4. 博客后台

4.1 准备工作

1. 在admin模块中创建启动类

```
@SpringBootApplication
@MapperScan("com.wh.blog.dao")
public class AdminApplication {
    public static void main(String[] args) {
        SpringApplication.run(AdminApplication.class, args);}
}
```

2. 在admin模块中创建application.yml配置文件（从blog模块中复制，修改端口号）

```
server:
  port: 8989

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/blog?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8
    username: root
    password: 123456
  servlet:
    multipart:
      max-file-size: 2MB
      max-request-size: 5MB

mybatis-plus:
#  configuration:
#    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
  global-config:
    db-config:
      logic-delete-field: delFlag
      logic-delete-value: 1
      logic-not-delete-value: 0
      id-type: auto
```

3. 导入新的 blog.sql，添加博客后台会使用到的表格
4. 在admin模块中添加security相关类

   在config包中添加 **SecurityConfig**（从blog模块中复制，修改请求的访问认证）

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws
Exception {
        return super.authenticationManagerBean();
    }
```

```
    @Autowired
    private JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter;
    @Autowired
    AuthenticationEntryPoint authenticationEntryPoint;
    @Autowired
    AccessDeniedHandler accessDeniedHandler;


    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                //关闭csrf
                .csrf().disable()
                //不通过Session获取SecurityContext

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                // 对于登录接口 允许匿名访问
                .antMatchers("/user/login").anonymous()
                // 除上面外的所有请求需要认证才能访问
                .anyRequest().authenticated();

        //配置异常处理器
        http.exceptionHandling()
                .authenticationEntryPoint(authenticationEntryPoint)
                .accessDeniedHandler(accessDeniedHandler);
        //关闭默认的注销功能
        http.logout().disable();
        //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中
        http.addFilterBefore(jwtAuthenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);
        //允许跨域
        http.cors();
    }

    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
}
```

在filter包中添加 **JwtAuthenticationTokenFilter**（从blog模块中复制，将获取用户信息的key的前缀修改为adminlogin）

```
@Component
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {

    @Autowired
    private RedisCache redisCache;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
        //获取请求头中的token
```

```
        String token = request.getHeader("token");
        if (!StringUtils.hasText(token)) {
            //说明该接口不需要登录  直接放行
            filterChain.doFilter(request, response);
            return;
        }
        //解析获取userid
        Claims claims = null;
        try {
            claims = JwtUtil.parseJWT(token);
        } catch (Exception e) {
            e.printStackTrace();
            //token超时  token非法
            //响应告诉前端需要重新登录
            ResponseResult result =
ResponseResult.errorResult(AppHttpCodeEnum.NEED_LOGIN);
            WebUtils.renderString(response, JSON.toJSONString(result));
            return;
        }
        String userId = claims.getSubject();
        //从redis中获取用户信息
        LoginUser loginUser = redisCache.getCacheObject("adminlogin:" +
userId);
        //如果获取不到
        if (Objects.isNull(loginUser)) {
            //说明登录过期  提示重新登录
            ResponseResult result =
ResponseResult.errorResult(AppHttpCodeEnum.NEED_LOGIN);
            WebUtils.renderString(response, JSON.toJSONString(result));
            return;
        }
        //存入SecurityContextHolder
        UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(loginUser, null, null);

 SecurityContextHolder.getContext().setAuthentication(authenticationToken);

        filterChain.doFilter(request, response);
    }
}
```

4.2 后台登录

4.2.1 需求

需要实现登录功能，后台所有功能都必须登录才能使用。后台的认证授权也使用SpringSecurity安全框架来实现。

4.2.2 接口设计

请求方式	请求路径
POST	/user/login

请求体：

```
{
    "userName":"test",
    "password":"1234"
}
```

响应格式：

```
{
    "code": 200,
    "data": {
        "token":
"eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI0ODBmOThmYmJkNmI0NjM0OWUyZjY2NTM0NGNjZWY2NSIsIn
N1YiI6IjEiLCJpc3MiOiJzZyIsImlhdCI6MTY0Mzg3NDMxNiwiZXhwIjoxNjQzOTYwNzE2fQ.ldLBUvN
IxQCGemkCoMgT_OYsjsWndTg5tqfJb77pabk"
    },
    "msg": "操作成功"
}
```

4.2.3 思路分析

- 登录：
  - 自定义登录接口
    - 调用ProviderManager的方法进行认证，如果认证通过生成jwt
    - 把用户信息存入redis中
  - 自定义UserDetailsService
    - 在这个实现类中去查询数据库
    - 注意配置passwordEncoder为BCryptPasswordEncoder
- 校验：
  - 定义Jwt认证过滤器
    - 获取token
    - 解析token获取其中的userid
    - 从redis中获取用户信息
    - 存入SecurityContextHolder

4.2.4 代码实现

AdminLoginController

复制一份BlogLoginController，命名为AdminLoginController，其中注入 AdminLoginService，并将请求地址修改为/user/login

```
@RestController
public class AdminLoginController {

    @Autowired
    private IAdminLoginService adminLoginService;

    @PostMapping("/user/login")
    public ResponseResult login(@RequestBody User user){
        if(!StringUtils.hasText(user.getUserName())){
            //提示 必须要传用户名
            throw new SystemException(AppHttpCodeEnum.REQUIRE_USERNAME);
        }
```

```
        return adminLoginService.login(user);
    }
}
```

IAdminLoginService

复制一份IBlogLoginService命名为IAdminLoginService即可

```
public interface IAdminLoginService {
    ResponseResult login(User user);
}
```

AdminLoginServiceImpl

复制一份，BlogLoginServiceImpl，命名为AdminLoginServiceImpl 实现 IAdminLoginService

login方法中存redis的key的前缀修改为adminlogin

返回的数据中只要返回token

```
@Service
public class AdminLoginServiceImpl implements IAdminLoginService {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private RedisCache redisCache;

    @Override
    public ResponseResult login(User user) {
        UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(user.getUserName(),user.getPassword());
        Authentication authenticate =
authenticationManager.authenticate(authenticationToken);
        //判断是否认证通过
        if(Objects.isNull(authenticate)){
            throw new RuntimeException("用户名或密码错误");
        }
        //获取userid 生成token
        LoginUser loginUser = (LoginUser) authenticate.getPrincipal();
        String userId = loginUser.getUser().getId().toString();
        String jwt = JwtUtil.createJWT(userId);
        //把用户信息存入redis
        redisCache.setCacheObject("adminlogin:"+userId,loginUser);

        //把token封装 返回
        Map<String,String> map = new HashMap<>();
        map.put("token",jwt);
        return ResponseResult.okResult(map);
    }
}
```

4.3 前端权限控制

4.3.1 需求

用户只能使用他的权限所允许使用的功能。

4.3.2 接口设计

请求方式	请求地址	请求头
GET	/getInfo	需要token请求头

响应格式:


```
{
    "code":200,
    "data":{
        "permissions":[
            "content:category:list",
            "content:article:list",
            "content:tag:index",
            "content:article:writer",
            "content:category:export"
        ],
        "roles":[
            "common"
        ],
        "user":{
            "avatar":"https://gss0.baidu.com/-
Po3dSag_xI4khGko9WTAnF6hhy/zhidao/pic/item/574e9258d109b3de57070594cbbf6c81810a4
c96.jpg",
            "email":"test@qq.com",
            "id":"2",
            "nickName":"test",
            "sex":"0"
        }
    },
    "msg":"操作成功"
}
```


4.3.3 代码实现

使用代码生成器生成menu和role表对应的实体类、Mapper接口、Service接口以及 ServiceImpl实现类

AdminUserInfoVo

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class AdminUserInfoVo {

    private List<String> permissions;

    private List<String> roles;

    private UserInfoVo user;
}
```

AdminLoginController

```
@Autowired
private IMenuService menuService;

@Autowired
private IRoleService roleService;

@GetMapping("/getInfo")
public ResponseResult<AdminUserInfoVo> getInfo(){
    //获取当前登录的用户
    LoginUser loginUser = SecurityUtils.getLoginUser();
    //根据用户id查询权限信息
    List<String> perms =
menuService.selectPermsByUserId(loginUser.getUser().getId());
    //根据用户id查询角色信息
    List<String> roleKeyList =
roleService.selectRoleKeyByUserId(loginUser.getUser().getId());

    //获取用户信息
    User user = loginUser.getUser();
    UserInfoVo userInfoVo = BeanCopyUtils.copyBean(user, UserInfoVo.class);

    //封装数据返回
    AdminUserInfoVo adminUserInfoVo = new
AdminUserInfoVo(perms,roleKeyList,userInfoVo);
    return ResponseResult.okResult(adminUserInfoVo);
}
```

IMenuService

```
List<String> selectPermsByUserId(Long id);
```

MenuServiceImpl

```
@Autowired
MenuMapper menuMapper;

@Override
public List<String> selectPermsByUserId(Long id) {
    //如果是管理员，返回所有的权限
```

```
    if (id == 1L) {
        LambdaQueryWrapper<Menu> wrapper = new LambdaQueryWrapper<>();
        wrapper.in(Menu::getMenuType, SystemConstants.MENU,
SystemConstants.BUTTON);
        wrapper.eq(Menu::getStatus, SystemConstants.STATUS_NORMAL);
        List<Menu> menus = menuMapper.selectList(wrapper);
        List<String> perms = menus.stream()
            .map(Menu::getPerms)
            .collect(Collectors.toList());
        return perms;
    }
    //否则返回所具有的权限
    return menuMapper.selectPermsByUserId(id);
}
```

SystemConstants

```
public static final String MENU = "C";
public static final String BUTTON = "F";
```

MenuMapper

```
List<String> selectPermsByUserId(Long userId);
```

MenuMapper.xml

```
<?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.wh.blog.dao.MenuMapper">

    <select id="selectPermsByUserId" resultType="java.lang.String">
        SELECT
            DISTINCT m.perms
        FROM
            sys_user_role ur
            LEFT JOIN sys_role_menu rm ON ur.role_id = rm.role_id
            LEFT JOIN sys_menu m ON m.id = rm.menu_id
        WHERE
            ur.user_id = #{userId} AND
            m.menu_type IN ('C','F') AND
            m.status = 0 AND
            m.del_flag = 0
    </select>
</mapper>
```

IRoleService

```
List<String> selectRoleKeyByUserId(Long id);
```

RoleServiceImpl

```
@Autowired
```

```
RoleMapper roleMapper;

@Override
public List<String> selectRoleKeyByUserId(Long id) {
    //判断是否是管理员 如果是返回集合中只需要有admin
    if(id == 1L){
        List<String> roleKeys = new ArrayList<>();
        roleKeys.add("admin");
        return roleKeys;
    }
    //否则查询用户所具有的角色信息
    return roleMapper.selectRoleKeyByUserId(id);
}
```

RoleMapper

```
List<String> selectRoleKeyByUserId(Long userId);
```

RoleMapper.xml

```
<?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.wh.blog.dao.RoleMapper">
    <select id="selectRoleKeyByUserId" resultType="java.lang.String">
        SELECT r.role_key
        FROM sys_user_role ur
        LEFT JOIN sys_role r ON ur.role_id = r.id
        WHERE
            ur.user_id = #{userId} AND
            r.status = 0 AND r.del_flag = 0
    </select>
</mapper>
```

4.4 动态路由

4.4.1 需求

后台系统需要能实现不同的用户权限可以看到不同的功能。

4.4.2 接口设计

请求方式	请求地址	请求头
GET	/getRouters	需要token请求头

响应格式:

- 前端为了实现动态路由的效果，需要后端有接口能返回用户所能访问的菜单数据。返回的菜单数据需要体现父子菜单的层级关系
- 返回所有菜单类型为C或者M的，状态为正常的，未被删除的菜单


```
{
    "code":200,
    "data":{
        "menus":[
            {
                "component":"content/article/write/index",
                "createTime":"2022-01-08 03:39:58",
                "icon":"build",
                "id":"2023",
                "menuName":"写博文",
                "menuType":"C",
                "orderNum":0,
                "parentId":"0",
                "path":"write",
                "perms":"content:article:writer",
                "status":"0",
                "visible":"0",
                "children":[]
            },
            {
                "createTime":"2022-01-08 02:44:38",
                "icon":"table",
                "id":"2017",
                "menuName":"内容管理",
                "menuType":"M",
                "orderNum":4,
                "parentId":"0",
                "path":"content",
                "perms":"",
                "status":"0",
                "visible":"0",
                "children":[
                    {
                        "children":[],
                        "component":"content/article/index",
                        "createTime":"2022-01-08 02:53:10",
                        "icon":"build",
                        "id":"2019",
                        "menuName":"文章管理",
                        "menuType":"C",
                        "orderNum":0,
                        "parentId":"2017",
                        "path":"article",
                        "perms":"content:article:list",
                        "status":"0",
                        "visible":"0"
                    },
```

```
                    {
                        "children":[],
                        "component":"content/category/index",
                        "createTime":"2022-01-08 02:51:45",
                        "icon":"example",
                        "id":"2018",
                        "menuName":"分类管理",
                        "menuType":"C",
                        "orderNum":1,
                        "parentId":"2017",
                        "path":"category",
                        "perms":"content:category:list",
                        "status":"0",
                        "visible":"0"
                    },
                    {
                        "children":[],
                        "component":"content/tag/index",
                        "createTime":"2022-01-08 02:55:37",
                        "icon":"button",
                        "id":"2021",
                        "menuName":"标签管理",
                        "menuType":"C",
                        "orderNum":6,
                        "parentId":"2017",
                        "path":"tag",
                        "perms":"content:tag:index",
                        "status":"0",
                        "visible":"0"
                    }
                ]
            }
        ]
    },
    "msg":"操作成功"
}
```


4.4.3 代码实现

RoutersVo

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class RoutersVo {

    private List<Menu> menus;
}
```

在Menu中添加变量children，存储子菜单，由于表中不存在该字段，使用注解@TableField(exist = false)

```
@TableName("sys_menu")
@Data
@Accessors(chain = true)
public class Menu implements Serializable {

    ...

    @TableField(exist = false)
    private List<Menu> children;
}
```

AdminLoginController

```
@GetMapping("/getRouters")
public ResponseResult<RoutersVo> getRouters(){
    Long userId = SecurityUtils.getUserId();
    //查询menu，返回的menus以tree形式表示父子菜单的层级关系
    List<Menu> menus = menuService.selectRouterMenuTreeByUserId(userId);

    //封装数据返回
    return ResponseResult.okResult(new RoutersVo(menus));
}
```

IMenuService

```
List<Menu> selectRouterMenuTreeByUserId(Long userId);
```

MenuServiceImpl

```
@Override
public List<Menu> selectRouterMenuTreeByUserId(Long userId) {
    List<Menu> menus = null;
    //判断是否是管理员
    if(SecurityUtils.isAdmin()){
        //如果是，获取所有符合要求的Menu
        menus = menuMapper.selectAllRouterMenu();
    }else{
        //否则，获取当前用户所具有的Menu
        menus = menuMapper.selectRouterMenuByUserId(userId);
    }

    //构建tree
    //先找出一级菜单，然后去找他们的子菜单设置到children属性中
    List<Menu> menuTree = builderMenuTree(menus);
    return menuTree;
}

private List<Menu> builderMenuTree(List<Menu> menus) {
    List<Menu> menuTree = menus.stream()
        // 获取一级菜单
        .filter(menu -> menu.getParentId().equals(0L))
        // 查询并设置一级菜单下的子菜单
        .map(menu -> menu.setChildren(getChildren(menus, menu.getId())))
        .collect(Collectors.toList());
    return menuTree;
```

```
}

private List<Menu> getChildren(List<Menu> menus, Long menuId) {
    List<Menu> childrenList = menus.stream()
        .filter(m -> m.getParentId().equals(menuId))
        .collect(Collectors.toList());
    return childrenList;
}
```

MenuMapper.java

```
List<Menu> selectAllRouterMenu();

List<Menu> selectRouterMenuByUserId(Long userId);
```

MenuMapper.xml

```
<select id="selectAllRouterMenu" resultType="com.wh.blog.domain.entity.Menu">
    SELECT
    DISTINCT m.id, m.parent_id, m.menu_name, m.path, m.component, m.visible,
m.status, IFNULL(m.perms,'') AS perms, m.is_frame,  m.menu_type, m.icon,
m.order_num, m.create_time
    FROM
    sys_menu m
    WHERE
    m.menu_type IN ('C','M') AND
    m.status = 0 AND
    m.del_flag = 0
    ORDER BY
    m.parent_id,m.order_num
</select>

<select id="selectRouterMenuByUserId"
resultType="com.wh.blog.domain.entity.Menu">
    SELECT
    DISTINCT m.id, m.parent_id, m.menu_name, m.path, m.component, m.visible,
m.status, IFNULL(m.perms,'') AS perms, m.is_frame,  m.menu_type, m.icon,
m.order_num, m.create_time
    FROM
    sys_user_role ur
    LEFT JOIN sys_role_menu rm ON ur.role_id = rm.role_id
    LEFT JOIN sys_menu m ON m.id = rm.menu_id
    WHERE
    ur.user_id = #{userId} AND
    m.menu_type IN ('C','M') AND
    m.status = 0 AND
    m.del_flag = 0
    ORDER BY
    m.parent_id,m.order_num
</select>
```

4.5 后端权限控制

以后台的文章列表接口为例，对该接口做权限控制。

4.5.1 文章列表接口

请求方式	请求路径	是否需求token头
Get	/content/article/list	是

Query格式请求参数：

pageNum: 页码

pageSize: 每页条数

title：文章标题

summary：文章摘要

响应格式：

```
{
    "code": 200,
    "data": {
        "rows": [
            {
                "createTime": "2023-04-22 14:58:34",
                "id": "3",
                "summary": "适合初学者的Pycharm安装和使用教程",
                "thumbnail": "https://cdn2.byhy.net/imgs/gh/36257654_36417077-
e7c80698-1665-11e8-9b0d-fcc33fa1e34a.png",
                "title": "Pycharm的安装和使用",
                "viewCount": "115"
            }
        ],
        "total": "1"
    },
    "msg": "操作成功"
}
```

在admin模块中新建一个AdminArticleController

```
@RestController
@RequestMapping("/content/article")
public class AdminArticleController {

    @Autowired
    IArticleService articleService;

    @GetMapping("/list")
    public ResponseResult getArticleList(Integer pageNum, Integer pageSize,
String title, String summary) {

        ResponseResult articles = articleService.getAdminArticleList(pageNum,
pageSize, title, summary);
```

```
        return articles;
    }
}
```

IArticleService

```
ResponseResult getAdminArticleList(Integer pageNum, Integer pageSize, String
title, String summary);
```

ArticleServiceImpl

```
@Override
public ResponseResult getAdminArticleList(Integer pageNum, Integer pageSize,
String title, String summary) {
    //设置条件查询
    LambdaQueryWrapper<Article> queryWrapper = new LambdaQueryWrapper<>();
    queryWrapper.like(title != null, Article::getTitle, title)
        .like(summary != null, Article::getSummary, summary)
        .orderByDesc(Article::getIsTop)
        .orderByDesc(Article::getCreateTime);
    //设置分页查询
    Page<Article> articlePage = new Page<>(pageNum, pageSize);
    //查询文章列表
    articlePage = articleMapper.selectPage(articlePage, queryWrapper);
    //封装进VO
    List<ArticleListVo> adminArticleVoList =
BeanCopyUtils.copyBeanList(articlePage.getRecords(), ArticleListVo.class);
    PageVo pageVo = new PageVo(adminArticleVoList, articlePage.getTotal());
    return ResponseResult.okResult(pageVo);
}
```

4.5.2 实现后端权限控制

在SecurityConfig类上加上注解

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

LoginUser增加权限信息属性

```
private List<String> permissions;
```

UserDetailsServiceImpl，加入权限信息封装

```
@Autowired
private MenuMapper menuMapper;

@Override
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
    //根据用户名查询用户信息
    LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();
    queryWrapper.eq(User::getUserName,username);
    User user = userMapper.selectOne(queryWrapper);
```

```
        //判断是否查到用户  如果没查到抛出异常
        if(Objects.isNull(user)){
            throw new RuntimeException("用户不存在");
        }

        //查询权限信息，封装入用户信息返回
        List<String> permissions = menuMapper.selectPermsByUserId(user.getId());
        return new LoginUser(user, permissions);
    }
```

PermissionService，添加判断用户是否具有指定权限的方法

```
@Service
public class PermissionService {

    public boolean hasPermission(String permission){
        //如果是超级管理员，直接返回true
        if(SecurityUtils.isAdmin()){
            return true;
        }
        //否则，获取当前登录用户所具有的权限列表
        List<String> permissions =
SecurityUtils.getLoginUser().getPermissions();
        if(permissions == null)
            return false;
        return permissions.contains(permission);
    }
}
```

AdminArticleController，在需要权限控制的接口方法上添加@PreAuthorize注解

```
@GetMapping("/list")
@PreAuthorize("@permissionService.hasPermission('content:article:list')")
public ResponseResult getArticleList(...) {...}
```

4.6 退出登录接口

4.6.1 接口设计

请求方式	请求地址	请求头
POST	/user/logout	需要token请求头

响应格式:

```
{
    "code": 200,
    "msg": "操作成功"
}
```

4.6.2 代码实现

AdminLoginController

```
@PostMapping("/user/logout")
public ResponseResult logout(){
    return adminLoginService.logout();
}
```

IAdminLoginService

```
ResponseResult logout();
```

AdminLoginServiceImpl

```
@Override
public ResponseResult logout() {
    //获取当前登录的用户id
    Long userId = SecurityUtils.getUserId();
    //删除redis中对应的值
    redisCache.deleteObject("adminlogin:"+userId);
    return ResponseResult.okResult();
}
```