云计算应用开发实训

实 训 指 导 书

厦门城市职业学院

云计算技术应用教研室

2024 年 9 月修订

一、实训目的与要求

1.1 实训目的

1）掌握云计算应用项目开发的流程；

2）掌握云计算应用开发常用的工具软件；

3）掌握 Spring boot、Spring Security、Redis & Jwt 及 Vue 等开发工具包的使用；

4）掌握 Web 前端程序设计技术，包括 vue，vuex，vue-route，element-ui，axios；

5）掌握单元测试及调试技术；

6）初步掌握软件的测试方法；

7）掌握软件部署方法。

1.2 实训任务

利用 Spring Tool Suites 集成工具开发一个基于 JavaWeb 的 APP 项目——用户管理系统；通过该项目的部分实现以熟悉 Web 项目开发的基本流程，从而检验并提高学生综合运用所学知识解决实际问题的能力。主要完成以下功能模块：

1) 用户管理：用户是系统操作者，该功能主要完成系统用户配置。

2) 部门管理：配置系统组织机构（公司、部门、小组），树结构展现支持数据权限。

3) 菜单管理：配置系统菜单，操作权限，按钮权限标识等。

4) 岗位管理：配置系统用户所属担任职务。

5) 角色管理：角色菜单权限分配、设置角色按机构进行数据范围权限划分。

6) 个人中心：修改个性信息。

7) 前后端应用部署。

1.3 实训环境

1）高性能 PC（内存 8G 以上），电子教室及局域网；

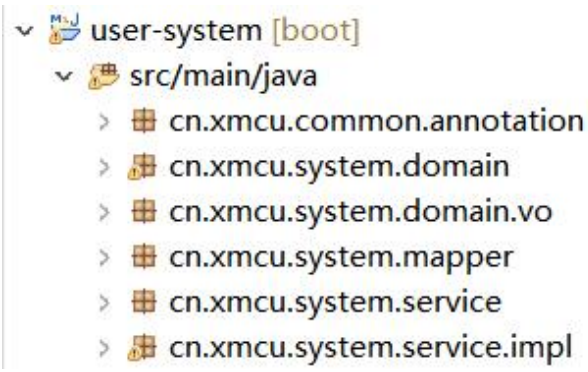
2）Tomcat9.0 以上，Mysql5.7+, Spring Boot 2.6.7, Redis；

二、实训内容及步骤

2.1 后端应用

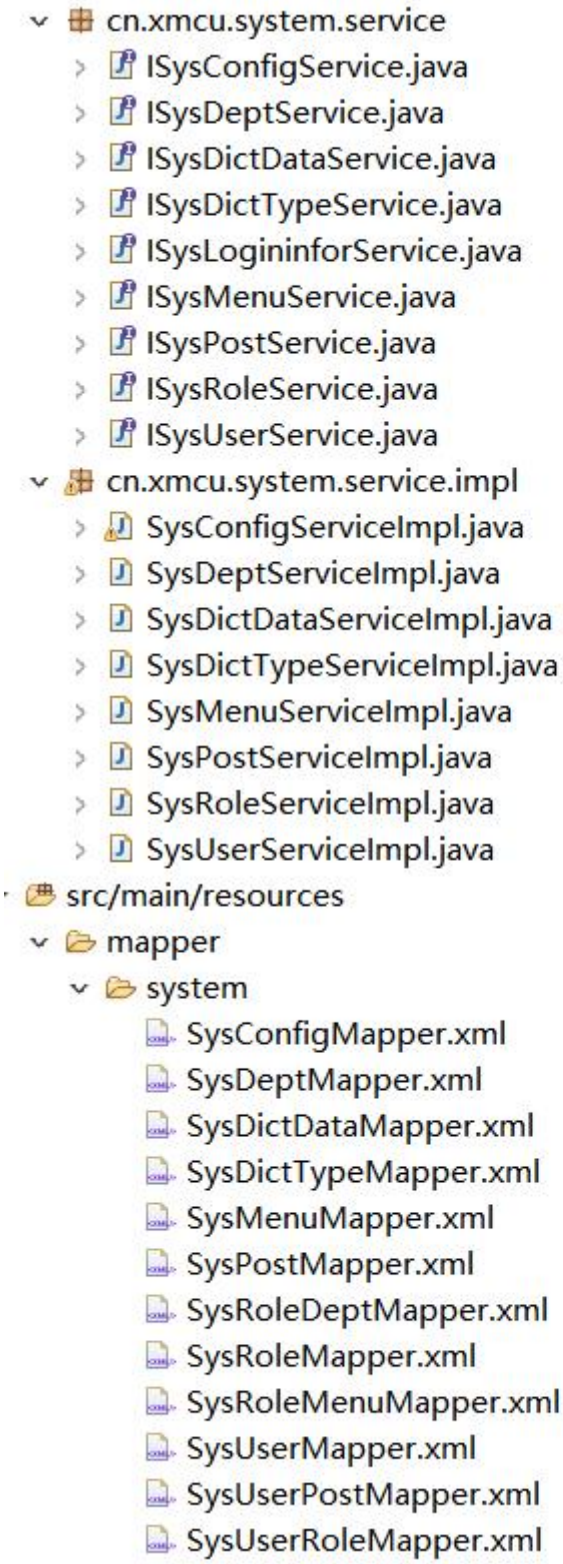
2.1.4 系统模块

1、创建包



2、包内容

- cn.xmcu.common.annotation
  - DataScope.java
- cn.xmcu.system.domain
  - Person.java
  - SysConfig.java
  - SysLogininfor.java
  - SysRoleDept.java
  - SysRoleMenu.java
  - SysUserPost.java
  - SysUserRole.java
- cn.xmcu.system.domain.vo
  - MetaVo.java
  - RouterVo.java
- cn.xmcu.system.mapper
  - SysConfigMapper.java
  - SysDeptMapper.java
  - SysDictDataMapper.java
  - SysDictTypeMapper.java
  - SysMenuMapper.java
  - SysPostMapper.java
  - SysRoleDeptMapper.java
  - SysRoleMapper.java
  - SysRoleMenuMapper.java
  - SysUserMapper.java
  - SysUserPostMapper.java
  - SysUserRoleMapper.java


cn.xmcu.system.service
ISysConfigService.java
ISysDeptService.java
ISysDictDataService.java
ISysDictTypeService.java
ISysLogininforService.java
ISysMenuService.java
ISysPostService.java
ISysRoleService.java
ISysUserService.java
cn.xmcu.system.service.impl
SysConfigServiceImpl.java
SysDeptServiceImpl.java
SysDictDataServiceImpl.java
SysDictTypeServiceImpl.java
SysMenuServiceImpl.java
SysPostServiceImpl.java
SysRoleServiceImpl.java
SysUserServiceImpl.java
src/main/resources
mapper
system
SysConfigMapper.xml
SysDeptMapper.xml
SysDictDataMapper.xml
SysDictTypeMapper.xml
SysMenuMapper.xml
SysPostMapper.xml
SysRoleDeptMapper.xml
SysRoleMapper.xml
SysRoleMenuMapper.xml
SysUserMapper.xml
SysUserPostMapper.xml
SysUserRoleMapper.xml

3、拷贝 user-system 中的文件到相应包

4、代码

DataScope

```
package cn.xmcu.common.annotation;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * 数据权限过滤注解
 *
 */
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface DataScope
{
    /**
     * 部门表的别名
     */
    public String deptAlias() default "";

    /**
     * 用户表的别名
     */
    public String userAlias() default "";

    /**
     * 权限字符（用于多个角色匹配符合要求的权限）默认根据权限注解@ss 获取，多个权限用逗号分隔开来
     */
    public String permission() default "";
}
```

SysConfigMapper

```
package cn.xmcu.system.mapper;

import java.util.List;
import org.apache.ibatis.annotations.Mapper;
import cn.xmcu.system.domain.SysConfig;

/**
 * 参数配置 数据层
 *
 */
@Mapper
public interface SysConfigMapper
{
    /**
     * 查询参数配置信息
     *
     * @param config 参数配置信息
     * @return 参数配置信息
     */
    public SysConfig selectConfig(SysConfig config);

    /**
     * 通过 ID 查询配置
     *
     * @param configId 参数 ID
     * @return 参数配置信息
     */
    public SysConfig selectConfigById(Long configId);

    /**
     * 查询参数配置列表
     *
     * @param config 参数配置信息
     * @return 参数配置集合
     */
    public List<SysConfig> selectConfigList(SysConfig config);

    /**
     * 根据键名查询参数配置信息
     *
     * @param configKey 参数键名
```

```
     * @return 参数配置信息
     */
    public SysConfig checkConfigKeyUnique(String configKey);

    /**
     * 新增参数配置
     *
     * @param config 参数配置信息
     * @return 结果
     */
    public int insertConfig(SysConfig config);

    /**
     * 修改参数配置
     *
     * @param config 参数配置信息
     * @return 结果
     */
    public int updateConfig(SysConfig config);

    /**
     * 删除参数配置
     *
     * @param configId 参数 ID
     * @return 结果
     */
    public int deleteConfigById(Long configId);

    /**
     * 批量删除参数信息
     *
     * @param configIds 需要删除的参数 ID
     * @return 结果
     */
    public int deleteConfigByIds(Long[] configIds);
}
```

SysDeptMapper

```
package cn.xmcu.system.mapper;

import java.util.List;
```

```
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import cn.xmcu.common.core.domain.entity.SysDept;

/**
 * 部门管理 数据层
 *
 */
@Mapper
public interface SysDeptMapper
{
    /**
     * 查询部门管理数据
     *
     * @param dept 部门信息
     * @return 部门信息集合
     */
    public List<SysDept> selectDeptList(SysDept dept);

    /**
     * 根据角色 ID 查询部门树信息
     *
     * @param roleId 角色 ID
     * @param deptCheckStrictly 部门树选择项是否关联显示
     * @return 选中部门列表
     */
    public List<Long> selectDeptListByRoleId(@Param("roleId") Long roleId,
@Param("deptCheckStrictly") boolean deptCheckStrictly);

    /**
     * 根据部门 ID 查询信息
     *
     * @param deptId 部门 ID
     * @return 部门信息
     */
    public SysDept selectDeptById(Long deptId);

    /**
     * 根据 ID 查询所有子部门
     *
     * @param deptId 部门 ID
     * @return 部门列表
     */
    public List<SysDept> selectChildrenDeptById(Long deptId);
```

```
    /**
     * 根据 ID 查询所有子部门（正常状态）
     *
     * @param deptId 部门 ID
     * @return 子部门数
     */
    public int selectNormalChildrenDeptById(Long deptId);

    /**
     * 是否存在子节点
     *
     * @param deptId 部门 ID
     * @return 结果
     */
    public int hasChildByDeptId(Long deptId);

    /**
     * 查询部门是否存在用户
     *
     * @param deptId 部门 ID
     * @return 结果
     */
    public int checkDeptExistUser(Long deptId);

    /**
     * 校验部门名称是否唯一
     *
     * @param deptName 部门名称
     * @param parentId 父部门 ID
     * @return 结果
     */
    public SysDept checkDeptNameUnique(@Param("deptName") String deptName, @Param("parentId") Long parentId);

    /**
     * 新增部门信息
     *
     * @param dept 部门信息
     * @return 结果
     */
    public int insertDept(SysDept dept);

    /**
```

```
     * 修改部门信息
     *
     * @param dept 部门信息
     * @return 结果
     */
    public int updateDept(SysDept dept);

    /**
     * 修改所在部门正常状态
     *
     * @param deptIds 部门 ID 组
     */
    public void updateDeptStatusNormal(Long[] deptIds);

    /**
     * 修改子元素关系
     *
     * @param depts 子元素
     * @return 结果
     */
    public int updateDeptChildren(@Param("depts") List<SysDept> depts);

    /**
     * 删除部门管理信息
     *
     * @param deptId 部门 ID
     * @return 结果
     */
    public int deleteDeptById(Long deptId);
}
```

SysDictDataMapper

```
package cn.xmcu.system.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import cn.xmcu.common.core.domain.entity.SysDictData;

/**
 * 字典表 数据层
```

```
 */
@Mapper
public interface SysDictDataMapper
{
    /**
     * 根据条件分页查询字典数据
     *
     * @param dictData 字典数据信息
     * @return 字典数据集合信息
     */
    public List<SysDictData> selectDictDataList(SysDictData dictData);

    /**
     * 根据字典类型查询字典数据
     *
     * @param dictType 字典类型
     * @return 字典数据集合信息
     */
    public List<SysDictData> selectDictDataByType(String dictType);

    /**
     * 根据字典类型和字典键值查询字典数据信息
     *
     * @param dictType 字典类型
     * @param dictValue 字典键值
     * @return 字典标签
     */
    public String selectDictLabel(@Param("dictType") String dictType, @Param("dictValue") String
dictValue);

    /**
     * 根据字典数据 ID 查询信息
     *
     * @param dictCode 字典数据 ID
     * @return 字典数据
     */
    public SysDictData selectDictDataById(Long dictCode);

    /**
     * 查询字典数据
     *
     * @param dictType 字典类型
     * @return 字典数据
     */
```

```
    public int countDictDataByType(String dictType);

    /**
     * 通过字典 ID 删除字典数据信息
     *
     * @param dictCode 字典数据 ID
     * @return 结果
     */
    public int deleteDictDataById(Long dictCode);

    /**
     * 批量删除字典数据信息
     *
     * @param dictCodes 需要删除的字典数据 ID
     * @return 结果
     */
    public int deleteDictDataByIds(Long[] dictCodes);

    /**
     * 新增字典数据信息
     *
     * @param dictData 字典数据信息
     * @return 结果
     */
    public int insertDictData(SysDictData dictData);

    /**
     * 修改字典数据信息
     *
     * @param dictData 字典数据信息
     * @return 结果
     */
    public int updateDictData(SysDictData dictData);

    /**
     * 同步修改字典类型
     *
     * @param oldDictType 旧字典类型
     * @param newDictType 新旧字典类型
     * @return 结果
     */
    public int updateDictDataType(@Param("oldDictType") String oldDictType,
@Param("newDictType") String newDictType);
}
```

SysDictTypeMapper

```
package cn.xmcu.system.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;

import cn.xmcu.common.core.domain.entity.SysDictType;

/**
 * 字典表 数据层
 *
 */
@Mapper
public interface SysDictTypeMapper
{
    /**
     * 根据条件分页查询字典类型
     *
     * @param dictType 字典类型信息
     * @return 字典类型集合信息
     */
    public List<SysDictType> selectDictTypeList(SysDictType dictType);

    /**
     * 根据所有字典类型
     *
     * @return 字典类型集合信息
     */
    public List<SysDictType> selectDictTypeAll();

    /**
     * 根据字典类型 ID 查询信息
     *
     * @param dictId 字典类型 ID
     * @return 字典类型
     */
    public SysDictType selectDictTypeById(Long dictId);

    /**
     * 根据字典类型查询信息
     *
```

```
     * @param dictType 字典类型
     * @return 字典类型
     */
    public SysDictType selectDictTypeByType(String dictType);

    /**
     * 通过字典 ID 删除字典信息
     *
     * @param dictId 字典 ID
     * @return 结果
     */
    public int deleteDictTypeById(Long dictId);

    /**
     * 批量删除字典类型信息
     *
     * @param dictIds 需要删除的字典 ID
     * @return 结果
     */
    public int deleteDictTypeByIds(Long[] dictIds);

    /**
     * 新增字典类型信息
     *
     * @param dictType 字典类型信息
     * @return 结果
     */
    public int insertDictType(SysDictType dictType);

    /**
     * 修改字典类型信息
     *
     * @param dictType 字典类型信息
     * @return 结果
     */
    public int updateDictType(SysDictType dictType);

    /**
     * 校验字典类型称是否唯一
     *
     * @param dictType 字典类型
     * @return 结果
     */
    public SysDictType checkDictTypeUnique(String dictType);
```

```
    }
```

SysMenuMapper

```
package cn.xmcu.system.mapper;

import java.util.List;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import cn.xmcu.common.core.domain.entity.SysMenu;

/**
 * 菜单表 数据层
 *
 */
@Mapper
public interface SysMenuMapper
{
    /**
     * 查询系统菜单列表
     *
     * @param menu 菜单信息
     * @return 菜单列表
     */
    public List<SysMenu> selectMenuList(SysMenu menu);

    /**
     * 根据用户所有权限
     *
     * @return 权限列表
     */
    public List<String> selectMenuPerms();

    /**
     * 根据用户查询系统菜单列表
     *
     * @param menu 菜单信息
     * @return 菜单列表
     */
    public List<SysMenu> selectMenuListByUserId(SysMenu menu);

    /**
     * 根据角色 ID 查询权限
```

```
     *
     * @param roleId 角色 ID
     * @return 权限列表
     */
    public List<String> selectMenuPermsByRoleId(Long roleId);

    /**
     * 根据用户 ID 查询权限
     *
     * @param userId 用户 ID
     * @return 权限列表
     */
    public List<String> selectMenuPermsByUserId(Long userId);

    /**
     * 根据用户 ID 查询菜单
     *
     * @return 菜单列表
     */
    public List<SysMenu> selectMenuTreeAll();

    /**
     * 根据用户 ID 查询菜单
     *
     * @param userId 用户 ID
     * @return 菜单列表
     */
    public List<SysMenu> selectMenuTreeByUserId(Long userId);

    /**
     * 根据角色 ID 查询菜单树信息
     *
     * @param roleId 角色 ID
     * @param menuCheckStrictly 菜单树选择项是否关联显示
     * @return 选中菜单列表
     */
    public List<Long> selectMenuListByRoleId(@Param("roleId") Long roleId,
@Param("menuCheckStrictly") boolean menuCheckStrictly);

    /**
     * 根据菜单 ID 查询信息
     *
     * @param menuId 菜单 ID
     * @return 菜单信息
```

```
     */
    public SysMenu selectMenuById(Long menuId);

    /**
     * 是否存在菜单子节点
     *
     * @param menuId 菜单 ID
     * @return 结果
     */
    public int hasChildByMenuId(Long menuId);

    /**
     * 新增菜单信息
     *
     * @param menu 菜单信息
     * @return 结果
     */
    public int insertMenu(SysMenu menu);

    /**
     * 修改菜单信息
     *
     * @param menu 菜单信息
     * @return 结果
     */
    public int updateMenu(SysMenu menu);

    /**
     * 删除菜单管理信息
     *
     * @param menuId 菜单 ID
     * @return 结果
     */
    public int deleteMenuById(Long menuId);

    /**
     * 校验菜单名称是否唯一
     *
     * @param menuName 菜单名称
     * @param parentId 父菜单 ID
     * @return 结果
     */
    public SysMenu checkMenuNameUnique(@Param("menuName") String menuName,
@Param("parentId") Long parentId);
```

```
}
```

SysPostMapper

```
package cn.xmcu.system.mapper;

import java.util.List;
import org.apache.ibatis.annotations.Mapper;
import cn.xmcu.system.domain.SysPost;

/**
 * 岗位信息 数据层
 */
@Mapper
public interface SysPostMapper
{
    /**
     * 查询岗位数据集合
     *
     * @param post 岗位信息
     * @return 岗位数据集合
     */
    public List<SysPost> selectPostList(SysPost post);

    /**
     * 查询所有岗位
     *
     * @return 岗位列表
     */
    public List<SysPost> selectPostAll();

    /**
     * 通过岗位 ID 查询岗位信息
     *
     * @param postId 岗位 ID
     * @return 角色对象信息
     */
    public SysPost selectPostById(Long postId);

    /**
     * 根据用户 ID 获取岗位选择框列表
     *
     * @param userId 用户 ID
```

```
     * @return 选中岗位 ID 列表
     */
    public List<Long> selectPostListByUserId(Long userId);

    /**
     * 查询用户所属岗位组
     *
     * @param userName 用户名
     * @return 结果
     */
    public List<SysPost> selectPostsByUserName(String userName);

    /**
     * 删除岗位信息
     *
     * @param postId 岗位 ID
     * @return 结果
     */
    public int deletePostById(Long postId);

    /**
     * 批量删除岗位信息
     *
     * @param postIds 需要删除的岗位 ID
     * @return 结果
     */
    public int deletePostByIds(Long[] postIds);

    /**
     * 修改岗位信息
     *
     * @param post 岗位信息
     * @return 结果
     */
    public int updatePost(SysPost post);

    /**
     * 新增岗位信息
     *
     * @param post 岗位信息
     * @return 结果
     */
    public int insertPost(SysPost post);
```

```
    /**
     * 校验岗位名称
     *
     * @param postName 岗位名称
     * @return 结果
     */
    public SysPost checkPostNameUnique(String postName);

    /**
     * 校验岗位编码
     *
     * @param postCode 岗位编码
     * @return 结果
     */
    public SysPost checkPostCodeUnique(String postCode);
}
```

SysRoleDeptMapper

```
package cn.xmcu.system.mapper;

import java.util.List;
import org.apache.ibatis.annotations.Mapper;
import cn.xmcu.system.domain.SysRoleDept;

/**
 * 角色与部门关联表 数据层
 *
 */
@Mapper
public interface SysRoleDeptMapper
{
    /**
     * 通过角色 ID 删除角色和部门关联
     *
     * @param roleId 角色 ID
     * @return 结果
     */
    public int deleteRoleDeptByRoleId(Long roleId);

    /**
     * 批量删除角色部门关联信息
     *
```

```
     * @param ids  需要删除的数据 ID
     * @return  结果
     */
    public int deleteRoleDept(Long[] ids);

    /**
     *  查询部门使用数量
     *
     * @param deptId  部门 ID
     * @return  结果
     */
    public int selectCountRoleDeptByDeptId(Long deptId);

    /**
     *  批量新增角色部门信息
     *
     * @param roleDeptList  角色部门列表
     * @return  结果
     */
    public int batchRoleDept(List<SysRoleDept> roleDeptList);
}
```

SysRoleMapper

```
package cn.xmcu.system.mapper;

import java.util.List;
import org.apache.ibatis.annotations.Mapper;
import cn.xmcu.common.core.domain.entity.SysRole;

/**
 *  角色表  数据层
 */
@Mapper
public interface SysRoleMapper
{
    /**
     *  根据条件分页查询角色数据
     *
     * @param role  角色信息
     * @return  角色数据集合信息
     */
    public List<SysRole> selectRoleList(SysRole role);
```

```
/**
 * 根据用户 ID 查询角色
 *
 * @param userId 用户 ID
 * @return 角色列表
 */
public List<SysRole> selectRolePermissionByUserId(Long userId);

/**
 * 查询所有角色
 *
 * @return 角色列表
 */
public List<SysRole> selectRoleAll();

/**
 * 根据用户 ID 获取角色选择框列表
 *
 * @param userId 用户 ID
 * @return 选中角色 ID 列表
 */
public List<Long> selectRoleListByUserId(Long userId);

/**
 * 通过角色 ID 查询角色
 *
 * @param roleId 角色 ID
 * @return 角色对象信息
 */
public SysRole selectRoleById(Long roleId);

/**
 * 根据用户 ID 查询角色
 *
 * @param userName 用户名
 * @return 角色列表
 */
public List<SysRole> selectRolesByUserName(String userName);

/**
 * 校验角色名称是否唯一
 *
 * @param roleName 角色名称
```

```
     * @return 角色信息
     */
    public SysRole checkRoleNameUnique(String roleName);

    /**
     * 校验角色权限是否唯一
     *
     * @param roleKey 角色权限
     * @return 角色信息
     */
    public SysRole checkRoleKeyUnique(String roleKey);

    /**
     * 修改角色信息
     *
     * @param role 角色信息
     * @return 结果
     */
    public int updateRole(SysRole role);

    /**
     * 新增角色信息
     *
     * @param role 角色信息
     * @return 结果
     */
    public int insertRole(SysRole role);

    /**
     * 通过角色 ID 删除角色
     *
     * @param roleId 角色 ID
     * @return 结果
     */
    public int deleteRoleById(Long roleId);

    /**
     * 批量删除角色信息
     *
     * @param roleIds 需要删除的角色 ID
     * @return 结果
     */
    public int deleteRoleByIds(Long[] roleIds);
}
```

SysRoleMenuMapper

```
package cn.xmcu.system.mapper;

import java.util.List;
import org.apache.ibatis.annotations.Mapper;
import cn.xmcu.system.domain.SysRoleMenu;

/**
 * 角色与菜单关联表 数据层
 */
@Mapper
public interface SysRoleMenuMapper
{
    /**
     * 查询菜单使用数量
     *
     * @param menuId 菜单 ID
     * @return 结果
     */
    public int checkMenuExistRole(Long menuId);

    /**
     * 通过角色 ID 删除角色和菜单关联
     *
     * @param roleId 角色 ID
     * @return 结果
     */
    public int deleteRoleMenuByRoleId(Long roleId);

    /**
     * 批量删除角色菜单关联信息
     *
     * @param ids 需要删除的数据 ID
     * @return 结果
     */
    public int deleteRoleMenu(Long[] ids);

    /**
     * 批量新增角色菜单信息
```

```
     *
     * @param roleMenuList 角色菜单列表
     * @return 结果
     */
    public int batchRoleMenu(List<SysRoleMenu> roleMenuList);
}
```

SysUserMapper

```
package cn.xmcu.system.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import cn.xmcu.common.core.domain.entity.SysUser;

/**
 * 用户表 数据层
 */
@Mapper
public interface SysUserMapper
{
    /**
     * 根据条件分页查询用户列表
     *
     * @param sysUser 用户信息
     * @return 用户信息集合信息
     */
    public List<SysUser> selectUserList(SysUser sysUser);

    /**
     * 根据条件分页查询已配用户角色列表
     *
     * @param user 用户信息
     * @return 用户信息集合信息
     */
    public List<SysUser> selectAllocatedList(SysUser user);

    /**
     * 根据条件分页查询未分配用户角色列表
     *
     * @param user 用户信息
```

```
     * @return 用户信息集合信息
     */
    public List<SysUser> selectUnallocatedList(SysUser user);

    /**
     * 通过用户名查询用户
     *
     * @param userName 用户名
     * @return 用户对象信息
     */
    public SysUser selectUserByUserName(String userName);

    /**
     * 通过用户 ID 查询用户
     *
     * @param userId 用户 ID
     * @return 用户对象信息
     */
    public SysUser selectUserById(Long userId);

    /**
     * 新增用户信息
     *
     * @param user 用户信息
     * @return 结果
     */
    public int insertUser(SysUser user);

    /**
     * 修改用户信息
     *
     * @param user 用户信息
     * @return 结果
     */
    public int updateUser(SysUser user);

    /**
     * 修改用户头像
     *
     * @param userName 用户名
     * @param avatar 头像地址
     * @return 结果
     */
    public int updateUserAvatar(@Param("userName") String userName, @Param("avatar") String
```

```
avatar);

    /**
     * 重置用户密码
     *
     * @param userName 用户名
     * @param password 密码
     * @return 结果
     */
    public int resetUserPwd(@Param("userName") String userName, @Param("password") String password);

    /**
     * 通过用户 ID 删除用户
     *
     * @param userId 用户 ID
     * @return 结果
     */
    public int deleteUserById(Long userId);

    /**
     * 批量删除用户信息
     *
     * @param userIds 需要删除的用户 ID
     * @return 结果
     */
    public int deleteUserByIds(Long[] userIds);

    /**
     * 校验用户名称是否唯一
     *
     * @param userName 用户名称
     * @return 结果
     */
    public SysUser checkUserNameUnique(String userName);

    /**
     * 校验手机号码是否唯一
     *
     * @param phonenumber 手机号码
     * @return 结果
     */
    public SysUser checkPhoneUnique(String phonenumber);
```

```
    /**
     * 校验 email 是否唯一
     *
     * @param email 用户邮箱
     * @return 结果
     */
    public SysUser checkEmailUnique(String email);
}
```

SysUserPostMapper

```
package cn.xmcu.system.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;

import cn.xmcu.system.domain.SysUserPost;

/**
 * 用户与岗位关联表 数据层
 */
@Mapper
public interface SysUserPostMapper
{
    /**
     * 通过用户 ID 删除用户和岗位关联
     *
     * @param userId 用户 ID
     * @return 结果
     */
    public int deleteUserPostByUserId(Long userId);

    /**
     * 通过岗位 ID 查询岗位使用数量
     *
     * @param postId 岗位 ID
     * @return 结果
     */
    public int countUserPostById(Long postId);

    /**
     * 批量删除用户和岗位关联
```

```
     *
     * @param ids  需要删除的数据 ID
     * @return  结果
     */
    public int deleteUserPost(Long[] ids);

    /**
     *  批量新增用户岗位信息
     *
     * @param userPostList  用户角色列表
     * @return  结果
     */
    public int batchUserPost(List<SysUserPost> userPostList);
}
```

SysUserRoleMapper

```
package cn.xmcu.system.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import cn.xmcu.system.domain.SysUserRole;

/**
 * 用户与角色关联表  数据层
 */

@Mapper
public interface SysUserRoleMapper
{
    /**
     *  通过用户 ID 删除用户和角色关联
     *
     * @param userId  用户 ID
     * @return  结果
     */
    public int deleteUserRoleByUserId(Long userId);

    /**
     *  批量删除用户和角色关联
     *
```

```
     * @param ids  需要删除的数据 ID
     * @return  结果
     */
    public int deleteUserRole(Long[] ids);

    /**
     * 通过角色 ID 查询角色使用数量
     *
     * @param roleId  角色 ID
     * @return  结果
     */
    public int countUserRoleByRoleId(Long roleId);

    /**
     * 批量新增用户角色信息
     *
     * @param userRoleList  用户角色列表
     * @return  结果
     */
    public int batchUserRole(List<SysUserRole> userRoleList);

    /**
     * 删除用户和角色关联信息
     *
     * @param userRole  用户和角色关联信息
     * @return  结果
     */
    public int deleteUserRoleInfo(SysUserRole userRole);

    /**
     * 批量取消授权用户角色
     *
     * @param roleId  角色 ID
     * @param userIds  需要删除的用户数据 ID
     * @return  结果
     */
    public int deleteUserRoleInfos(@Param("roleId") Long roleId, @Param("userIds") Long[] userIds);
}
```

SysDeptServiceImpl

```
package cn.xmcu.system.service.impl;
```

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.stream.Collectors;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import cn.xmcu.common.annotation.DataScope;
import cn.xmcu.common.constant.UserConstants;
import cn.xmcu.common.core.domain.TreeSelect;
import cn.xmcu.common.core.domain.entity.SysDept;
import cn.xmcu.common.core.domain.entity.SysRole;
import cn.xmcu.common.core.domain.entity.SysUser;
import cn.xmcu.common.core.text.Convert;
import cn.xmcu.common.exception.ServiceException;
import cn.xmcu.common.utils.SecurityUtils;
import cn.xmcu.common.utils.StringUtils;
import cn.xmcu.common.utils.spring.SpringUtils;
import cn.xmcu.system.mapper.SysDeptMapper;
import cn.xmcu.system.mapper.SysRoleMapper;
import cn.xmcu.system.service.ISysDeptService;

/**
 * 部门管理 服务实现
 */
@Service
public class SysDeptServiceImpl implements ISysDeptService
{
    @Autowired
    private SysDeptMapper deptMapper;

    @Autowired
    private SysRoleMapper roleMapper;

    /**
     * 查询部门管理数据
     *
     * @param dept 部门信息
     * @return 部门信息集合
     */
    @Override
    @DataScope(deptAlias = "d")
    public List<SysDept> selectDeptList(SysDept dept)
    {
```

```
        return deptMapper.selectDeptList(dept);
    }

    /**
     * 查询部门树结构信息
     *
     * @param dept 部门信息
     * @return 部门树信息集合
     */
    @Override
    public List<TreeSelect> selectDeptTreeList(SysDept dept)
    {
        List<SysDept> depts = SpringUtils.getAopProxy(this).selectDeptList(dept);
        return buildDeptTreeSelect(depts);
    }

    /**
     * 构建前端所需要树结构
     *
     * @param depts 部门列表
     * @return 树结构列表
     */
    @Override
    public List<SysDept> buildDeptTree(List<SysDept> depts)
    {
        List<SysDept> returnList = new ArrayList<SysDept>();
        List<Long> tempList =
depts.stream().map(SysDept::getDeptId).collect(Collectors.toList());
        for (SysDept dept : depts)
        {
            // 如果是顶级节点, 遍历该父节点的所有子节点
            if (!tempList.contains(dept.getParentId()))
            {
                recursionFn(depts, dept);
                returnList.add(dept);
            }
        }
        if (returnList.isEmpty())
        {
            returnList = depts;
        }
        return returnList;
    }
```

```
/**
 * 构建前端所需要下拉树结构
 *
 * @param depts 部门列表
 * @return 下拉树结构列表
 */
@Override
public List<TreeSelect> buildDeptTreeSelect(List<SysDept> depts)
{
    List<SysDept> deptTrees = buildDeptTree(depts);
    return deptTrees.stream().map(TreeSelect::new).collect(Collectors.toList());
}

/**
 * 根据角色 ID 查询部门树信息
 *
 * @param roleId 角色 ID
 * @return 选中部门列表
 */
@Override
public List<Long> selectDeptListByRoleId(Long roleId)
{
    SysRole role = roleMapper.selectRoleById(roleId);
    return deptMapper.selectDeptListByRoleId(roleId, role.isDeptCheckStrictly());
}

/**
 * 根据部门 ID 查询信息
 *
 * @param deptId 部门 ID
 * @return 部门信息
 */
@Override
public SysDept selectDeptById(Long deptId)
{
    return deptMapper.selectDeptById(deptId);
}

/**
 * 根据 ID 查询所有子部门（正常状态）
 *
 * @param deptId 部门 ID
 * @return 子部门数
 */
```

```
    @Override
    public int selectNormalChildrenDeptById(Long deptId)
    {
        return deptMapper.selectNormalChildrenDeptById(deptId);
    }

    /**
     * 是否存在子节点
     *
     * @param deptId 部门 ID
     * @return 结果
     */
    @Override
    public boolean hasChildByDeptId(Long deptId)
    {
        int result = deptMapper.hasChildByDeptId(deptId);
        return result > 0;
    }

    /**
     * 查询部门是否存在用户
     *
     * @param deptId 部门 ID
     * @return 结果 true 存在 false 不存在
     */
    @Override
    public boolean checkDeptExistUser(Long deptId)
    {
        int result = deptMapper.checkDeptExistUser(deptId);
        return result > 0;
    }

    /**
     * 校验部门名称是否唯一
     *
     * @param dept 部门信息
     * @return 结果
     */
    @Override
    public boolean checkDeptNameUnique(SysDept dept)
    {
        Long deptId = StringUtils.isNull(dept.getDeptId()) ? -1L : dept.getDeptId();
        SysDept info = deptMapper.checkDeptNameUnique(dept.getDeptName(),
dept.getParentId());
```

```
        if (StringUtils.isNotNull(info) && info.getDeptId().longValue() != deptId.longValue())
        {
            return UserConstants.NOT_UNIQUE;
        }
        return UserConstants.UNIQUE;
    }

    /**
     * 校验部门是否有数据权限
     *
     * @param deptId 部门 id
     */
    @Override
    public void checkDeptDataScope(Long deptId)
    {
        if (!SysUser.isAdmin(SecurityUtils.getUserId()))
        {
            SysDept dept = new SysDept();
            dept.setDeptId(deptId);
            List<SysDept> depts = SpringUtils.getAopProxy(this).selectDeptList(dept);
            if (StringUtils.isEmpty(depts))
            {
                throw new ServiceException("没有权限访问部门数据！");
            }
        }
    }

    /**
     * 新增保存部门信息
     *
     * @param dept 部门信息
     * @return 结果
     */
    @Override
    public int insertDept(SysDept dept)
    {
        SysDept info = deptMapper.selectDeptById(dept.getParentId());
        // 如果父节点不为正常状态,则不允许新增子节点
        if (!UserConstants.DEPT_NORMAL.equals(info.getStatus()))
        {
            throw new ServiceException("部门停用，不允许新增");
        }
        dept.setAncestors(info.getAncestors() + "," + dept.getParentId());
        return deptMapper.insertDept(dept);
```

```
    }

    /**
     * 修改保存部门信息
     *
     * @param dept 部门信息
     * @return 结果
     */
    @Override
    public int updateDept(SysDept dept)
    {
        SysDept newParentDept = deptMapper.selectDeptById(dept.getParentId());
        SysDept oldDept = deptMapper.selectDeptById(dept.getDeptId());
        if (StringUtils.isNotNull(newParentDept) && StringUtils.isNotNull(oldDept))
        {
            String newAncestors = newParentDept.getAncestors() + "," + newParentDept.getDeptId();
            String oldAncestors = oldDept.getAncestors();
            dept.setAncestors(newAncestors);
            updateDeptChildren(dept.getDeptId(), newAncestors, oldAncestors);
        }
        int result = deptMapper.updateDept(dept);
        if (UserConstants.DEPT_NORMAL.equals(dept.getStatus()) && StringUtils.isNotEmpty(dept.getAncestors())
                && !StringUtils.equals("0", dept.getAncestors()))
        {
            // 如果该部门是启用状态，则启用该部门的所有上级部门
            updateParentDeptStatusNormal(dept);
        }
        return result;
    }

    /**
     * 修改该部门的父级部门状态
     *
     * @param dept 当前部门
     */
    private void updateParentDeptStatusNormal(SysDept dept)
    {
        String ancestors = dept.getAncestors();
        Long[] deptIds = Convert.toLongArray(ancestors);
        deptMapper.updateDeptStatusNormal(deptIds);
    }
```

```
/**
 * 修改子元素关系
 *
 * @param deptId 被修改的部门 ID
 * @param newAncestors 新的父 ID 集合
 * @param oldAncestors 旧的父 ID 集合
 */
public void updateDeptChildren(Long deptId, String newAncestors, String oldAncestors)
{
    List<SysDept> children = deptMapper.selectChildrenDeptById(deptId);
    for (SysDept child : children)
    {
        child.setAncestors(child.getAncestors().replaceFirst(oldAncestors, newAncestors));
    }
    if (children.size() > 0)
    {
        deptMapper.updateDeptChildren(children);
    }
}

/**
 * 删除部门管理信息
 *
 * @param deptId 部门 ID
 * @return 结果
 */
@Override
public int deleteDeptById(Long deptId)
{
    return deptMapper.deleteDeptById(deptId);
}

/**
 * 递归列表
 */
private void recursionFn(List<SysDept> list, SysDept t)
{
    // 得到子节点列表
    List<SysDept> childList = getChildList(list, t);
    t.setChildren(childList);
    for (SysDept tChild : childList)
    {
        if (hasChild(list, tChild))
        {
```

```
                    recursionFn(list, tChild);
                }
            }
        }

        /**
         * 得到子节点列表
         */
        private List<SysDept> getChildList(List<SysDept> list, SysDept t)
        {
            List<SysDept> tlist = new ArrayList<SysDept>();
            Iterator<SysDept> it = list.iterator();
            while (it.hasNext())
            {
                SysDept n = (SysDept) it.next();
                if (StringUtils.isNotNull(n.getParentId()) && n.getParentId().longValue() == t.getDeptId().longValue())
                {
                    tlist.add(n);
                }
            }
            return tlist;
        }

        /**
         * 判断是否有子节点
         */
        private boolean hasChild(List<SysDept> list, SysDept t)
        {
            return getChildList(list, t).size() > 0;
        }
    }
```

SysMenuServiceImpl

```
package cn.xmcu.system.service.impl;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
```

```
import java.util.Set;
import java.util.stream.Collectors;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import cn.xmcu.common.constant.Constants;
import cn.xmcu.common.constant.UserConstants;
import cn.xmcu.common.core.domain.TreeSelect;
import cn.xmcu.common.core.domain.entity.SysMenu;
import cn.xmcu.common.core.domain.entity.SysRole;
import cn.xmcu.common.core.domain.entity.SysUser;
import cn.xmcu.common.utils.SecurityUtils;
import cn.xmcu.common.utils.StringUtils;
import cn.xmcu.system.domain.vo.MetaVo;
import cn.xmcu.system.domain.vo.RouterVo;
import cn.xmcu.system.mapper.SysMenuMapper;
import cn.xmcu.system.mapper.SysRoleMapper;
import cn.xmcu.system.mapper.SysRoleMenuMapper;
import cn.xmcu.system.service.ISysMenuService;

/**
 * 菜单 业务层处理
 */
@Service
public class SysMenuServiceImpl implements ISysMenuService
{
    public static final String PREMISSION_STRING = "perms[\"{0}\"]";

    @Autowired
    private SysMenuMapper menuMapper;

    @Autowired
    private SysRoleMapper roleMapper;

    @Autowired
    private SysRoleMenuMapper roleMenuMapper;

    /**
     * 根据用户查询系统菜单列表
     *
     * @param userId 用户 ID
     * @return 菜单列表
     */
    @Override
    public List<SysMenu> selectMenuList(Long userId)
```

```
{
    return selectMenuList(new SysMenu(), userId);
}

/**
 * 查询系统菜单列表
 *
 * @param menu 菜单信息
 * @return 菜单列表
 */
@Override
public List<SysMenu> selectMenuList(SysMenu menu, Long userId)
{
    List<SysMenu> menuList = null;
    // 管理员显示所有菜单信息
    if (SysUser.isAdmin(userId))
    {
        menuList = menuMapper.selectMenuList(menu);
    }
    else
    {
        menu.getParams().put("userId", userId);
        menuList = menuMapper.selectMenuListByUserId(menu);
    }
    return menuList;
}

/**
 * 根据用户 ID 查询权限
 *
 * @param userId 用户 ID
 * @return 权限列表
 */
@Override
public Set<String> selectMenuPermsByUserId(Long userId)
{
    List<String> perms = menuMapper.selectMenuPermsByUserId(userId);
    Set<String> permsSet = new HashSet<>();
    for (String perm : perms)
    {
        if (StringUtils.isNotEmpty(perm))
        {
            permsSet.addAll(Arrays.asList(perm.trim().split(",")));
        }
```

```
        }
        return permsSet;
    }

    /**
     * 根据角色 ID 查询权限
     *
     * @param roleId 角色 ID
     * @return 权限列表
     */
    @Override
    public Set<String> selectMenuPermsByRoleId(Long roleId)
    {
        List<String> perms = menuMapper.selectMenuPermsByRoleId(roleId);
        Set<String> permsSet = new HashSet<>();
        for (String perm : perms)
        {
            if (StringUtils.isNotEmpty(perm))
            {
                permsSet.addAll(Arrays.asList(perm.trim().split(",")));
            }
        }
        return permsSet;
    }

    /**
     * 根据用户 ID 查询菜单
     *
     * @param userId 用户名称
     * @return 菜单列表
     */
    @Override
    public List<SysMenu> selectMenuTreeByUserId(Long userId)
    {
        List<SysMenu> menus = null;
        if (SecurityUtils.isAdmin(userId))
        {
            menus = menuMapper.selectMenuTreeAll();
        }
        else
        {
            menus = menuMapper.selectMenuTreeByUserId(userId);
        }
        return getChildPerms(menus, 0);
```

```
    }

    /**
     * 根据角色 ID 查询菜单树信息
     *
     * @param roleId 角色 ID
     * @return 选中菜单列表
     */
    @Override
    public List<Long> selectMenuListByRoleId(Long roleId)
    {
        SysRole role = roleMapper.selectRoleById(roleId);
        return menuMapper.selectMenuListByRoleId(roleId, role.isMenuCheckStrictly());
    }

    /**
     * 构建前端路由所需要的菜单
     *
     * @param menus 菜单列表
     * @return 路由列表
     */
    @Override
    public List<RouterVo> buildMenus(List<SysMenu> menus)
    {
        List<RouterVo> routers = new LinkedList<RouterVo>();
        for (SysMenu menu : menus)
        {
            RouterVo router = new RouterVo();
            router.setHidden("1".equals(menu.getVisible()));
            router.setName(getRouteName(menu));
            router.setPath(getRouterPath(menu));
            router.setComponent(getComponent(menu));
            router.setQuery(menu.getQuery());
            router.setMeta(new MetaVo(menu.getMenuName(), menu.getIcon(),
StringUtils.equals("1", menu.getIsCache()), menu.getPath()));
            List<SysMenu> cMenus = menu.getChildren();
            if (StringUtils.isNotEmpty(cMenus) &&
UserConstants.TYPE_DIR.equals(menu.getMenuType()))
            {
                router.setAlwaysShow(true);
                router.setRedirect("noRedirect");
                router.setChildren(buildMenus(cMenus));
            }
            else if (isMenuFrame(menu))
```

```
                {
                    router.setMeta(null);
                    List<RouterVo> childrenList = new ArrayList<RouterVo>();
                    RouterVo children = new RouterVo();
                    children.setPath(menu.getPath());
                    children.setComponent(menu.getComponent());
                    children.setName(StringUtils.capitalize(menu.getPath()));
                    children.setMeta(new MetaVo(menu.getMenuName(), menu.getIcon(),
StringUtils.equals("1", menu.getIsCache()), menu.getPath()));
                    children.setQuery(menu.getQuery());
                    childrenList.add(children);
                    router.setChildren(childrenList);
                }
                else if (menu.getParentId().intValue() == 0 && isInnerLink(menu))
                {
                    router.setMeta(new MetaVo(menu.getMenuName(), menu.getIcon()));
                    router.setPath("/");
                    List<RouterVo> childrenList = new ArrayList<RouterVo>();
                    RouterVo children = new RouterVo();
                    String routerPath = innerLinkReplaceEach(menu.getPath());
                    children.setPath(routerPath);
                    children.setComponent(UserConstants.INNER_LINK);
                    children.setName(StringUtils.capitalize(routerPath));
                    children.setMeta(new MetaVo(menu.getMenuName(), menu.getIcon(),
menu.getPath()));
                    childrenList.add(children);
                    router.setChildren(childrenList);
                }
                routers.add(router);
            }
            return routers;
        }

        /**
         * 构建前端所需要树结构
         *
         * @param menus 菜单列表
         * @return 树结构列表
         */
        @Override
        public List<SysMenu> buildMenuTree(List<SysMenu> menus)
        {
            List<SysMenu> returnList = new ArrayList<SysMenu>();
            List<Long> tempList =
```

```
menus.stream().map(SysMenu::getMenuId).collect(Collectors.toList());
        for (Iterator<SysMenu> iterator = menus.iterator(); iterator.hasNext();)
        {
            SysMenu menu = (SysMenu) iterator.next();
            // 如果是顶级节点, 遍历该父节点的所有子节点
            if (!tempList.contains(menu.getParentId()))
            {
                recursionFn(menus, menu);
                returnList.add(menu);
            }
        }
        if (returnList.isEmpty())
        {
            returnList = menus;
        }
        return returnList;
    }

    /**
     * 构建前端所需要下拉树结构
     *
     * @param menus 菜单列表
     * @return 下拉树结构列表
     */
    @Override
    public List<TreeSelect> buildMenuTreeSelect(List<SysMenu> menus)
    {
        List<SysMenu> menuTrees = buildMenuTree(menus);
        return menuTrees.stream().map(TreeSelect::new).collect(Collectors.toList());
    }

    /**
     * 根据菜单 ID 查询信息
     *
     * @param menuId 菜单 ID
     * @return 菜单信息
     */
    @Override
    public SysMenu selectMenuById(Long menuId)
    {
        return menuMapper.selectMenuById(menuId);
    }

    /**
```

```
     * 是否存在菜单子节点
     *
     * @param menuId 菜单 ID
     * @return 结果
     */
    @Override
    public boolean hasChildByMenuId(Long menuId)
    {
        int result = menuMapper.hasChildByMenuId(menuId);
        return result > 0;
    }

    /**
     * 查询菜单使用数量
     *
     * @param menuId 菜单 ID
     * @return 结果
     */
    @Override
    public boolean checkMenuExistRole(Long menuId)
    {
        int result = roleMenuMapper.checkMenuExistRole(menuId);
        return result > 0;
    }

    /**
     * 新增保存菜单信息
     *
     * @param menu 菜单信息
     * @return 结果
     */
    @Override
    public int insertMenu(SysMenu menu)
    {
        return menuMapper.insertMenu(menu);
    }

    /**
     * 修改保存菜单信息
     *
     * @param menu 菜单信息
     * @return 结果
     */
    @Override
```

```
    public int updateMenu(SysMenu menu)
    {
        return menuMapper.updateMenu(menu);
    }

    /**
     * 删除菜单管理信息
     *
     * @param menuId 菜单 ID
     * @return 结果
     */
    @Override
    public int deleteMenuById(Long menuId)
    {
        return menuMapper.deleteMenuById(menuId);
    }

    /**
     * 校验菜单名称是否唯一
     *
     * @param menu 菜单信息
     * @return 结果
     */
    @Override
    public boolean checkMenuNameUnique(SysMenu menu)
    {
        Long menuId = StringUtils.isNull(menu.getMenuId()) ? -1L : menu.getMenuId();
        SysMenu info = menuMapper.checkMenuNameUnique(menu.getMenuName(),
menu.getParentId());
        if (StringUtils.isNotNull(info) && info.getMenuId().longValue() != menuId.longValue())
        {
            return UserConstants.NOT_UNIQUE;
        }
        return UserConstants.UNIQUE;
    }

    /**
     * 获取路由名称
     *
     * @param menu 菜单信息
     * @return 路由名称
     */
    public String getRouteName(SysMenu menu)
    {
```

```
            String routerName = StringUtils.capitalize(menu.getPath());
            // 非外链并且是一级目录（类型为目录）
            if (isMenuFrame(menu))
            {
                routerName = StringUtils.EMPTY;
            }
            return routerName;
        }

        /**
         * 获取路由地址
         *
         * @param menu 菜单信息
         * @return 路由地址
         */
        public String getRouterPath(SysMenu menu)
        {
            String routerPath = menu.getPath();
            // 内链打开外网方式
            if (menu.getParentId().intValue() != 0 && isInnerLink(menu))
            {
                routerPath = innerLinkReplaceEach(routerPath);
            }
            // 非外链并且是一级目录（类型为目录）
            if (0 == menu.getParentId().intValue() &&
UserConstants.TYPE_DIR.equals(menu.getMenuType())
                    && UserConstants.NO_FRAME.equals(menu.getIsFrame()))
            {
                routerPath = "/" + menu.getPath();
            }
            // 非外链并且是一级目录（类型为菜单）
            else if (isMenuFrame(menu))
            {
                routerPath = "/";
            }
            return routerPath;
        }

        /**
         * 获取组件信息
         *
         * @param menu 菜单信息
         * @return 组件信息
         */
```

```
    public String getComponent(SysMenu menu)
    {
        String component = UserConstants.LAYOUT;
        if (StringUtils.isNotEmpty(menu.getComponent()) && !isMenuFrame(menu))
        {
            component = menu.getComponent();
        }
        else if (StringUtils.isEmpty(menu.getComponent()) && menu.getParentId().intValue() != 0
&& isInnerLink(menu))
        {
            component = UserConstants.INNER_LINK;
        }
        else if (StringUtils.isEmpty(menu.getComponent()) && isParentView(menu))
        {
            component = UserConstants.PARENT_VIEW;
        }
        return component;
    }

    /**
     * 是否为菜单内部跳转
     *
     * @param menu 菜单信息
     * @return 结果
     */
    public boolean isMenuFrame(SysMenu menu)
    {
        return menu.getParentId().intValue() == 0 &&
UserConstants.TYPE_MENU.equals(menu.getMenuType())
                && menu.getIsFrame().equals(UserConstants.NO_FRAME);
    }

    /**
     * 是否为内链组件
     *
     * @param menu 菜单信息
     * @return 结果
     */
    public boolean isInnerLink(SysMenu menu)
    {
        return menu.getIsFrame().equals(UserConstants.NO_FRAME) &&
StringUtils.ishttp(menu.getPath());
    }
```

```
    /**
     * 是否为 parent_view 组件
     *
     * @param menu 菜单信息
     * @return 结果
     */
    public boolean isParentView(SysMenu menu)
    {
        return menu.getParentId().intValue() != 0 && UserConstants.TYPE_DIR.equals(menu.getMenuType());
    }

    /**
     * 根据父节点的 ID 获取所有子节点
     *
     * @param list 分类表
     * @param parentId 传入的父节点 ID
     * @return String
     */
    public List<SysMenu> getChildPerms(List<SysMenu> list, int parentId)
    {
        List<SysMenu> returnList = new ArrayList<SysMenu>();
        for (Iterator<SysMenu> iterator = list.iterator(); iterator.hasNext();)
        {
            SysMenu t = (SysMenu) iterator.next();
            // 一、根据传入的某个父节点 ID,遍历该父节点的所有子节点
            if (t.getParentId() == parentId)
            {
                recursionFn(list, t);
                returnList.add(t);
            }
        }
        return returnList;
    }

    /**
     * 递归列表
     *
     * @param list 分类表
     * @param t 子节点
     */
    private void recursionFn(List<SysMenu> list, SysMenu t)
    {
        // 得到子节点列表
```

```
        List<SysMenu> childList = getChildList(list, t);
        t.setChildren(childList);
        for (SysMenu tChild : childList)
        {
            if (hasChild(list, tChild))
            {
                recursionFn(list, tChild);
            }
        }
    }

    /**
     * 得到子节点列表
     */
    private List<SysMenu> getChildList(List<SysMenu> list, SysMenu t)
    {
        List<SysMenu> tlist = new ArrayList<SysMenu>();
        Iterator<SysMenu> it = list.iterator();
        while (it.hasNext())
        {
            SysMenu n = (SysMenu) it.next();
            if (n.getParentId().longValue() == t.getMenuId().longValue())
            {
                tlist.add(n);
            }
        }
        return tlist;
    }

    /**
     * 判断是否有子节点
     */
    private boolean hasChild(List<SysMenu> list, SysMenu t)
    {
        return getChildList(list, t).size() > 0;
    }

    /**
     * 内链域名特殊字符替换
     *
     * @return 替换后的内链域名
     */
    public String innerLinkReplaceEach(String path)
    {
```

```
            return StringUtils.replaceEach(path, new String[] { Constants.HTTP, Constants.HTTPS, Constants.WWW, "." },
                    new String[] { "", "", "", "/" });
        }
    }
```

SysPostServiceImpl

```
package cn.xmcu.system.service.impl;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import cn.xmcu.common.constant.UserConstants;
import cn.xmcu.common.exception.ServiceException;
import cn.xmcu.common.utils.StringUtils;
import cn.xmcu.system.domain.SysPost;
import cn.xmcu.system.mapper.SysPostMapper;
import cn.xmcu.system.mapper.SysUserPostMapper;
import cn.xmcu.system.service.ISysPostService;

/**
 * 岗位信息 服务层处理
 *
 */
@Service
public class SysPostServiceImpl implements ISysPostService
{
    @Autowired
    private SysPostMapper postMapper;

    @Autowired
    private SysUserPostMapper userPostMapper;

    /**
     * 查询岗位信息集合
     *
     * @param post 岗位信息
     * @return 岗位信息集合
     */
    @Override
    public List<SysPost> selectPostList(SysPost post)
    {
```

```
        return postMapper.selectPostList(post);
    }

    /**
     * 查询所有岗位
     *
     * @return 岗位列表
     */
    @Override
    public List<SysPost> selectPostAll()
    {
        return postMapper.selectPostAll();
    }

    /**
     * 通过岗位 ID 查询岗位信息
     *
     * @param postId 岗位 ID
     * @return 角色对象信息
     */
    @Override
    public SysPost selectPostById(Long postId)
    {
        return postMapper.selectPostById(postId);
    }

    /**
     * 根据用户 ID 获取岗位选择框列表
     *
     * @param userId 用户 ID
     * @return 选中岗位 ID 列表
     */
    @Override
    public List<Long> selectPostListByUserId(Long userId)
    {
        return postMapper.selectPostListByUserId(userId);
    }

    /**
     * 校验岗位名称是否唯一
     *
     * @param post 岗位信息
     * @return 结果
     */
```

```
@Override
public boolean checkPostNameUnique(SysPost post)
{
    Long postId = StringUtils.isNull(post.getPostId()) ? -1L : post.getPostId();
    SysPost info = postMapper.checkPostNameUnique(post.getPostName());
    if (StringUtils.isNotNull(info) && info.getPostId().longValue() != postId.longValue())
    {
        return UserConstants.NOT_UNIQUE;
    }
    return UserConstants.UNIQUE;
}

/**
 * 校验岗位编码是否唯一
 *
 * @param post 岗位信息
 * @return 结果
 */
@Override
public boolean checkPostCodeUnique(SysPost post)
{
    Long postId = StringUtils.isNull(post.getPostId()) ? -1L : post.getPostId();
    SysPost info = postMapper.checkPostCodeUnique(post.getPostCode());
    if (StringUtils.isNotNull(info) && info.getPostId().longValue() != postId.longValue())
    {
        return UserConstants.NOT_UNIQUE;
    }
    return UserConstants.UNIQUE;
}

/**
 * 通过岗位 ID 查询岗位使用数量
 *
 * @param postId 岗位 ID
 * @return 结果
 */
@Override
public int countUserPostById(Long postId)
{
    return userPostMapper.countUserPostById(postId);
}

/**
 * 删除岗位信息
```

```
     *
     * @param postId 岗位 ID
     * @return 结果
     */
    @Override
    public int deletePostById(Long postId)
    {
        return postMapper.deletePostById(postId);
    }

    /**
     * 批量删除岗位信息
     *
     * @param postIds 需要删除的岗位 ID
     * @return 结果
     */
    @Override
    public int deletePostByIds(Long[] postIds)
    {
        for (Long postId : postIds)
        {
            SysPost post = selectPostById(postId);
            if (countUserPostById(postId) > 0)
            {
                throw new ServiceException(String.format("%1$s 已分配,不能删除",
post.getPostName()));
            }
        }
        return postMapper.deletePostByIds(postIds);
    }

    /**
     * 新增保存岗位信息
     *
     * @param post 岗位信息
     * @return 结果
     */
    @Override
    public int insertPost(SysPost post)
    {
        return postMapper.insertPost(post);
    }

    /**
```

```
     * 修改保存岗位信息
     *
     * @param post 岗位信息
     * @return 结果
     */
    @Override
    public int updatePost(SysPost post)
    {
        return postMapper.updatePost(post);
    }
}
```

SysRoleServiceImpl

```
package cn.xmcu.system.service.impl;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import cn.xmcu.common.annotation.DataScope;
import cn.xmcu.common.constant.UserConstants;
import cn.xmcu.common.core.domain.entity.SysRole;
import cn.xmcu.common.core.domain.entity.SysUser;
import cn.xmcu.common.exception.ServiceException;
import cn.xmcu.common.utils.SecurityUtils;
import cn.xmcu.common.utils.StringUtils;
import cn.xmcu.common.utils.spring.SpringUtils;
import cn.xmcu.system.domain.SysRoleDept;
import cn.xmcu.system.domain.SysRoleMenu;
import cn.xmcu.system.domain.SysUserRole;
import cn.xmcu.system.mapper.SysRoleDeptMapper;
import cn.xmcu.system.mapper.SysRoleMapper;
import cn.xmcu.system.mapper.SysRoleMenuMapper;
import cn.xmcu.system.mapper.SysUserRoleMapper;
import cn.xmcu.system.service.ISysRoleService;

/**
 * 角色 业务层处理
```

```
 */
@Service
public class SysRoleServiceImpl implements ISysRoleService
{
    @Autowired
    private SysRoleMapper roleMapper;

    @Autowired
    private SysRoleMenuMapper roleMenuMapper;

    @Autowired
    private SysUserRoleMapper userRoleMapper;

    @Autowired
    private SysRoleDeptMapper roleDeptMapper;

    /**
     * 根据条件分页查询角色数据
     *
     * @param role 角色信息
     * @return 角色数据集合信息
     */
    @Override
    @DataScope(deptAlias = "d")
    public List<SysRole> selectRoleList(SysRole role)
    {
        return roleMapper.selectRoleList(role);
    }

    /**
     * 根据用户 ID 查询角色
     *
     * @param userId 用户 ID
     * @return 角色列表
     */
    @Override
    public List<SysRole> selectRolesByUserId(Long userId)
    {
        List<SysRole> userRoles = roleMapper.selectRolePermissionByUserId(userId);
        List<SysRole> roles = selectRoleAll();
        for (SysRole role : roles)
        {
            for (SysRole userRole : userRoles)
            {
```

```
                if (role.getRoleId().longValue() == userRole.getRoleId().longValue())
                {
                    role.setFlag(true);
                    break;
                }
            }
        }
        return roles;
    }

    /**
     * 根据用户 ID 查询权限
     *
     * @param userId 用户 ID
     * @return 权限列表
     */
    @Override
    public Set<String> selectRolePermissionByUserId(Long userId)
    {
        List<SysRole> perms = roleMapper.selectRolePermissionByUserId(userId);
        Set<String> permsSet = new HashSet<>();
        for (SysRole perm : perms)
        {
            if (StringUtils.isNotNull(perm))
            {
                permsSet.addAll(Arrays.asList(perm.getRoleKey().trim().split(",")));
            }
        }
        return permsSet;
    }

    /**
     * 查询所有角色
     *
     * @return 角色列表
     */
    @Override
    public List<SysRole> selectRoleAll()
    {
        return SpringUtils.getAopProxy(this).selectRoleList(new SysRole());
    }

    /**
     * 根据用户 ID 获取角色选择框列表
```

```
     *
     * @param userId 用户 ID
     * @return 选中角色 ID 列表
     */
    @Override
    public List<Long> selectRoleListByUserId(Long userId)
    {
        return roleMapper.selectRoleListByUserId(userId);
    }

    /**
     * 通过角色 ID 查询角色
     *
     * @param roleId 角色 ID
     * @return 角色对象信息
     */
    @Override
    public SysRole selectRoleById(Long roleId)
    {
        return roleMapper.selectRoleById(roleId);
    }

    /**
     * 校验角色名称是否唯一
     *
     * @param role 角色信息
     * @return 结果
     */
    @Override
    public boolean checkRoleNameUnique(SysRole role)
    {
        Long roleId = StringUtils.isNull(role.getRoleId()) ? -1L : role.getRoleId();
        SysRole info = roleMapper.checkRoleNameUnique(role.getRoleName());
        if (StringUtils.isNotNull(info) && info.getRoleId().longValue() != roleId.longValue())
        {
            return UserConstants.NOT_UNIQUE;
        }
        return UserConstants.UNIQUE;
    }

    /**
     * 校验角色权限是否唯一
     *
     * @param role 角色信息
```

```
     * @return 结果
     */
    @Override
    public boolean checkRoleKeyUnique(SysRole role)
    {
        Long roleId = StringUtils.isNull(role.getRoleId()) ? -1L : role.getRoleId();
        SysRole info = roleMapper.checkRoleKeyUnique(role.getRoleKey());
        if (StringUtils.isNotNull(info) && info.getRoleId().longValue() != roleId.longValue())
        {
            return UserConstants.NOT_UNIQUE;
        }
        return UserConstants.UNIQUE;
    }

    /**
     * 校验角色是否允许操作
     *
     * @param role 角色信息
     */
    @Override
    public void checkRoleAllowed(SysRole role)
    {
        if (StringUtils.isNotNull(role.getRoleId()) && role.isAdmin())
        {
            throw new ServiceException("不允许操作超级管理员角色");
        }
    }

    /**
     * 校验角色是否有数据权限
     *
     * @param roleId 角色 id
     */
    @Override
    public void checkRoleDataScope(Long roleId)
    {
        if (!SysUser.isAdmin(SecurityUtils.getUserId()))
        {
            SysRole role = new SysRole();
            role.setRoleId(roleId);
            List<SysRole> roles = SpringUtils.getAopProxy(this).selectRoleList(role);
            if (StringUtils.isEmpty(roles))
            {
                throw new ServiceException("没有权限访问角色数据！");
```

```
            }
        }
    }

    /**
     * 通过角色 ID 查询角色使用数量
     *
     * @param roleId 角色 ID
     * @return 结果
     */
    @Override
    public int countUserRoleByRoleId(Long roleId)
    {
        return userRoleMapper.countUserRoleByRoleId(roleId);
    }

    /**
     * 新增保存角色信息
     *
     * @param role 角色信息
     * @return 结果
     */
    @Override
    @Transactional
    public int insertRole(SysRole role)
    {
        // 新增角色信息
        roleMapper.insertRole(role);
        return insertRoleMenu(role);
    }

    /**
     * 修改保存角色信息
     *
     * @param role 角色信息
     * @return 结果
     */
    @Override
    @Transactional
    public int updateRole(SysRole role)
    {
        // 修改角色信息
        roleMapper.updateRole(role);
        // 删除角色与菜单关联
```

```
        roleMenuMapper.deleteRoleMenuByRoleId(role.getRoleId());
        return insertRoleMenu(role);
    }

    /**
     * 修改角色状态
     *
     * @param role 角色信息
     * @return 结果
     */
    @Override
    public int updateRoleStatus(SysRole role)
    {
        return roleMapper.updateRole(role);
    }

    /**
     * 修改数据权限信息
     *
     * @param role 角色信息
     * @return 结果
     */
    @Override
    @Transactional
    public int authDataScope(SysRole role)
    {
        // 修改角色信息
        roleMapper.updateRole(role);
        // 删除角色与部门关联
        roleDeptMapper.deleteRoleDeptByRoleId(role.getRoleId());
        // 新增角色和部门信息（数据权限）
        return insertRoleDept(role);
    }

    /**
     * 新增角色菜单信息
     *
     * @param role 角色对象
     */
    public int insertRoleMenu(SysRole role)
    {
        int rows = 1;
        // 新增用户与角色管理
        List<SysRoleMenu> list = new ArrayList<SysRoleMenu>();
```

```
        for (Long menuId : role.getMenuIds())
        {
            SysRoleMenu rm = new SysRoleMenu();
            rm.setRoleId(role.getRoleId());
            rm.setMenuId(menuId);
            list.add(rm);
        }
        if (list.size() > 0)
        {
            rows = roleMenuMapper.batchRoleMenu(list);
        }
        return rows;
    }

    /**
     * 新增角色部门信息(数据权限)
     *
     * @param role 角色对象
     */
    public int insertRoleDept(SysRole role)
    {
        int rows = 1;
        // 新增角色与部门（数据权限）管理
        List<SysRoleDept> list = new ArrayList<SysRoleDept>();
        for (Long deptId : role.getDeptIds())
        {
            SysRoleDept rd = new SysRoleDept();
            rd.setRoleId(role.getRoleId());
            rd.setDeptId(deptId);
            list.add(rd);
        }
        if (list.size() > 0)
        {
            rows = roleDeptMapper.batchRoleDept(list);
        }
        return rows;
    }

    /**
     * 通过角色 ID 删除角色
     *
     * @param roleId 角色 ID
     * @return 结果
     */
```

```
    @Override
    @Transactional
    public int deleteRoleById(Long roleId)
    {
        // 删除角色与菜单关联
        roleMenuMapper.deleteRoleMenuByRoleId(roleId);
        // 删除角色与部门关联
        roleDeptMapper.deleteRoleDeptByRoleId(roleId);
        return roleMapper.deleteRoleById(roleId);
    }

    /**
     * 批量删除角色信息
     *
     * @param roleIds 需要删除的角色 ID
     * @return 结果
     */
    @Override
    @Transactional
    public int deleteRoleByIds(Long[] roleIds)
    {
        for (Long roleId : roleIds)
        {
            checkRoleAllowed(new SysRole(roleId));
            checkRoleDataScope(roleId);
            SysRole role = selectRoleById(roleId);
            if (countUserRoleByRoleId(roleId) > 0)
            {
                throw new ServiceException(String.format("%1$s 已分配,不能删除",
role.getRoleName()));
            }
        }
        // 删除角色与菜单关联
        roleMenuMapper.deleteRoleMenu(roleIds);
        // 删除角色与部门关联
        roleDeptMapper.deleteRoleDept(roleIds);
        return roleMapper.deleteRoleByIds(roleIds);
    }

    /**
     * 取消授权用户角色
     *
     * @param userRole 用户和角色关联信息
     * @return 结果
```

```
     */
    @Override
    public int deleteAuthUser(SysUserRole userRole)
    {
        return userRoleMapper.deleteUserRoleInfo(userRole);
    }

    /**
     * 批量取消授权用户角色
     *
     * @param roleId 角色 ID
     * @param userIds 需要取消授权的用户数据 ID
     * @return 结果
     */
    @Override
    public int deleteAuthUsers(Long roleId, Long[] userIds)
    {
        return userRoleMapper.deleteUserRoleInfos(roleId, userIds);
    }

    /**
     * 批量选择授权用户角色
     *
     * @param roleId 角色 ID
     * @param userIds 需要授权的用户数据 ID
     * @return 结果
     */
    @Override
    public int insertAuthUsers(Long roleId, Long[] userIds)
    {
        // 新增用户与角色管理
        List<SysUserRole> list = new ArrayList<SysUserRole>();
        for (Long userId : userIds)
        {
            SysUserRole ur = new SysUserRole();
            ur.setUserId(userId);
            ur.setRoleId(roleId);
            list.add(ur);
        }
        return userRoleMapper.batchUserRole(list);
    }
}
```

SysUserServiceImpl

```
package cn.xmcu.system.service.impl;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import javax.validation.Validator;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.util.CollectionUtils;
import cn.xmcu.common.annotation.DataScope;
import cn.xmcu.common.constant.UserConstants;
import cn.xmcu.common.core.domain.entity.SysRole;
import cn.xmcu.common.core.domain.entity.SysUser;
import cn.xmcu.common.exception.ServiceException;
import cn.xmcu.common.utils.SecurityUtils;
import cn.xmcu.common.utils.StringUtils;
import cn.xmcu.common.utils.bean.BeanValidators;
import cn.xmcu.common.utils.spring.SpringUtils;
import cn.xmcu.system.domain.SysPost;
import cn.xmcu.system.domain.SysUserPost;
//import cn.xmcu.system.domain.SysUserPost;
import cn.xmcu.system.domain.SysUserRole;
import cn.xmcu.system.mapper.SysPostMapper;
//import cn.xmcu.system.mapper.SysPostMapper;
import cn.xmcu.system.mapper.SysRoleMapper;
import cn.xmcu.system.mapper.SysUserMapper;
import cn.xmcu.system.mapper.SysUserPostMapper;
//import cn.xmcu.system.mapper.SysUserPostMapper;
import cn.xmcu.system.mapper.SysUserRoleMapper;
import cn.xmcu.system.service.ISysConfigService;
//import cn.xmcu.system.service.ISysConfigService;
import cn.xmcu.system.service.ISysUserService;

@Service
public class SysUserServiceImpl implements ISysUserService{
    private static final Logger log = LoggerFactory.getLogger(SysUserServiceImpl.class);

    @Autowired
```

```
private SysUserMapper userMapper;

@Autowired
private SysRoleMapper roleMapper;

@Autowired
private SysPostMapper postMapper;

@Autowired
private SysUserRoleMapper userRoleMapper;

@Autowired
private SysUserPostMapper userPostMapper;

@Autowired
private ISysConfigService configService;

@Autowired
protected Validator validator;

/**
 * 根据条件分页查询用户列表
 *
 * @param user 用户信息
 * @return 用户信息集合信息
 */
@Override
@DataScope(deptAlias = "d", userAlias = "u")
public List<SysUser> selectUserList(SysUser user)
{
    return userMapper.selectUserList(user);
}

/**
 * 根据条件分页查询已分配用户角色列表
 *
 * @param user 用户信息
 * @return 用户信息集合信息
 */
@Override
@DataScope(deptAlias = "d", userAlias = "u")
public List<SysUser> selectAllocatedList(SysUser user)
{
    return userMapper.selectAllocatedList(user);
```

```
}

/**
 * 根据条件分页查询未分配用户角色列表
 *
 * @param user 用户信息
 * @return 用户信息集合信息
 */
@Override
@DataScope(deptAlias = "d", userAlias = "u")
public List<SysUser> selectUnallocatedList(SysUser user)
{
    return userMapper.selectUnallocatedList(user);
}

/**
 * 通过用户名查询用户
 *
 * @param userName 用户名
 * @return 用户对象信息
 */
@Override
public SysUser selectUserByUserName(String userName)
{
    return userMapper.selectUserByUserName(userName);
}

/**
 * 通过用户 ID 查询用户
 *
 * @param userId 用户 ID
 * @return 用户对象信息
 */
@Override
public SysUser selectUserById(Long userId)
{
    return userMapper.selectUserById(userId);
}

/**
 * 查询用户所属角色组
 *
 * @param userName 用户名
 * @return 结果
```

```
     */
    @Override
    public String selectUserRoleGroup(String userName)
    {
        List<SysRole> list = roleMapper.selectRolesByUserName(userName);
        if (CollectionUtils.isEmpty(list))
        {
            return StringUtils.EMPTY;
        }
        return list.stream().map(SysRole::getRoleName).collect(Collectors.joining(","));
    }

    /**
     * 查询用户所属岗位组
     *
     * @param userName 用户名
     * @return 结果
     */
    @Override
    public String selectUserPostGroup(String userName)
    {
        List<SysPost> list = postMapper.selectPostsByUserName(userName);
        if (CollectionUtils.isEmpty(list))
        {
            return StringUtils.EMPTY;
        }
        return list.stream().map(SysPost::getPostName).collect(Collectors.joining(","));
    }

    /**
     * 校验用户名称是否唯一
     *
     * @param user 用户信息
     * @return 结果
     */
    @Override
    public boolean checkUserNameUnique(SysUser user)
    {
        Long userId = StringUtils.isNull(user.getUserId()) ? -1L : user.getUserId();
        SysUser info = userMapper.checkUserNameUnique(user.getUserName());
        if (StringUtils.isNotNull(info) && info.getUserId().longValue() != userId.longValue())
        {
            return UserConstants.NOT_UNIQUE;
        }
```

```
        return UserConstants.UNIQUE;
    }

    /**
     * 校验手机号码是否唯一
     *
     * @param user 用户信息
     * @return
     */
    @Override
    public boolean checkPhoneUnique(SysUser user)
    {
        Long userId = StringUtils.isNull(user.getUserId()) ? -1L : user.getUserId();
        SysUser info = userMapper.checkPhoneUnique(user.getPhonenumber());
        if (StringUtils.isNotNull(info) && info.getUserId().longValue() != userId.longValue())
        {
            return UserConstants.NOT_UNIQUE;
        }
        return UserConstants.UNIQUE;
    }

    /**
     * 校验 email 是否唯一
     *
     * @param user 用户信息
     * @return
     */
    @Override
    public boolean checkEmailUnique(SysUser user)
    {
        Long userId = StringUtils.isNull(user.getUserId()) ? -1L : user.getUserId();
        SysUser info = userMapper.checkEmailUnique(user.getEmail());
        if (StringUtils.isNotNull(info) && info.getUserId().longValue() != userId.longValue())
        {
            return UserConstants.NOT_UNIQUE;
        }
        return UserConstants.UNIQUE;
    }

    /**
     * 校验用户是否允许操作
     *
     * @param user 用户信息
     */
```

```
@Override
public void checkUserAllowed(SysUser user)
{
    if (StringUtils.isNotNull(user.getUserId()) && user.isAdmin())
    {
        throw new ServiceException("不允许操作超级管理员用户");
    }
}

/**
 * 校验用户是否有数据权限
 *
 * @param userId 用户 id
 */
@Override
public void checkUserDataScope(Long userId)
{
    if (!SysUser.isAdmin(SecurityUtils.getUserId()))
    {
        SysUser user = new SysUser();
        user.setUserId(userId);
        List<SysUser> users = SpringUtils.getAopProxy(this).selectUserList(user);
        if (StringUtils.isEmpty(users))
        {
            throw new ServiceException("没有权限访问用户数据！");
        }
    }
}

/**
 * 新增保存用户信息
 *
 * @param user 用户信息
 * @return 结果
 */
@Override
@Transactional
public int insertUser(SysUser user)
{
    // 新增用户信息
    int rows = userMapper.insertUser(user);
    // 新增用户岗位关联
    insertUserPost(user);
    // 新增用户与角色管理
```

```
        insertUserRole(user);
        return rows;
    }

    /**
     * 注册用户信息
     *
     * @param user 用户信息
     * @return 结果
     */
    @Override
    public boolean registerUser(SysUser user)
    {
        return userMapper.insertUser(user) > 0;
    }

    /**
     * 修改保存用户信息
     *
     * @param user 用户信息
     * @return 结果
     */
    @Override
    @Transactional
    public int updateUser(SysUser user)
    {
        Long userId = user.getUserId();
        // 删除用户与角色关联
        userRoleMapper.deleteUserRoleByUserId(userId);
        // 新增用户与角色管理
        insertUserRole(user);
        // 删除用户与岗位关联
        userPostMapper.deleteUserPostByUserId(userId);
        // 新增用户与岗位管理
        insertUserPost(user);
        return userMapper.updateUser(user);
    }

    /**
     * 用户授权角色
     *
     * @param userId 用户 ID
     * @param roleIds 角色组
     */
```

```
@Override
@Transactional
public void insertUserAuth(Long userId, Long[] roleIds)
{
    userRoleMapper.deleteUserRoleByUserId(userId);
    insertUserRole(userId, roleIds);
}

/**
 * 修改用户状态
 *
 * @param user 用户信息
 * @return 结果
 */
@Override
public int updateUserStatus(SysUser user)
{
    return userMapper.updateUser(user);
}

/**
 * 修改用户基本信息
 *
 * @param user 用户信息
 * @return 结果
 */
@Override
public int updateUserProfile(SysUser user)
{
    return userMapper.updateUser(user);
}

/**
 * 修改用户头像
 *
 * @param userName 用户名
 * @param avatar 头像地址
 * @return 结果
 */
@Override
public boolean updateUserAvatar(String userName, String avatar)
{
    return userMapper.updateUserAvatar(userName, avatar) > 0;
}
```

```
/**
 * 重置用户密码
 *
 * @param user 用户信息
 * @return 结果
 */
@Override
public int resetPwd(SysUser user)
{
    return userMapper.updateUser(user);
}

/**
 * 重置用户密码
 *
 * @param userName 用户名
 * @param password 密码
 * @return 结果
 */
@Override
public int resetUserPwd(String userName, String password)
{
    return userMapper.resetUserPwd(userName, password);
}

/**
 * 新增用户角色信息
 *
 * @param user 用户对象
 */
public void insertUserRole(SysUser user)
{
    this.insertUserRole(user.getUserId(), user.getRoleIds());
}

/**
 * 新增用户岗位信息
 *
 * @param user 用户对象
 */
public void insertUserPost(SysUser user)
{
    Long[] posts = user.getPostIds();
```

```
        if (StringUtils.isNotEmpty(posts))
        {
            // 新增用户与岗位管理
            List<SysUserPost> list = new ArrayList<SysUserPost>(posts.length);
            for (Long postId : posts)
            {
                SysUserPost up = new SysUserPost();
                up.setUserId(user.getUserId());
                up.setPostId(postId);
                list.add(up);
            }
            userPostMapper.batchUserPost(list);
        }
    }

    /**
     * 新增用户角色信息
     *
     * @param userId 用户 ID
     * @param roleIds 角色组
     */
    public void insertUserRole(Long userId, Long[] roleIds)
    {
        if (StringUtils.isNotEmpty(roleIds))
        {
            // 新增用户与角色管理
            List<SysUserRole> list = new ArrayList<SysUserRole>(roleIds.length);
            for (Long roleId : roleIds)
            {
                SysUserRole ur = new SysUserRole();
                ur.setUserId(userId);
                ur.setRoleId(roleId);
                list.add(ur);
            }
            userRoleMapper.batchUserRole(list);
        }
    }

    /**
     * 通过用户 ID 删除用户
     *
     * @param userId 用户 ID
     * @return 结果
     */
```

```
@Override
@Transactional
public int deleteUserById(Long userId)
{
    // 删除用户与角色关联
    userRoleMapper.deleteUserRoleByUserId(userId);
    // 删除用户与岗位表
    userPostMapper.deleteUserPostByUserId(userId);
    return userMapper.deleteUserById(userId);
}

/**
 * 批量删除用户信息
 *
 * @param userIds 需要删除的用户 ID
 * @return 结果
 */
@Override
@Transactional
public int deleteUserByIds(Long[] userIds)
{
    for (Long userId : userIds)
    {
        checkUserAllowed(new SysUser(userId));
        checkUserDataScope(userId);
    }
    // 删除用户与角色关联
    userRoleMapper.deleteUserRole(userIds);
    // 删除用户与岗位关联
    userPostMapper.deleteUserPost(userIds);
    return userMapper.deleteUserByIds(userIds);
}

/**
 * 导入用户数据
 *
 * @param userList 用户数据列表
 * @param isUpdateSupport 是否更新支持，如果已存在，则进行更新数据
 * @param operName 操作用户
 * @return 结果
 */
@Override
public String importUser(List<SysUser> userList, Boolean isUpdateSupport, String operName)
{
```

```
        if (StringUtils.isNull(userList) || userList.size() == 0)
        {
            throw new ServiceException("导入用户数据不能为空！");
        }
        int successNum = 0;
        int failureNum = 0;
        StringBuilder successMsg = new StringBuilder();
        StringBuilder failureMsg = new StringBuilder();
        String password = configService.selectConfigByKey("sys.user.initPassword");
        for (SysUser user : userList)
        {
            try
            {
                // 验证是否存在这个用户
                SysUser u = userMapper.selectUserByUserName(user.getUserName());
                if (StringUtils.isNull(u))
                {
                    BeanValidators.validateWithException(validator, user);
                    user.setPassword(SecurityUtils.encryptPassword(password));
                    user.setCreateBy(operName);
                    userMapper.insertUser(user);
                    successNum++;
                    successMsg.append("<br/>" + successNum + "、账号 " +
user.getUserName() + " 导入成功");
                }
                else if (isUpdateSupport)
                {
                    BeanValidators.validateWithException(validator, user);
                    checkUserAllowed(u);
                    checkUserDataScope(u.getUserId());
                    user.setUserId(u.getUserId());
                    user.setUpdateBy(operName);
                    userMapper.updateUser(user);
                    successNum++;
                    successMsg.append("<br/>" + successNum + "、账号 " +
user.getUserName() + " 更新成功");
                }
                else
                {
                    failureNum++;
                    failureMsg.append("<br/>" + failureNum + "、账号 " + user.getUserName()
+ " 已存在");
                }
            }
```

```
                catch (Exception e)
                {
                    failureNum++;
                    String msg = "<br/>" + failureNum + "、账号 " + user.getUserName() + " 导入失败：";
                    failureMsg.append(msg + e.getMessage());
                    log.error(msg, e);
                }
            }
            if (failureNum > 0)
            {
                failureMsg.insert(0, "很抱歉，导入失败！共 " + failureNum + " 条数据格式不正确，错误如下：");
                throw new ServiceException(failureMsg.toString());
            }
            else
            {
                successMsg.insert(0, "恭喜您，数据已全部导入成功！共 " + successNum + " 条，数据如下：");
            }
            return successMsg.toString();

        }


    }
```

5、创建 MyBatis 遇射器文件

在 resources 目录下创建包 mapper.system。拷贝 mapper.system 目录下的文件到包 mapper.system 中。

SysUserMapper.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="cn.xmcu.system.mapper.SysUserMapper">

    <resultMap type="SysUser" id="SysUserResult">
```

```
        <id property="userId" column="user_id" />
        <result property="deptId" column="dept_id" />
        <result property="userName" column="user_name" />
        <result property="nickName" column="nick_name" />
        <result property="email" column="email" />
        <result property="phonenumber" column="phonenumber" />
        <result property="sex" column="sex" />
        <result property="avatar" column="avatar" />
        <result property="password" column="password" />
        <result property="status" column="status" />
        <result property="delFlag" column="del_flag" />
        <result property="loginIp" column="login_ip" />
        <result property="loginDate" column="login_date" />
        <result property="createBy" column="create_by" />
        <result property="createTime" column="create_time" />
        <result property="updateBy" column="update_by" />
        <result property="updateTime" column="update_time" />
        <result property="remark" column="remark" />
        <association property="dept" javaType="SysDept" resultMap="deptResult" />
        <collection property="roles" javaType="java.util.List" resultMap="RoleResult" />
    </resultMap>

    <resultMap id="deptResult" type="SysDept">
        <id property="deptId" column="dept_id" />
        <result property="parentId" column="parent_id" />
        <result property="deptName" column="dept_name" />
        <result property="ancestors" column="ancestors" />
        <result property="orderNum" column="order_num" />
        <result property="leader" column="leader" />
        <result property="status" column="dept_status" />
    </resultMap>

    <resultMap id="RoleResult" type="SysRole">
        <id property="roleId" column="role_id" />
        <result property="roleName" column="role_name" />
        <result property="roleKey" column="role_key" />
        <result property="roleSort" column="role_sort" />
        <result property="dataScope" column="data_scope" />
        <result property="status" column="role_status" />
    </resultMap>

    <sql id="selectUserVo">
        select u.user_id, u.dept_id, u.user_name, u.nick_name, u.email, u.avatar, u.phonenumber,
```

```
u.password, u.sex, u.status, u.del_flag, u.login_ip, u.login_date, u.create_by, u.create_time, u.remark,
            d.dept_id, d.parent_id, d.ancestors, d.dept_name, d.order_num, d.leader, d.status as dept_status,
            r.role_id, r.role_name, r.role_key, r.role_sort, r.data_scope, r.status as role_status
            from sys_user u
                left join sys_dept d on u.dept_id = d.dept_id
                left join sys_user_role ur on u.user_id = ur.user_id
                left join sys_role r on r.role_id = ur.role_id
        </sql>

        <select id="selectUserList" parameterType="SysUser" resultMap="SysUserResult">
            select u.user_id, u.dept_id, u.nick_name, u.user_name, u.email, u.avatar, u.phonenumber, u.sex, u.status, u.del_flag, u.login_ip, u.login_date, u.create_by, u.create_time, u.remark, d.dept_name, d.leader from sys_user u
            left join sys_dept d on u.dept_id = d.dept_id
            where u.del_flag = '0'
            <if test="userId != null and userId != 0">
                AND u.user_id = #{userId}
            </if>
            <if test="userName != null and userName != "">
                AND u.user_name like concat('%', #{userName}, '%')
            </if>
            <if test="status != null and status != "">
                AND u.status = #{status}
            </if>
            <if test="phonenumber != null and phonenumber != "">
                AND u.phonenumber like concat('%', #{phonenumber}, '%')
            </if>
            <if test="params.beginTime != null and params.beginTime != ""><!-- 开始时间检索 -->
                AND date_format(u.create_time,'%y%m%d') &gt;= date_format(#{params.beginTime},'%y%m%d')
            </if>
            <if test="params.endTime != null and params.endTime != ""><!-- 结束时间检索 -->
                AND date_format(u.create_time,'%y%m%d') &lt;= date_format(#{params.endTime},'%y%m%d')
            </if>
            <if test="deptId != null and deptId != 0">
                AND (u.dept_id = #{deptId} OR u.dept_id IN ( SELECT t.dept_id FROM sys_dept t WHERE find_in_set(#{deptId}, ancestors) ))
            </if>
            <!-- 数据范围过滤 -->
            ${params.dataScope}
        </select>
```

```
    <select id="selectAllocatedList" parameterType="SysUser" resultMap="SysUserResult">
        select distinct u.user_id, u.dept_id, u.user_name, u.nick_name, u.email, u.phonenumber, u.status, u.create_time
        from sys_user u
            left join sys_dept d on u.dept_id = d.dept_id
            left join sys_user_role ur on u.user_id = ur.user_id
            left join sys_role r on r.role_id = ur.role_id
        where u.del_flag = '0' and r.role_id = #{roleId}
        <if test="userName != null and userName != """>
            AND u.user_name like concat('%', #{userName}, '%')
        </if>
        <if test="phonenumber != null and phonenumber != """>
            AND u.phonenumber like concat('%', #{phonenumber}, '%')
        </if>
        <!-- 数据范围过滤 -->
        ${params.dataScope}
    </select>

    <select id="selectUnallocatedList" parameterType="SysUser" resultMap="SysUserResult">
        select distinct u.user_id, u.dept_id, u.user_name, u.nick_name, u.email, u.phonenumber, u.status, u.create_time
        from sys_user u
            left join sys_dept d on u.dept_id = d.dept_id
            left join sys_user_role ur on u.user_id = ur.user_id
            left join sys_role r on r.role_id = ur.role_id
        where u.del_flag = '0' and (r.role_id != #{roleId} or r.role_id IS NULL)
        and u.user_id not in (select u.user_id from sys_user u inner join sys_user_role ur on u.user_id = ur.user_id and ur.role_id = #{roleId})
        <if test="userName != null and userName != """>
            AND u.user_name like concat('%', #{userName}, '%')
        </if>
        <if test="phonenumber != null and phonenumber != """>
            AND u.phonenumber like concat('%', #{phonenumber}, '%')
        </if>
        <!-- 数据范围过滤 -->
        ${params.dataScope}
    </select>

    <select id="selectUserByUserName" parameterType="String" resultMap="SysUserResult">
        <include refid="selectUserVo"/>
        where u.user_name = #{userName} and u.del_flag = '0'
    </select>

    <select id="selectUserById" parameterType="Long" resultMap="SysUserResult">
```

```
        <include refid="selectUserVo"/>
        where u.user_id = #{userId}
    </select>

    <select id="checkUserNameUnique" parameterType="String" resultMap="SysUserResult">
        select user_id, user_name from sys_user where user_name = #{userName} and del_flag = '0' limit 1
    </select>

    <select id="checkPhoneUnique" parameterType="String" resultMap="SysUserResult">
        select user_id, phonenumber from sys_user where phonenumber = #{phonenumber} and del_flag = '0' limit 1
    </select>

    <select id="checkEmailUnique" parameterType="String" resultMap="SysUserResult">
        select user_id, email from sys_user where email = #{email} and del_flag = '0' limit 1
    </select>

    <insert id="insertUser" parameterType="SysUser" useGeneratedKeys="true" keyProperty="userId">
        insert into sys_user(
            <if test="userId != null and userId != 0">user_id,</if>
            <if test="deptId != null and deptId != 0">dept_id,</if>
            <if test="userName != null and userName != ''">user_name,</if>
            <if test="nickName != null and nickName != ''">nick_name,</if>
            <if test="email != null and email != ''">email,</if>
            <if test="avatar != null and avatar != ''">avatar,</if>
            <if test="phonenumber != null and phonenumber != ''">phonenumber,</if>
            <if test="sex != null and sex != ''">sex,</if>
            <if test="password != null and password != ''">password,</if>
            <if test="status != null and status != ''">status,</if>
            <if test="createBy != null and createBy != ''">create_by,</if>
            <if test="remark != null and remark != ''">remark,</if>
            create_time
        )values(
            <if test="userId != null and userId != ''">#{userId},</if>
            <if test="deptId != null and deptId != ''">#{deptId},</if>
            <if test="userName != null and userName != ''">#{userName},</if>
            <if test="nickName != null and nickName != ''">#{nickName},</if>
            <if test="email != null and email != ''">#{email},</if>
            <if test="avatar != null and avatar != ''">#{avatar},</if>
            <if test="phonenumber != null and phonenumber != ''">#{phonenumber},</if>
            <if test="sex != null and sex != ''">#{sex},</if>
            <if test="password != null and password != ''">#{password},</if>
```

```
            <if test="status != null and status != ''">#{status},</if>
            <if test="createBy != null and createBy != ''">#{createBy},</if>
            <if test="remark != null and remark != ''">#{remark},</if>
            sysdate()
        )
</insert>

<update id="updateUser" parameterType="SysUser">
        update sys_user
        <set>
            <if test="deptId != null and deptId != 0">dept_id = #{deptId},</if>
            <if test="userName != null and userName != ''">user_name = #{userName},</if>
            <if test="nickName != null and nickName != ''">nick_name = #{nickName},</if>
            <if test="email != null ">email = #{email},</if>
            <if test="phonenumber != null ">phonenumber = #{phonenumber},</if>
            <if test="sex != null and sex != ''">sex = #{sex},</if>
            <if test="avatar != null and avatar != ''">avatar = #{avatar},</if>
            <if test="password != null and password != ''">password = #{password},</if>
            <if test="status != null and status != ''">status = #{status},</if>
            <if test="loginIp != null and loginIp != ''">login_ip = #{loginIp},</if>
            <if test="loginDate != null">login_date = #{loginDate},</if>
            <if test="updateBy != null and updateBy != ''">update_by = #{updateBy},</if>
            <if test="remark != null">remark = #{remark},</if>
            update_time = sysdate()
        </set>
        where user_id = #{userId}
</update>

<update id="updateUserStatus" parameterType="SysUser">
        update sys_user set status = #{status} where user_id = #{userId}
</update>

<update id="updateUserAvatar" parameterType="SysUser">
        update sys_user set avatar = #{avatar} where user_name = #{userName}
</update>

<update id="resetUserPwd" parameterType="SysUser">
        update sys_user set password = #{password} where user_name = #{userName}
</update>

<delete id="deleteUserById" parameterType="Long">
        update sys_user set del_flag = '2' where user_id = #{userId}
</delete>
```

```
    <delete id="deleteUserByIds" parameterType="Long">
        update sys_user set del_flag = '2' where user_id in
        <foreach collection="array" item="userId" open="(" separator="," close=")">
            #{userId}
        </foreach>
    </delete>

</mapper>
```