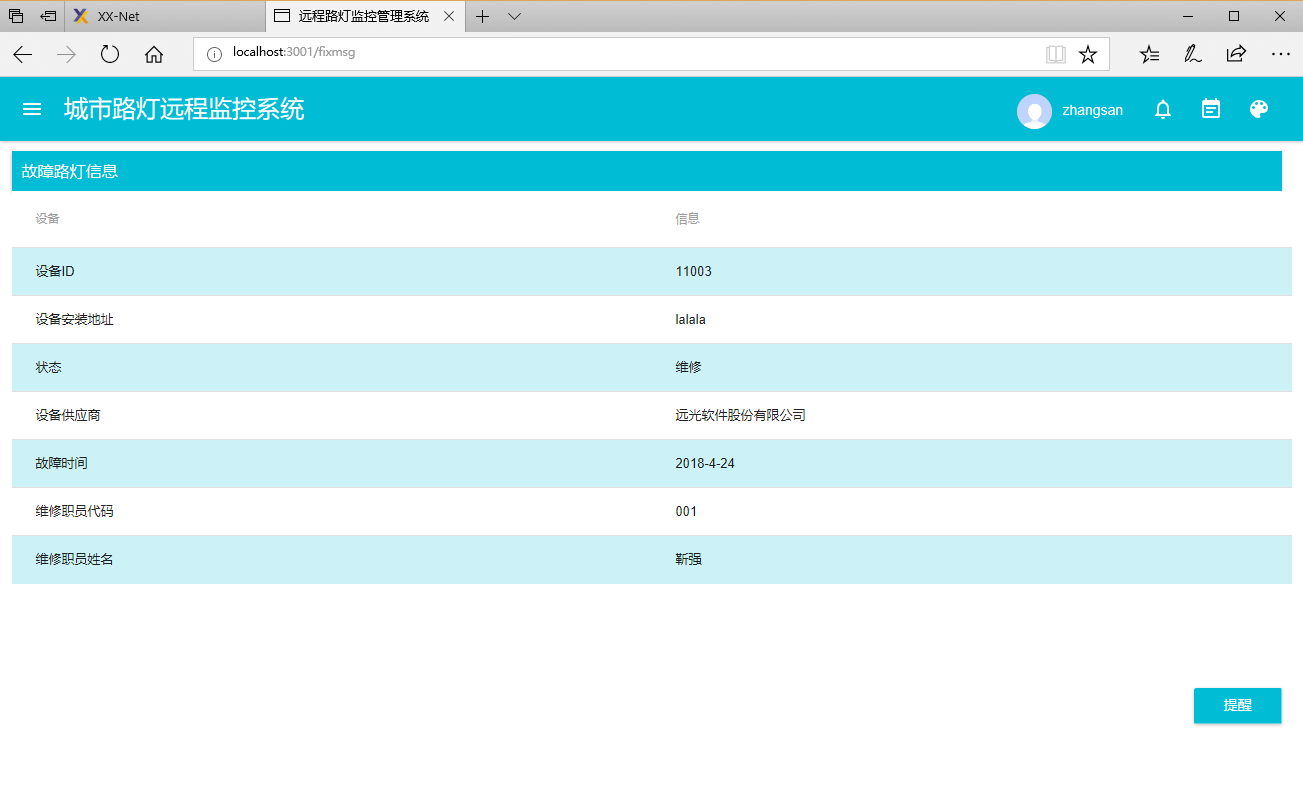
#指定默认消费者group id

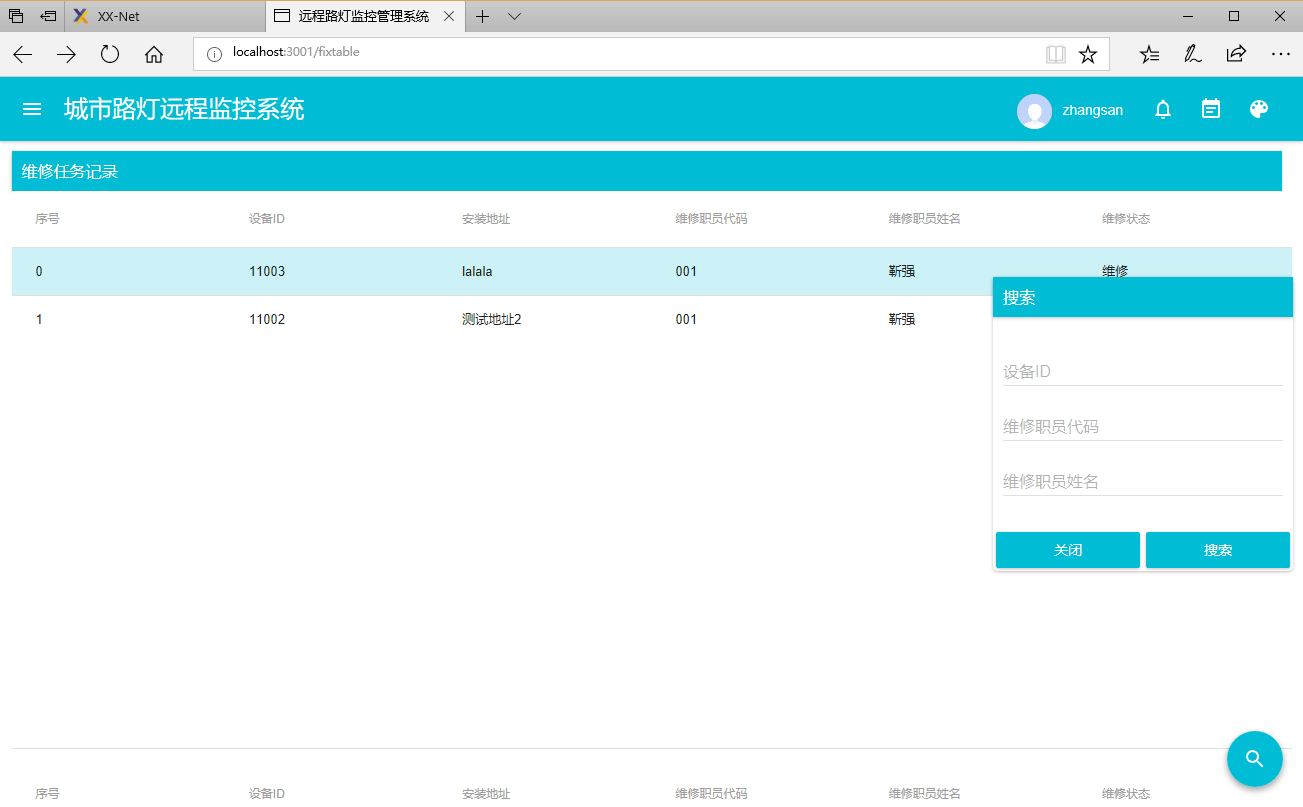
spring.kafka.consumer.group-id=test-consumer-group

一些kafka的配置

可以看出使用方法还是很相似的。

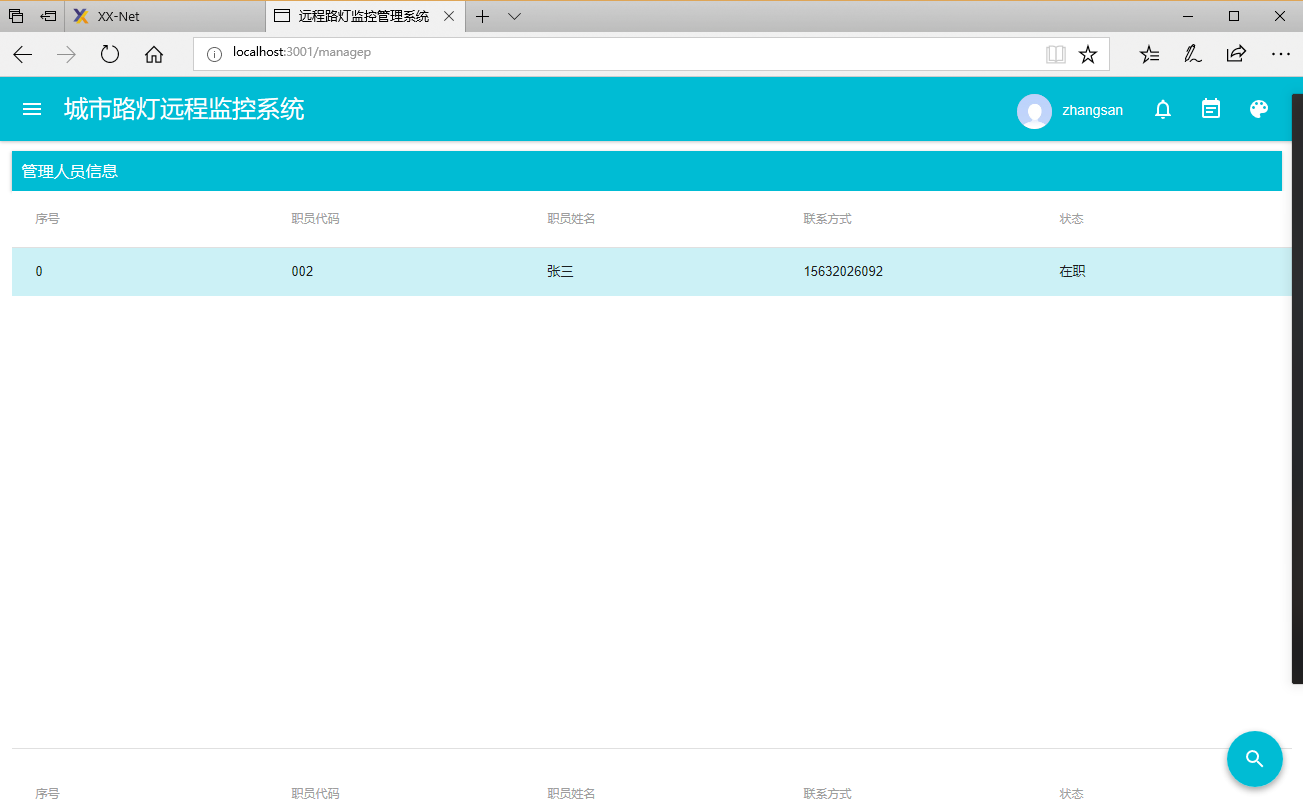






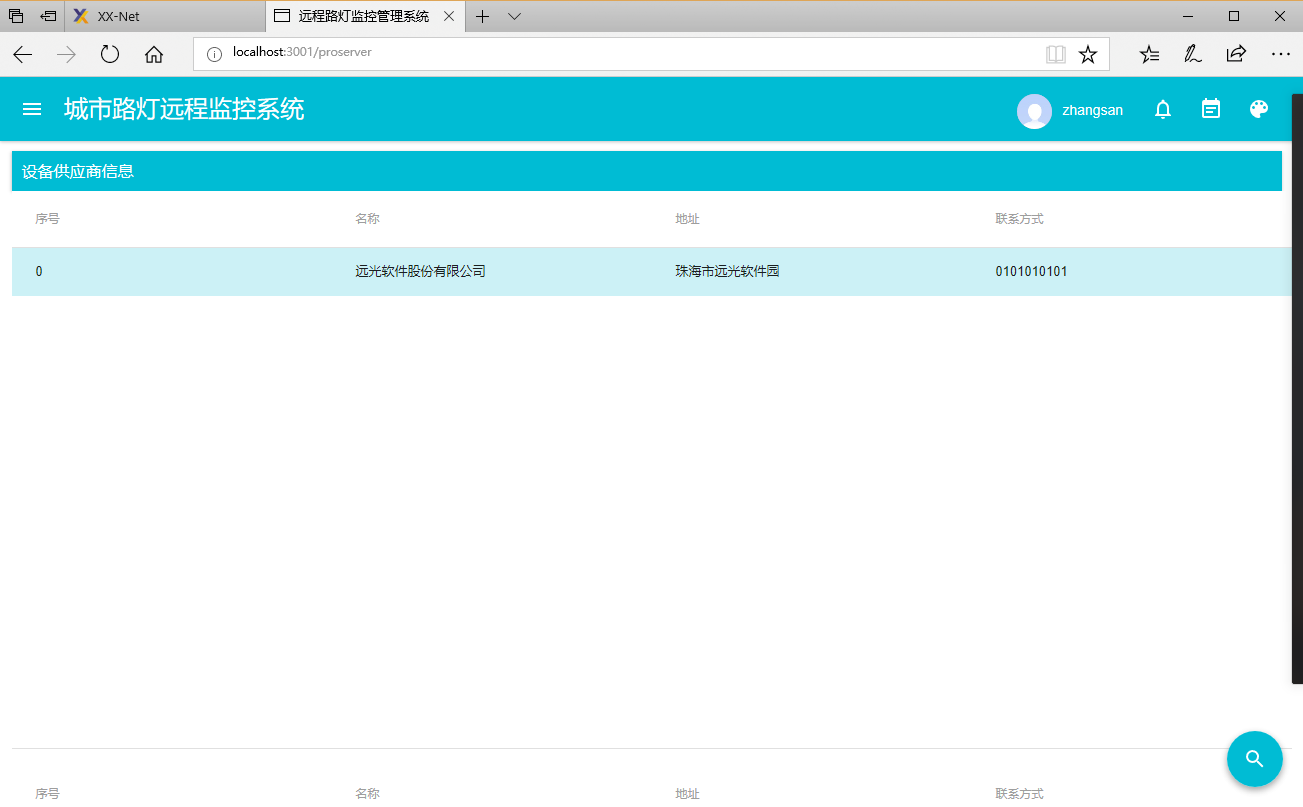
1. 管理人员信息

显示管理人员的一些个人信息，包括联系方式，职工代码，状态等信息，用户可以通过一些信息进行查询。



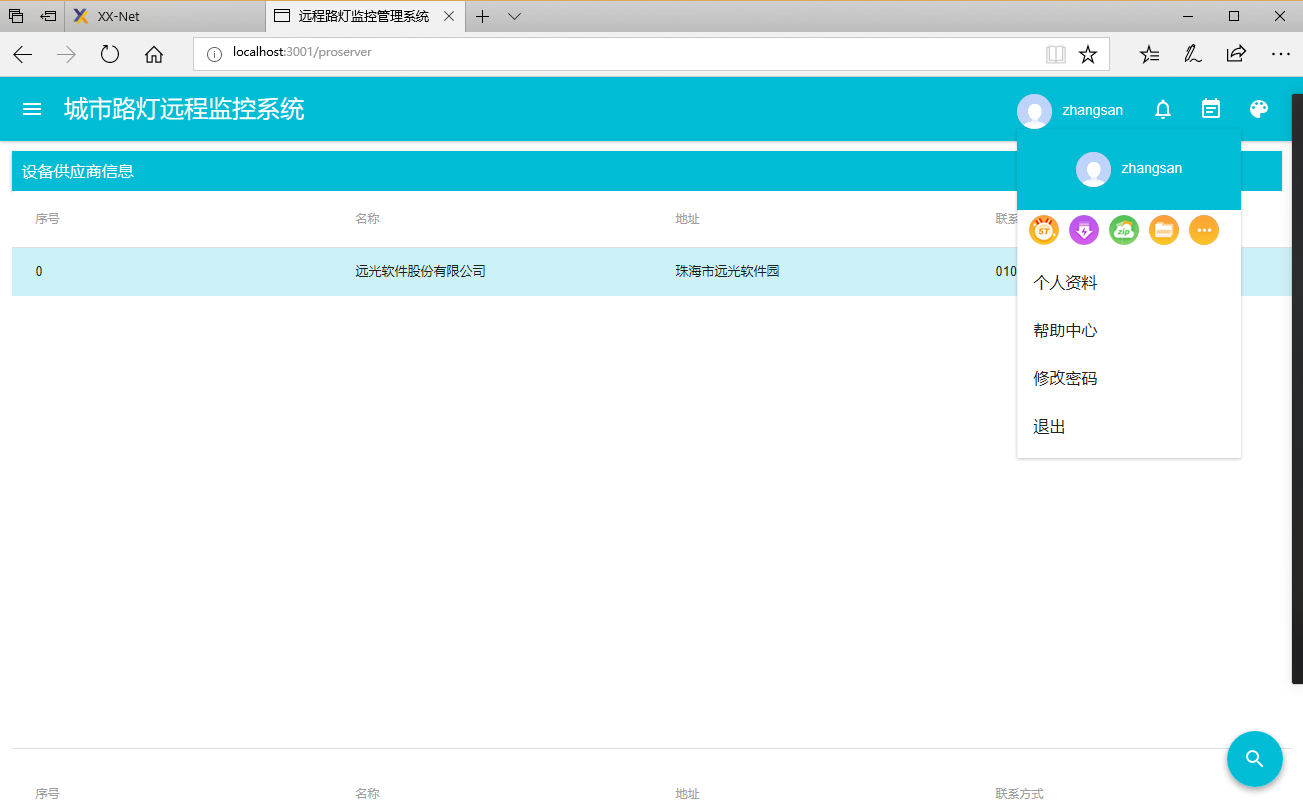
7.维修人员页面和管理人员页面相同

8.设备供应商页面

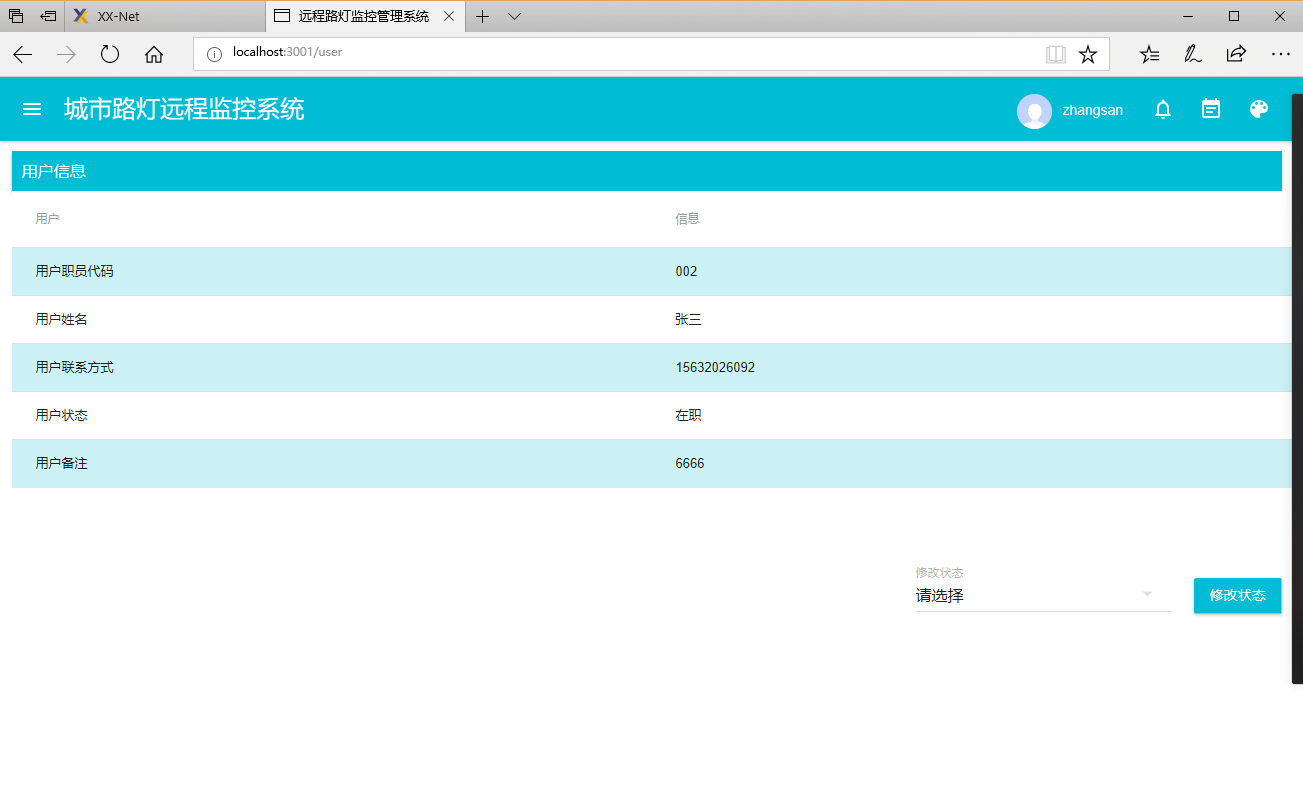


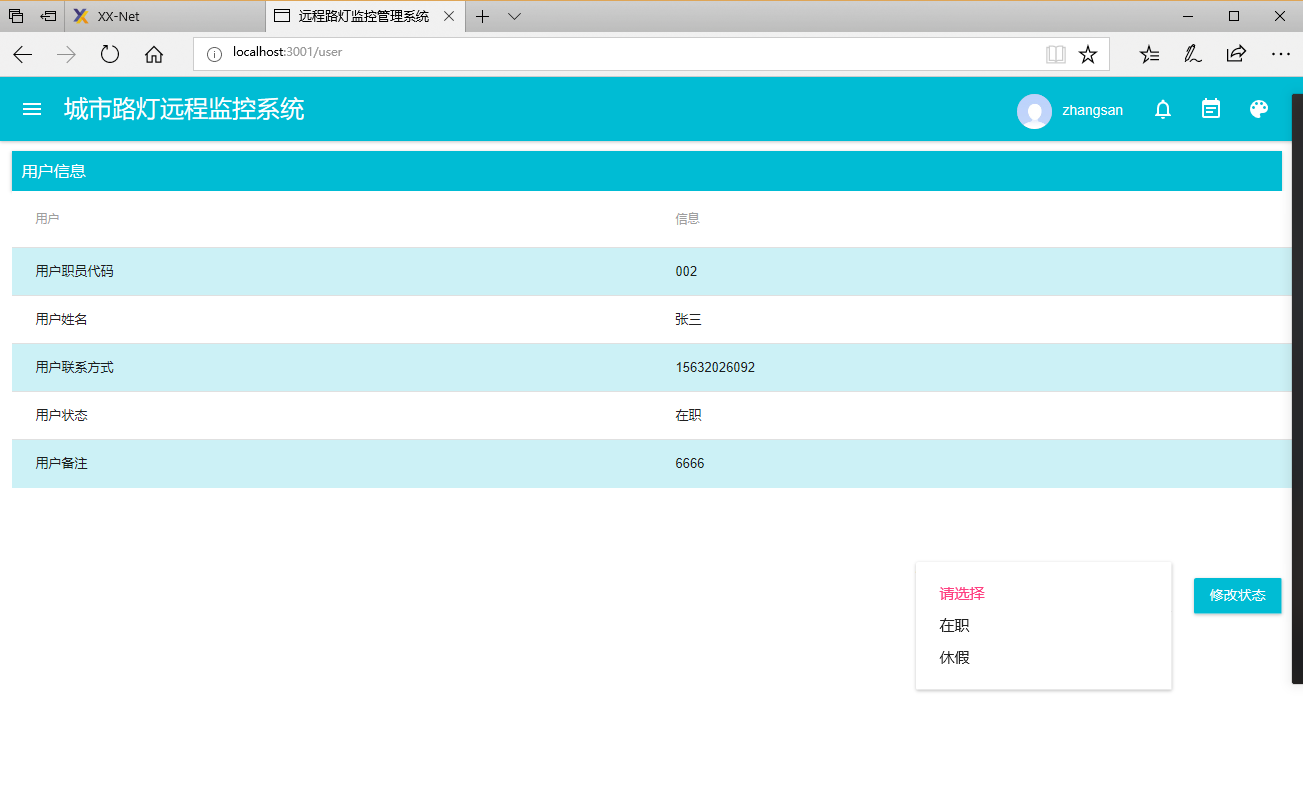
1. 功能栏

当前登录用户的一些操作，报错修改登录密码，修改工作状态等功能。

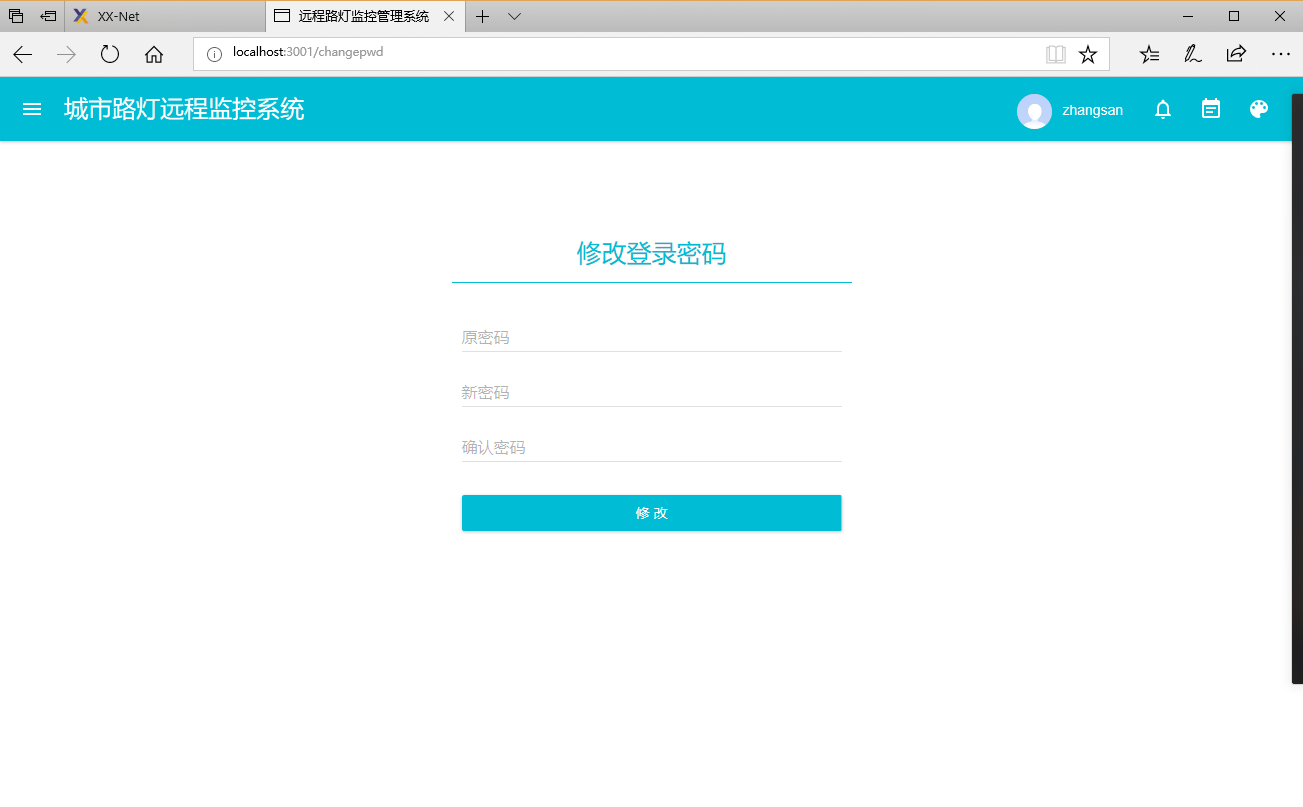


10.用户详细信息、修改状态





11.修改密码，修改之后会重新登录



下面我来介绍一下高德地图的自定义图标的实现方法：

import React, { Component } from 'react';

import { Map, Markers } from 'react-amap';

import { withStyles } from '@material-ui/core/styles';

import Tooltip from '@material-ui/core/Tooltip';

import FormLabel from '@material-ui/core/FormLabel';

import FormControl from '@material-ui/core/FormControl';

import FormGroup from '@material-ui/core/FormGroup';

import FormControlLabel from '@material-ui/core/FormControlLabel';

import green from '@material-ui/core/colors/green';

import blue from '@material-ui/core/colors/blue';

import yellow from '@material-ui/core/colors/yellow';

import red from '@material-ui/core/colors/red';

import purple from '@material-ui/core/colors/purple';

import Checkbox from '@material-ui/core/Checkbox';

import U181 from '../../image/green.png';

import U183 from '../../image/yellow.png';

import U185 from '../../image/blue.png';

import U187 from '../../image/purple.png';

import U189 from '../../image/red.png';

import axios from 'axios';

import history from '../../history';

const h = document.documentElement.clientHeight || document.body.clientHeight;

const styles = {

feng: {

color: green[600],

'&$checked': {

color: green[500],

},

},

huo: {

color: red[600],

'&$checked': {

color: red[500],

},

},

shui: {

color: blue[600],

'&$checked': {

color: blue[500],

},

},

guang: {

color: yellow[600],

'&$checked': {

color: yellow[500],

},

},

he: {

color: purple[600],

'&$checked': {

color: purple[500],

},

},

checked: {},

};

const stylegreen = {

padding: '12px',

backgroundImage: `url(${U181})`,

backgroundSize: '100%',

};

const mouseovergreenStyle = {

padding: '15px',

backgroundImage: `url(${U181})`,

backgroundSize: '100%',

};

const styleyellow = {

padding: '12px',

backgroundImage: `url(${U183})`,

backgroundSize: '100%',

};

const mouseoveryellowStyle = {

padding: '15px',

backgroundImage: `url(${U183})`,

backgroundSize: '100%',

};

const styleblue = {

padding: '12px',

backgroundImage: `url(${U185})`,

backgroundSize: '100%',

};

const mouseoverblueStyle = {

padding: '15px',

backgroundImage: `url(${U185})`,

backgroundSize: '100%',

}

const stylepurple = {

padding: '12px',

backgroundImage: `url(${U187})`,

backgroundSize: '100%',

};

const mouseoverpurpleStyle = {

padding: '15px',

backgroundImage: `url(${U187})`,

backgroundSize: '100%',

};

const stylered = {

padding: '12px',

backgroundImage: `url(${U189})`,

backgroundSize: '100%',

};

const mouseoverredStyle = {

padding: '15px',

backgroundImage: `url(${U189})`,

backgroundSize: '100%',

}

class App extends Component {

state = {

markers: [],

data: [],

mapCenter: {longitude: 110, latitude: 38},

huo: true,

guang: true,

feng: true,

shui: true,

he: true,

markerEvents: {

mouseover:(e, marker) => {

marker.render(this.renderMouseoverLayout);

},

mouseout: (e, marker) => {

marker.render(this.renderMarkerLayout);

}

}

}

componentWillMount() {

axios.post('http://localhost:8080/index/getIndex/')

.then((response) => {

let d = response.data;

let tempdata = [];

for(let i = 0; i < d.length; i++) {

let ttdata = {};

ttdata.plantName = d[i].plantName;

ttdata.powerType = d[i].powerType;

let tt = {};

tt.longitude = d[i].coordinate.split(',')[0];

tt.latitude = d[i].coordinate.split(',')[1];

ttdata.position = tt;

tempdata.push(ttdata);

}

this.setState({data: tempdata, markers: tempdata});

})

.catch(function (error) {

console.log(error);

});

}

renderMouseoverLayout = (extData) => {

if (extData.powerType === '风电'){

return <Tooltip id="tooltip-top" title={extData.plantName} placement="top"><div style={mouseovergreenStyle} onClick={() => history.push('/fac/123456')}>{extData.myLabel}</div></Tooltip>;

}

if (extData.powerType === '水电'){

return <Tooltip id="tooltip-top" title={extData.plantName} placement="top"><div style={mouseoverblueStyle}>{extData.myLabel}</div></Tooltip>;

}

if (extData.powerType === '核电'){

return <Tooltip id="tooltip-top" title={extData.plantName} placement="top"><div style={mouseoverpurpleStyle}>{extData.myLabel}</div></Tooltip>;

}

if (extData.powerType === '火电'){

return <Tooltip id="tooltip-top" title={extData.plantName} placement="top"><div style={mouseoverredStyle}>{extData.myLabel}</div></Tooltip>;

}

return <Tooltip id="tooltip-top" title={extData.plantName} placement="top"><div style={mouseoveryellowStyle}>{extData.myLabel}</div></Tooltip>

}

renderMarkerLayout = (extData) => {

if (extData.powerType === '风电'){

return <div style={stylegreen}>{extData.myLabel}</div>;

}

if (extData.powerType === '水电'){

return <div style={styleblue}>{extData.myLabel}</div>;

}

if (extData.powerType === '核电'){

return <div style={stylepurple}>{extData.myLabel}</div>;

}

if (extData.powerType === '火电'){

return <div style={stylered} >{extData.myLabel}</div>;

}

return <div style={styleyellow}>{extData.myLabel}</div>;

}

handleChange = name => event => {

this.setState({ [name]: event.target.checked});

setTimeout(this.f, 100);

};

f = () => {

let tempdata = this.state.data;

if(!this.state.guang) {

tempdata = tempdata.filter(t => t.powerType !== '太阳能发电');

}

if(!this.state.shui) {

tempdata = tempdata.filter(t => t.powerType !== '水电');

}

if(!this.state.he) {

tempdata = tempdata.filter(t => t.powerType !== '核电');

}

if(!this.state.huo) {

tempdata = tempdata.filter(t => t.powerType !== '火电');

}

if(!this.state.feng) {

tempdata = tempdata.filter(t => t.powerType !== '风电');

}

this.setState({markers: tempdata});

}

render(){

const { classes } = this.props;

return <div>

<div style={{width: '100%', height: h - 66 + 'px'}}>

{/\* <div style={{position: 'fixed', width: '181px', height: '250px', top: '126px', zIndex: 10, left: '50px'}}>

<div style={{background: `url(${feng})`, backgroundSize: '100%', height: '41px', cursor: 'pointer'}}></div>

<div style={{background: `url(${guang})`, backgroundSize: '100%', height: '41px', marginTop: '10px', cursor: 'pointer'}}></div>

<div style={{background: `url(${he})`, backgroundSize: '100%', height: '41px', marginTop: '10px', cursor: 'pointer'}}></div>

<div style={{background: `url(${huo})`, backgroundSize: '100%', height: '41px', marginTop: '10px', cursor: 'pointer'}}></div>

<div style={{background: `url(${shui})`, backgroundSize: '100%', height: '41px', marginTop: '10px', cursor: 'pointer'}}></div>

</div> \*/}

<FormControl component="fieldset" style={{position: 'fixed', top: '126px', zIndex: 10, left: '50px'}}>

<FormLabel component="legend">发电类型</FormLabel>

<FormGroup>

<FormControlLabel

control={

<Checkbox

checked={this.state.huo}

onChange={this.handleChange('huo')}

value="huo"

classes={{

root: classes.red,

checked: classes.checked,

}}

/>

}

label="火电"

/>

<FormControlLabel

control={

<Checkbox

checked={this.state.guang}

onChange={this.handleChange('guang')}

value="guang"

classes={{

root: classes.guang,

checked: classes.checked,

}}

/>

}

label="太阳能发电"

/>

<FormControlLabel

control={

<Checkbox

checked={this.state.feng}

onChange={this.handleChange('feng')}

value="feng"

classes={{

root: classes.feng,

checked: classes.checked,

}}

/>

}

label="风电"

/>

<FormControlLabel

control={

<Checkbox

checked={this.state.shui}

onChange={this.handleChange('shui')}

value="shui"

classes={{

root: classes.shui,

checked: classes.checked,

}}

/>

}

label="水电"

/>

<FormControlLabel

control={

<Checkbox

checked={this.state.he}

onChange={this.handleChange('he')}

value="he"

classes={{

root: classes.he,

checked: classes.checked,

}}

/>

}

label="核电"

/>

</FormGroup>

</FormControl>

<Map plugins={['ToolBar']} center={this.state.mapCenter} zoom={4.5}>

<Markers

events={this.state.markerEvents}

markers={this.state.markers}

render={this.renderMarkerLayout}

/>

</Map>

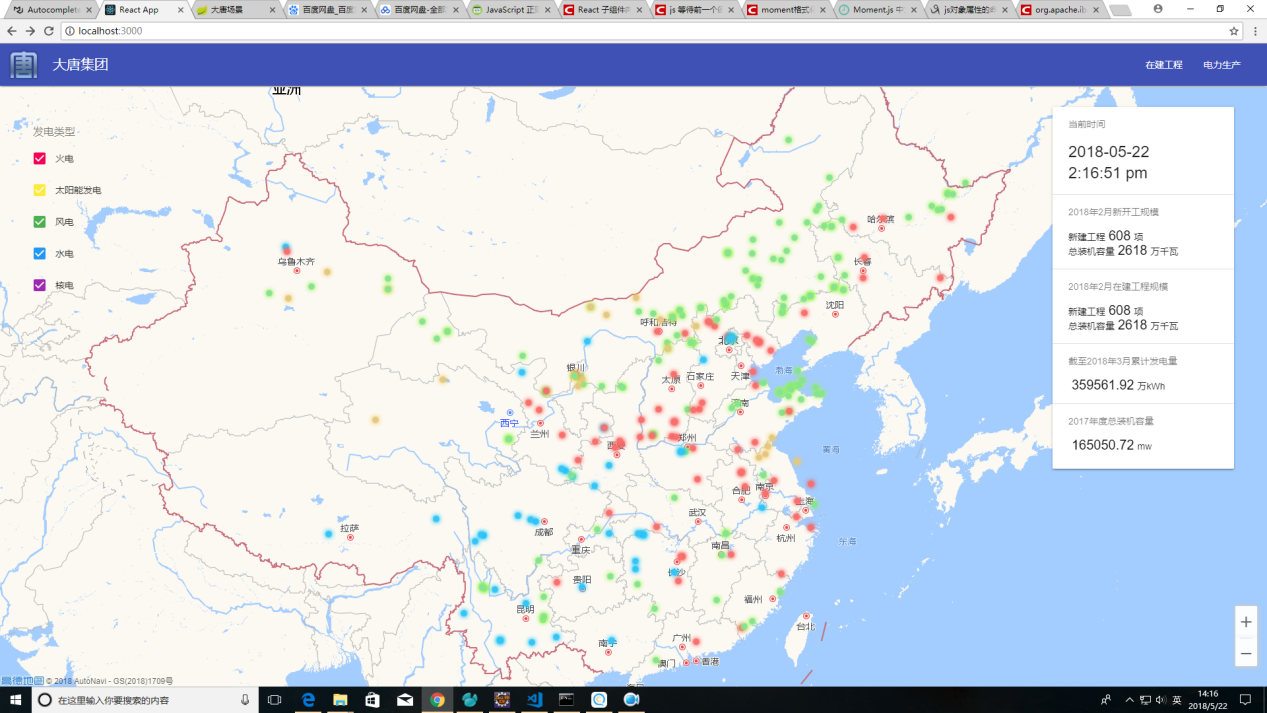
</div>

</div>

}

}

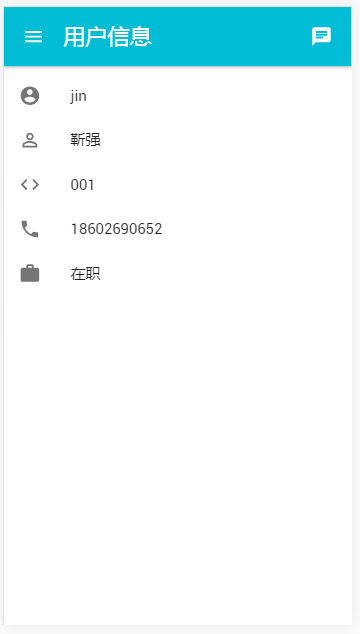
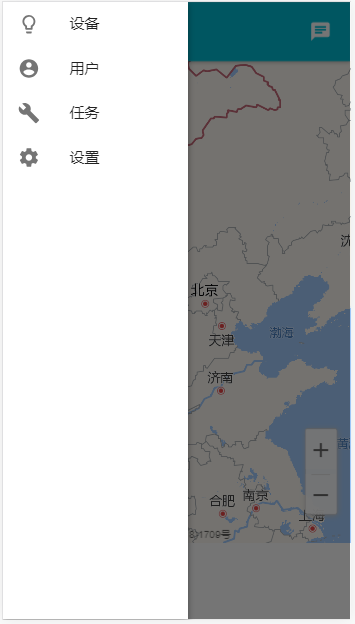
export default withStyles(styles)(App);

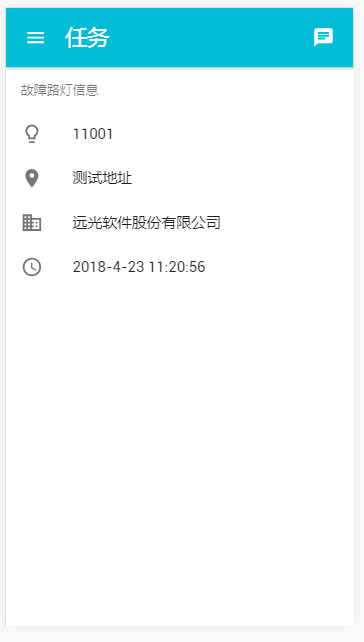
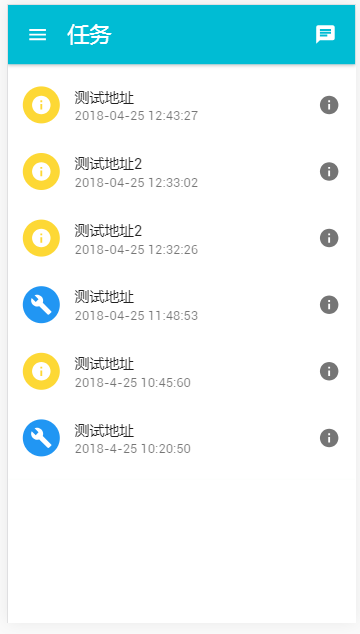


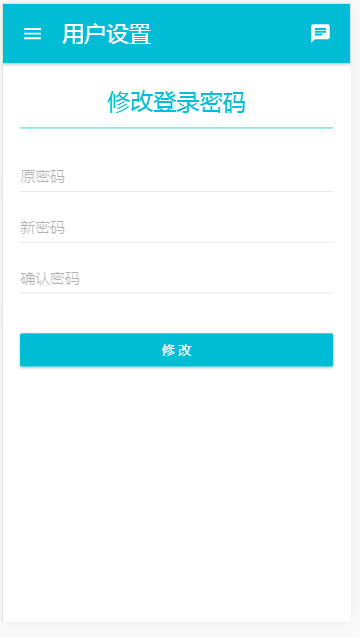
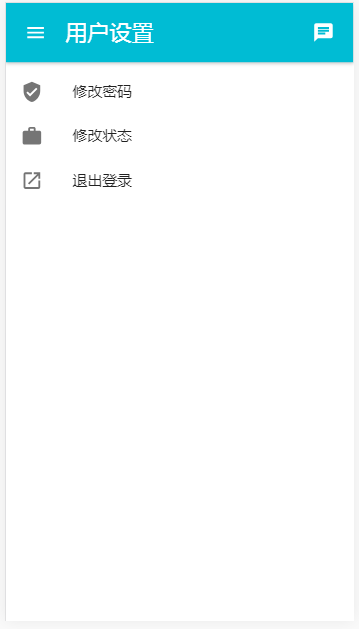
实现效果是这样的，我可以通过左边选择框来控制显示哪些位置信息，地图上的点在hover时会弹出该店的名称信息，次处也可以根据需求来展示信息。同时位置信息也是从后台获取的，可以和动态路由进行配合使用，点击某点，就可以进入详细信息页面，通过在路由上设置不同的位置id即可实现。这只是后期探讨性内容，我并没有把他运用到毕业设计上。

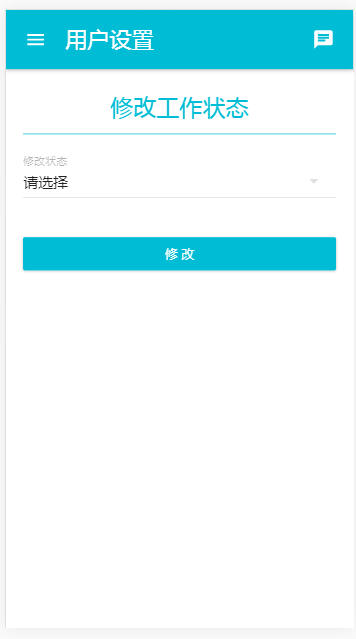
**3.3.5 app页面展示**

****

****

****

****

****

**3.4 后台设计**

**3.4.1 aop拦截web请求、使用自定义注解绑定所需要拦截的controller**

Auth.java自定义注解

package com.jin.road.auth;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target({ElementType.METHOD, ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

public @interface Auth {

String s() default "auth";

}

这里定义了注解@Auth，在方法前标注此注解时会进行检查。

AuthAspect.java

package com.jin.road.auth;

import static org.hamcrest.CoreMatchers.nullValue;

import javax.servlet.http.HttpServletRequest;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Pointcut;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

import org.springframework.web.context.request.RequestContextHolder;

import org.springframework.web.context.request.ServletRequestAttributes;

@Component

@Aspect

public class AuthAspect {

private static final Logger logger = LoggerFactory.getLogger(AuthAspect.class);

@Pointcut(value="@annotation(com.jin.road.Auth)")

public void access() {

}

@Around("@annotation(auth)")

public Object around(ProceedingJoinPoint point, Auth auth) {

ServletRequestAttributes attributes = (ServletRequestAttributes)RequestContextHolder.getRequestAttributes();

HttpServletRequest request = attributes.getRequest();

if(request.getHeader("auth").equals("road2018")) {

try {

return point.proceed();

} catch (Throwable e) {

e.printStackTrace();

return null;// TODO: handle exception

}

} else {

return null;

}

}

}

上面定义了切面类，对标注了@Auth注解的方法进行拦截，通过@Around进行前后处理，对请求头的检查，对结果的封装都可以在此处进行。

只有请求header中包含auth时才将请求分发到controller，进行处理，否则不进行处理。

Aop其实是对cglib的又一次封装，也是通过代理的方式在切点附近进行对切点类功能的一些扩充。还有很多注解经常使用，@Before，@After，@AfterReturning等，都可以在切点之前或之后进行数据的处理。

**3.4.2 controller接口**

package com.jin.road.Controller;

import com.jin.road.Dao.RoadUser;

import com.jin.road.Mapper.UserMapper;

import com.jin.road.auth.Auth;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;

import java.util.Map;

@RestController

public class LoginController {

@Autowired

private UserMapper userMapper;

@Auth

@RequestMapping(value = "/login", method = RequestMethod.POST)

public Map<String, Object> login(@RequestParam String logname, @RequestParam String pwd) throws Exception {

System.err.println(logname);

Map<String, Object> map = new HashMap<>();

RoadUser roaduser = userMapper.findByLogname(logname);

if(pwd.equals(roaduser.getPwd())) {

map.put("success", true);

map.put("data", roaduser);

} else {

map.put("success", false);

}

return map;

}

@Auth

@RequestMapping(value = "/loginwork", method = RequestMethod.POST)

public Map<String, Object> loginwork(@RequestParam String logname, @RequestParam String pwd) throws Exception {

System.err.println(logname);

Map<String, Object> map = new HashMap<>();

RoadUser roaduser = userMapper.findWorkByLogname(logname);

if(pwd.equals(roaduser.getPwd())) {

map.put("success", true);

map.put("data", roaduser);

} else {

map.put("success", false);

}

return map;

}

@Auth

@RequestMapping(value = "/changepwd", method = RequestMethod.POST)

public Map<String, Object> changepwd(@RequestParam String logname, @RequestParam String oldpwd, @RequestParam String newpwd) throws Exception {

Map<String, Object> map = new HashMap<>();

RoadUser roaduser = userMapper.findByLogname(logname);

if(oldpwd.equals(roaduser.getPwd())) {

userMapper.changepwd(newpwd, logname);

map.put("success", true);

} else {

map.put("success", false);

}

return map;

}

@Auth

@RequestMapping(value = "/changeworkpwd", method = RequestMethod.POST)

public Map<String, Object> changeworkpwd(@RequestParam String logname, @RequestParam String oldpwd, @RequestParam String newpwd) throws Exception {

Map<String, Object> map = new HashMap<>();

RoadUser roaduser = userMapper.findWorkByLogname(logname);

if(oldpwd.equals(roaduser.getPwd())) {

userMapper.changeworkpwd(newpwd, logname);

map.put("success", true);

} else {

map.put("success", false);

}

return map;

}

@Auth

@RequestMapping(value = "/changestate", method = RequestMethod.POST)

public Map<String, Object> changestate(@RequestParam String state, @RequestParam String code) throws Exception {

Map<String, Object> map = new HashMap<>();

userMapper.changestate(state, code);

map.put("success", true);

return map;

}

}

响应请求，进行数据查询，并将json格式的数据返回到客户端。如果发生异常会使用自定义的全局异常处理类进行处理，返回错误信息。

**3.4.3 配置跨域**

因为前后端分别部署，需要进行跨域设置。这里我设置任何ip都可以访问，现实情况可以根据部署情况进行设置，设置ip白名单。

简化的方式也可以使用@CrossOorign注解实现跨域。

private CorsConfiguration buildConfig() {

CorsConfiguration corsConfiguration = new CorsConfiguration();

corsConfiguration.addAllowedOrigin("\*");

corsConfiguration.addAllowedHeader("\*");

corsConfiguration.addAllowedMethod("\*");

return corsConfiguration;

}

/\*\*

\* 跨域过滤器

\* @return

\*/

@Bean

public CorsFilter corsFilter() {

UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();

source.registerCorsConfiguration("/\*\*", buildConfig()); // 4

return new CorsFilter(source);

}

**3.4.4 全局异常处理**

在数据请求出现错误时进行异常处理，将错误信息以json格式返回给客户端。

使用@ControllerAdvice注解说明这是异常处理方法，在此处也可以自定义异常类，分别返回不同的结果。

@RestController注解会使返回信息以json的格式。

package com.jin.road.exception;

import java.util.HashMap;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.bind.annotation.RestController;

@ControllerAdvice

@RestController

public class GlobalExceptionHandler {

private static final Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.class);

@ExceptionHandler(value = Exception.class)

public Map<String, Object> defaultErrorHandler(HttpServletRequest req, Exception e) throws Exception {

logger.error(e.getMessage());

Map<String, Object> map = new HashMap<>();

map.put("success", false);

return map;

}

}

**3.4.5 mybatis配置**

application.properties中设置包扫描范围

mybatis.mapper-locations=classpath:MapperXml/\*.xml

pom中加入配置

<resources>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.xml</include>

</includes>

</resource>

<resource>

<directory>src/main/resources</directory>

</resource>

</resources>

Mapper 接口

package com.jin.road.Mapper;

import com.jin.road.Dao.RoadUser;

import com.jin.road.Dao.WorkUser;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;

import org.apache.ibatis.annotations.Param;

import org.apache.ibatis.annotations.Select;

import org.apache.ibatis.annotations.Update;

import org.springframework.stereotype.Component;

@Component

@Mapper

public interface UserMapper {

RoadUser findByLogname(@Param("logname") String logname);

RoadUser findWorkByLogname(@Param("logname") String logname);

@Update("UPDATE LOGUSER SET pwd = #{pwd} WHERE logname LIKE #{logname}")

void changepwd(@Param("pwd") String pwd, @Param("logname") String logname);

@Update("UPDATE LOGINWORK SET pwd = #{pwd} WHERE logname LIKE #{logname}")

void changeworkpwd(@Param("pwd") String pwd, @Param("logname") String logname);

@Update("UPDATE USER SET state = #{state} WHERE code LIKE #{code}")

void changestate(@Param("state") String state, @Param("code") String code);

@Update("UPDATE USER SET STATE = #{state} WHERE code LIKE #{code}")

void fix(@Param("state") String state, @Param("code") String code);

@Select("SELECT \* FROM USER WHERE POSITION LIKE #{position}")

List<WorkUser> findByPosition(@Param("position") String position);

@Select("SELECT \* FROM USER WHERE POSITION LIKE '维修工' AND STATE LIKE '在职'")

List<WorkUser> findWork();

}

复杂的sql语句可以使用xml格式

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.jin.road.Mapper.UserMapper" >

<select id="findByLogname" resultType="com.jin.road.Dao.RoadUser">

SELECT loguser.id, loguser.logname, loguser.pwd, loguser.code , user.name, user.phone, user.state, user.remark

FROM loguser

LEFT JOIN user

ON loguser.code like user.code

WHERE logname like #{logname}

</select>

<select id="findWorkByLogname" resultType="com.jin.road.Dao.RoadUser">

SELECT loginwork.id, loginwork.logname, loginwork.pwd, loginwork.code , user.name, user.phone, user.state, user.remark

FROM loginwork

LEFT JOIN user

ON loginwork.code like user.code

WHERE logname like #{logname}

</select>

</mapper>

命名空间写需要引入的接口名称。

**3.4.6 mysql配置**

application.properties加入

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.datasource.url=jdbc:mysql://localhost:3306/road?useUnicode=true&characterEncoding=utf8&useSSL=false

spring.datasource.username=root

spring.datasource.password=

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.initialSize=5

spring.datasource.minIdle=5

spring.datasource.maxActive=20

spring.datasource.maxWait=60000

spring.datasource.timeBetweenEvictionRunsMillis=60000

spring.datasource.minEvictableIdleTimeMillis=300000

spring.datasource.validationQuery=SELECT 1 FROM DUAL

spring.datasource.testWhileIdle=true

spring.datasource.testOnBorrow=false

spring.datasource.testOnReturn=false

spring.datasource.poolPreparedStatements=true

spring.datasource.maxPoolPreparedStatementPerConnectionSize=20

spring.datasource.filters=stat,wall,log4j

spring.datasource.connectionProperties=druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000

这里我使用了durid连接池。Druid是阿里巴巴开源平台上的一个项目，整个项目由数据库连接池、插件框架和SQL解析器组成。该项目主要是为了扩展JDBC的一些限制，可以让程序员实现一些特殊的需求，比如向密钥服务请求凭证、统计SQL信息、SQL性能收集、SQL注入检查、SQL翻译等，程序员可以通过定制来实现自己需要的功能。

**3.4.7 logback配置**

Logback-boot.xml

对日志的记录进行配置，这里设置了记录日志级别高于DEBUG的都会被记录，控制台只会记录INFO级别以上的日志。文件大小超过10MB将会自动截断，创建新的文件继续记录。

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>

%d %p (%file:%line\)- %m%n

</pattern>

<charset>UTF-8</charset>

</encoder>

</appender>

<appender name="roadlog" class="ch.qos.logback.core.rolling.RollingFileAppender">

<File>log/sys.log</File>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>log/road.%d.%i.log</fileNamePattern>

<maxHistory>30</maxHistory>

<timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">

<maxFileSize>10MB</maxFileSize>

</timeBasedFileNamingAndTriggeringPolicy>

</rollingPolicy>

<encoder>

<pattern>

%d %p (%file:%line\)- %m%n

</pattern>

<charset>UTF-8</charset>

</encoder>

</appender>

<root level="info">

<appender-ref ref="STDOUT" />

</root>

<logger name="com.jin" level="DEBUG">

<appender-ref ref="roadlog" />

</logger>

</configuration>

这里配置了日志记录的格式，显示级别，命名方式，存储位置。

Application.properties加入配置文件位置

logging.config=classpath:logback-boot.xml

这样就可以实现日志的记录。

**3.4.8 数据分页**

考虑到系统运行的数据量应该不是很大，我没有在数据展示页实现分页效果。但是我也对分页实现方法进行了一些研究，比较好用的是使用pagehelper，下面介绍一下pagehelper的使用。

首先要在pom中引入pagehelper依赖，在github上或maven中心库都可以查到。

和mybatis结合：

@Configuration

public class MybatisConf {

@Bean

public PageHelper pageHelper() {

PageHelper pageHelper = new PageHelper();

Properties properties = new Properties();

properties.setProperty("offsetAsPageNum", "true");

properties.setProperty("rowBoundsWithCount", "true");

properties.setProperty("reasonable", "true");

pageHelper.setProperties(properties);

return pageHelper;

}

}

这里设置了一些参数，可根据需要进行添加或修改。

使用起来很简单，只要在mybatis mapper返回数据前添加PageHelper.startPage(1, 2);

第一个参数是页数，第二个参数是每页显示的条数。分页处理是很简单的。

**3.4.9 定时任务配置**

记录最近七天的路灯信息记录时需要创建定时任务，每天的23点查询路灯故障数，维修数，正常数，写入数据库。

@Transactional

@Scheduled(cron="0 0 23 \* \* ?")

public void setcountrecord() {

int error = countLampMapper.errorcount();

int fix = countLampMapper.fixcount();

int ok = countLampMapper.okcount();

SimpleDateFormat d = new SimpleDateFormat("yyyy-MM-dd");

String time = d.format(new Date());

countLampMapper.setrecord(error, fix, ok, time);

}

使用@Scheduled注解实现定时任务，cron表达式设置为每日的23点，这样每天的23点就会将当天的路等情况写入数据库。使用这个注解还需要在spring boot启动类上加入@EnableScheduling注解，同时设置定时任务类为@component，这样项目启动时就会执行这个容器类。

**3.4.10 分布式架构想法**

在写完这个项目时，我也接触到了spring clooud微服务，我在思考这个项目是否也可以改成分布式的架构，把不同的功能模块进行拆分，下面是我的一些项目该写方案。

首先将路灯信息服务和用户信息拆分为两个生产者client，使用eureka作为注册中心，eureka ribbon项目作为消费者来调用生产者的接口，添加断路器hystris，并将不同的消费者通过统一的zuul进行网关管理，项目为gateway，设置accessfilter来添加拦截器对外部请求进行拦截，符合条件的进行分发到消费者。

下面是我是用的一些重点的代码：

Applications.properties中写入

注册中心地址：eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/

服务名称：spring.application.name=eureka-consumer

Ribbon调用client方法：

@RestController

public class DcController {

@Autowired

Dcservice dcservice;

@GetMapping("/consumer")

public String dc() {

return dcservice.consumer();

}

}

Dcservice接口：

@Service

public class Dcservice {

@Autowired

RestTemplate restTemplate;

@HystrixCommand(fallbackMethod = "fallback")

public String consumer() {

return restTemplate.getForObject("http://eureka-client/dc", String.class);

}

public String fallback() {

return "dcfallback";

}

}

这里使用了RestTemplate来进行请求转发，@HystrixCommand注解设置断路器，当请求出现延迟或失败时会触发fallback函数。

还需要在启动类中添加

@EnableHystrix

@EnableDiscoveryClient

开启断路器服务，服务发现，装配RestTemplate

@Bean

@LoadBalanced

public RestTemplate restTemplate() {

return new RestTemplate();

}

@LoadBalanced 负载均衡

统一网关

spring.application.name=api-gateway

server.port=1101

eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/

//定义路由服务

zuul.routes.eureka-consumer=/consumer/\*\*

启动类中开启@EnableZuulProxy

拦截器：

public class AccessFilter extends ZuulFilter {

private static Logger logger = LoggerFactory.getLogger(AccessFilter.class);

@Override

public String filterType() {

return "pre";

}

@Override

public int filterOrder() {

return 0;

}

@Override

public boolean shouldFilter() {

return true;

}

@Override

public Object run() {

RequestContext context = RequestContext.getCurrentContext();

HttpServletRequest request = context.getRequest();

logger.info("send {} request to {}", request.getMethod(), request.getRequestURL().toString());

String accessToken = request.getHeader("tooken");

System.out.println(accessToken);

if(accessToken == null) {

logger.error("access token is empty");

context.setSendZuulResponse(false);

context.setResponseStatusCode(401);

return null;

}

logger.info("access token ok");

return null;

}

}

对请求进行拦截，只有符合要求的才会被转发，不符合的将会返回401错误。

在启动类装配该拦截类为Bean。

@Bean

public AccessFilter accessFilter() {

return new AccessFilter();

}

这样就可以通过统一的入口来访问服务了。

1. **项目测试**

**4.1 测试结果说明**

编写好前后端后，进行项目的测试，查看数据导入是否正确，登录是否能够成功，处理故障路灯信息是否成功，提醒维修是否成功，搜索结果显示是否正确，记住密码功能是否正常，用户操作是否正常，是否流畅，app端接收信息是否成功。

继续按此顺序进行其他测试。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 用户 | 密码 | 注册 | 信息 | 修改密码 | 退出登录 | 修改状态 |
| zhangsan | 123456 | 成功 | 准确 | 成功 | 成功 | 成功 |
| jin | 123456 | 成功 | 准确 | 成功 | 成功 | 成功 |
| lisi | 123456 | 成功 | 准确 | 成功 | 成功 | 成功 |

测试效果均成功。

**4.2 测试局限性分析**

测试时是在chrome下调试运行，可能测试结果可信度不高，测试时数据量不大，可能存在数据处理方面的不足，数据的实时反映可能会使系统卡顿。

路灯的位置会比较密集，在地图上显示可能会变得连接在一起，不易区分，测试用的数据还是比较少，不能实现实际中的效果，最好分区域统计，在一张地图上显示太过拥挤了。

项目如何与硬件结合是测试时不能满足的条件，我假设的是路灯每过一段时间就会发送运行状态到服务器，服务器根据状态信息更新数据库中的信息，如果路灯数量非常大，服务器如何处理这些请求是难点。同时我展示的信息也比较少，可能实际情况会有数倍的信息要进行前后端的传输，这是分页应该是必须的，所以我后来有研究了分页的方法。

但系统设计逻辑清晰，基本实现了预期功能，测试用例反应也良好，操作很流畅。可以说此次系统的设计基本算是成功的。

**结论**

完成这个毕业设计前前后后大约花费了2个月的时间，总体来说，过程还是比较顺利的，但有时候也会遇到一些棘手，难以解决的问题，比如mybatis扫描不到xml配置，axios的跨域问题，多表联查，统一异常处理，aop连接请求。都需要查看大量的资料，但是经过网络查找解决方法，这些问题都得到了解决，同时也让我有了很多的项目实战经验，为以后的工作打下了好的基础。

经过此次毕业设计，我不仅熟悉了react、redux、springboot等等框架的使用方法，同时也完成了研究内容，react相比于传统的html，更加简洁，更加高效。Springboot简化了sprng的xml配置，自动装配，简洁的resource配置就可以完成庞大的工作，极大的简化了开发人员的工作量，以后一定会更加火爆。uo经过这个系统的设计与实现，我对如何开发单页面应用有了一定的了解，对后台的编写也有了经验，现在流行的技术框架基本在本项目中都用到了，虽然系统可能功能不能达到商业用途，但是也是一个好的开端。总之，这次毕业设计我收获了很多，同时作品效果也令人满意。

在项目基本完成后我也尝试了一些项目中没有用到的技术，pagehelper，自定义地图标志，微服务架构spring boot,druid，rabbitmq和kafka，都有了一些使用经历，可以根据需求对项目进行改进，添加新的功能。做这个毕业设计时我正在实习，实习过程中也接触到了做完整项目的流程，我们属于开发人员，需要根据需求制定的方案，由ui设计师来设计具体的页面，最后才是开发人员编写代码实现，毕业设计完全是自己一个人编写，以前也没有开发经验，所以项目中也会有很多不足，很多需要改进的地方。只有真的实践了才能够体会到真正的开发过程。

在开发过程中我也在不断熟悉编写代码的方式，在进行项目编写时一定要先确定产品要实现的功能，根据功能进行系统的模块设计，前期的准备越详细，后面的开发越顺利，如果功能都没有确定，很容易在开发时出现无效的工作。所以如何在开发之前就明确需求是非常重要的，做好调查，确定软件需要实现的功能，确定技术方案，这样可以减少很多不必要的劳动。在这次设计过程中我对这点感触很深，因为我开始时没有明确自己想实现的功能，所以在编写代码时不断的推翻重来，写完之后又想加其他功能，结果之前的代码可能要全部修改，甚至要重写。严重的拉长了开发时间。直到最后我还在想不断的改变功能，修改页面，虽然由于时间关系没有在修改项目，但是我还是在其他地方分别实现了一下，看是否可以达到我的需求，如果我开发之前准备的更充分一些，可能做出的产品功能会更加丰富，界面更美观。

**附录**

**附录1 主要参考资料**

[1]顾勇.浅析城市路灯的建设与管理[J].经济视野，2013（24）：23～24.

[2]翟芳，冯学奎，张悦.城市路灯建设与管理工作思考[J].城市，2009（9）：76～78.

[3]耿桂英.浅谈路灯建设与管理的主要问题[J].城市建设理论研究：电子版，2012：11～12.

[4]郑远鹏.城市路灯建设措施及管理工作开展分析[J].建筑工程技术与设计，2014（32）：36～37.

[5]陈屹，深入React技术栈.人民邮电出版社

[6]程墨，深入浅出React和Redux，机械工业出版社.

[7]克雷格.沃斯，Spring Boot实战，人民邮电出版社

[8]Brian Goetz,Tim Peierls,Joshua Bloch，Java Concurrentcy in Practice，机械工业出版社.

[9]Bohme T, Rahm E. Supporting Efficient Streaming and Insertion of XML Data inRDBMS[C] / /Proc. 3rd Int. Workshop Data Integration over the Web (D IWeb) ,2004.

[10]陈韶健，深入实践Spring Boot，机械工业出版社.

[11]杨开振，深入浅出MyBatis技术原理与实战，电子工业出版社

[12]福塔，MySQL必知必会，人民邮电出版社.

[13]Dilwyn Edards, Mike Hamson. Guide to Mathematical Mdellng. WorldPublishing Corp. 1989

[14]James Laudon, Lawrence Spracklen. The Coming Wave of Multithreaded ChipMultiprocessors[J]. International Journal of Parallel Programming . 2007 (3)

[15]Y.Liu , C.J.Liu. Scheduling strategy design of thread pool for real-time controlsystem,Computer Engineering and Applications. 2010. 46(32): 71-73