目录

[一． 前言 1](#_Toc28493)

[二． 修改项目需知 2](#_Toc14008)

[三． 数据库表格关系 4](#_Toc9074)

[四． 重要的查询语句 5](#_Toc25799)

[五． 部署 5](#_Toc22764)

[六． 鉴权相关java代码 5](#_Toc10239)

1. 前言

版本要求：

node-v8.11.3-x64



参考1：

<https://mp.weixin.qq.com/s?__biz=MzI1NDY0MTkzNQ==&mid=2247485303&idx=1&sn=0b04836496f2d22dd2dcb116041570e7&scene=21#wechat_redirect>

参考2：

<https://blog.csdn.net/shu15121856/article/details/80790277>

参考3：

<https://github.com/lenve/vhr>

说明：

本项目完全参考Vhr项目。不得用于任何商业用途。

版本号：

20211222V2

1. 修改项目需知

如果想加入自己的前端模块。需要配置：

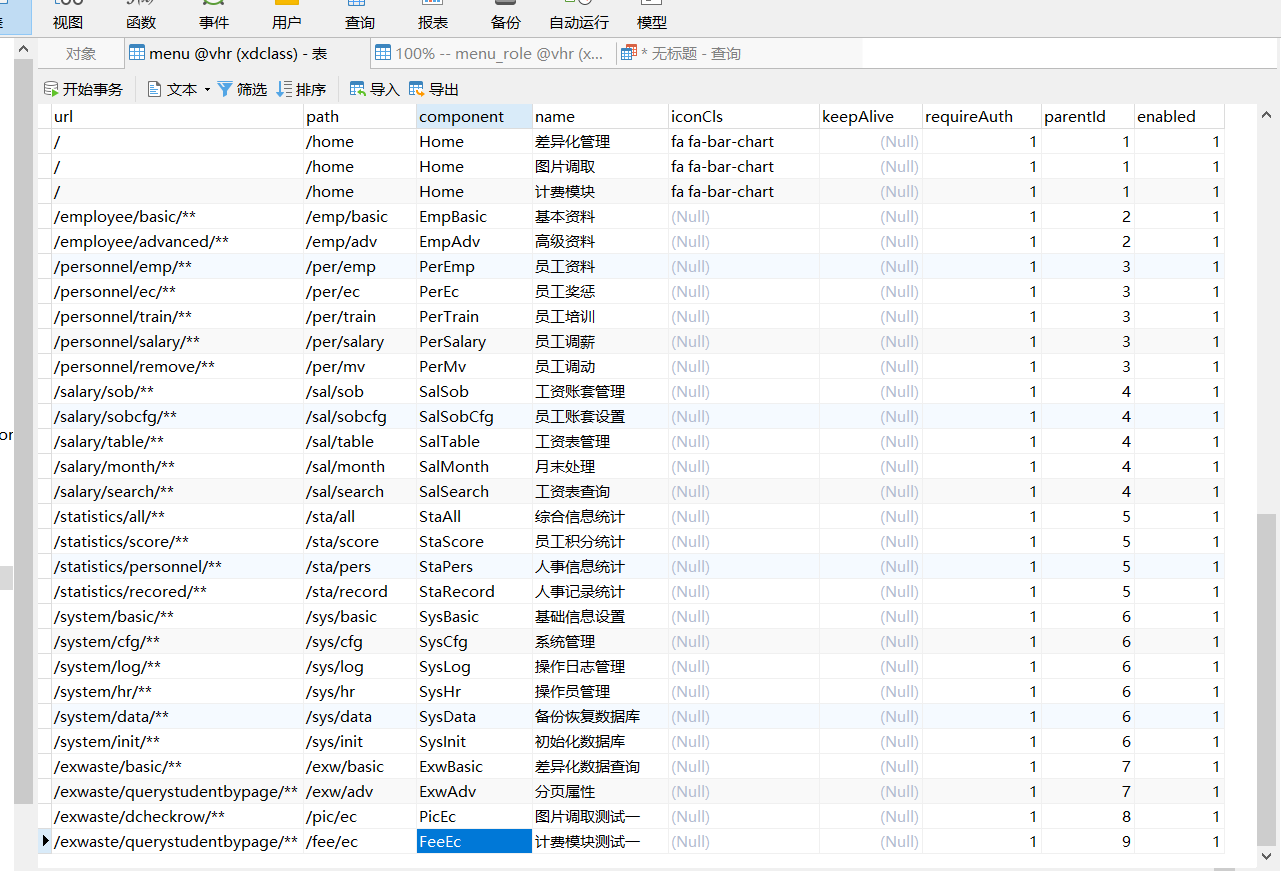
1. java项目pom.xml去掉flyway配置。

<!-- <dependency>-->  
<!-- <groupId>org.flywaydb</groupId>-->  
<!-- <artifactId>flyway-core</artifactId>-->  
<!-- </dependency>-->

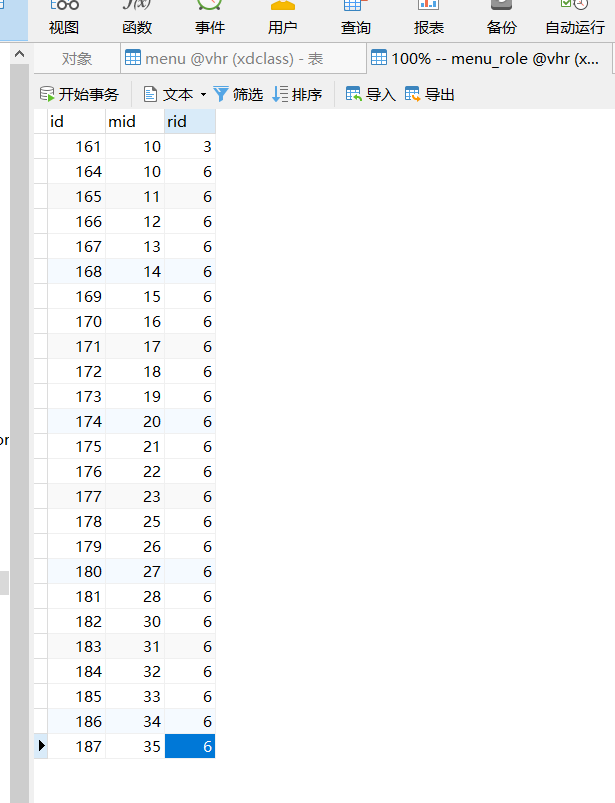
其原因是需要手动先修改数据库。后期如果想用flyway，可以释放本段代码。

如果不释放，直接导入数据库sql文件也可以。

1. menu中配置路径



1. 在menu\_role中给角色权限



1. 在vue menu.js配置component解析路径



1. 数据库表格关系

表1-1各个表格

|  |  |  |
| --- | --- | --- |
| 表格名 | 作用 | 是否启用 |
| nation | 民族表 | 否 |
| politicstatus | 政治角色表 | 否 |
| position | 职位表 | 是 |
| salary | 工资表 | 是 |
| department | 部门表 | 是 |
| employee | 员工表 | 是 |
| Empsalary | 员工工资关系表 | 是 |
| Flyway\_schema\_history | flyway相关表 | 否 |
| hr | 人员表 | 是 |
| hr\_role | 人员角色关系表 | 是 |
| role | 角色表 | 是 |
| joblevel | 工作等级表 | 是 |
| menu | 菜单路径配置表 | 是 |
| menu\_role | 菜单角色关系表 | 是 |
| msgcontent | 不知道干啥用的表 | 否 |
| sysmsg | 不知道干啥用的表 | 否 |

1. 重要的查询语句

在menuMapper.xml中

<select id="getAllMenusWithRole" resultMap="MenuWithRole">  
 select m.\*,r.`id` as rid,r.`name` as rname,r.`nameZh` as rnameZh from menu m,menu\_role mr,role r where m.`id`=mr.`mid` and mr.`rid`=r.`id` order by m.`id`  
</select>

1. 部署

参考git中nginx.conf和自动部署脚本。主要使用docker容器.

1. 鉴权相关java代码

copy自https://github.com/lenve/vhr

这是非常核心的一块内容。

创建Hr和HrService

首先我们需要创建Hr类，即我们的用户类，该类实现了UserDetails接口，该类的属性如下：

public class Hr implements UserDetails {

private Long id;

private String name;

private String phone;

private String telephone;

private String address;

private boolean enabled;

private String username;

private String password;

private String remark;

private List<Role> roles;

private String userface;

//getter/setter省略

}

如果小伙伴对属性的含义有疑问，可以参考[1.权限数据库设计](https://github.com/lenve/vhr/wiki/1.%E6%9D%83%E9%99%90%E6%95%B0%E6%8D%AE%E5%BA%93%E8%AE%BE%E8%AE%A1).

UserDetails接口默认有几个方法需要实现，这几个方法中，除了isEnabled返回了正常的enabled之外，其他的方法我都统一返回true，因为我这里的业务逻辑并不涉及到账户的锁定、密码的过期等等，只有账户是否被禁用，因此只处理了isEnabled方法，这一块小伙伴可以根据自己的实际情况来调整。另外，UserDetails中还有一个方法叫做getAuthorities，该方法用来获取当前用户所具有的角色，但是小伙伴也看到了，我的Hr中有一个roles属性用来描述当前用户的角色，因此我的getAuthorities方法的实现如下：

public Collection<? extends GrantedAuthority> getAuthorities() {

List<GrantedAuthority> authorities = new ArrayList<>();

for (Role role : roles) {

authorities.add(new SimpleGrantedAuthority(role.getName()));

}

return authorities;

}

即直接从roles中获取当前用户所具有的角色，构造SimpleGrantedAuthority然后返回即可。

创建好Hr之后，接下来我们需要创建HrService，用来执行登录等操作，HrService需要实现UserDetailsService接口，如下：

@Service

@Transactional

public class HrService implements UserDetailsService {

@Autowired

HrMapper hrMapper;

@Override

public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException {

Hr hr = hrMapper.loadUserByUsername(s);

if (hr == null) {

throw new UsernameNotFoundException("用户名不对");

}

return hr;

}

}

这里最主要是实现了UserDetailsService接口中的loadUserByUsername方法，在执行登录的过程中，这个方法将根据用户名去查找用户，如果用户不存在，则抛出UsernameNotFoundException异常，否则直接将查到的Hr返回。HrMapper用来执行数据库的查询操作，这个不在本系列的介绍范围内，所有涉及到数据库的操作都将只介绍方法的作用。

自定义FilterInvocationSecurityMetadataSource

FilterInvocationSecurityMetadataSource有一个默认的实现类DefaultFilterInvocationSecurityMetadataSource，该类的主要功能就是通过当前的请求地址，获取该地址需要的用户角色，我们照猫画虎，自己也定义一个FilterInvocationSecurityMetadataSource，如下：

@Component

public class UrlFilterInvocationSecurityMetadataSource implements FilterInvocationSecurityMetadataSource {

@Autowired

MenuService menuService;

AntPathMatcher antPathMatcher = new AntPathMatcher();

@Override

public Collection<ConfigAttribute> getAttributes(Object o) throws IllegalArgumentException {

//获取请求地址

String requestUrl = ((FilterInvocation) o).getRequestUrl();

if ("/login\_p".equals(requestUrl)) {

return null;

}

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);

}

}

关于自定义这个类，我说如下几点：

1.一开始注入了MenuService，MenuService的作用是用来查询数据库中url pattern和role的对应关系，查询结果是一个List集合，集合中是Menu类，Menu类有两个核心属性，一个是url pattern，即匹配规则(比如/admin/\*\*)，还有一个是List,即这种规则的路径需要哪些角色才能访问。

2.我们可以从getAttributes(Object o)方法的参数o中提取出当前的请求url，然后将这个请求url和数据库中查询出来的所有url pattern一一对照，看符合哪一个url pattern，然后就获取到该url pattern所对应的角色，当然这个角色可能有多个，所以遍历角色，最后利用SecurityConfig.createList方法来创建一个角色集合。

3.第二步的操作中，涉及到一个优先级问题，比如我的地址是/employee/basic/hello,这个地址既能被/employee/\*\*匹配，也能被/employee/basic/\*\*匹配，这就要求我们从数据库查询的时候对数据进行排序，将/employee/basic/\*\*类型的url pattern放在集合的前面去比较。

4.如果getAttributes(Object o)方法返回null的话，意味着当前这个请求不需要任何角色就能访问，甚至不需要登录。但是在我的整个业务中，并不存在这样的请求，我这里的要求是，所有未匹配到的路径，都是认证(登录)后可访问，因此我在这里返回一个ROLE\_LOGIN的角色，这种角色在我的角色数据库中并不存在，因此我将在下一步的角色比对过程中特殊处理这种角色。

5.如果地址是/login\_p，这个是登录页，不需要任何角色即可访问，直接返回null。

6.getAttributes(Object o)方法返回的集合最终会来到AccessDecisionManager类中，接下来我们再来看AccessDecisionManager类。

自定义AccessDecisionManager

自定义UrlAccessDecisionManager类实现AccessDecisionManager接口，如下：

@Component

public class UrlAccessDecisionManager implements AccessDecisionManager {

@Override

public void decide(Authentication authentication, Object o, Collection<ConfigAttribute> collection) throws AccessDeniedException, AuthenticationException {

Iterator<ConfigAttribute> iterator = collection.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 (authentication instanceof AnonymousAuthenticationToken) {

throw new BadCredentialsException("未登录");

} else

return;

}

//当前用户所具有的权限

Collection<? extends GrantedAuthority> authorities = authentication.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;

}

}

关于这个类，我说如下几点：

1.decide方法接收三个参数，其中第一个参数中保存了当前登录用户的角色信息，第三个参数则是UrlFilterInvocationSecurityMetadataSource中的getAttributes方法传来的，表示当前请求需要的角色（可能有多个）。

2.如果当前请求需要的权限为ROLE\_LOGIN则表示登录即可访问，和角色没有关系，此时我需要判断authentication是不是AnonymousAuthenticationToken的一个实例，如果是，则表示当前用户没有登录，没有登录就抛一个BadCredentialsException异常，登录了就直接返回，则这个请求将被成功执行。

3.遍历collection，同时查看当前用户的角色列表中是否具备需要的权限，如果具备就直接返回，否则就抛异常。

4.这里涉及到一个all和any的问题：假设当前用户具备角色A、角色B，当前请求需要角色B、角色C，那么是要当前用户要包含所有请求角色才算授权成功还是只要包含一个就算授权成功？我这里采用了第二种方案，即只要包含一个即可。小伙伴可根据自己的实际情况调整decide方法中的逻辑。

自定义AccessDeniedHandler

通过自定义AccessDeniedHandler我们可以自定义403响应的内容，如下：

@Component

public class AuthenticationAccessDeniedHandler implements AccessDeniedHandler {

@Override

public void handle(HttpServletRequest httpServletRequest, HttpServletResponse resp, AccessDeniedException e) throws IOException, ServletException {

resp.setStatus(HttpServletResponse.SC\_FORBIDDEN);

resp.setCharacterEncoding("UTF-8");

PrintWriter out = resp.getWriter();

out.write("{\"status\":\"error\",\"msg\":\"权限不足，请联系管理员!\"}");

out.flush();

out.close();

}

}

配置WebSecurityConfig

最后在webSecurityConfig中完成简单的配置即可，如下：

@Configuration

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

HrService hrService;

@Autowired

UrlFilterInvocationSecurityMetadataSource urlFilterInvocationSecurityMetadataSource;

@Autowired

UrlAccessDecisionManager urlAccessDecisionManager;

@Autowired

AuthenticationAccessDeniedHandler authenticationAccessDeniedHandler;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(hrService);

}

@Override

public void configure(WebSecurity web) throws Exception {

web.ignoring().antMatchers("/index.html", "/static/\*\*");

}

@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(urlFilterInvocationSecurityMetadataSource);

o.setAccessDecisionManager(urlAccessDecisionManager);

return o;

}

}).and().formLogin().loginPage("/login\_p").loginProcessingUrl("/login").usernameParameter("username").passwordParameter("password").permitAll().failureHandler(new AuthenticationFailureHandler() {

@Override

public void onAuthenticationFailure(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, AuthenticationException e) throws IOException, ServletException {

httpServletResponse.setContentType("application/json;charset=utf-8");

PrintWriter out = httpServletResponse.getWriter();

StringBuffer sb = new StringBuffer();

sb.append("{\"status\":\"error\",\"msg\":\"");

if (e instanceof UsernameNotFoundException || e instanceof BadCredentialsException) {

sb.append("用户名或密码输入错误，登录失败!");

} else if (e instanceof DisabledException) {

sb.append("账户被禁用，登录失败，请联系管理员!");

} else {

sb.append("登录失败!");

}

sb.append("\"}");

out.write(sb.toString());

out.flush();

out.close();

}

}).successHandler(new AuthenticationSuccessHandler() {

@Override

public void onAuthenticationSuccess(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Authentication authentication) throws IOException, ServletException {

httpServletResponse.setContentType("application/json;charset=utf-8");

PrintWriter out = httpServletResponse.getWriter();

ObjectMapper objectMapper = new ObjectMapper();

String s = "{\"status\":\"success\",\"msg\":" + objectMapper.writeValueAsString(HrUtils.getCurrentHr()) + "}";

out.write(s);

out.flush();

out.close();

}

}).and().logout().permitAll().and().csrf().disable().exceptionHandling().accessDeniedHandler(authenticationAccessDeniedHandler);

}

}

关于这个配置，我说如下几点：

1.在configure(HttpSecurity http)方法中，通过withObjectPostProcessor将刚刚创建的UrlFilterInvocationSecurityMetadataSource和UrlAccessDecisionManager注入进来。到时候，请求都会经过刚才的过滤器（除了configure(WebSecurity web)方法忽略的请求）。

2.successHandler中配置登录成功时返回的JSON，登录成功时返回当前用户的信息。

3.failureHandler表示登录失败，登录失败的原因可能有多种，我们根据不同的异常输出不同的错误提示即可。

OK，这些操作都完成之后，我们可以通过POSTMAN或者RESTClient来发起一个登录请求，看到如下结果则表示登录成功: