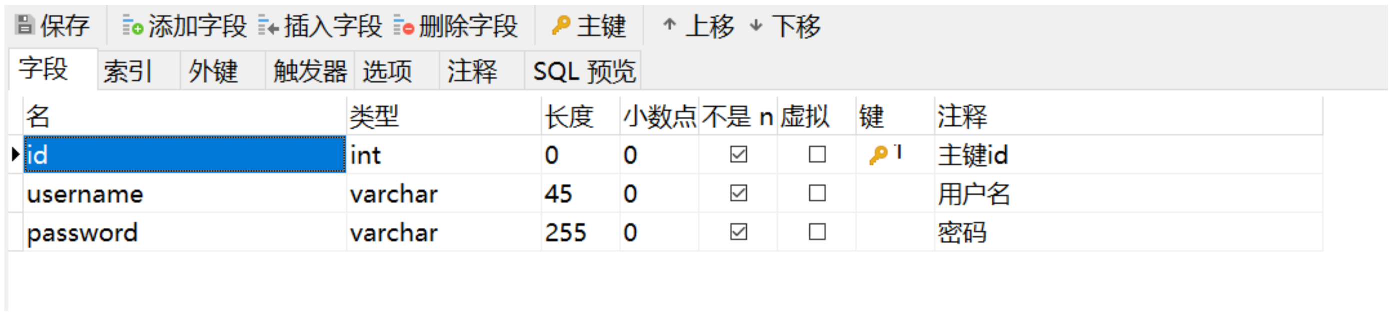
权限

> *管理台因为有订单相关的管理，可能会涉及的退款操作，所以管理台必须加权限，不是所有人都能进行操作。*
>
> *权限仍旧使用spring security完成。*
>
> *一个用户可以拥有多个角色，一个角色可以拥有多个权限，一个角色可以拥有多个菜单*
>
> *涉及到用户表，角色表，权限表，菜单表，以及用户角色关联表，角色权限关联表，角色菜单关联表*

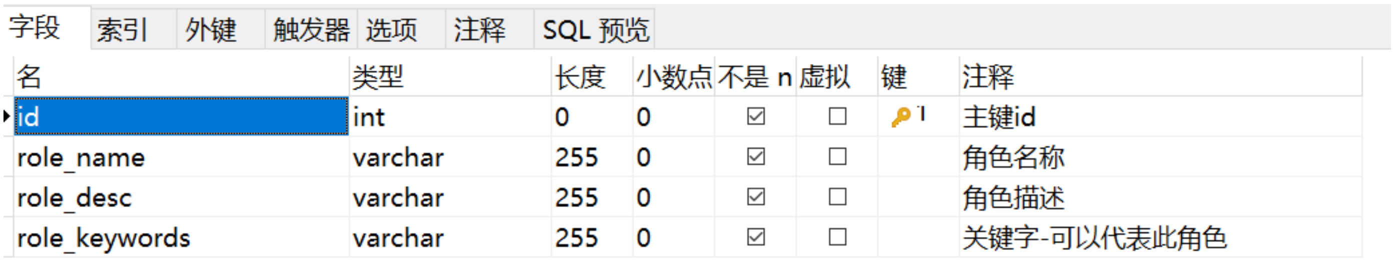
1. 数据库设计

用户表： 之前已经做过登录



名	类型	长度	小数点	不是 null	虚拟	键	注释
id	int	0	0	☑	☐	🔑1	主键id
username	varchar	45	0	☑	☐		用户名
password	varchar	255	0	☑	☐		密码

角色表:

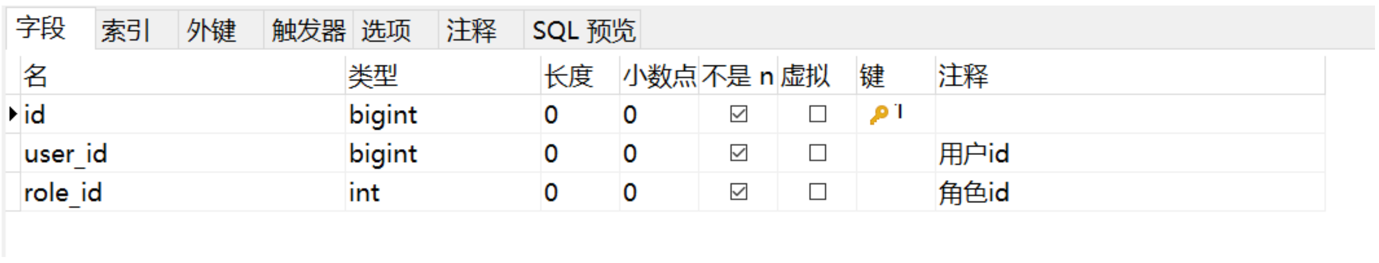


名	类型	长度	小数点	不是 null	虚拟	键	注释
id	int	0	0	☑	☐	🔑1	主键id
role_name	varchar	255	0	☑	☐		角色名称
role_desc	varchar	255	0	☑	☐		角色描述
role_keywords	varchar	255	0	☑	☐		关键字-可以代表此角色

```
CREATE TABLE `xt`.`t_admin_role`  (
  `id` int(0) NOT NULL AUTO_INCREMENT COMMENT '主键id',
  `role_name` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '角色名称',
  `role_desc` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '角色描述',
  `role_keywords` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '关键字-可以代表此角色',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci
ROW_FORMAT = Dynamic;
```

```java
package com.mszlu.xt.admin.dao.data;

import lombok.Data;

@Data
public class AdminRole {

    private Integer id;

    private String roleName;

    private String roleDesc;

    private String roleKeywords;
}
```

用户角色关联表:



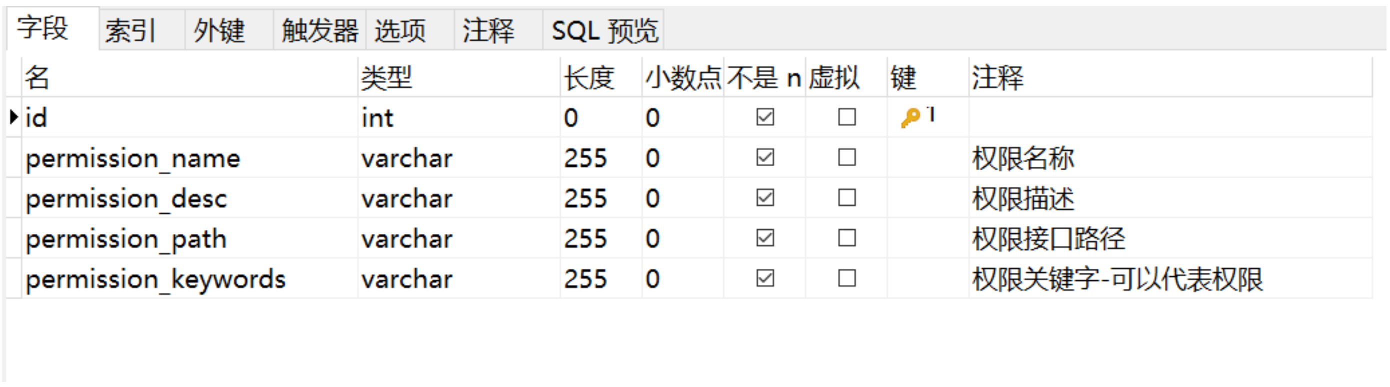
字段 | 索引 | 外键 | 触发器 | 选项 | 注释 | SQL 预览

名	类型	长度	小数点	不是 n	虚拟	键	注释
id	bigint	0	0	☑	☐	🔑1	
user_id	bigint	0	0	☑	☐		用户id
role_id	int	0	0	☑	☐		角色id

```
CREATE TABLE `xt`.`t_admin_user_role`  (
  `id` bigint(0) NOT NULL AUTO_INCREMENT,
  `user_id` bigint(0) NOT NULL COMMENT '用户id',
  `role_id` int(0) NOT NULL COMMENT '角色id',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci
ROW_FORMAT = Dynamic;
```

```
package com.mszlu.xt.admin.dao.data;

import lombok.Data;

@Data
public class AdminUserRole {

    private Long id;

    private Long userId;

    private Integer roleId;

}
```

权限表:



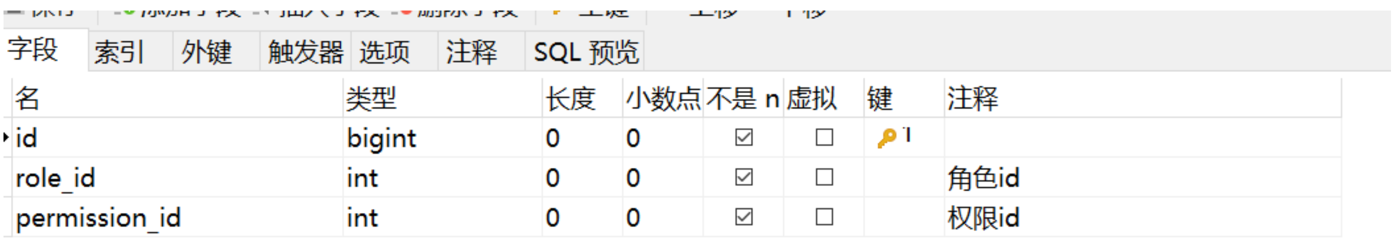
字段 索引 外键 触发器 选项 注释 SQL 预览

名	类型	长度	小数点	不是 n	虚拟	键	注释
id	int	0	0	☑	☐	🔑1	
permission_name	varchar	255	0	☑	☐		权限名称
permission_desc	varchar	255	0	☑	☐		权限描述
permission_path	varchar	255	0	☑	☐		权限接口路径
permission_keywords	varchar	255	0	☑	☐		权限关键字-可以代表权限

```sql
CREATE TABLE `xt`.`t_admin_permission`  (
  `id` int(0) NOT NULL AUTO_INCREMENT,
  `permission_name` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '权限名称',
  `permission_desc` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '权限描述',
  `permission_path` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '权限接口路径',
  `permission_keywords` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '权限关键字-可以代表权限',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci
ROW_FORMAT = Dynamic;
```

```java
package com.mszlu.xt.admin.dao.data;

import lombok.Data;

@Data
public class AdminPermission {

    private Integer id;

    private String permissionName;

    private String permissionDesc;

    private String permissionPath;

    private String permissionKeywords;
}
```

角色权限关联表:

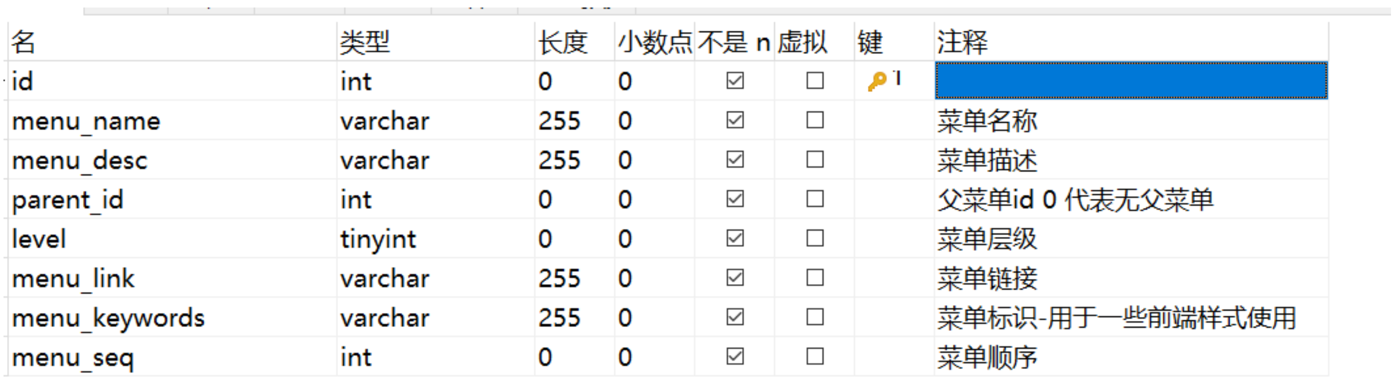


名	类型	长度	小数点	不是 n	虚拟	键	注释
id	bigint	0	0	☑	☐	🔑 1	
role_id	int	0	0	☑	☐		角色id
permission_id	int	0	0	☑	☐		权限id

```sql
CREATE TABLE `xt`.`t_admin_role_permission`  (
  `id` bigint(0) NOT NULL AUTO_INCREMENT,
  `role_id` int(0) NOT NULL COMMENT '角色id',
  `permission_id` int(0) NOT NULL COMMENT '权限id',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci
ROW_FORMAT = Dynamic;
```

```java
package com.mszlu.xt.admin.dao.data;

import lombok.Data;

@Data
public class AdminRolePermission {

    private Long id;

    private Integer roleId;

    private Integer permissionId;
}
```

菜单表：

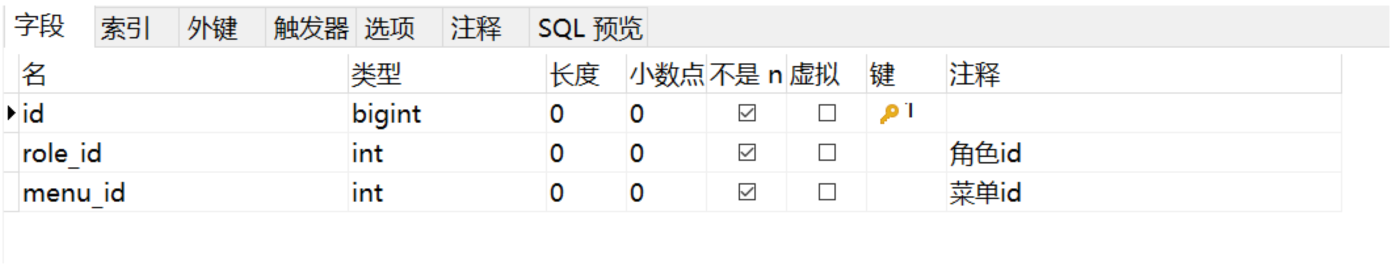


名	类型	长度	小数点	不是 n	虚拟	键	注释
id	int	0	0	☑	☐	🔑1	
menu_name	varchar	255	0	☑	☐		菜单名称
menu_desc	varchar	255	0	☑	☐		菜单描述
parent_id	int	0	0	☑	☐		父菜单id 0 代表无父菜单
level	tinyint	0	0	☑	☐		菜单层级
menu_link	varchar	255	0	☑	☐		菜单链接
menu_keywords	varchar	255	0	☑	☐		菜单标识-用于一些前端样式使用
menu_seq	int	0	0	☑	☐		菜单顺序

```sql
CREATE TABLE `xt`.`t_admin_menu`  (
  `id` int(0) NOT NULL AUTO_INCREMENT,
  `menu_name` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '菜单名称',
  `menu_desc` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '菜单描述',
  `parent_id` int(0) NOT NULL COMMENT '父菜单id 0 代表无父菜单',
  `level` tinyint(0) NOT NULL COMMENT '菜单层级',
  `menu_link` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '菜单链接',
  `menu_keywords` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL COMMENT '菜单标识-用于一些前端样式使用',
  `menu_seq` int(0) NOT NULL COMMENT '菜单顺序',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci
ROW_FORMAT = Dynamic;
```

```java
package com.mszlu.xt.admin.dao.data;

import lombok.Data;

@Data
public class AdminMenu {

    private Integer id;

    private String menuName;

    private String menuDesc;

    private Integer parentId;

    private Integer level;

    private String menuLink;

    private String menuKeywords;

    private Integer menuSeq;
}
```

角色菜单关联表：



名	类型	长度	小数点	不是 null	虚拟	键	注释
id	bigint	0	0	☑	☐	🔑1	
role_id	int	0	0	☑	☐		角色id
menu_id	int	0	0	☑	☐		菜单id

```sql
CREATE TABLE `xt`.`t_admin_role_menu`  (
  `id` bigint(0) NOT NULL AUTO_INCREMENT,
  `role_id` int(0) NOT NULL COMMENT '角色id',
  `menu_id` int(0) NOT NULL COMMENT '菜单id',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci
ROW_FORMAT = Dynamic;
```

```java
package com.mszlu.xt.admin.dao.data;

import lombok.Data;

@Data
public class AdminRoleMenu {

    private Long id;

    private Integer roleId;

    private Integer menuId;

}
```

2. 实现权限

> *通过用户请求的接口路径来进行匹配权限，先去查询用户所拥有的所有权限，然后进行比对*

2.1 配置需要权限认证的接口

```
package com.mszlu.xt.admin.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.A
uthenticationManagerBuilder;
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.WebSecu
rityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    public PasswordEncoder getPasswordEncoder(){
        return new BCryptPasswordEncoder();
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.authorizeRequests()
                .antMatchers("/login.html").permitAll()
                .antMatchers("/css/**").permitAll()
                .antMatchers("/js/**").permitAll()
                .antMatchers("/plugins/**").permitAll()
                .antMatchers(
                        "/course/**",
                        "/news/**",
                        "/subject/**",
                        "/topic/**",

 "/order/**").access("@authService.auth(request,authentication)")
                .anyRequest().authenticated()
                .and().headers().frameOptions().disable()
                .and()
                .formLogin()
                .defaultSuccessUrl("/pages/main.html")
                .permitAll()
                .and().logout()
                .and().csrf().disable()
                ;
//          super.configure(http);
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        super.configure(auth);
    }

    @Override
    public void configure(WebSecurity web) throws Exception {
        super.configure(web);
    }
}
```

2.2 认证实现

逻辑：

1. 判断如果是匿名，未登录
2. 已登录，根据用户名查询用户
3. 根据用户id查询角色
4. 根据角色id 查询权限
5. 拿到权限设置的path路径和请求路径进行匹配

```
package com.mszlu.xt.admin.security;

import com.mszlu.common.model.CallResult;
import com.mszlu.xt.admin.dao.data.AdminRole;
import com.mszlu.xt.admin.dao.data.AdminUser;
import com.mszlu.xt.admin.service.AdminUserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;
```

```java
import javax.servlet.http.HttpServletRequest;

@Component
public class AuthService {

    @Value("${server.servlet.context-path}")
    private String contextPath;
    @Autowired
    private AdminUserService adminUserService;

    public boolean auth(HttpServletRequest request, Authentication
authentication){
        Object principal = authentication.getPrincipal();
        if ("anonymousUser".equals(principal)){
            //未登录
            return false;
        }
        UserDetails user = (UserDetails) principal;
        String username = user.getUsername();
        CallResult callResult = adminUserService.findUser(username);
        if (callResult.isSuccess()){
            AdminUser adminUser = (AdminUser) callResult.getResult();
            String requestURI = request.getRequestURI();
            //由于 路径有 前缀 需要先将 requestURI 去除前缀 即 去除 /lzadmin
            requestURI = requestURI.replace(contextPath,"");
            return adminUserService.auth(requestURI,adminUser.getId());
        }
        return false;
    }
}
```

```
@Override
    public boolean auth(String requestURI, Long adminUserId) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(null);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Boolean>() {
            @Override
            public CallResult<Boolean> doAction() {
                return adminUserDomain.auth(requestURI,adminUserId);
            }
        }).getResult();
    }
```

```
public CallResult<Boolean> auth(String requestURI, Long adminUserId) {
        //根据用户id 查询角色
        List<Integer> adminRoleIdList =
this.adminUserDomainRepository.findAdminRoleIdListByUserId(adminUserId);
        if (adminRoleIdList.size() <= 0){
            return CallResult.fail(false);
        }
        //根据角色 查询权限
        List<AdminPermission> permissions =
this.adminUserDomainRepository.findPermissionListByRoleIds(adminRoleIdLi
st);
        for (AdminPermission permission : permissions) {
            if (new
AntPathMatcher().match(permission.getPermissionPath(),requestURI)){
                return CallResult.success(true);
            }
        }
        return CallResult.fail(false);
    }
```

```
public List<Integer> findAdminRoleIdListByUserId(Long adminUserId) {
        LambdaQueryWrapper<AdminUserRole> queryWrapper =
Wrappers.lambdaQuery();
        queryWrapper.eq(AdminUserRole::getUserId, adminUserId);
        queryWrapper.select(AdminUserRole::getRoleId);
        List<AdminUserRole> adminUserRoleList =
this.adminUserRoleMapper.selectList(queryWrapper);
        List<Integer> roleIdList =
adminUserRoleList.stream().map(AdminUserRole::getRoleId).collect(Collect
ors.toList());
```

```java
        return roleIdList;
    }

    public List<AdminPermission>
findPermissionListByRoleIds(List<Integer> adminRoleIdList) {
        LambdaQueryWrapper<AdminRolePermission> queryWrapper =
Wrappers.lambdaQuery();
        queryWrapper.in(AdminRolePermission::getRoleId,adminRoleIdList);
        queryWrapper.select(AdminRolePermission::getPermissionId);
        List<AdminRolePermission> adminRolePermissions =
this.adminRolePermissionMapper.selectList(queryWrapper);
        List<Integer> permissionIdList =
adminRolePermissions.stream().map(AdminRolePermission::getPermissionId).
collect(Collectors.toList());

        if (permissionIdList.size() <= 0){
            return new ArrayList<>();
        }
        LambdaQueryWrapper<AdminPermission> queryWrapper1 =
Wrappers.lambdaQuery();
        queryWrapper1.in(AdminPermission::getId,permissionIdList);
        queryWrapper1.select(AdminPermission::getPermissionPath);
        List<AdminPermission> adminPermissions =
this.adminPermissionMapper.selectList(queryWrapper1);
        return adminPermissions;
    }
```

mapper:

```java
package com.mszlu.xt.admin.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mszlu.xt.admin.dao.data.AdminPermission;
import com.mszlu.xt.admin.dao.data.AdminRole;

public interface AdminPermissionMapper extends
BaseMapper<AdminPermission> {
}



package com.mszlu.xt.admin.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mszlu.xt.admin.dao.data.AdminRole;
import com.mszlu.xt.admin.dao.data.AdminRolePermission;

public interface AdminRolePermissionMapper extends
BaseMapper<AdminRolePermission> {
}

package com.mszlu.xt.admin.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mszlu.xt.admin.dao.data.AdminRole;
import com.mszlu.xt.admin.dao.data.AdminUser;

public interface AdminRoleMapper extends BaseMapper<AdminRole> {
}


package com.mszlu.xt.admin.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mszlu.xt.admin.dao.data.AdminUserRole;

public interface AdminUserRoleMapper extends BaseMapper<AdminUserRole> {
}
```

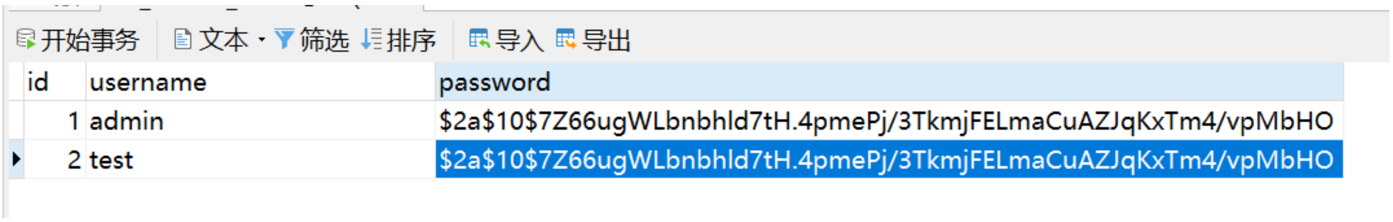
2.3 前端添加无权限提示

```
 axios.post("/lzadmin/news/findPage",params).then((res)=>{
                        if (res.data.success){
                            this.dataList = res.data.result.list;
                            this.pagination.total =
res.data.result.total;
                        }
                    }).catch((err)=>{
                        if ('Error: Request failed with status code 403'
== err){
                            this.$message({
                                message: '无权限，请联系管理员',
                                type: 'error'
                            });
                        }
                    });
```

> *前端的每一个请求接口 都需要加入以上判断，这样无权限时 就会进行提示*

3. 测试权限

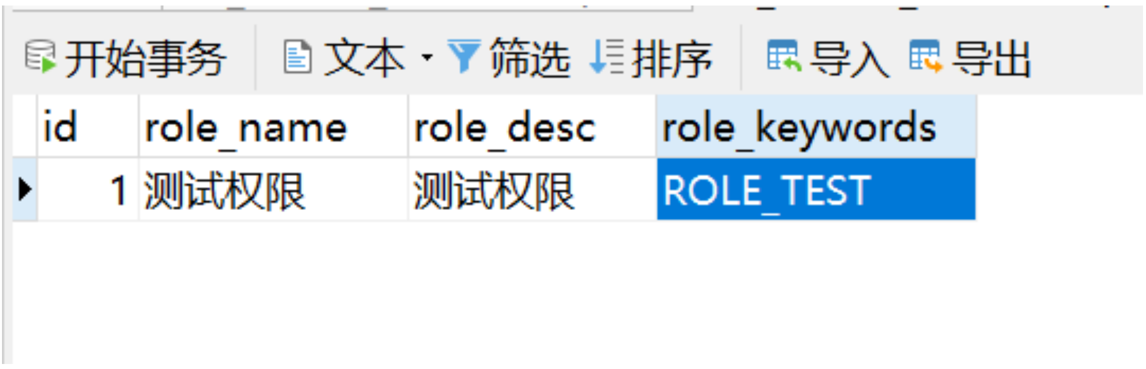
添加一个用户2，用户名为 test



开始事务 文本 · 筛选 排序 导入 导出

id	username	password
1	admin	$2a$10$7Z66ugWLbnbhld7tH.4pmePj/3TkmjFELmaCuAZJqKxTm4/vpMbHO
2	test	$2a$10$7Z66ugWLbnbhld7tH.4pmePj/3TkmjFELmaCuAZJqKxTm4/vpMbHO

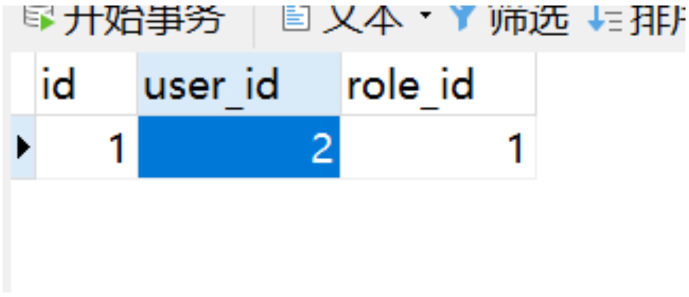
添加一个测试角色



开始事务 文本 · 筛选 排序 导入 导出

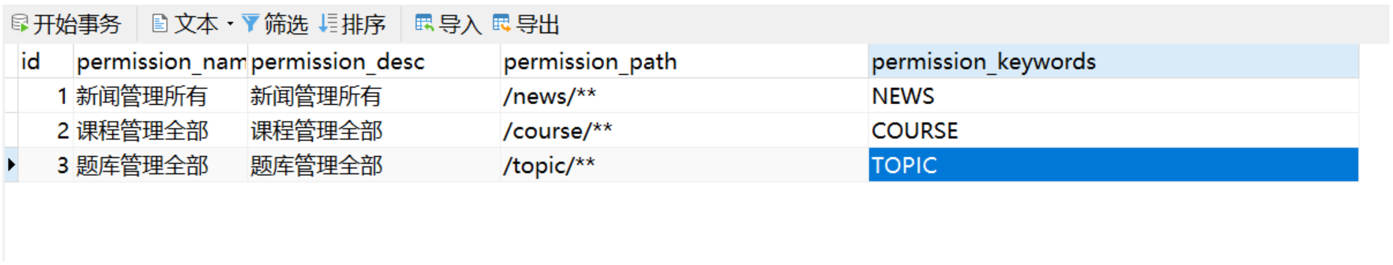
id	role_name	role_desc	role_keywords
1	测试权限	测试权限	ROLE_TEST

用户2 拥有此 测试权限的角色



id	user_id	role_id
1	2	1

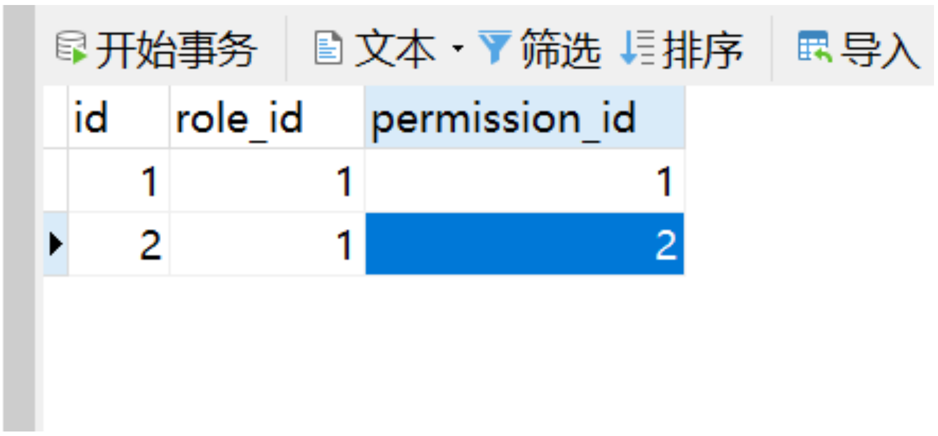
设置权限：



开始事务 文本 · 筛选 排序 导入 导出

id	permission_nam	permission_desc	permission_path	permission_keywords
1	新闻管理所有	新闻管理所有	/news/**	NEWS
2	课程管理全部	课程管理全部	/course/**	COURSE
3	题库管理全部	题库管理全部	/topic/**	TOPIC

测试权限的 角色 拥有 1,2的权限，即 新闻管理和课程管理的权限



id	role_id	permission_id
1	1	1
2	1	2

登录用户test 进行测试

4. 角色管理

在main.html的菜单列表中加入一下配置：

```
{
                    "path": "5",
                    "title": "系统管理",
                    "icon":"fa-tty",
                    "children": [
                        {
                            "path": "/5-1",
                            "title": "角色管理",
                            "linkUrl":"role.html",
                            "children":[]
                        },
                        {
                            "path": "/5-2",
                            "title": "权限管理",
                            "linkUrl":"coupon.html",
                            "children":[]
                        },
                        {
                            "path": "/5-3",
                            "title": "用户管理",
                            "linkUrl":"user.html",
                            "children":[]
                        },
                    ]
                }
```

将资料中的role.html拷贝到pages下

4.1 分页查询

```java
package com.mszlu.xt.admin.model.param;

import lombok.Data;

import java.util.List;

@Data
public class AdminUserParam {

    private String username;

    private String password;

    private String newPassword;

    private int page;

    private int pageSize;

    private Long id;

    private Integer roleId;

    private Integer permissionId;

    private Integer menuId;

    private String roleName;

    private String roleDesc;

    private String roleKeywords;

    private List<Integer> permissionIdList;

    private String permissionName;

    private String permissionDesc;

    private String permissionKeywords;

    private String permissionPath;

    private List<Integer> roleIdList;

    private List<Integer> menuIdList;

    private String menuName;

    private String menuDesc;

    private String menuKeywords;

    private Integer parentId;

    private Integer level;

    private String menuLink;

    private Integer menuSeq;
}
```

```java
package com.mszlu.xt.admin.controller;

import com.mszlu.common.model.CallResult;
import com.mszlu.xt.admin.model.param.AdminUserParam;
import com.mszlu.xt.admin.model.param.CourseParam;
import com.mszlu.xt.admin.service.AdminUserService;
import com.mszlu.xt.admin.service.CourseService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @author Jarno
 */
@RestController
@RequestMapping("user")
public class AdminUserController {
    @Autowired
    private AdminUserService adminUserService;

    @RequestMapping(value = "role/findRolePage")
    public CallResult findRolePage(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.findRolePage(adminUserParam);
    }
}
```

```java
 @Override
    public CallResult findRolePage(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.findRolePage();
            }
        });
    }
```

```java
public CallResult<Object> findRolePage() {
        int page = this.adminUserParam.getPage();
        int pageSize = this.adminUserParam.getPageSize();
        Page<AdminRole> adminRolePage =
this.adminUserDomainRepository.findRoleList(page,pageSize);
        ListModel listModel = new ListModel();
        listModel.setTotal((int) adminRolePage.getTotal());
        List<AdminRole> result = adminRolePage.getRecords();
        listModel.setList(result);
        return CallResult.success(listModel);
    }
```

```java
public Page<AdminRole> findRoleList(int page, int pageSize) {
        return this.adminRoleMapper.selectPage(new Page<>
(page,pageSize),Wrappers.lambdaQuery());
    }
```

4.2 新增角色

> *新增角色的时候，需要选择对应的权限*

```
//角色添加
@RequestMapping(value = "role/add")
    public CallResult add(@RequestBody AdminUserParam adminUserParam){
        return adminUserService.add(adminUserParam);
    }
```

```
  //查询所有权限
    @RequestMapping(value = "permission/all")
    public CallResult permissionAll(){
        return adminUserService.permissionAll();
    }
```

```
@Override
    @Transactional
    public CallResult add(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain = 
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.execute(new 
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.add();
            }
        });
    }
@Override
    public CallResult permissionAll() {
        AdminUserDomain adminUserDomain = 
this.adminUserDomainRepository.createDomain(null);
        return this.serviceTemplate.executeQuery(new 
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.permissionAll();
            }
        });
    }
```

Domain:

```
public CallResult<Object> add() {
        AdminRole role = new AdminRole();
        BeanUtils.copyProperties(this.adminUserParam,role);
        List<Integer> permissionIdList =
this.adminUserParam.getPermissionIdList();
        this.adminUserDomainRepository.saveRole(role);
        Integer roleId = role.getId();

 this.adminUserDomainRepository.saveRolePermission(roleId,permissionIdLi
st);
        return CallResult.success();
    }

    public CallResult<Object> permissionAll() {
        List<AdminPermission> allPermission =
this.adminUserDomainRepository.findAllPermission();
        return CallResult.success(allPermission);
    }
```

```
 public void saveRole(AdminRole role) {
        this.adminRoleMapper.insert(role);
    }

    public void saveRolePermission(Integer roleId, List<Integer>
permissionIdList) {
        for (Integer permissionId : permissionIdList) {
            AdminRolePermission adminRolePermission = new
AdminRolePermission();
            adminRolePermission.setRoleId(roleId);
            adminRolePermission.setPermissionId(permissionId);
            this.adminRolePermissionMapper.insert(adminRolePermission);
        }
    }

    public List<AdminPermission> findAllPermission() {
       return
this.adminPermissionMapper.selectList(Wrappers.lambdaQuery());
    }
```

4.3 编辑角色

4.3.1 回显数据

```
  @RequestMapping(value = "role/findRoleById")
    public CallResult findRoleById(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.findRoleById(adminUserParam);
    }
```

```
   @Override
    public CallResult findRoleById(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.findRoleById();
            }
        });
    }
```

Domain:

```
 public CallResult<Object> findRoleById() {
        Integer roleId = this.adminUserParam.getRoleId();
        //查询角色
        AdminRole role =
this.adminUserDomainRepository.findRoleId(roleId);
        //根据id查询选中的权限id列表
        List<Integer> permissionIdList =
this.adminUserDomainRepository.findPermissionIdListByRoleId(roleId);
        Map<String,Object> result = new HashMap<>();
        result.put("role",role);
        result.put("permissionIdList",permissionIdList);
        return CallResult.success(result);
    }
 public AdminRole findRoleId(Integer roleId) {
        return this.adminRoleMapper.selectById(roleId);
    }

    public List<Integer> findPermissionIdListByRoleId(Integer roleId) {
        LambdaQueryWrapper<AdminRolePermission> queryWrapper =
Wrappers.lambdaQuery();
        queryWrapper.eq(AdminRolePermission::getRoleId,roleId);
        List<AdminRolePermission> adminRolePermissions =
this.adminRolePermissionMapper.selectList(queryWrapper);
        return
adminRolePermissions.stream().map(AdminRolePermission::getPermissionId).
collect(Collectors.toList());
    }
```

4.3.2 编辑

> *编辑和新增类似，只不过在更新的时候，需要将原有的角色和权限的关联关系先删除，在新增*

```
 @RequestMapping(value = "role/update")
    public CallResult updateRole(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.updateRole(adminUserParam);
    }
```

```
@Override
    @Transactional
    public CallResult updateRole(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.execute(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.updateRole();
            }
        });
    }
```

Domain:

```
 public CallResult<Object> updateRole() {
        AdminRole role = new AdminRole();
        BeanUtils.copyProperties(this.adminUserParam,role);
        role.setId(this.adminUserParam.getRoleId());
        List<Integer> permissionIdList =
this.adminUserParam.getPermissionIdList();
        this.adminUserDomainRepository.updateRole(role);
        Integer roleId = role.getId();
        //先删除关联关系

 this.adminUserDomainRepository.deleteRolePermissionByRoleId(roleId);

 this.adminUserDomainRepository.saveRolePermission(roleId,permissionIdLi
st);
        return CallResult.success();
    }
```

```
 public void updateRole(AdminRole role) {
        this.adminRoleMapper.updateById(role);
    }

    public void deleteRolePermissionByRoleId(Integer roleId) {
        LambdaQueryWrapper<AdminRolePermission> queryWrapper =
Wrappers.lambdaQuery();
        queryWrapper.eq(AdminRolePermission::getRoleId,roleId);
        this.adminRolePermissionMapper.delete(queryWrapper);
    }
```

5. 权限管理

> *将资料中的permission.html拷贝到pages目录，同时修改main.html中权限管理的映射*

5.1 分页查询

```
@RequestMapping(value = "permission/findPermissionPage")
    public CallResult findPermissionPage(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.findPermissionPage(adminUserParam);
    }
```

```java
@Override
    public CallResult findPermissionPage(AdminUserParam adminUserParam)
{
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.findPermissionPage();
            }
        });
    }
```

```java
 public CallResult<Object> findPermissionPage() {
        int page = this.adminUserParam.getPage();
        int pageSize = this.adminUserParam.getPageSize();
        Page<AdminPermission> adminPermissionPage =
this.adminUserDomainRepository.findPermissionList(page,pageSize);
        ListModel listModel = new ListModel();
        listModel.setTotal((int) adminPermissionPage.getTotal());
        List<AdminPermission> result = adminPermissionPage.getRecords();
        listModel.setList(result);
        return CallResult.success(listModel);
    }
```

```java
public Page<AdminPermission> findPermissionList(int page, int pageSize)
{
        return this.adminPermissionMapper.selectPage(new Page<>
(page,pageSize),Wrappers.lambdaQuery());
    }
```

5.2 新增权限

```java
 @RequestMapping(value = "permission/add")
    public CallResult addPermission(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.addPermission(adminUserParam);
    }
 @Override
    public CallResult addPermission(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.execute(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.addPermission();
            }
        });
    }
```

```
public CallResult<Object> addPermission() {
        AdminPermission adminPermission = new AdminPermission();
        BeanUtils.copyProperties(adminUserParam,adminPermission);
        this.adminUserDomainRepository.savePermission(adminPermission);
        return CallResult.success();
    }
```

```
public void savePermission(AdminPermission adminPermission) {
        this.adminPermissionMapper.insert(adminPermission);
    }
```

5.3 编辑权限

5.3.1 回显

```
@RequestMapping(value = "permission/findPermissionById")
    public CallResult findPermissionById(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.findPermissionById(adminUserParam);
    }
```

```
 @Override
    public CallResult findPermissionById(AdminUserParam adminUserParam)
{
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.findPermissionById();
            }
        });
    }
```

```
public CallResult<Object> findPermissionById() {
        Integer permissionId = this.adminUserParam.getPermissionId();
       AdminPermission adminPermission =
this.adminUserDomainRepository.findPermissionById(permissionId);
        return CallResult.success(adminPermission);
    }
```

```
public AdminPermission findPermissionById(Integer permissionId) {
        return adminPermissionMapper.selectById(permissionId);
    }
```

5.3.2 编辑

```
@RequestMapping(value = "permission/update")
    public CallResult updatePermission(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.updatePermission(adminUserParam);
    }
```

```
 @Override
    public CallResult updatePermission(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.execute(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.updatePermission();
            }
        });
    }
```

```
 public CallResult<Object> updatePermission() {
        AdminPermission adminPermission = new AdminPermission();
        BeanUtils.copyProperties(adminUserParam,adminPermission);
        adminPermission.setId(adminUserParam.getPermissionId());

 this.adminUserDomainRepository.updatePermission(adminPermission);
        return CallResult.success();
    }
```

```
public void updatePermission(AdminPermission adminPermission) {
        this.adminPermissionMapper.updateById(adminPermission);
    }
```

6. 管理用户管理

> *将资料中的user.html拷贝到pages目录下，修改main.html中用户管理的对应链接*

6.1 分页查询

```
@RequestMapping(value = "findPage")
    public CallResult findPage(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.findPage(adminUserParam);
    }
```

```
 @Override
    public CallResult findPage(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.findPage();
            }
        });
    }
```

```
 public CallResult<Object> findPage() {
        int page = this.adminUserParam.getPage();
        int pageSize = this.adminUserParam.getPageSize();
        Page<AdminUser> adminUserPage =
this.adminUserDomainRepository.findUserList(page,pageSize);
        ListModel listModel = new ListModel();
        listModel.setTotal((int) adminUserPage.getTotal());
        List<AdminUser> result = adminUserPage.getRecords();
        listModel.setList(result);
        return CallResult.success(listModel);
    }
```

```
public Page<AdminUser> findUserList(int page, int pageSize) {
        return adminUserMapper.selectPage(new Page<>
(page,pageSize),Wrappers.lambdaQuery());
    }
```

6.2 新增用户

查询所有的角色

```
 @RequestMapping(value = "role/all")
    public CallResult roleAll(){
        return adminUserService.roleAll();
    }
```

```
@Override
    public CallResult roleAll() {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(null);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.roleAll();
            }
        });
    }
```

```
   public CallResult<Object> roleAll() {
        List<AdminRole> adminRoleList =
this.adminUserDomainRepository.findAllRole();
        return CallResult.success(adminRoleList);
    }
```

```
public List<AdminRole> findAllRole() {
        return adminRoleMapper.selectList(Wrappers.lambdaQuery());
    }
```

新增:

```
 @RequestMapping(value = "add")
    public CallResult add(@RequestBody AdminUserParam adminUserParam){
        return adminUserService.addUser(adminUserParam);
    }
```

```java
@Override
    @Transactional
    public CallResult addUser(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.execute(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.addUser();
            }
        });
    }
```

```java
public CallResult<Object> addUser() {
        AdminUser adminUser = new AdminUser();
        adminUser.setUsername(this.adminUserParam.getUsername());
        adminUser.setPassword(new
BCryptPasswordEncoder().encode(this.adminUserParam.getPassword()));
        this.adminUserDomainRepository.saveUser(adminUser);
        List<Integer> roleIdList = this.adminUserParam.getRoleIdList();
        for (Integer roleId : roleIdList) {

 this.adminUserDomainRepository.saveUserRole(adminUser.getId(),roleId);
        }
        return CallResult.success();
    }
```

```java
public void saveUserRole(Long userId, Integer roleId) {
        AdminUserRole adminUserRole = new AdminUserRole();
        adminUserRole.setRoleId(roleId);
        adminUserRole.setUserId(userId);
        this.adminUserRoleMapper.insert(adminUserRole);
    }
```

6.3 编辑

6.3.1 回显

```

    @RequestMapping(value = "findUserById")
    public CallResult findUserById(@RequestBody AdminUserParam 
adminUserParam){
        return adminUserService.findUserById(adminUserParam);
    }

```

```
@Override
    public CallResult findUserById(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain = 
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.executeQuery(new 
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.findUserById();
            }
        });
    }
```

```
public CallResult<Object> findUserById() {
        AdminUser adminUser = 
this.adminUserDomainRepository.findUserById(this.adminUserParam.getId())
;
        List<Integer> adminRoleIdListByUserId = 
this.adminUserDomainRepository.findAdminRoleIdListByUserId(this.adminUse
rParam.getId());
        Map<String,Object> result = new HashMap<>();
        result.put("user",adminUser);
        result.put("roleIdList",adminRoleIdListByUserId);
        return CallResult.success(result);
    }
```

```
public AdminUser findUserById(Long id) {
        return adminUserMapper.selectById(id);
    }
```

```
public List<Integer> findAdminRoleIdListByUserId(Long adminUserId) {
        LambdaQueryWrapper<AdminUserRole> queryWrapper =
Wrappers.lambdaQuery();
        queryWrapper.eq(AdminUserRole::getUserId, adminUserId);
        queryWrapper.select(AdminUserRole::getRoleId);
        List<AdminUserRole> adminUserRoleList =
this.adminUserRoleMapper.selectList(queryWrapper);
        List<Integer> roleIdList =
adminUserRoleList.stream().map(AdminUserRole::getRoleId).collect(Collect
ors.toList());
        return roleIdList;
    }
```

6.3.2 编辑

```
@RequestMapping(value = "update")
    public CallResult update(@RequestBody AdminUserParam adminUserParam)
{
        return adminUserService.update(adminUserParam);
    }
```

```
@Override
    @Transactional
    public CallResult update(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.execute(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.update();
            }
        });
    }
```

```
public CallResult<Object> update() {
        AdminUser adminUser = new AdminUser();
        adminUser.setUsername(this.adminUserParam.getUsername());
        String newPassword = this.adminUserParam.getNewPassword();
        if (StringUtils.isNotBlank(newPassword)) {
            adminUser.setPassword(new
BCryptPasswordEncoder().encode(this.adminUserParam.getNewPassword()));
        }
        adminUser.setId(this.adminUserParam.getId());
        this.adminUserDomainRepository.updateUser(adminUser);

 this.adminUserDomainRepository.deleteUserRoleByUserId(adminUser.getId()
);
        List<Integer> roleIdList = this.adminUserParam.getRoleIdList();
        for (Integer roleId : roleIdList) {

 this.adminUserDomainRepository.saveUserRole(adminUser.getId(),roleId);
        }
        return CallResult.success();
    }
```

```
public void updateUser(AdminUser adminUser) {
        this.adminUserMapper.updateById(adminUser);
    }

    public void deleteUserRoleByUserId(Long userId) {
        LambdaQueryWrapper<AdminUserRole> queryWrapper =
Wrappers.lambdaQuery();
        queryWrapper.eq(AdminUserRole::getUserId,userId);
        this.adminUserRoleMapper.delete(queryWrapper);
    }
```

7. 菜单

> *菜单和权限无关，和角色有关*
>
> *根据用户的角色，可以在前端决定是否展示某个菜单,但是菜单的展示与否和权限无必然关系*

7.1 分页查询

> *将资料中的menu.html拷贝到pages中，并在main.html中的系统管理添加菜单管理*

```java
   @RequestMapping(value = "menu/findMenuPage")
    public CallResult findMenuPage(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.findMenuPage(adminUserParam);
    }
```

```java
 @Override
    public CallResult findMenuPage(AdminUserParam adminUserParam) {

        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.findMenuPage();
            }
        });
    }
```

```java
public CallResult<Object> findMenuPage() {
        int page = this.adminUserParam.getPage();
        int pageSize = this.adminUserParam.getPageSize();
        Page<AdminMenu> adminMenuPage =
this.adminUserDomainRepository.findMenuPage(page,pageSize);
        ListModel listModel = new ListModel();
        listModel.setTotal((int) adminMenuPage.getTotal());
        List<AdminMenu> result = adminMenuPage.getRecords();
        listModel.setList(result);
        return CallResult.success(listModel);
    }
```

```java
public Page<AdminMenu> findMenuPage(int page, int pageSize) {
        return adminMenuMapper.selectPage(new Page<>
(page,pageSize),Wrappers.lambdaQuery());
    }
```

```
package com.mszlu.xt.admin.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mszlu.xt.admin.dao.data.AdminMenu;

public interface AdminMenuMapper extends BaseMapper<AdminMenu> {
}
```

7.2 新增菜单

> *弹出添加菜单框的时候，需要先查询所有的菜单，用于在选择父菜单的位置展示*

7.2.1 查询所有的菜单

```
@RequestMapping(value = "menu/all")
    public CallResult menuAll(){
        return adminUserService.menuAll();
    }
```

```
 @Override
    public CallResult menuAll() {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(null);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.menuAll();
            }
        });
    }
```

```
public CallResult<Object> menuAll() {
        List<AdminMenu> menuAll =
this.adminUserDomainRepository.findMenuAll();
        AdminMenu parent = new AdminMenu();
        parent.setId(0);
        parent.setLevel(0);
        parent.setMenuName("无父菜单");
        menuAll.add(parent);
        return CallResult.success(menuAll);
    }
```

```
public List<AdminMenu> findMenuAll() {
        return adminMenuMapper.selectList(Wrappers.lambdaQuery());
    }
```

7.2.2 新增

```
@RequestMapping(value = "menu/add")
    public CallResult saveMenu(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.saveMenu(adminUserParam);
    }
```

```
 @Override
    public CallResult saveMenu(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.execute(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.saveMenu();
            }
        });
    }
```

```
public CallResult<Object> saveMenu() {
        AdminMenu menu = new AdminMenu();
        BeanUtils.copyProperties(this.adminUserParam,menu);
        this.adminUserDomainRepository.saveMenu(menu);

        return CallResult.success();
    }
```

7.3 编辑菜单

7.3.1 回显数据

```
 @RequestMapping(value = "menu/findMenuById")
    public CallResult findMenuById(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.findMenuById(adminUserParam);
    }
```

```
CallResult findMenuById(AdminUserParam adminUserParam);
```

```
 @Override
    public CallResult findMenuById(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.findMenuById();
            }
        });
    }
```

```
public CallResult<Object> findMenuById() {
        AdminMenu menu =
this.adminUserDomainRepository.findMenuById(this.adminUserParam.getMenuI
d());

        return CallResult.success(menu);
    }
```

7.3.2 编辑

```
@RequestMapping(value = "menu/update")
    public CallResult updateMenu(@RequestBody AdminUserParam
adminUserParam){
        return adminUserService.updateMenu(adminUserParam);
    }
```

```
  @Override
    public CallResult updateMenu(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.execute(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.updateMenu();
            }
        });
    }
```

```
public CallResult<Object> updateMenu() {
        AdminMenu menu = new AdminMenu();
        BeanUtils.copyProperties(this.adminUserParam,menu);
        menu.setId(this.adminUserParam.getMenuId());
        this.adminUserDomainRepository.updateMenu(menu);

        return CallResult.success();
    }
```

```
public void updateMenu(AdminMenu menu) {
        adminMenuMapper.updateById(menu);
    }
```

8. 菜单动态获取

> *登录后，根据当前用户所拥有的角色来获取菜单*

main.html:

```
created:function () {
            axios.post("/lzadmin/user/userInfo").then((res)=>{
                this.username = res.data.result;
            });
            axios.post("/lzadmin/user/userMenuList").then((res)=>{
                this.menuList = res.data.result;
            });
        },
```

获取登录用户id:

在配置文件中，添加/user/userMenuList需要认证

```
.antMatchers(
                        "/course/**",
                        "/news/**",
                        "/subject/**",
                        "/topic/**",
                        "/order/**",

 "/user/userMenuList").access("@authService.auth(request,authentication)
")
                .anyRequest().authenticated()
```

```
package com.mszlu.xt.admin.security;

import com.mszlu.common.model.CallResult;
import com.mszlu.common.utils.UserThreadLocal;
import com.mszlu.xt.admin.dao.data.AdminRole;
import com.mszlu.xt.admin.dao.data.AdminUser;
import com.mszlu.xt.admin.service.AdminUserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;

@Component
public class AuthService {

    @Value("${server.servlet.context-path}")
    private String contextPath;
    @Autowired
    private AdminUserService adminUserService;

    public boolean auth(HttpServletRequest request, Authentication
authentication){
        Object principal = authentication.getPrincipal();
        if ("anonymousUser".equals(principal)){
            //未登录
            return false;
        }
        UserDetails user = (UserDetails) principal;
        String username = user.getUsername();
        CallResult callResult = adminUserService.findUser(username);
        if (callResult.isSuccess()){
            AdminUser adminUser = (AdminUser) callResult.getResult();
            UserThreadLocal.put(adminUser.getId());
            String requestURI = request.getRequestURI();
            //由于 路径有 前缀 需要先将 requestURI 去除前缀 即 去除 /lzadmin
            requestURI = requestURI.replace(contextPath,"");
            if (requestURI.equals("/user/userMenuList")){
                //不进行权限验证
                return true;
            }
            return adminUserService.auth(requestURI,adminUser.getId());
        }
        return false;
    }
}
```

8.1 实现接口

```
 @RequestMapping(value = "userMenuList")
    public CallResult userMenuList(){

        return adminUserService.userMenuList(new AdminUserParam());
    }
```

```
   @Override
    public CallResult userMenuList(AdminUserParam adminUserParam) {
        AdminUserDomain adminUserDomain =
this.adminUserDomainRepository.createDomain(adminUserParam);
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return adminUserDomain.userMenuList();
            }
        });
    }
```

```
public CallResult<Object> userMenuList() {
        Long userId = UserThreadLocal.get();
        List<Integer> roleIdList =
this.adminUserDomainRepository.findAdminRoleIdListByUserId(userId);
        List<AdminMenu> adminMenus =
this.adminUserDomainRepository.findAllMenuBySeqAndRoleId(roleIdList);
        //构建树形结构
        List<AdminMenuModel> adminMenuModelList = new ArrayList<>();
        for (AdminMenu adminMenu : adminMenus) {
            if (adminMenu.getLevel() == 1) {
                AdminMenuModel adminMenuModel = new AdminMenuModel();
                adminMenuModel.setId(adminMenu.getId());
                adminMenuModel.setLevel(adminMenu.getLevel());
                adminMenuModel.setIcon("fa-user-md");
                adminMenuModel.setTitle(adminMenu.getMenuName());

 adminMenuModel.setChildren(childrenMenu(adminMenuModel,adminMenus));
                adminMenuModelList.add(adminMenuModel);
            }
        }
        return CallResult.success(adminMenuModelList);
    }

    private List<AdminMenuModel> childrenMenu(AdminMenuModel
adminMenuModel, List<AdminMenu> adminMenus) {
        List<AdminMenuModel> adminMenuModelList = new ArrayList<>();
        if (adminMenuModel.getLevel() == 2){
            return adminMenuModelList;
        }
        for (AdminMenu adminMenu : adminMenus) {
            if (adminMenu.getLevel() == 1){
                continue;
            }
            if (adminMenu.getParentId().equals(adminMenuModel.getId())){
                AdminMenuModel amm = new AdminMenuModel();
                amm.setId(adminMenu.getId());
                amm.setLevel(adminMenu.getLevel());
                amm.setIcon("fa-user-md");
                amm.setLinkUrl(adminMenu.getMenuLink());
                amm.setTitle(adminMenu.getMenuName());
                amm.setChildren(childrenMenu(amm,adminMenus));
                adminMenuModelList.add(amm);
            }
        }
        return adminMenuModelList;
    }
```

```
@Resource
    private AdminRoleMenuMapper adminRoleMenuMapper;

    public List<AdminMenu> findAllMenuBySeqAndRoleId(List<Integer>
roleIdList) {
        if (roleIdList.isEmpty()){
            return new ArrayList<>();
        }
        LambdaQueryWrapper<AdminRoleMenu> queryWrapper =
Wrappers.lambdaQuery();
        queryWrapper.in(AdminRoleMenu::getRoleId,roleIdList);
        List<AdminRoleMenu> adminRoleMenus =
adminRoleMenuMapper.selectList(queryWrapper);
        List<Integer> menuIdList =
adminRoleMenus.stream().map(AdminRoleMenu::getMenuId).collect(Collectors
.toList());
        LambdaQueryWrapper<AdminMenu> queryWrapper1 =
Wrappers.lambdaQuery();
        queryWrapper1.in(AdminMenu::getId,menuIdList);
        queryWrapper1.orderByAsc(AdminMenu::getMenuSeq);
        return adminMenuMapper.selectList(queryWrapper1);
    }
```