# 《中间件技术课程报告》

姓名：赵亚星

学号：2018104163

### 基于springboot+vue的人事管理系统

**项目介绍**

本项目目的是实现机关和事业单位的人事管理信息系统。机关和事业单位的

人事管理在政府工作中占有极其重要的地位，如何加强各个部门之间的协调和提

高工作效率是人事管理的当务之急。通过人事管理系统人事组织部门能做到以人

为中心，各部门之间使用共有的资源，即时通信，现时提高工作效率，简化繁琐

的手工统计、信息汇总和工资业务大量人工工作，让人事组织和工资管理工作在

人事组织相关的各部门之间活起来。本项目就是在这样背景下提出的。

本项目在技术方面采用了较为先进的 J2SE 信息技术，本项目旨在实现人事

管理信息系统，包括员工资料、人事管理、工资管理、统计管理和系统管理系统

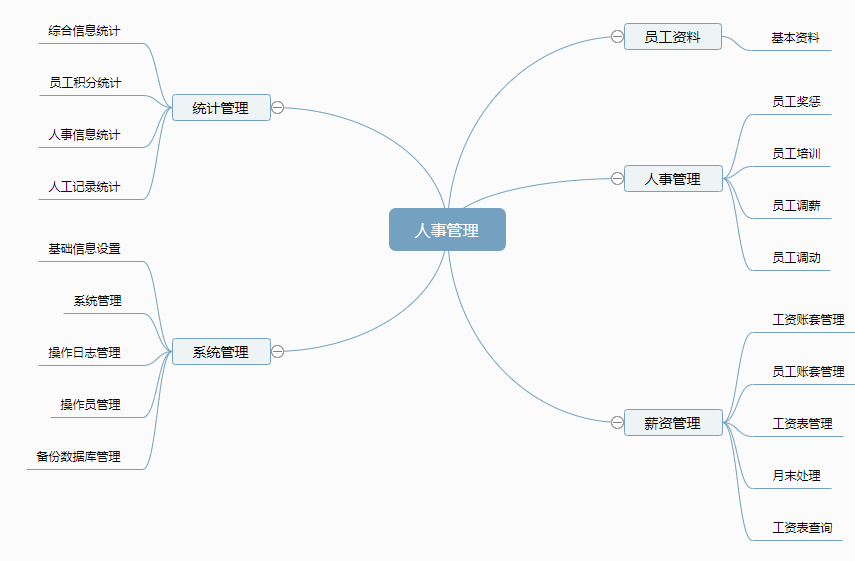
五大模块。

本项目是一个前后端分离的人力资源管理系统，项目采用SpringBoot+Vue开发。

该项目前端采用vue+element-ui开发，后端采用springboot+security+mybatis技术，数据库使用mysql。

**功能列表**

项目功能包括员工资料、人事管理、薪资调整、统计管理、系统管理几大模块。



**后端源码说明**

1、pom文件 定义了项目的基本信息，用于描述项目如何构建，声明项目依赖等。

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.sang</groupId>  
 <artifactId>hrserver</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>hrserver</name>  
 <description>Demo project for Spring Boot</description>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.0.4.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 <exclusions>  
 <exclusion>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-tomcat</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-undertow</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>1.3.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
 <exclusions>  
 <exclusion>  
 <groupId>io.lettuce</groupId>  
 <artifactId>lettuce-core</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 <dependency>  
 <groupId>redis.clients</groupId>  
 <artifactId>jedis</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>1.1.10</version>  
 </dependency>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.poi</groupId>  
 <artifactId>poi</artifactId>  
 <version>3.17</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-mail</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-websocket</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-cache</artifactId>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 <resources>  
 <resource>  
 <directory>src/main/java</directory>  
 <includes>  
 <include>\*\*/\*.xml</include>  
 </includes>  
 </resource>  
 <resource>  
 <directory>src/main/resources</directory>  
 </resource>  
 </resources>  
 </build>  
</project>

2、基于SpringSecurity处理权限问题

配置SpringSecurity访问策略，包括登录处理，登出处理，资源访问，密码基本加密等等。

package org.sang.config;  
  
import com.fasterxml.jackson.databind.ObjectMapper;  
import org.sang.bean.RespBean;  
import org.sang.common.HrUtils;  
import org.sang.service.HrService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.\*;  
import org.springframework.security.config.annotation.ObjectPostProcessor;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.builders.WebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.AuthenticationException;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.web.access.intercept.FilterSecurityInterceptor;  
import org.springframework.security.web.authentication.AuthenticationFailureHandler;  
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;  
import org.springframework.security.web.authentication.logout.LogoutSuccessHandler;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.io.PrintWriter;  
  
@Configuration  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
 @Autowired  
 HrService hrService;  
 @Autowired  
 CustomMetadataSource metadataSource;  
 @Autowired  
 UrlAccessDecisionManager urlAccessDecisionManager;  
 @Autowired  
 AuthenticationAccessDeniedHandler deniedHandler;  
  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(hrService)  
 .passwordEncoder(new BCryptPasswordEncoder());  
 }  
  
 @Override  
 public void configure(WebSecurity web) throws Exception {  
 web.ignoring().antMatchers("/index.html", "/static/\*\*", "/login\_p", "/favicon.ico");  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.authorizeRequests()  
 .withObjectPostProcessor(new ObjectPostProcessor<FilterSecurityInterceptor>() {  
 @Override  
 public <O extends FilterSecurityInterceptor> O postProcess(O o) {  
 o.setSecurityMetadataSource(metadataSource);  
 o.setAccessDecisionManager(urlAccessDecisionManager);  
 return o;  
 }  
 })  
 .and()  
 .formLogin().loginPage("/login\_p").loginProcessingUrl("/login")  
 .usernameParameter("username").passwordParameter("password")  
 .failureHandler(new AuthenticationFailureHandler() {  
 @Override  
 public void onAuthenticationFailure(HttpServletRequest req,  
 HttpServletResponse resp,  
 AuthenticationException e) throws IOException {  
 resp.setContentType("application/json;charset=utf-8");  
 RespBean respBean = null;  
 if (e instanceof BadCredentialsException ||  
 e instanceof UsernameNotFoundException) {  
 respBean = RespBean.*error*("账户名或者密码输入错误!");  
 } else if (e instanceof LockedException) {  
 respBean = RespBean.*error*("账户被锁定，请联系管理员!");  
 } else if (e instanceof CredentialsExpiredException) {  
 respBean = RespBean.*error*("密码过期，请联系管理员!");  
 } else if (e instanceof AccountExpiredException) {  
 respBean = RespBean.*error*("账户过期，请联系管理员!");  
 } else if (e instanceof DisabledException) {  
 respBean = RespBean.*error*("账户被禁用，请联系管理员!");  
 } else {  
 respBean = RespBean.*error*("登录失败!");  
 }  
 resp.setStatus(401);  
 ObjectMapper om = new ObjectMapper();  
 PrintWriter out = resp.getWriter();  
 out.write(om.writeValueAsString(respBean));  
 out.flush();  
 out.close();  
 }  
 })  
 .successHandler(new AuthenticationSuccessHandler() {  
 @Override  
 public void onAuthenticationSuccess(HttpServletRequest req,  
 HttpServletResponse resp,  
 Authentication auth) throws IOException {  
 resp.setContentType("application/json;charset=utf-8");  
 RespBean respBean = RespBean.*ok*("登录成功!", HrUtils.*getCurrentHr*());  
 ObjectMapper om = new ObjectMapper();  
 PrintWriter out = resp.getWriter();  
 out.write(om.writeValueAsString(respBean));  
 out.flush();  
 out.close();  
 }  
 })  
 .permitAll()  
 .and()  
 .logout()  
 .logoutUrl("/logout")  
 .logoutSuccessHandler(new LogoutSuccessHandler() {  
 @Override  
 public void onLogoutSuccess(HttpServletRequest req, HttpServletResponse resp, Authentication authentication) throws IOException, ServletException {  
 resp.setContentType("application/json;charset=utf-8");  
 RespBean respBean = RespBean.*ok*("注销成功!");  
 ObjectMapper om = new ObjectMapper();  
 PrintWriter out = resp.getWriter();  
 out.write(om.writeValueAsString(respBean));  
 out.flush();  
 out.close();  
 }  
 })  
 .permitAll()  
 .and().csrf().disable()  
 .exceptionHandling().accessDeniedHandler(deniedHandler);  
 }  
}

自定义投票鉴权决策管理器AccessDecisionManager

@Component  
public class UrlAccessDecisionManager implements AccessDecisionManager {  
 @Override  
 public void decide(Authentication auth, Object o, Collection<ConfigAttribute> cas){  
 Iterator<ConfigAttribute> iterator = cas.iterator();  
 while (iterator.hasNext()) {  
 ConfigAttribute ca = iterator.next();  
 //当前请求需要的权限  
 String needRole = ca.getAttribute();  
 if ("ROLE\_LOGIN".equals(needRole)) {  
 if (auth instanceof AnonymousAuthenticationToken) {  
 throw new BadCredentialsException("未登录");  
 } else  
 return;  
 }  
 //当前用户所具有的权限  
 Collection<? extends GrantedAuthority> authorities = auth.getAuthorities();  
 for (GrantedAuthority authority : authorities) {  
 if (authority.getAuthority().equals(needRole)) {  
 return;  
 }  
 }  
 }  
 throw new AccessDeniedException("权限不足!");  
 }  
 @Override  
 public boolean supports(ConfigAttribute configAttribute) {  
 return true;  
 }  
 @Override  
 public boolean supports(Class<?> aClass) {  
 return true;  
 }  
}

自定义AccessDeniedHandler，当一个已授权(或已登陆)的用户请求访问他权限之外的资源时, 这个类的handle方法将会被调用, 定义如何处理这个请求。

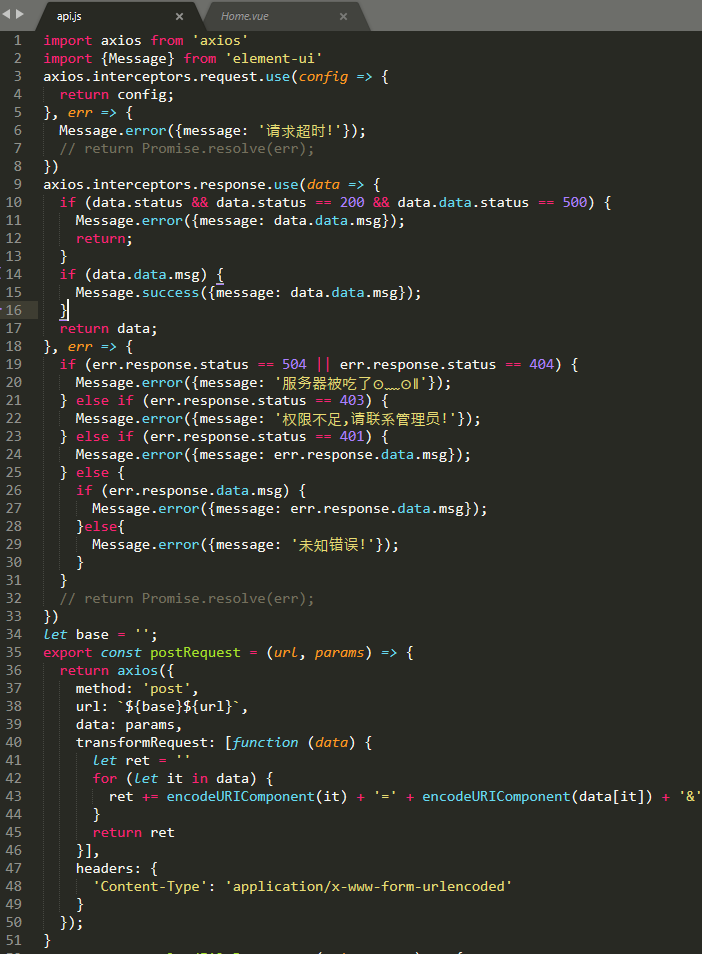
@Component  
public class AuthenticationAccessDeniedHandler implements AccessDeniedHandler {  
 @Override  
 public void handle(HttpServletRequest httpServletRequest, HttpServletResponse resp,  
 AccessDeniedException e) throws IOException {  
 resp.setStatus(HttpServletResponse.*SC\_FORBIDDEN*);  
 resp.setContentType("application/json;charset=UTF-8");  
 PrintWriter out = resp.getWriter();  
 RespBean error = RespBean.*error*("权限不足，请联系管理员!");  
 out.write(new ObjectMapper().writeValueAsString(error));  
 out.flush();  
 out.close();  
 }  
}

自定义securityMetadataSource，实现FilterInvocationSecurityMetadataSource接口，用来储存请求与权限的对应关系.

@Component  
public class CustomMetadataSource implements FilterInvocationSecurityMetadataSource {  
 @Autowired  
 MenuService menuService;  
 AntPathMatcher antPathMatcher = new AntPathMatcher();  
 @Override  
 public Collection<ConfigAttribute> getAttributes(Object o) {  
 String requestUrl = ((FilterInvocation) o).getRequestUrl();  
 List<Menu> allMenu = menuService.getAllMenu();  
 for (Menu menu : allMenu) {  
 if (antPathMatcher.match(menu.getUrl(), requestUrl)  
 &&menu.getRoles().size()>0) {  
 List<Role> roles = menu.getRoles();  
 int size = roles.size();  
 String[] values = new String[size];  
 for (int i = 0; i < size; i++) {  
 values[i] = roles.get(i).getName();  
 }  
 return SecurityConfig.*createList*(values);  
 }  
 }  
 //没有匹配上的资源，都是登录访问  
 return SecurityConfig.*createList*("ROLE\_LOGIN");  
 }  
 @Override  
 public Collection<ConfigAttribute> getAllConfigAttributes() {  
 return null;  
 }  
 @Override  
 public boolean supports(Class<?> aClass) {  
 return FilterInvocation.class.isAssignableFrom(aClass);  
 }  
}

**前端源码**

axios请求封装,请求异常统一处理对axios进行了简单的封装，基于axios中的拦截器功能。

****

在登录操作执行成功之后，通过commit操作将数据提交到store中，核心代码如下：

