摘要

随着信息技术的发展，现在已经由工业时代进入了信息时代。从Web1.0、Web2.0直到现在的Web3.0时代，从最开始的只有PC机能够浏览信息直到现在的手机、平板电脑、专用终端都能够浏览信息，进行数据交互。Web网站在近几年也进入了高速发展期，各种开发语言、框架、模式较之以前有了很大的变化。

此次毕业设计采用Spring-boot、Vue.js、Ajax技术，实现监控公司组织结构管理系统，具有一个RPC服务风格的服务器，以及MVVM风格的前端Web静态网站。

关键词：前后分离、Spring-boot、Vue.js、Ajax。

ABSTRACT

With the development of information technology, the industrial age has now entered the information age.From Web1.0 and Web2.0 to the present Web3.0 era, from the very beginning only PC can browse information to the present mobile phone, tablet computer and dedicated terminal can browse information and conduct data interaction.Web sites in recent years have also entered a period of rapid development, a variety of development language, framework, pattern than before has changed a lot.

This graduation project USES spring-boot, Vue. Js, Ajax technology, to achieve the monitoring of the company's organizational structure management system, with an RPC service style server, as well as MVVM style front-end Web static website.

Key Words: before and after the separation, spring-boot, Vue. Js, Ajax.

目录

1. 课题整体框架。
   1. 课题任务

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。

权限控制：针对不同职位的员工具有不同的操作权限。

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。

* 1. 课题要求

操作逻辑人性化，数据结构正确，程序功能完善，程序健壮性好。

* 1. 研究意义
     1. 研究背景

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。

后端服务Java语言语法一直以厚重繁杂，官方标准EJB实现起来复杂难懂，后来出现的Spring框架缓解了EJB的复杂，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，现在大大的缓解了Java开发服务器应用繁杂厚重的困难。

* + 1. 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. 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. 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. Spring boot概述

Spring boot是一些库的集合，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：

1. 创建独立的Spring应用程序。
2. 嵌入的Tomcat，无需部署WAR文件。
3. 简化Maven配置。
4. 自动配置Spring。
5. 提供生产就绪型功能，如指标，健康检查和外部配置。
6. 绝对没有代码生成并且对XML也没有配置要求。
   * 1. 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. 设计方案
   1. 开发环境

操作系统：Windows10 x64、JDK1.8、Node.js 10。

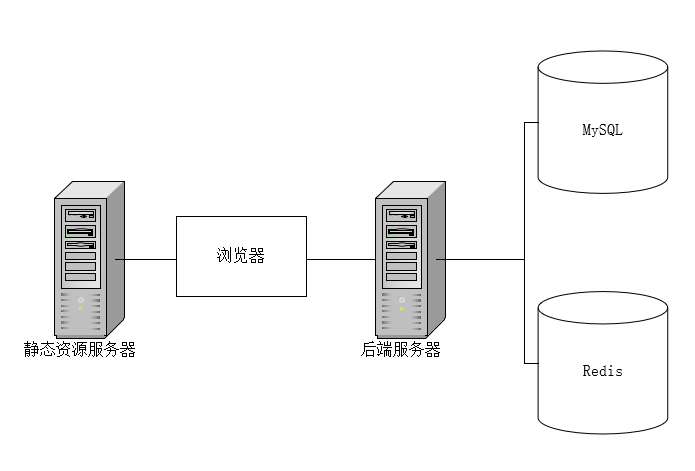
开发工具：IntelliJ IDEA、Vue CLI。

服务器版本：Tomcat9。

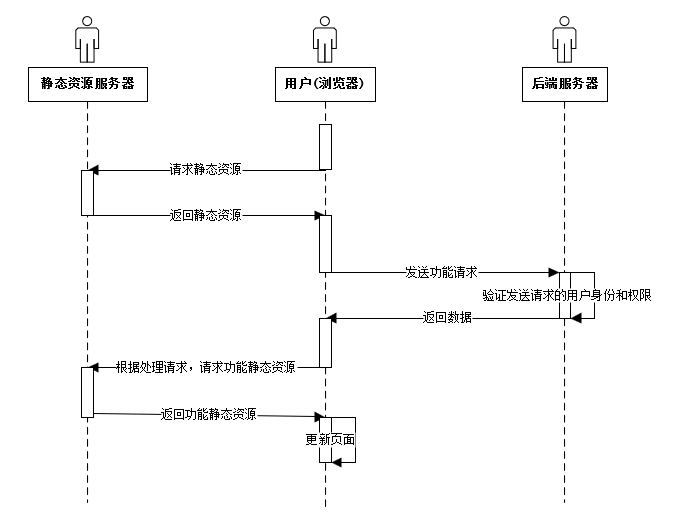
数据库：MySQL5.7、Redis3.2.1。

浏览器：Chrome

* 1. 软件结构图



* 1. 功能时序图



1. 详细功能设计
   1. 功能实现思路

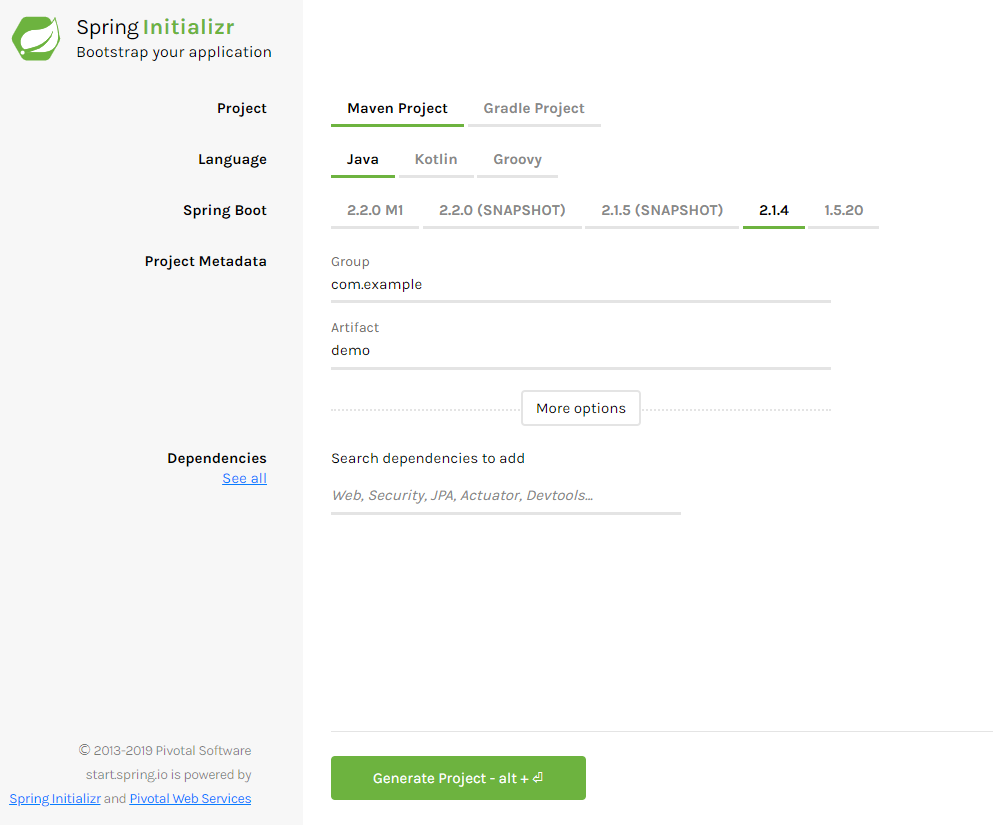
本项目使用前后分离概念，后端服务器采用RPC风格服务器，前端服务为HTML5、CSS3、JavaScript实现的单页应用，通过HTTP协议与后端服务进行数据交互。前端使用的网络库为Axios。流程如下：

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。

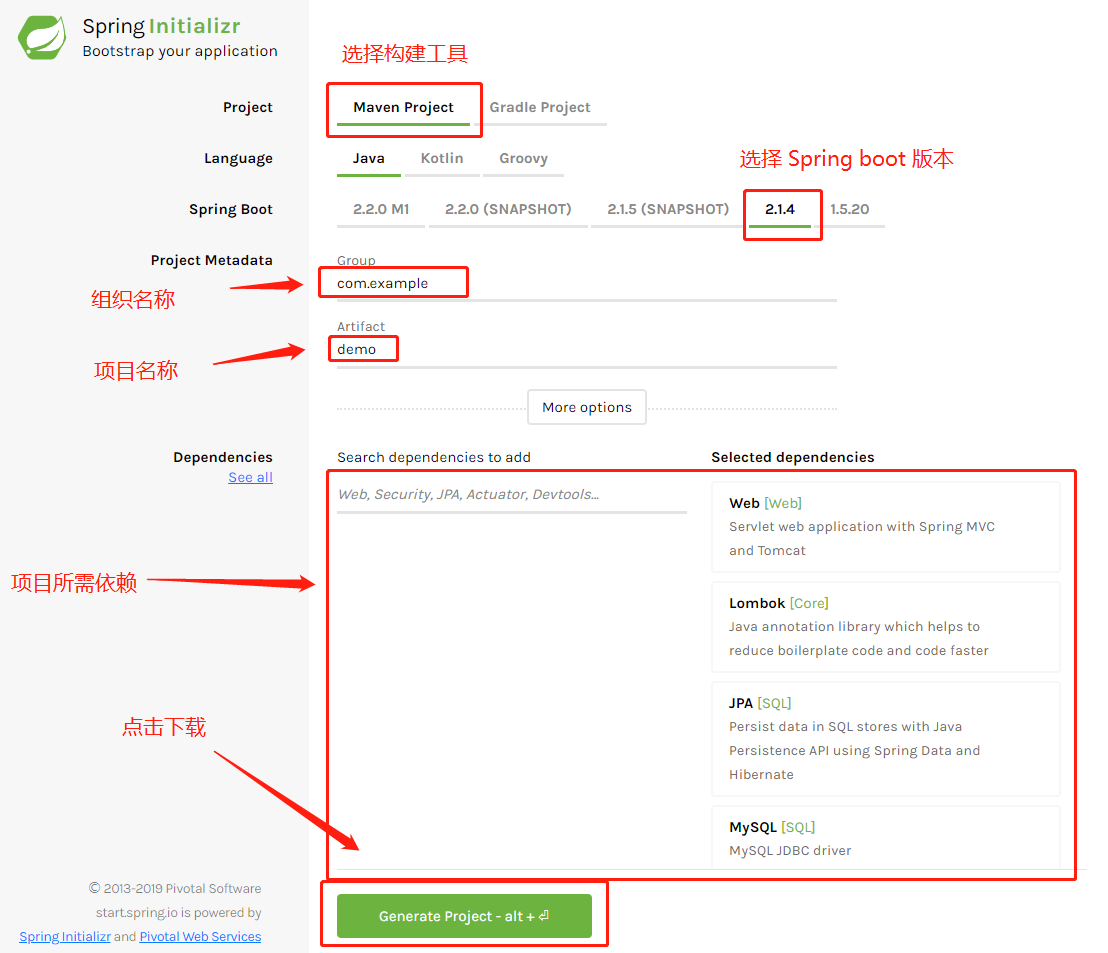
整个过程中前后分离，仅通过HTTP协议交互数据，降低前后端耦合，使整个项目可以并行开发，大大提高开发速度与项目质量。

* 1. 项目生成
     1. 后端开发流程

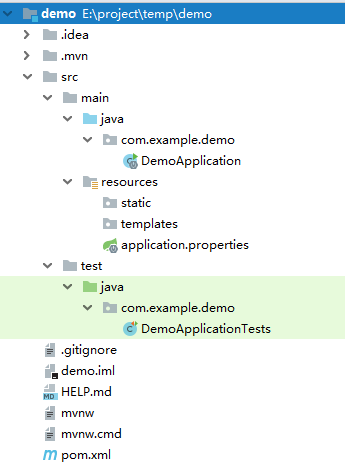
1. 打开Spring官网的Spring boot 的初始化页面 <https://start.spring.io/>



1. 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

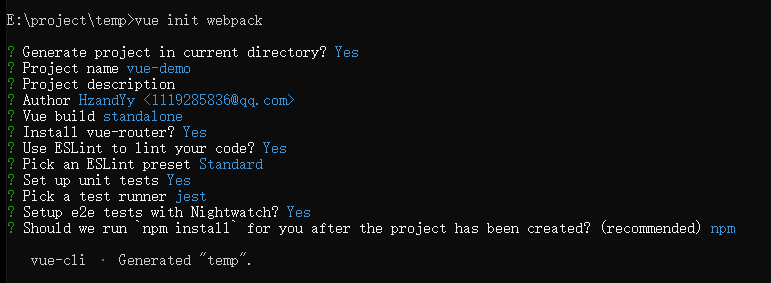


1. 生成的Spring boot项目工程
   1. src：Java代码文件和测试Java代码文件以及资源。
   2. .mvn：Maven工程管理工具Jar包。
   3. .gitignore：存放版本控制工具排除跟踪文件的正则表达式。
   4. Pom.xml：Maven工程依赖管理文件。

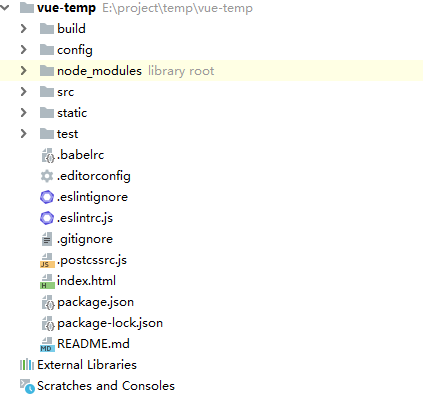


* + 1. 前端开发流程

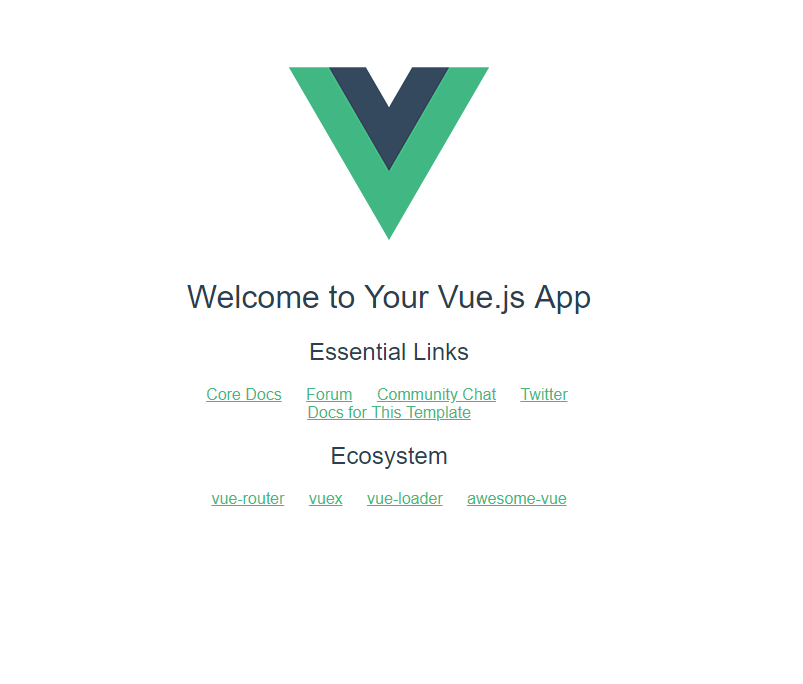
1. 打开命令行，使用Vue CLI生成项目。



1. 生成工程目录。

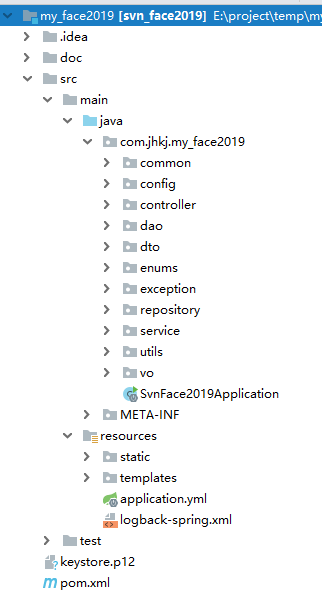


1. 测试是否正确。



* 1. 后端项目详细介绍

1. 工程目录结构



* 1. common包：存放AOP程序。
  2. config包：存放用户缓存、Token过滤器、默认Token、图片验证码生成器、权限框架认证授权配置。
  3. controller包：存放功能控制器类。
  4. dao包：存放数据库实体、数据库数据信息类。
  5. dto包：存放API请求参数类与控制器与服务层之间的数据交互类。
  6. enums包：存放各种状态位枚举。
  7. exception包：存放自定义异常类。
  8. repository包：存放数据库操作接口。
  9. service包：存放功能服务接口和功能服务接口实现类。
  10. utils包：存放全局功能类。
  11. vo包：存放HTTP响应数据类。
  12. application.yml：全局配置类。
  13. keystore.p12：HTTPS加密密钥。
  14. pom.xml：依赖管理文件。

1. 模块简介
   1. 依赖项目如下

// Spring boot 核心包

spring-boot-starter-aop

spring-boot-starter-tomcat

spring-boot-starter-data-jpa

spring-boot-starter-data-redis

spring-boot-devtools

spring-boot-starter-test

springfox-swagger2

springfox-swagger-ui

pagehelper

// shiro 认证授权包

shiro-all

shiro-redis

// 数据持久化包

jedis

mysql-connector-java

//工具包

zip4j

poi

poi-ooxml

guava

kaptcha

java-jwt

fastjson

commons-io

commons-lang3

commons-fileupload

encrypt-body-spring-boot-starter

lombok

* 1. 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。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。

// 定义切点

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

public void webLog() {

}

// 定义切面和结合切点以及增强逻辑实现

@Around(value = "webLog()")

public Object doAroundAdvice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {

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

Signature signature = proceedingJoinPoint.getSignature();

LogEntity logEntity = new LogEntity();

//获取ip

String ip = Tools.getClientIp();

//获取url

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 {

//获取执行方法的返回值

Object object = proceedingJoinPoint.proceed();

long endTime = System.currentTimeMillis();

log.info("程序运行时间：" + (endTime - startTime) + "ms");

//如果是执行的页面跳转方法，返回值为String不为JsonData

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");

//存储日志信息

logEntity.setRequestURL(url);

logEntity.setOperatorIp(ip);

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

// 以功能为划分进行日志存储

setLogInfo(logEntity, signature);

return result;

} finally {

//只有当日志类型不为空时，才进行记录

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

logRepository.save(logEntity);

}

}

}

* 1. 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。

// 让Spring boot 读取这个配置

@Configuration

public class RedisCacheConfig extends CachingConfigurerSupport {

// 声明为一个Spring 的Bean 且读取application.yml的spring.redis 配置文件

// 得到一个与Redis的连接

@Bean

@ConfigurationProperties("spring.redis")

public JedisConnectionFactory jedisConnectionFactory(){

return new JedisConnectionFactory();

}

// 配置Redis Key的生成策略

@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();

};

}

// 管理缓存

@Bean

public CacheManager cacheManager(RedisConnectionFactory connectionFactory){

//此处可以设置缓存过期时间

RedisCacheManager rcm = RedisCacheManager.create(connectionFactory);

return rcm;

}

// 配置Redis 模板 进行redis操作 与JDBC Template类似

@Bean

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

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

// 设置数据库连接

template.setConnectionFactory(jedisConnectionFactory);

// 设置key 序列化方案

template.setKeySerializer(new StringRedisSerializer());

// 设置 value 序列化方案

template.setValueSerializer(new GenericJackson2JsonRedisSerializer());

// 初始化操作

template.afterPropertiesSet();

return template;

}

}

* 1. 跨域处理模块：由于浏览器对安全有考虑，浏览器对JavaScript施加了跨域限制。所以在服务端为响应添加了跨域支持。默认是没有的，需要自行添加，故添加处理跨域的过滤器。

@Component

public class CorsFilter implements Filter {

@Override

public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {

HttpServletResponse response = (HttpServletResponse) servletResponse;

HttpServletRequest request = (HttpServletRequest) servletRequest;

String origin = request.getHeader("Origin");

response.setHeader("Access-Control-Allow-Origin", origin);

response.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS, DELETE");

response.setHeader("Access-Control-Max-Age", "3600");

response.setHeader("Access-Control-Allow-Headers", "Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");

response.setHeader("Access-Control-Allow-Credentials", "true");

String method = request.getMethod();

if (method.equalsIgnoreCase("OPTIONS")) {

servletResponse.getOutputStream().write("Success".getBytes("utf-8"));

} else {

filterChain.doFilter(servletRequest, servletResponse);

}

}

}

* 1. 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。

@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();

// 图片边框

properties.setProperty("kaptcha.border", "yes");

// 边框颜色,国网绿

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

// 字体颜色

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

// 图片宽

properties.setProperty("kaptcha.image.width", "180");

// 图片高

properties.setProperty("kaptcha.image.height", "50");

// 字体大小

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

// 验证码长度

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

// 字体

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

Config config = new Config(properties);

defaultKaptcha.setConfig(config);

return defaultKaptcha;

}

}

* 1. 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（基于角色的权限访问控制）的授权方式。

下面是一段验证逻辑

// 封装成Shiro能够验证的Token

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

// 得到用户主体

Subject subject = SecurityUtils.getSubject();

try {

// 登录,即身份验证

// Subject.login(token) 会根据配置的属性 验证用户身份以及授权给用户 并返回主体认证、授权信息

subject.login(token);

// 根据认证信息 确定是否是真正的用户

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);

}

以下是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;

/\*\*

\* 验证此Token 是否为一个有效的Token

\* 有效Token 都会实现一个 AuthenticationToken 接口

\*/

@Override

public boolean supports(AuthenticationToken token) {

return token instanceof JwtToken;

}

/\*\*

\* 授权认证

\*/

@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());

}

/\*\*

\* 获取用户权限信息

\*/

@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) {

//权限信息对象info,用来存放查出的用户的所有的角色（role）及权限（permission）,此消息放到

//Redis数据库中进行存储

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

// Shiro有时间间隔机制，2分钟内不会重复执行该方法

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

if (findUser != null) {

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

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

//用户的角色集合

info.addRole(roleEntity.getName());

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

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

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

//用户的角色对应的所有权限

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) {

//获取所有session

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);

}

}

* 1. 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DispatchServlect，将请求分发到不同的URI上面。在此举一个例子。

// 让Spring 知道这是一个控制器

@Controller

// 此控制器的前缀

@RequestMapping("/demo")

public class DemoController {

// 这个方法会处理 http://host:port/demo/test 的请求

@RequestMapping("/test")

public String test() {

return "test";

}

}

dao模块：此模块存放实体类，实体关系映射。以下是数据库设计。

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |
| positioin\_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 |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

* 1. 前端项目详细介绍

1. 调试与实现
   1. 调试中遇到的重点与难点
      1. 调试重点
2. 功能调试。
   1. 功能是否正确实现。
3. 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。
4. 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。
      1. 调试难点
5. 对HTTP协议还不够熟悉，时常把携带数据放错位置。
6. 时常会出现Web页面、服务器日志、数据库数据出现乱码。
7. 服务打不开，端口被占用。
   1. 解决方案
8. 仔细的看了一遍HTTP资料，针对每个API写好了测试文件。
9. 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。
10. 查看端口被什么程序占用，杀死该程序。
    1. 实现展示
11. 总结
12. 参考文献
13. 致谢