摘要

某监控公司近10年组织结构进行调整，业务扩展。原有的组织层级已经不能很好的展现出各个层级之间的关系和该层级相对应的责任，业务数据、设备信息、员工信息等数据信息仍是由人工记录，效率较低，错误率较高。

针对以上问题，决定设计实现一套能够对公司的组织结构、设备信息、业务数据、员工信息等信息进行统一管理的管理系统。具体采用前后端分离这种架构进行设计，使用Spring boot、Vue、Ajax、MySQL、Redis等技术进行开发。系统最终能够以简明清晰的方式展示出公司的各个层级的责任与层级间的关系，以及以数字化的方式对公司的各类信息进行管理。

**关键词：**管理系统,前后分离，Spring boot，Vue

ABSTRACT

In the past 10 years, a monitoring company has adjusted its organizational structure and expanded its business. The original organizational hierarchy has not been able to show the relationship between each level and the corresponding responsibilities of the hierarchy. The data of business data, equipment information, employee information and so on are still recorded manually, with low efficiency and error rate. high.

In response to the above problems, it was decided to design and implement a management system that can manage the company's organizational structure, equipment information, business data, employee information and other information. Specifically, the architecture is designed by separating the front and back ends, and using Spring boot, Vue, Ajax, MySQL, Redis and other technologies for development. The system ultimately demonstrates the relationship between responsibilities and levels at all levels of the company in a clear and concise manner, as well as the management of the company's various types of information in a digital manner.

**Key Words:** Management System,Separation Before and After, Spring boot, Vue.

**目 录**

[第1章 引言 1](#_Toc8141736)

[1.1 课题来源 1](#_Toc8141737)

[1.2 研究意义和目标 1](#_Toc8141738)

[1.2.1 研究意义 1](#_Toc8141739)

[1.2.2 研究目标 1](#_Toc8141740)

[1.3 研究思路 2](#_Toc8141741)

[1.3.1 技术选型 2](#_Toc8141742)

[1.3.2 HTML5概述 2](#_Toc8141743)

[1.3.3 JavaScript概述 2](#_Toc8141744)

[1.3.4 Web App概述 3](#_Toc8141745)

[1.3.5 Spring boot概述 3](#_Toc8141746)

[1.3.6 Ajax概述 3](#_Toc8141747)

[1.3.7 前后端分离概述 4](#_Toc8141748)

[第2章 设计方案 5](#_Toc8141749)

[2.1 开发环境 5](#_Toc8141751)

[2.2 软件结构图 5](#_Toc8141752)

[2.3 功能调用时序图 6](#_Toc8141753)

[第3章 功能实现 7](#_Toc8141754)

[3.1 功能实现思路 7](#_Toc8141756)

[3.2 项目生成 7](#_Toc8141757)

[3.2.1 后端开发流程 7](#_Toc8141758)

[3.2.2 前端开发流程 10](#_Toc8141759)

[3.3 后端项目详细介绍 12](#_Toc8141760)

[3.3.1 程序实现 12](#_Toc8141761)

[3.3.2 日志模块 13](#_Toc8141762)

[3.3.3 缓存模块 15](#_Toc8141763)

[3.3.4 验证码模块 17](#_Toc8141764)

[3.3.5 认证授权模块 18](#_Toc8141765)

[3.3.6 控制器模块 24](#_Toc8141766)

[3.3.7 数据库设计 24](#_Toc8141767)

[3.3.8 异常模块 28](#_Toc8141768)

[3.3.9 数据库操作模块 28](#_Toc8141769)

[3.3.10 Service模块 29](#_Toc8141770)

[3.4 前端项目详细介绍 31](#_Toc8141771)

[3.4.1 SPA介绍 31](#_Toc8141772)

[3.4.2 模块化 32](#_Toc8141773)

[3.4.3 Vue 32](#_Toc8141774)

[3.4.4 路由控制 34](#_Toc8141775)

[第4章 调试与实现 36](#_Toc8141776)

[4.1 调试中遇到的重点与难点 36](#_Toc8141778)

[4.1.1 调试重点 36](#_Toc8141779)

[4.1.2 调试难点 36](#_Toc8141780)

[4.2 解决方案 36](#_Toc8141781)

[4.3 实现展示 37](#_Toc8141782)

[4.3.1 登录页面 37](#_Toc8141783)

[4.3.2 组织结构管理页面 37](#_Toc8141784)

[4.3.3 设备管理页面 38](#_Toc8141785)

[4.3.4 人脸信息管理页面 38](#_Toc8141786)

[4.3.5 权限管理页面 39](#_Toc8141787)

[4.3.6 日志管理 39](#_Toc8141788)

[4.3.7 资源结构管理页面 40](#_Toc8141789)

[第5章 总结 41](#_Toc8141790)

[参考文献 43](#_Toc8141791)

[致谢 44](#_Toc8141792)

第1章 引言

1.1 课题来源

某监控公司紧10年内发展迅速，组织结构多次变化，原有系统已不能明确的展示出组织层级以及展示出某个组织层级的详细信息与该层级的负责人信息、层级职责。

该公司的监控设备在全国各地都有部署，因为监控设备不再仅作为监控设备使用，现在添加了人脸识别功能还有另外一些附加功能，原有系统不能很好的扩展这些功能。

因此委托我们公司开发一套管理系统，具有组织结构查看、平台资源管理的功能。

1.2 研究意义和目标

1.2.1 研究意义

近几年Web前端技术发展迅速，2010年MV\*框架出现，2014年HTML5正式发布，2015年ECMAScript6发布，2016年Chrome浏览器占据市场半壁江山。可以说现在的前端技术已经形成了一套规范，网页已经由最初的Web Site演变成了Web App。

常用来开发后端服务Java语言语法一直以厚重繁杂为特点，官方标准EJB实现起来复杂，后来出现的Spring框架缓解了EJB带来的痛楚，但又出现了繁杂的XML配置。Java程序员们才从EJB坑中跳出来，又进了一个新坑。直到现在Spring-boot的出现，它大大的缓解了Java开发服务器应用繁杂厚重的困难。

本科四年期间，我对Web技术很感兴趣，当面对一个实际需求时，我已跃跃欲试，欲使用在本科期间学习的知识实现一个前端应用为单页应用，后端服务仅提供数据，页面渲染、数据交由浏览器的低耦合性的管理系统。

1.2.2 研究目标

公司组织结构数据展示：根据监控公司组织结构设计出一套合理的数据结构，并把数据结构实现且将它在浏览器上合理展示出来。

公司业务数据结构数据展示：对公司主营监控业务有查看业务数据的功能。

权限控制：针对不同职位的员工具有不同的操作权限。

1.3 研究思路

1.3.1 技术选型

因需要开发前后端应用，经过综合考虑选了以下解决方案。

前端应用使用HTML5+JavaScript(EMCAScript6+)+CSS3实现的Vue框架开发。

后端应用使用Spring boot开发。

架构方面使用前后端分离架构。

1.3.2 HTML5概述

定义：万维网的核心语言、标准通用标记语言下的一个应用超文本标记语言（HTML）的第五次重大修改。HTML5的设计目的是为了在移动设备上支持多媒体。新的语法特征被引进以支持这一点，如video、audio和canvas 标记。

HTML5还引进了新的功能，可以真正改变用户与文档的交互方式，包括：新的解析规则增强了灵活性、新属性、淘汰过时的或冗余的属性、一个HTML5文档到另一个文档间的拖放功能、离线编辑、信息传递的增强、详细的解析规则、多用途互联网邮件扩展（MIME）和协议处理程序注册、在SQL数据库中存储数据的通用标准（Web SQL）。

HTML5未来会更侧重于移动优先，因为现在很大一部分的人已经到了离开手机就没法生活的地步了，移动优先已经成了一种趋势，不管开发什么，都以移动为主。

1.3.3 JavaScript概述

JavaScript一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为JavaScript引擎，为浏览器的一部分，广泛用于客户端的脚本语言，最早是在HTML（标准通用标记语言下的一个应用）网页上使用，用来给HTML网页增加动态功能。JavaScript包括ECMAScript、BOM、DOM。BOM是浏览器内置对象，DOM是文档树对象。

ECMAScript 6（简称ES6）是于2015年6月正式发布的JavaScript语言的标准，正式名为ECMAScript 2015（ES2015）。它的目标是使得JavaScript语言可以用来编写复杂的大型应用程序，成为企业级开发语言。

1.3.4 Web App概述

此处Web App指的是单页应用(Single Page Application)。早期web应用的前后端交互模式是这样的，每个html作为一个功能元件，通过刷新、超链接、表单提交等方式，将页面组织起来后给用户提供交互。浏览器只作为展示层，将MVC全置于后端，加重了服务端的体量，开发中主要以后端为主。页面展示依赖于不同的功能元件，所以必须依靠刷新页面，或者跳转路由来实现功能块的切换，这种方式严重耗费资源，同时用户体验很差。

而单页应用，没有了令人诟病的页面频繁刷新，同时节约浏览器资源，路由响应比较及时，提升了用户的体验。前端组件化是将独立完整的功能模块封装到一个组件中，代码结构更加规范，便于代码维护，同时模块化后的组件可以在不同的场景中进行复用，极大地加快了迭代开发的速度。

1.3.5 Spring boot概述

Spring boot是Spring各个项目的集合，它能够被任意项目的构建系统所使用，Spring Boot致力于在蓬勃发展的快速应用开发领域(rapid application development)成为领导者。具有以下特点：创建独立的Spring应用程序；嵌入的Tomcat，无需部署WAR文件；简化Maven配置；自动配置Spring；提供生产就绪型功能，如指标，健康检查和外部配置；绝对没有代码生成并且对XML也没有配置要求。

1.3.6 Ajax概述

Ajax 即“Asynchronous JavaScript And XML”（异步 JavaScript 和 XML），是指一种创建交互式网页应用的网页开发技术。使用 JavaScript 向服务器提出请求并处理响应而不阻塞用户核心对象XMLHttpRequest。通过这个对象，您的 JavaScript 可在不重载页面的情况与 Web 服务器交换数据，即在不需要刷新页面的情况下，就可以产生局部刷新的效果。这个特点奠定了SPA的基础。

1.3.7 前后端分离概述

以前的开发模式，前端页面往往是使用的动态模板技术，如PHP、JSP，它们在生成页面的时候将数据填充上去，属于后端渲染。

现阶段由于HTML5的定稿，浏览器有了行业标准，现在已可以仅提供一个HTML骨架，将数据完全交于浏览器渲染，这样就没了页面的更替，因此应用的流畅程度有了极大的提升。数据交互方面由于全部使用的是异步加载，即使在网络情况不好的情况下，用户也能够看到页面，不会出现一片空白，提高了用户体验感。

在开发的时候，可以预先定义好API文档，前后端应用针对API文档进行并行开发，较于以前的线型开发模式，极大的提高了程序的开发效率。

第2章 设计方案



2.1 开发环境

操作系统：Windows10 x64、JDK1.8、Node.js 10。

开发工具：IntelliJ IDEA、Vue CLI。

服务器版本：Tomcat9。

数据库：MySQL5.7、Redis3.2.1。

浏览器：Chrome70。

2.2 软件结构图

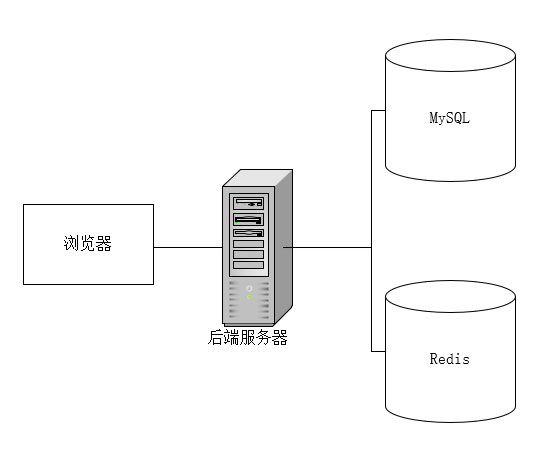


图2-1软件结构图

浏览器运行前端程序。

后端服务器运行后端程序、MySQL数据库、Redis数据库。

通过HTTP协议在前后端应用间进行数据交互。

2.3 功能调用时序图

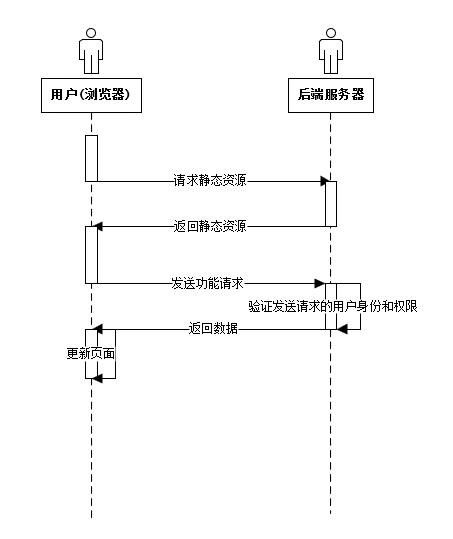


图2-2功能调用时序图

浏览器通过HTTP协议访问后端服务器资源，有功能资源和静态资源两种。

静态资源往往是不需要验证身份信息和权限的，后端服务器可直接将资源返回。

发送功能请求的时候，则必须验证身份信息和权限信息，通过后才能返回正确的数据。

第3章 功能实现



3.1 功能实现思路

本项目使用前后分离概念，后端服务器采用RPC风格服务器，前端服务为HTML5、CSS3、JavaScript实现的单页应用，通过HTTP协议与后端服务进行数据交互。前端使用的网络库为Axios。流程如下：

1. 用户通过浏览器访问页面。
2. 浏览器访问静态资源服务器，服务器返回整个单页应用，第一次加载时由于数据量较大且浏览器未做缓存，所以第一次加载会较慢，只要不清楚缓存，后面的页面加载和渲染将非常快。
3. 用户通过网页登录、获得身份Token。
4. 用户每次访问后端API时，需要携带Token，后端程序通过Token判断用户身份与权限进而返回相应的JSON数据和HTTP状态码。
5. 单页应用根据后端返回的JSON数据、HTTP状态码做出相应判断展现出数据或引导至错误页面。

整个过程中前后分离，仅通过HTTP协议交互数据，降低前后端耦合，使整个项目可以并行开发，大大提高开发速度与项目质量。

3.2 项目生成

3.2.1 后端开发流程

打开Spring官网的Spring boot 的初始化页面 <https://start.spring.io/>

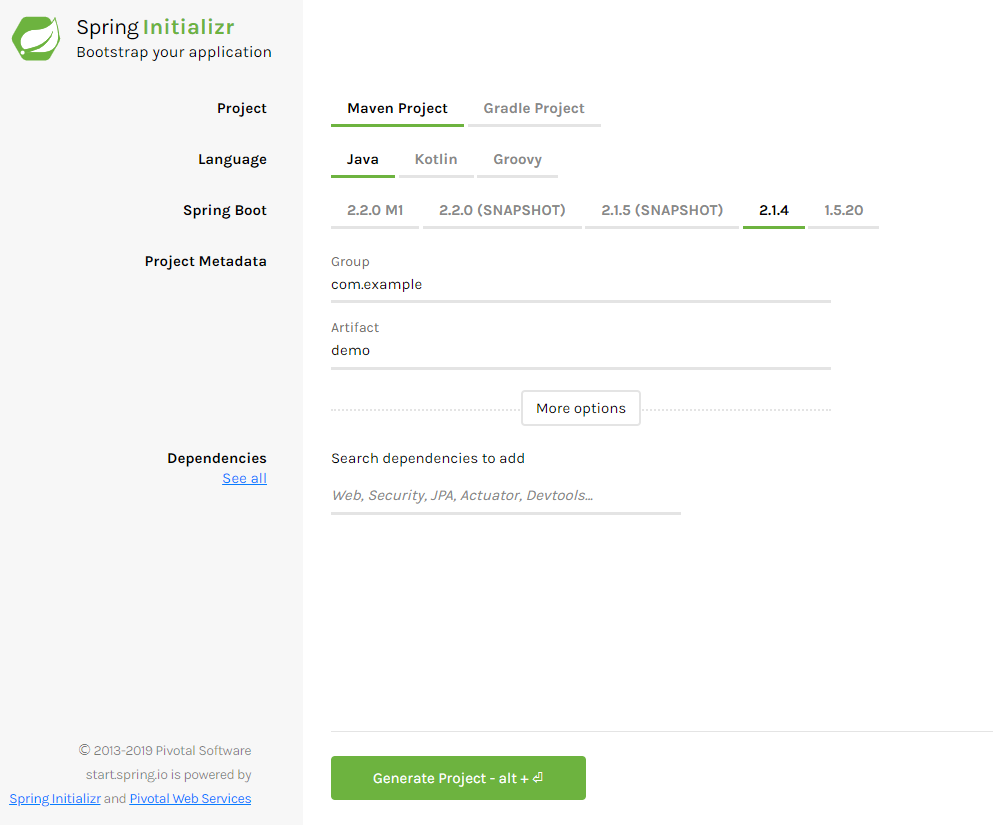


图3-1 Spring boot 初始化界面

填写Spring boot版本、构建工具、项目信息，以及开发所需要的功能模块

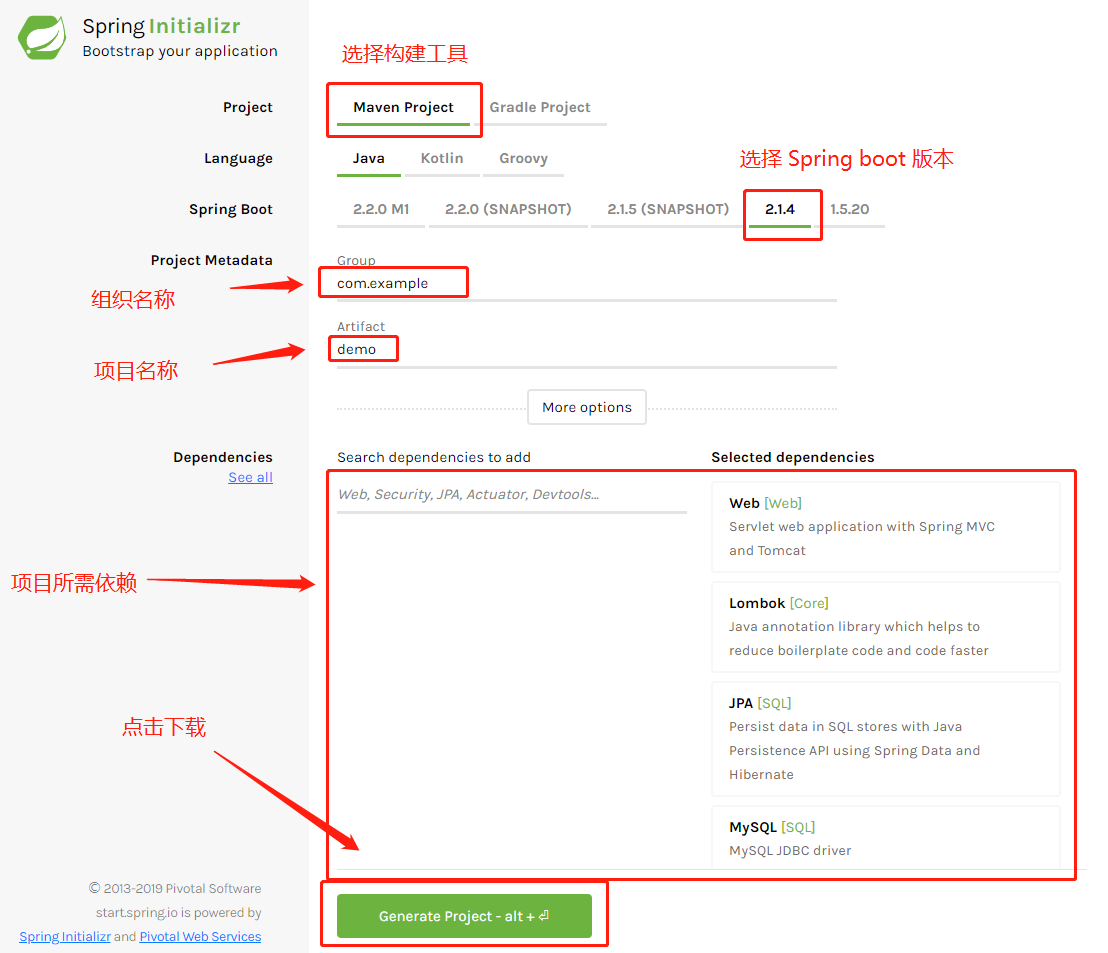


图3-2 Spring boot 填写信息界面

生成的Spring boot项目工程

src：Java代码文件和测试Java代码文件以及资源。

.mvn：Maven工程管理工具Jar包。

.gitignore：存放版本控制工具排除跟踪文件的正则表达式。

Pom.xml：Maven工程依赖管理文件。

其他文件：工具命令。

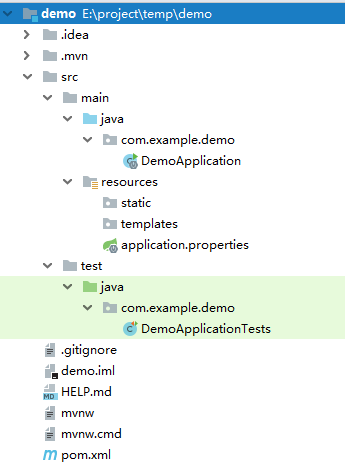


图3-3 工程结构图

3.2.2 前端开发流程

打开命令行，使用Vue CLI生成项目。

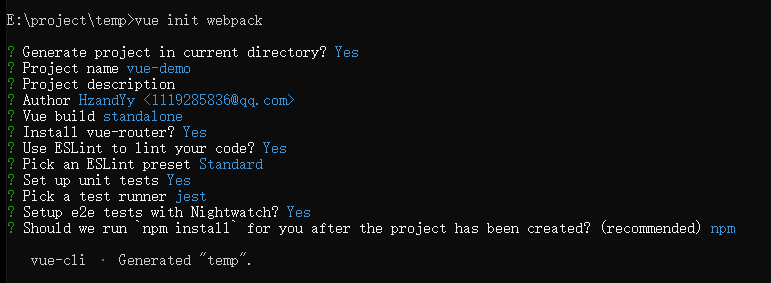


图3-4 Vue初始化图

Vue.CLI生成工程目录。

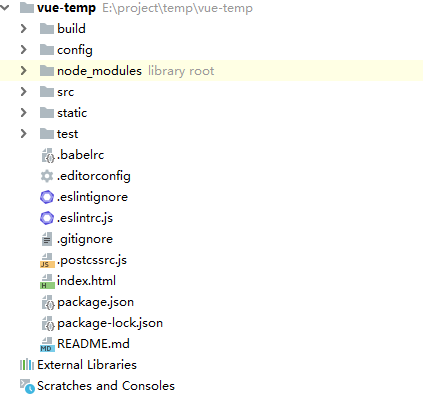


图3-5 Vue工程结构图

测试是否正确。

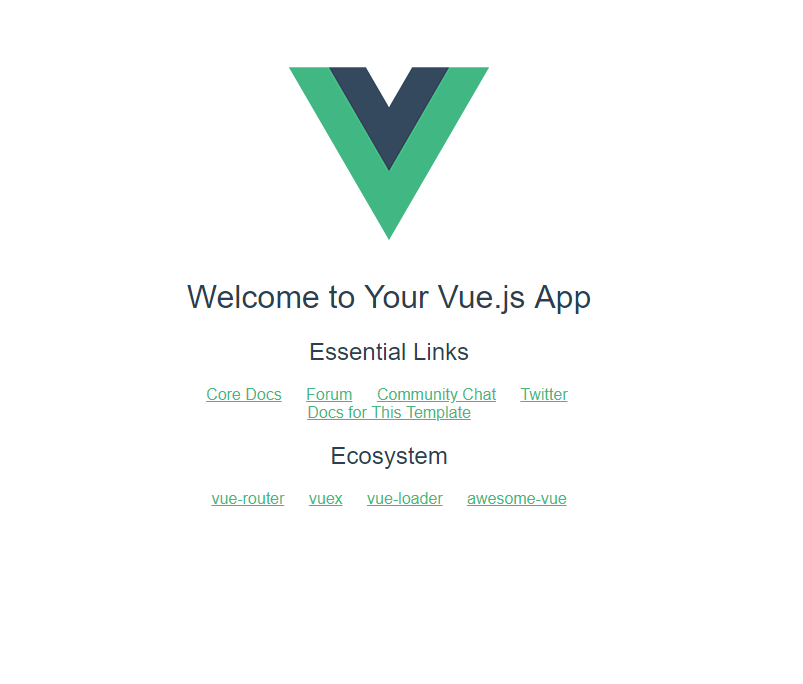


图3-6 Vue运行图

3.3 后端项目详细介绍

3.3.1 程序实现

后端应用收到HTTP请求后，首先通过认证授权的过滤器，当身份信息有错，数字签名被修改直接返回错误字段。验证通过后，通过DIspatchServlet在Controller层分发请求，在Controller层调用Service层暴露出来的功能接口，再通过Jackson包把处理后的数据格式化后返还给请求方。

Service层会暴露出功能接口给Controller层调用，包内含有私有实现，私有实现会使用容器中除了日志模块和控制器模块以外的所有组件。例如数据库操作会使用DAO层的实体，调用Repository数据库操作接口，抛出异常模块中的异常。

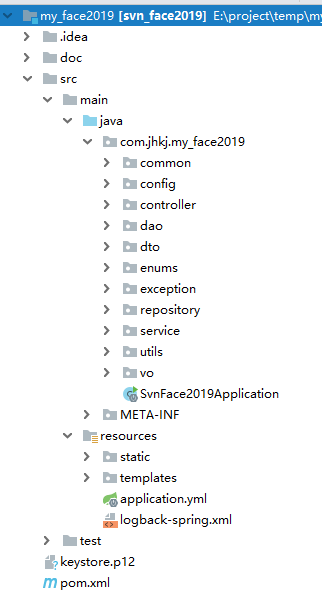


图3-7 实际项目工程结构图

3.3.2 日志模块

能通过对日志进行统计、分析、综合，就能有效地掌握网站运行状况，发现和排除错误原因，了解客户访问分布等，更好的加强系统的维护和管理。控制器本身的功能仅是对请求进行分发，如果添加了日志逻辑会造成代码耦合性增高，其次开发人员也会有阅读困难的问题。所以我们在本项目中使用Spring AOP的环绕增强为每个控制器添加日志功能。

AOP(Aspect Oriented Programming)，面向切面编程，此技术通过在程序编译阶段或使用动态代理的方法，为方法/函数添加新功能，进一步降低业务逻辑的耦合性。Spring AOP默认使用的基于JDK的动态代理技术。

以下是代码实现。

@Pointcut(“execution(\* com.jhkj.my\_face2019.controller.system.\*.\*(..))”)

public void eblog() {

}

@Around(value = “eblog()”)

public Object doAroundAdvice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {

HttpServletRequest request = ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getRequest();

Signature signature = proceedingJoinPoint.getSignature();

LogEntity logEntity = new LogEntity();

String ip = Tools.getClientIp();

String url = request.getRequestURL().toString();

log.info(“请求的URL：” + url);

log.info(signature.getDeclaringTypeName() + “的” + signature.getName() + “方法执行开始”);

long startTime = System.currentTimeMillis();

try {

Object object = proceedingJoinPoint.proceed();

long endTime = System.currentTimeMillis();

log.info(“程序运行时间：” + (endTime – startTime) + “ms”);

if (object instanceof String) {

String path = (String) object;

return path;

}

if (object instanceof HashMap) {

HashMap path = (HashMap) object;

return path;

}

ResponseResult result = (ResponseResult) object;

log.info(signature.getDeclaringTypeName() + “的” + signature.getName() + “方法执行结束”);

logEntity.setRequestURL(url);

logEntity.setOperatorIp(ip);

logEntity.setValueCode(result != null ? LogUtil.setValueCode(result.getStatus()) : “”);

setLogInfo(logEntity, signature);

return result;

} finally {

if (!StringUtils.isEmpty(logEntity.getLogType())) {

logRepository.save(logEntity);

}

}

}

3.3.3 缓存模块

持久化的数据如果每次都从数据库上去读，那样造成的IO负担是相当大的，极大的影响程序性能，故使用缓存机制。本项目使用Redis数据库作为缓存。

MySQL数据库，是一种基于磁盘的数据库，当面对频繁访问时，就会进行大量的I/O操作，这些操作相当的费时。面对这种快而准确的需求场景就不那么适用。

基于内存自己维护的队列，当数据存放在内存中，数据访问非常快，但是如果遇到断电问题，这些缓存就失效了，容错容灾机制不够好。

Redis是一种基于内存的数据库，它可以定时将数据持久化到磁盘中，它既能满足快速访问，也能解决断电容灾问题，非常适合使用缓存的这种需求场景。

以下是代码实现。

@Configuration

public class RedisCacheConfig extends CachingConfigurerSupport {

@Bean

@ConfigurationProperties(“spring.redis”)

public JedisConnectionFactory jedisConnectionFactory(){

return new JedisConnectionFactory();

}

@Bean

@Override

public KeyGenerator keyGenerator(){

return (target, method, params) -> {

StringBuilder sb = new StringBuilder();

sb.append(target.getClass().getName());

sb.append(method.getName());

for (Object obj : params) {

sb.append(obj.toString());

}

return sb.toString();

};

}

@Bean

public CacheManager cacheManager(RedisConnectionFactory connectionFactory){

RedisCacheManager rcm = RedisCacheManager.create(connectionFactory);

return rcm;

}

@Bean

public RedisTemplate<Object, Object> redisTemplate(JedisConnectionFactory jedisConnectionFactory) {

RedisTemplate<Object, Object> template = new RedisTemplate<>();

template.setConnectionFactory(jedisConnectionFactory);

template.setKeySerializer(new StringRedisSerializer());

template.setValueSerializer(new GenericJackson2JsonRedisSerializer());

template.afterPropertiesSet();

return template;

}

}

3.3.4 验证码模块

图片验证码生成器模块：如果没有验证码，如果不怀好意的使用者恶意登录或获取人员信息，故添加验证码，不法分子还要恶意登录难度就会高很多了。本项目使用的是Google下的kaptcha生成的验证码。

以下是代码实现。

@Component

public class KaptchaConfig {

@Bean

public DefaultKaptcha getDefaultKaptcha() {

com.google.code.kaptcha.impl.DefaultKaptcha defaultKaptcha = new com.google.code.kaptcha.impl.DefaultKaptcha();

Properties properties = new Properties();

properties.setProperty(“kaptcha.border”, “yes”);

properties.setProperty(“kaptcha.border.color”, “7,84,74”);

properties.setProperty(“kaptcha.textproducer.font.color”, “red”);

properties.setProperty(“kaptcha.image.width”, “180”);

properties.setProperty(“kaptcha.image.height”, “50”);

properties.setProperty(“kaptcha.textproducer.font.size”, “30”);

properties.setProperty(“kaptcha.textproducer.char.length”, “4”);

properties.setProperty(“kaptcha.textproducer.font.names”, “宋体,楷体,微软雅黑”);

Config config = new Config(properties);

defaultKaptcha.setConfig(config);

return defaultKaptcha;

}

}

3.3.5 认证授权模块

认证授权模块：一套系统是否安全，在认证授权是非常重要的。特别是像现在的RPC风格服务器、RESTFul风格服务器，都是基于Web API的服务器。如果没有认证授权，那么服务器资源将非常的危险。本项目使用的是Apache下的Shiro框架，认证是基于的用户名密码和Token两种认证模式，授权是以RBAC（Role-Based Access Control）的权限控制方式。

RBAC这种设计思想，它既足够灵活，也便于理解。它将程序按照一定的粒度划分成不同的资源，再让角色与资源进行绑定，用户再与角色进行绑定，有了角色这个中间件，开发人员和运维成员就能很便捷的对用户的权限进行管理。

以下是代码实现。

验证流程

JwtToken token = new JwtToken(null, dto.getUsername(), dto.getPassword());

Subject subject = SecurityUtils.getSubject();

try {

subject.login(token);

if (!subject.isAuthenticated()) {

return ResponseResult.e(ResponseCode.SIGN\_IN\_INPUT\_FAIL);

}

} catch (DisabledAccountException e) {

return ResponseResult.e(ResponseCode.SIGN\_IN\_INPUT\_FAIL);

} catch (Exception e) {

return ResponseResult.e(ResponseCode.SIGN\_IN\_FAIL);

}

Token配置

public class JwtToken implements AuthenticationToken {

private String token;

private String username;

private String password;

private String uid;

public JwtToken(String token, String username, String password) {

this.token = token;

this.username = username;

this.password = password;

}

@Override

public Object getPrincipal() {

return token;

}

@Override

public Object getCredentials() {

return token;

}

}

以下是认证流程和授权流程

public class MyRealm extends AuthorizingRealm {

@Autowired

private UserService userService;

@Autowired

private CacheManager cacheManager;

@Override

public oolean supports(AuthenticationToken token) {

return token instanceof JwtToken;

}

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

JwtToken token = (JwtToken) authenticationToken;

UserEntity user = new UserEntity();

//UsernamePasswordToken对象用来存放提交的登录信息

String username = token.getUsername() != null ? token.getUsername() : JwtUtil.getUsername(token.getToken());

try {

user = userService.findUserByUsername(username);

} catch (RequestException e) {

throw new DisabledAccountException(e.getMsg());

}

if (user == null) {

throw new DisabledAccountException(“用户不存在”);

}

if (user.getStatus() != 1) {

throw new DisabledAccountException(“用户账户已经锁定,暂时无法登陆”);

}

if (token.getUsername() == null) {

token.setUsername(user.getUsername());

}

String sign = JwtUtil.sign(user.getId(), user.getUsername(), user.getPassword());

if (token.getToken() == null) {

token.setToken(sign);

}

token.setUid(user.getId());

return new SimpleAuthenticationInfo(token, user.getPassword(), user.getId());

}

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {

log.info(“Shiro权限验证执行”);

JwtToken jwtToken = new JwtToken();

BeanUtils.copyProperties(principalCollection.getPrimaryPrincipal(), jwtToken);

if (jwtToken.getUsername() != null) {

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

UserEntity findUser = userService.findByUserName(jwtToken.getUsername(), true);

if (findUser != null) {

if (findUser.getRoles() != null) {

findUser.getRoles().forEach(roleEntity -> {

info.addRole(roleEntity.getName());

if (roleEntity.getResources() != null) {

roleEntity.getResources().forEach(v -> {

if (!””.equals(v.getPermission().trim())) {

info.addStringPermission(v.getPermission());

}

});

}

});

}

return info;

}

}

throw new DisabledAccountException(“用户信息异常，请重新登录！”);

}

public void clearAuthByUserId(String uid, Boolean author, Boolean out) {

Cache<Object, Object> cache = cacheManager

.getCache(MyRealm.class.getName() + “.authorizationCache”);

cache.remove(uid);

}

public void clearAuthByUserIdCollection(List<String> userList, Boolean author, Boolean out) {

Cache<Object, Object> cache = cacheManager

.getCache(MyRealm.class.getName() + “.authorizationCache”);

userList.forEach(cache::remove);

}

}

3.3.6 控制器模块

HTTP协议中，主机后的路径就代表资源的具体路径，为了区分不同的资源所以将不同的功能资源放到了不同的URI上面。本项目实际的控制器采用的是Spring MVC 项目的DispatchServlet，它可以把请求分发到相应的URL的控制器上。

以下是代码实现。

@Controller

@RequestMapping(“/demo”)

public class DemoController {

@RequestMapping(“/test”)

public String test() {

return “test”;

}

}

3.3.7 数据库设计

表3-1 设备表

|  |  |  |  |
| --- | --- | --- | --- |
| 设备表 | t\_equipment |  |  |
| 字段 | 类型 | 特殊 | 备注 |
| id | bingint | 自增 | 主键 |
| equipment\_number | varchar(255) | NOT NULL | 设备编号 |
| secretKey | varchar(255) | NOT NULL | 设备密钥 |
| status | int | NOT NULL | 状态 |
| model\_number | varchar(255) | NOT NULL | 设备型号 |
| software\_version | varchar(255) | NOT NULL | 软件版本 |
| equipment\_port | varchar(255) | NOT NULL | 设备端口号 |
| account | varchar(255) | NOT NULL | 登录账号 |
| password | varchar(255) | NOT NULL | 登录密码 |
| camera\_type | varchar(255) | NOT NULL | 摄像头类型 |
| coordinate | varchar(255) | NOT NULL | 坐标 |
| organization\_id | bigint | NOT NULL | 组织Id |
| gmt\_create | datetime | NOT NULL | 创建时间 |
| gmt\_modified | datetime | NOT NULL | 上次修改时间 |

表3-2 人脸图库表

|  |  |  |  |
| --- | --- | --- | --- |
| 人脸图库表 | t\_face\_user |  |  |
| 字段 | 类型 | 特殊 | 备注 |
| id | bingint | 自增 | 主键 |
| image\_status | int | NOT NULL | 人脸图片状态 |
| register\_status | int | NOT NULL | 图片注册状态 |
| thumbnail\_path | varchar(255) | NOT NULL | 微缩图路径 |
| real\_image\_path | varchar(255) | NOT NULL | 实际图路径 |
| characteristic\_file\_path | varchar(255) | NOT NULL | 特征文件路径 |
| map\_depot\_version | varchar(255) | NOT NULL | 人脸图库版本 |
| user\_id | bingint | NOT NULL | 员工Id |
| gmt\_create | datetime | NOT NULL | 创建时间 |
| gmt\_modified | datetime | NOT NULL | 上次修改时间 |

表3-3 日志表

|  |  |  |  |
| --- | --- | --- | --- |
| 日志表 | t\_log |  |  |
| 字段 | 类型 | 特殊 | 备注 |
| id | bingint | 自增 | 主键 |
| log\_type | int | NOT NULL | 日志类型 |
| log\_title | int | NOT NULL | 日志标题 |
| value\_code | varchar(255) | NOT NULL | 操作过后的信息 |
| request\_url | varchar(255) | NOT NULL | 请求的URL |
| operator | varchar(255) | NOT NULL | 操作人员账号 |
| operator\_ip | bigint | NOT NULL | 操作人员IP |
| gmt\_create | datetime | NOT NULL | 创建时间 |
| gmt\_modified | datetime | NOT NULL | 上次修改时间 |

表3-4 组织结构表

|  |  |  |  |
| --- | --- | --- | --- |
| 组织结构表 | t\_organization |  |  |
| 字段 | 类型 | 特殊 | 备注 |
| id | bingint | 自增 | 主键 |
| organization\_number | varchar(255) | NOT NULL | 组织编号 |
| name | varchar(255) | NOT NULL | 组织名称 |
| zip\_code | varchar(255) | NOT NULL | 组织邮编 |
| location | varchar(255) | NOT NULL | 组织地址 |
| description | varchar(255) | NOT NULL | 组织介绍 |
| sort\_weight | varchar(255) | NOT NULL | 排序权重 |
| admin\_name | varchar(255) | NOT NULL | 负责人姓名 |
| admin\_tel | varchar(255) | NOT NULL | 负责人电话 |
| status | int | NOT NULL | 组织状态 |
| type | int | NOT NULL | 组织类型 |
| level | int | NOT NULL | 组织层级 |
| path | varchar(255) | NOT NULL | 组织层级路径 |
| parent\_id | varchar(255) | NOT NULL | 父级组织ID |
| gmt\_create | datetime | NOT NULL | 创建时间 |
| gmt\_modified | datetime | NOT NULL | 上次修改时间 |

表3-5 用户表

|  |  |  |  |
| --- | --- | --- | --- |
| 用户表 | t\_user |  |  |
| 字段 | 类型 | 特殊 | 备注 |
| id | bigint | 主键 | 自增 |
| organization\_path | varchar(255) | NOT NULL | 所属组织 |
| username | varchar(255) | NOT NULL | 登录用户名 |
| name | varchar(255) | NOT NULL | 用户姓名 |
| password | varchar(255) | NOT NULL | 登陆密码 |
| tel | varchar(255) | NOT NULL | 电话 |
| positioin | varchar(255) | NOT NULL | 职位 |
| osition\_title | varchar(255) | NOT NULL | 岗位名称 |
| status | int | NOT NULL | 状态 |
| weight\_sort | int | NOT NULL | 排序权重 |
| face\_status | int | NOT NULL | 人脸信息注册状态 |
| gmt\_create | datetime | NOT NULL | 创建时间 |
| gmt\_modified | datetime | NOT NULL | 上次修改时间 |

表3-6 资源表

|  |  |  |  |
| --- | --- | --- | --- |
| 资源表 | t\_resource |  |  |
| 字段 | 类型 | 特殊 | 备注 |
| id | bigint | 主键 | 主键 |
| model\_name | varchar(255) | NOT NULL | 资源名称 |
| sort\_weight | int | NOT NULL | 排序权重 |
| url | varchar(255) | NOT NULL | 请求URL |
| permission | varchar(255) | NOT NULL | shiro验证权限 |
| parent\_id | bigint | NOT NULL | 父级id |
| type | int | NOT NULL | 资源类型 |
| verification | varchar(255) | NOT NULL | 是否开启权限验证 |
| remark | varchar(255) | NOT NULL | 备注 |
| gmt\_create | datetime | NOT NULL | 创建时间 |
| gmt\_modified | datetime | NOT NULL | 上次修改时间 |

表3-7 角色表

|  |  |  |  |
| --- | --- | --- | --- |
| 角色表 | t\_role |  |  |
| 字段 | 类型 | 特殊 | 备注 |
| id | bigint | 主键 | 主键 |
| name | varchar(255) | NOT NULL | 权限名称 |
| remark | varchar(255) | NOT NULL | 权限备注 |
| gmt\_create | datetime | NOT NULL | 创建时间 |
| gmt\_modified | datetime | NOT NULL | 上次修改时间 |

表3-8 资源权限表

|  |  |  |  |
| --- | --- | --- | --- |
| 资源权限表 | t\_resource\_role |  |  |
| 字段 | 类型 | 特殊 | 备注 |
| id | bigint | 主键 | 自增主键 |
| role\_id | bigint | 外键 | 角色表外键 |
| resource\_id | bigint | 外键 | 资源表外键 |

3.3.8 异常模块

Exception模块：程序时常会发生一些不被期望的事情，会阻止程序按照预期正常运行。当在单线程应用中，发生了异常，程序就会退出终止。Java中提供了优秀的解决方法，当程序异常发生时，可以按照程序预先设定好的异常处理逻辑，针对性地处理异常，让程序尽最大可能恢复正常并继续执行，且保持代码的清晰。

以下是代码实现。

// 创建一个自定义异常

public class EntityNotFoundException extends Exception {

private String message;

public EntityNotFoundException(String message) {

super(message);

}

}

// 异常处理

try(InputStream inputStream = new FileInputStream(“/filepath/filename”)) {

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

3.3.9 数据库操作模块

Repository接口是Spring-data项目下面的一个主要接口，Spring-data的目的是简化数据库访问,并支持云服务的开源框架。本项目主要用于处理关系型数据库中的数据库操作。使用的使它项目下的Spring-data-JPA模块，它是基于Hibernate这种ORM框架实现的。

因为本项目的业务关系并不复杂，而且需求变化大，所以使用原生SQL语言进行开发，成本较高。在这种业务场景下，使用ORM框架比较适合。

以下是代码实现。

@Repository

public interface UserRepository extends JpaRepository<UserEntity,String> {

@Query(“update UserEntity c set c.status = ?1 where c.id in ?2”)

void statusChange(Integer status, List<String> id);

@Query(“update UserEntity c set c.status = ?1 where c.id = ?2”)

void statusChangeSig(Integer status,String id);

@Query(“delete from UserEntity s where s.id in (?1)”)

void deleteBatch(String[] ids);

@Query(“update UserEntity c set c.organizationPath = ?1 where c.id = ?2”)

void updatePathById(String path, String id);

@Query(“select e from UserEntity e where e.userType = ‘0’ order by e.weightSort asc “)

List<UserEntity> findNormalUser();

}

3.3.10 Service模块

为了体现面向对象的思想，以及提高程序的扩展性以及封装性，为此，提供了service模块，将所有的实际功能逻辑都提供出一个接口方法，然后自己实现此接口。依靠Spring的依赖注入，对外程序只提供接口，实现类由Spring boot注入。这样的好处是如果修改了实现逻辑，只需要将注入的实现类修改即可，不用修改非service层的代码。体现了优质代码的高内聚、低耦合的特点。

以下是代码实现。

// 接口声明

public interface RoleResourceService{

List<ResourceEntity> findAllResourceByRoleId(String rid);

List<RoleResourceEntity> findAll();

List<RoleResourceEntity> findAllByRid(String rid);

}

// 接口实现

public class RoleResourceServiceImpl implements RoleResourceService {

@Autowired

private RoleResourceRepository roleResourceRepository;

@Autowired

private ResourceService resourceService;

@Override

public List<ResourceEntity> findAllResourceByRoleId(String rid) {

List<RoleResourceEntity> roles = roleResourceRepository.findAllByRid(rid);

if(roles != null){

List<String> pids = new ArrayList<>();

//lambda表达式遍历rps-r指代rps

List<ResourceEntity> resources = new ArrayList<>();

for(RoleResourceEntity role:roles){

pids.add(role.getPid());

}

for(String pid:pids){

resources.add(resourceService.findById(pid));

}

//rps.forEach(r->pids.add(r.getPid()));

if(pids.size() == 0){

return null;

}

return resources;

}

return null;

}

@Override

public List<RoleResourceEntity> findAll() {

List<RoleResourceEntity> roleResources = roleResourceRepository.findAll();

return roleResources;

}

@Override

public List<RoleResourceEntity> findAllByRid(String rid) {

return roleResourceRepository.findAllByRid(rid);

}

}

3.4 前端项目详细介绍

3.4.1 SPA介绍

本项目采用的SPA，单页应用， 就是只有一张HTML页面的应用。此应用页面不会发生刷新，它的速度和流畅会和原生应用相差不多，所有的数据交互、页面更新全部都由Ajax完成。所有的逻辑都由JavaScript来控制，因此程序的复杂性相较于以前那种Web模式就不是一个级别的了。

3.4.2 模块化

随着web2.0时代的到来，Ajax技术得到广泛应用，jQuery等前端库层出不穷，前端代码日益膨胀，此时在JS方面就会考虑使用模块化规范去管理。所以我们可以将复杂的程序依据一定的规则(规范)封装成几个块(文件), 并进行组合在一起，块的内部数据与实现是私有的, 只是向外部暴露一些接口(方法)与外部其它模块通信。这个和面向对象的编程思想有点异曲同工之妙。

但由于JavaScript的语法特性，最初的模块化命名空间污染严重，复用性、维护性差，直到有了CommonJS模块规范，每个文件就是一个模块，有自己的作用域。在一个文件里面定义的变量、函数、类，都是私有的，对其他文件不可见。这样就不会发生命名污染，且提高了复用性和维护性，而且在运行时，仅会第一次加载，之后会缓存，再次运行时就会直接读取缓存结果，想让模块在此运行则必须清除缓存。这样也就又一次提高了SPA的运行效率和流畅感。

3.4.3 Vue

Vue是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

组件：如果说CommonJs是JavaScript的一套模块化规范，那么组件就是SPA的一个模块化规范，只不过粒度是由开发人员所决定，可以是整个功能，也可能只是一个按钮。本次项目的粒度只达到了功能模块。

以下是代码实现。

<template>

<div class="login\_wrapper">

<div class="login">

<el-form :model="formLogin">

<el-form-item>

<h2 class="title">后台管理系统</h2>

</el-form-item>

<el-form-item>

<el-input v-model="formLogin.loginName" placeholder="账号"></el-input>

</el-form-item>

<el-form-item>

<el-input type="password" v-model="formLogin.password" placeholder="密码"></el-input>

</el-form-item>

<el-form-item>

<el-input style="width: 50%;display: inline-block" v-model="formLogin.verification\_code"

class="form-control" id="focusedInput" type="text" placeholder="请输入验证码"></el-input>

<img style="display: inline;width: 130px;float: right" v-bind:src="imgUrl" class="img-code"

id="kaptchaImage" @click="change()" title="点击换一张"/>

</el-form-item>

<el-form-item>

<el-button type="primary" @click="login">登录</el-button>

<span v-show="this.errorInfo.isShowError" class='error'>

{{this.errorInfo.text}}

</span>

</el-form-item>

</el-form>

</div>

</div>

</template>

3.4.4 路由控制

以前那种网站，浏览器所指URL是真实的后端资源路径，到了SPA里，路由起一个页面导航，数据传参的效果。实际访问的URL全部由JavaScript来决定。由于本项目做了权限访问控制，会根据用户的权限分发给不同的界面，变成了动态路由。

本项目由于权限是动态管理的，所以对于路由控制也相应的会使用动态控制。

以下是代码实现。

function lazy(code) {

//return resolve => require(["@/components/" + name + ".vue"], resolve)

return getComponetByPath('/' + code);

}

function getComponetByPath(path) {

for (var i = 0; i < dynamicRouter.length; i++) {

var route = dynamicRouter[i];

if (route.path == path) {

return route.component;

}

}

}

const dynamicRouter = [

{

name: 'index',

path: '/index',

component: resolve => require(['@/components/index.vue'], resolve),

meta: {

title: '系统首页'

}

}

// 以下均为静态定义，在此省略

]

export { lazy }

第4章 调试与实现



4.1 调试中遇到的重点与难点

4.1.1 调试重点

1. 功能调试。
   1. 功能是否正确实现。
2. 健壮性调试。
   1. 格式错误的数据是否存在提示。
   2. 是否存在注入攻击。
   3. 通过API访问是否有认证。
3. 用户体验感。
   1. 对于用户的错误操作，是否有能够接受的提示。
   2. 用户操作逻辑是否有问题。

4.1.2 调试难点

1. 对HTTP协议还不够熟悉，时常把携带数据放错位置。
2. 时常会出现Web页面、服务器日志、数据库数据出现乱码。
3. 服务打不开，端口被占用。

4.2 解决方案

1. 查阅HTTP协议文档，将数据放在正确的地方。
2. 为服务器添加字符集处理过滤器、前端的请求字符集统一设定请求头。
3. 查看端口被什么程序占用，杀死该程序。

4.3 实现展示

4.3.1 登录页面



图4-1 登录页面

4.3.2 组织结构管理页面



图4-2 组织结构管理页面

4.3.3 设备管理页面

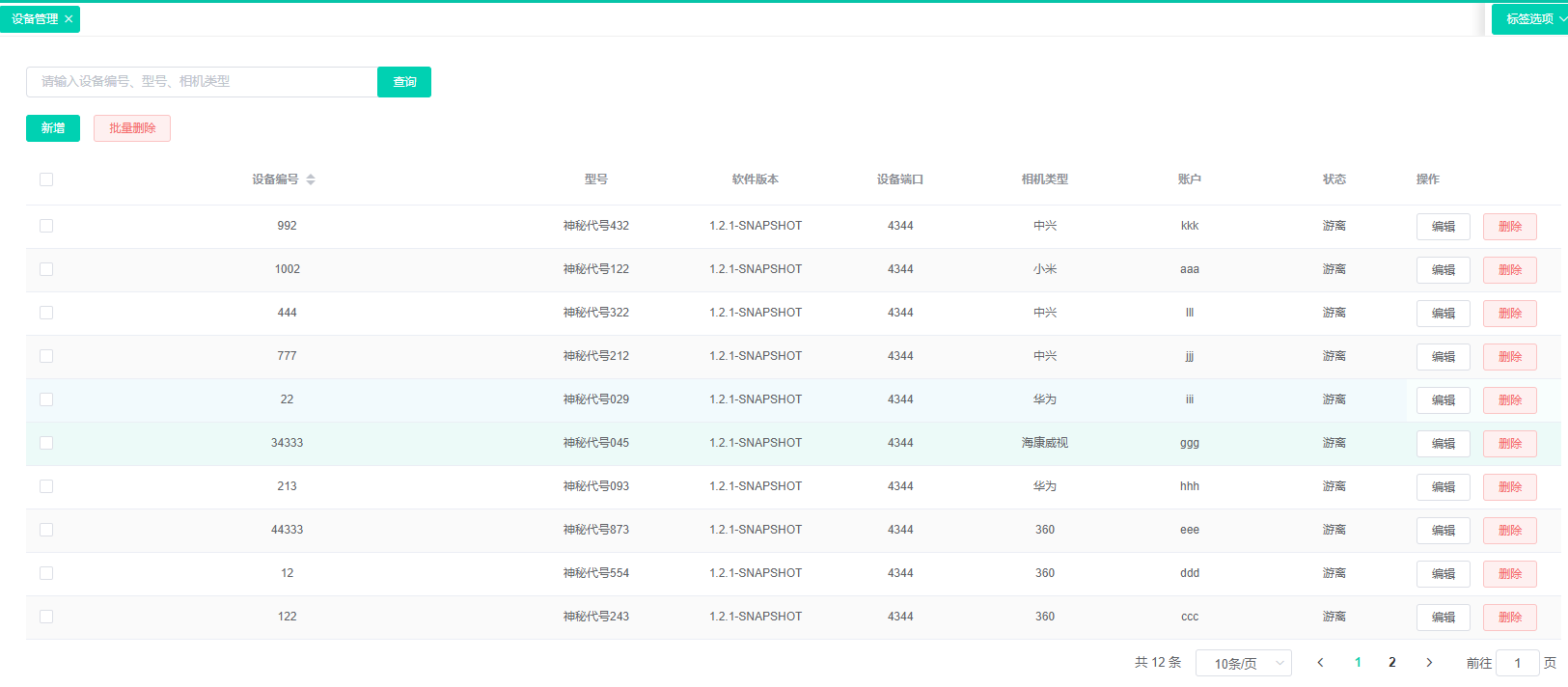


图4-3 设备管理页面

4.3.4 人脸信息管理页面



图4-4 人脸信息管理页面

4.3.5 权限管理页面



图4-5 权限管理页面

4.3.6 日志管理

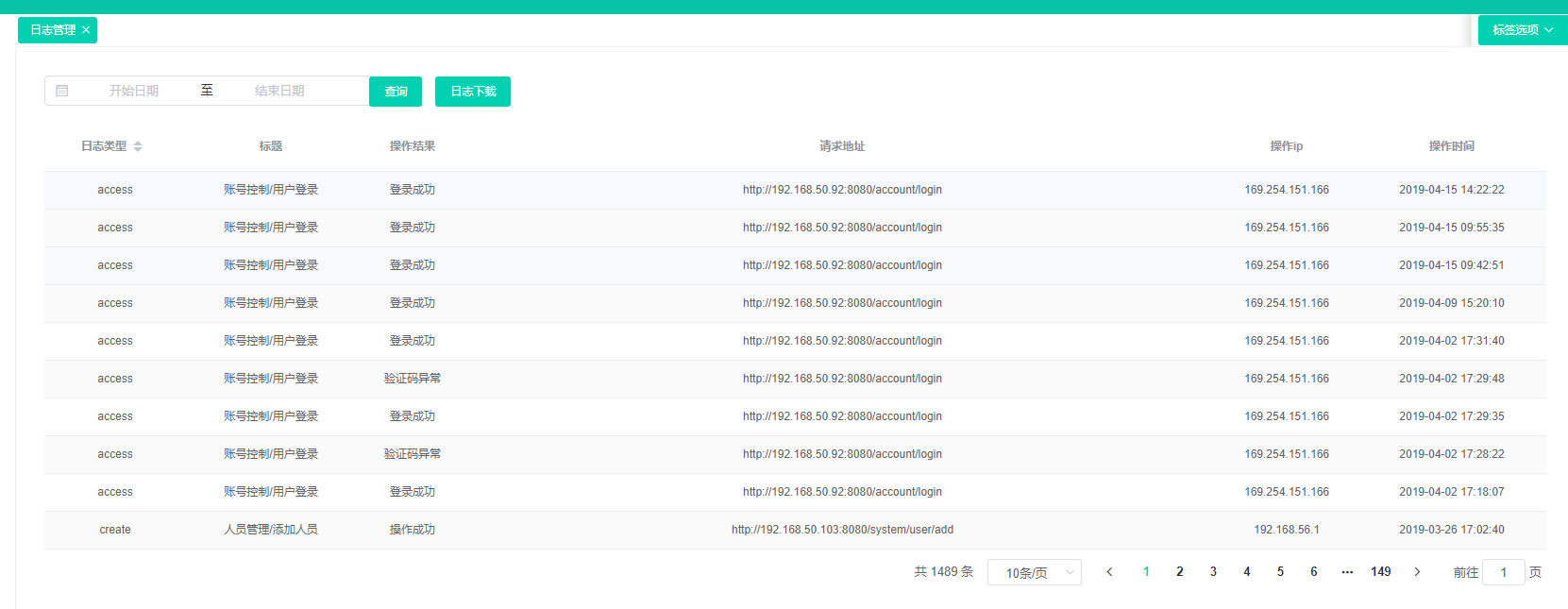


图4-6 日志管理页面

4.3.7 资源结构管理页面



图4-7 资源结构管理页面

第5章 总结

本项目选用技术有Spring boot、Vue.js、Axios，这些框架技术都基础技术在我们看来都对底层技术做了很好的封装。但只是使用这些技术是非常简单的，所以我们应该看到更深层次的东西，这些框架究竟是怎么实现的。

Spring boot，本质上面来讲就是一套Spring框架的全家桶，本项目有使用Spring MVC、Spring AOP、Spring Data JPA。Spring MVC，本质上面是对Java Web那一套体系的封装，使用一个DispatchServlet的类，分发请求得到的请求，个人认为这样做的好处是可以将除了请求这一个需要遵守Java Web的规范外，其他的实现全部都可以由Spring MVC来定义，这样做对修改关闭、对扩展开放就基本上能满足这个思想了。Spring AOP，面向切面编程，使用代理的方法针对实际方法逻辑做了很好的分层，让每个方法只做自己的事情，做到了高内聚、低耦合，极大的提高了程序代码的质量。Spring Data JPA，底层默认使用的Hibernate框架实现，众所周知，Hibernate学习门槛较高，理念繁多、优化困难，但Spring Data JPA大大的简化了这些，内部就把Hibernate的复杂性封装并实现了，大大的降低了使用者的门槛，让使用者仅仅需要很基础的Java知识和SQL知识就能实现自己想要的功能。

Vue.js，非常的简单易用，（本人主攻Angular2+），如果只需要使用它的一些简单的功能，只需要在<script>标签里面引用它的JS类库即可。做单页应用，只需使用脚手架搭建出项目，创建.vue文件即可，相较于本人之前使用的Angular2+，它没了那么多条条款款、规规矩矩。但Vue和Angular2+相比较后，个人发现了一些问题，可能是因为我是做后端的开发人员，深知一个好的工程项目是必须有那么一些规范的，不然也不会每个公司都有自己的那一套代码规范，前端的技术逐渐由直接引用的CSS、JavaScript，慢慢的变成了现在要使用的打包工具打包，逐渐的工程化了起来，而且现在的Web前端技术它的模式逐渐的向桌面应用靠齐，只不过一个是直接运行在操作系统上的，一个是运行在浏览器上的，如果我们把浏览器想象成一个虚拟机，那么现在的前端技术就和桌面应用一样了。所以前端技术需要工程化的问题，迫在眉睫。

Axios类库，本质上一个HTTP请求类库，要想使用好这个类库，不能只知道GET、POST、DELETE、PUT、怎么传输数据，更该学习一下HTTP事务、代理、网关、响应头、认证方法、不同的HTTP版本之间有什么区别。、

在功能设计上面，我们不能为了实现功能而去做功能。我们更需要的是去更好的了解需求，因为本质上面，我们做应用是为客户做应用，用户用着好用、程序能够正确运行才是用户最需要的。所以我们的每一条提示语句、每一个弹出框，要让使用者觉得实际有效，这样才能提高用户体验感。当有了用户需求，我们实现功能的时候，最好能够画一画UML图，因为功能是非常好实现的，但功能实现的好与不好才是关键，画UML图可以让我们知道整体程序的流程走向，功能点覆盖的是否完全，反馈是否合理。

在本项目中，遇到的问题还是挺多的，有技术使用上的、操作系统和程序环境版本的、有字符编码的。这些问题给了我们很多解决问题的思路，如果是一个相对大众的问题，去论坛博客查找解决方案，如果是一个不那么大众的问题，就去官方文档查找解决方案。这样做以后，大大的提高了我们解决问题的效率。

本次毕业设计，使我学习到了很多，加深了对技术的了解认识，此次毕业设计使我受益颇多。

参考文献

[1] “Spring-boot reference guide”，https://docs.spring.io/springboot/docs/2.1.4.RELEASE/reference/htmlsingle/

[2] “Vue reference guide”, https://cn.vuejs.org/v2/guide/

[3] “axios reference guide”，http://www.axios-js.com/zh-cn/docs/

[4] (德)Christian Bauer著. Hibernate实战(第二版).蒲成译. 清华大学出版社, 2016.9

[5] (美) David Gourley 著. HTTP权威指南. 陈涓. 赵振平译. 人民邮电出版社,2012.9

[6] (美) 尼古拉斯·泽卡斯著. JavaScript高级程序设计(第3版). 李松峰, 曹力译, 人民邮电出版社, 2012.3

[7] Baron Schwartz, Peter Zaitsev, Vadim Tkachenko著. 高性能MySQL(第3版). 宁海元, 周振兴, 彭立勋等译. 电子工业出版社, 2013.5

致谢

四年的大学学习生活在即将划上一个句号，而于我的人生来说却仅仅只是一个逗号，我将面对新的征程的开始。本研究及论文是在我的导师徐艳的亲切关怀和耐心的指导下完成的。伟人、名人固然为我所崇拜，可是我更迫切地想要把我的敬意献给一位平凡的人，我的导师徐艳老师。也许我不是您最出色的学生，但您却是我所最尊敬的老师。您是如此的治学严谨，学识渊博，视野广阔，思想深刻，您用心为我营造一种良好的学术氛围，让我的论文更加的严谨。至此论文付梓之际，我的心情无法保持平静，从开始选择课题到论文的顺利答辩，有无数可敬的师长、朋友给了我很多的帮助，在这里请您接受我诚挚的谢意! 最后，再次对那些在论文完成过程中，关心、帮助我的老师、同学和朋友们表示衷心地感谢！