通用权限管理系统

1. 技术选型与环境要求

1.1 项目技术选型

（1）后端技术

- Spring Boot
- MyBatis Plus
- Spring Security
- Jwt
- Redis
- MySQL

（2）前端技术

- HTML 5
- CSS 3
- JavaScript
- Vue
- Element UI

（3）其它技术栈

- git

1.2 环境要求

（1）后端环境要求

1. jdk 8 +
2. maven 3.6+
3. Spring Boot 2+
4. 开发工具：idea
5. git

（2）前端环境要求

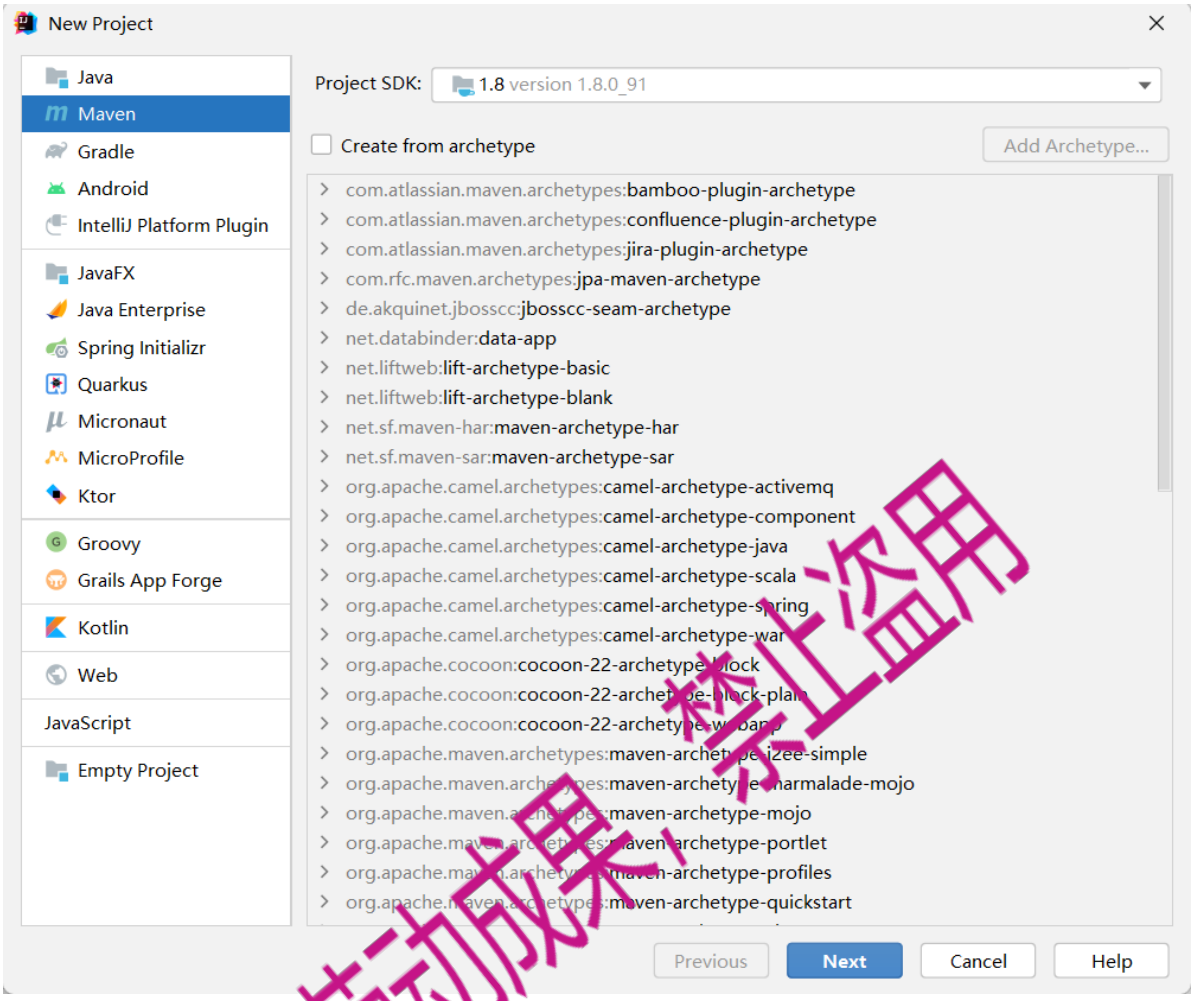
1. node.js
2. python 3+
3. 开发工具：WebStorm 或 Visual Studio Code
4. git

2.搭建后端项目环境

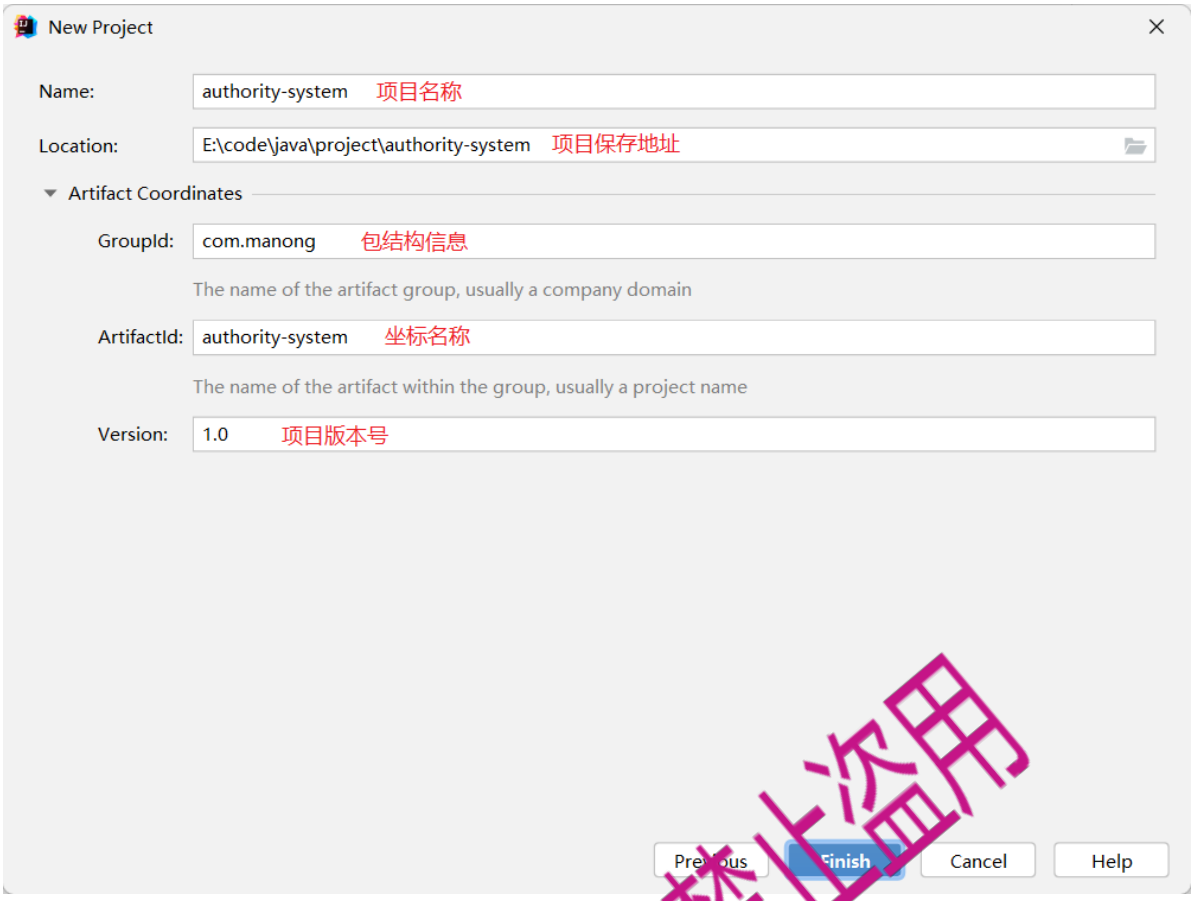
2.1 创建Spring Boot项目

温馨提示：创建Spring Boot项目可使用Spring Initializr方式创建，亦可以使用Maven方式创建。

（1）选择Maven项目



（2）填写项目信息



2.2 引入各模块依赖

在项目的pom.xml文件中加入所需要的依赖，具体如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!--Spring Boot版本 -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.6</version>
    </parent>

    <groupId>com.manong</groupId>
    <artifactId>authority-system</artifactId>
    <version>1.0</version>

    <properties>
        <java.version>1.8</java.version>
        <jwt.version>0.9.1</jwt.version>
        <mybatis-plus.version>3.5.1</mybatis-plus.version>
        <fastjson.version>1.2.80</fastjson.version>
    </properties>

    <dependencies>
```

```
        <!-- web -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- mysql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!-- mybatis plus -->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>${mybatis-plus.version}</version>
        </dependency>
        <!-- lombok -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!-- 热部署 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <!-- 单元测试 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!-- fast json -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>${fastjson.version}</version>
        </dependency>
        <!-- jwt -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>${jwt.version}</version>
        </dependency>
        <!-- redis -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
```

```
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>


</project>
```

2.3 编写全局配置文件

在 `resources` 资源目录下创建 `application.properties` 全局配置文件，代码如下所示：

```
#设置端口号
server.port=9999
#数据库驱动
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#数据库连接地址
spring.datasource.url=jdbc:mysql://localhost:3306/db_authority_system?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
#数据库用户名
spring.datasource.username=root
#数据库密码
spring.datasource.password=root

#加载映射文件
mybatis-plus.mapper-locations=classpath*:/mapper/**/*.xml
#设置别名
mybatis-plus.type-aliases-package=com.manong.entity
#关闭驼峰命名映射
#mybatis-plus.configuration.map-underscore-to-camel-case=false

#显示日志
logging.level.com.manong.dao=debug

#JSON日期格式化
spring.jackson.date-format= yyyy-MM-dd
#JSON日期格式化设置时区为上海
spring.jackson.time-zone=Asia/Shanghai

#日期格式化
spring.mvc.format.date=yyyy-MM-dd
spring.mvc.format.date-time=yyyy-MM-dd HH:mm:ss
```


2.4 编写项目启动器类

在 `com.manong` 包下创建 `AuthorityApplication` 项目启动器类，并加载Mapper接口。

```java
package com.manong;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@MapperScan("com.manong.dao")    //加载Mapper接口
@SpringBootApplication
public class AuthorityApplication {
    public static void main(String[] args) {
        SpringApplication.run(AuthorityApplication.class,args);
    }
}
```

2.5 生成相关接口和类

注意：使用MyBatis Plus代码生成器生成相关的接口和类，此处对生成过程不再赘述。

2.5.1 生成实体类

注意：将实体类中属性数据类型为LocalDate和LocalDateTime修改成java.util.Date类型。

（1）User

```java
package com.manong.entity;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import lombok.Data;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.io.Serializable;
import java.util.Collection;
import java.util.Date;
import java.util.List;

@Data
@TableName("sys_user")
public class User implements Serializable, UserDetails {

    private static final long serialVersionUID = 1L;

    /**
     * 用户编号
     */
    @TableId(value = "id", type = IdType.AUTO)
    private Long id;

    /**
     * 登录名称(用户名)
     */
    private String username;

    /**
     * 登录密码
```

```
35          */
36         private String password;
37
38         /**
39          * 真实姓名
40          */
41         private String realName;
42
43         /**
44          * 昵称
45          */
46         private String nickName;
47
48         /**
49          * 所属部门ID
50          */
51         private Long departmentId;
52
53         /**
54          * 所属部门名称
55          */
56         private String departmentName;
57
58         /**
59          * 性别(0-男, 1-女)
60          */
61         private Integer gender;
62
63         /**
64          * 电话
65          */
66         private String phone;
67
68         /**
69          * 用户头像
70          */
71         private String avatar;
72
73         /**
74          * 邮箱
75          */
76         private String email;
77
78         /**
79          * 是否是管理员(1-管理员)
80          */
81         private Integer isAdmin;
82
83         /**
84          * 创建时间
85          */
86         private Date createTime;
87
88         /**
89          * 修改时间
90          */
91         private Date updateTime;
92
```

```
93        /**
94         * 是否删除(0-未删除，1-已删除)
95         */
96        private Integer isDelete;
97
98        /**
99         * 帐户是否过期(1 未过期，0已过期)
100        */
101       private boolean isAccountNonExpired = true;
102
103       /**
104        * 帐户是否被锁定(1 未过期，0已过期)
105        */
106       private boolean isAccountNonLocked = true;
107
108       /**
109        * 密码是否过期(1 未过期，0已过期)
110        */
111       private boolean isCredentialsNonExpired = true;
112
113       /**
114        * 帐户是否可用(1 可用，0 删除用户)
115        */
116       private boolean isEnabled = true;
117
118       /**
119        * 权限列表
120        */
121       @TableField(exist = false)
122       Collection<? extends GrantedAuthority> authorities;
123
124       /**
125        * 查询用户权限列表
126        */
127       @TableField(exist = false)
128       private List<Permission> permissionList;
129   }
```

(2) Role

```
1   package com.manong.entity;
2
3   import com.baomidou.mybatisplus.annotation.IdType;
4   import com.baomidou.mybatisplus.annotation.TableId;
5   import com.baomidou.mybatisplus.annotation.TableName;
6   import lombok.Data;
7
8   import java.io.Serializable;
9   import java.util.Date;
10
11  @Data
12  @TableName("sys_role")
13  public class Role implements Serializable {
14
15      private static final long serialVersionUID = 1L;
16
17      /**
18       * 角色编号
```

```
     */
    @TableId(value = "id", type = IdType.AUTO)
    private Long id;

    /**
     * 角色编码
     */
    private String roleCode;

    /**
     * 角色名称
     */
    private String roleName;

    /**
     * 创建人
     */
    private Long createUser;

    /**
     * 创建时间
     */
    private Date createTime;

    /**
     * 修改时间
     */
    private Date updateTime;

    /**
     * 备注
     */
    private String remark;

    /**
     * 是否删除(0-未删除，1-已删除)
     */
    private Integer isDelete;


}
```

(3) Permission

```
package com.manong.entity;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import lombok.Data;

import java.io.Serializable;
import java.util.Date;

@Data
@TableName("sys_permission")
public class Permission implements Serializable {

```

```java
    private static final long serialVersionUID = 1L;

    /**
     * 权限编号
     */
    @TableId(value = "id", type = IdType.AUTO)
    private Long id;

    /**
     * 权限名称
     */
    private String label;

    /**
     * 父权限ID
     */
    private Long parentId;

    /**
     * 父权限名称
     */
    private String parentName;

    /**
     * 授权标识符
     */
    private String code;

    /**
     * 路由地址
     */
    private String path;

    /**
     * 路由名称
     */
    private String name;

    /**
     * 授权路径
     */
    private String url;

    /**
     * 权限类型(0-目录 1-菜单 2-按钮)
     */
    private Integer type;

    /**
     * 图标
     */
    private String icon;

    /**
     * 创建时间
     */
    private Date createTime;

```

```
    /**
     * 修改时间
     */
    private Date updateTime;

    /**
     * 备注
     */
    private String remark;

    /**
     * 是否删除(0-未删除，1-已删除)
     */
    private Integer isDelete;

    /**
     * 排序
     */
    private Integer orderNum;
}
```

(4) Department

```
package com.manong.entity;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import lombok.Data;

import java.io.Serializable;

@Data
@TableName("sys_department")
public class Department implements Serializable {

    private static final long serialVersionUID = 1L;

    /**
     * 部门编号
     */
    @TableId(value = "id", type = IdType.AUTO)
    private Long id;

    /**
     * 部门名称
     */
    private String departmentName;

    /**
     * 部门电话
     */
    private String phone;

    /**
     * 部门地址
     */
    private String address;
```

```
36
37      /**
38       * 所属部门编号
39       */
40      private Long pid;
41
42      /**
43       * 所属部门名称
44       */
45      private String parentName;
46
47      /**
48       * 是否删除(0-未删除 1-已删除)
49       */
50      private Integer isDelete;
51
52      /**
53       * 排序
54       */
55      private Integer orderNum;
56
57  }
```

2.5.2 生成Mapper接口

(1) UserMapper

```
1  package com.manong.dao;
2
3  import com.manong.entity.User;
4  import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6  public interface UserMapper extends BaseMapper<User> {
7
8  }
```

(2) RoleMapper

```
1  package com.manong.dao;
2
3  import com.manong.entity.Role;
4  import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6  public interface RoleMapper extends BaseMapper<Role> {
7
8  }
```

(3) PermissionMapper

```
1  package com.manong.dao;
2
3  import com.manong.entity.Permission;
4  import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6  public interface PermissionMapper extends BaseMapper<Permission> {
7
8  }
```

(4) DepartmentMapper

```java
package com.manong.dao;

import com.manong.entity.Department;
import com.baomidou.mybatisplus.core.mapper.BaseMapper;

public interface DepartmentMapper extends BaseMapper<Department> {

}
```

2.5.3 生成Mapper 映射文件

(1) UserMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.manong.dao.UserMapper">

    <!-- 通用查询映射结果 -->
    <resultMap id="BaseResultMap" type="com.manong.entity.User">
        <id column="id" property="id" />
        <result column="username" property="username" />
        <result column="password" property="password" />
        <result column="is_account_non_expired" property="isAccountNonExpired" />
        <result column="is_account_non_locked" property="isAccountNonLocked" />
        <result column="is_credentials_non_expired"
property="isCredentialsNonExpired" />
        <result column="is_enabled" property="isEnabled" />
        <result column="real_name" property="realName" />
        <result column="nick_name" property="nickName" />
        <result column="department_id" property="departmentId" />
        <result column="department_name" property="departmentName" />
        <result column="gender" property="gender" />
        <result column="phone" property="phone" />
        <result column="avatar" property="avatar" />
        <result column="email" property="email" />
        <result column="is_admin" property="isAdmin" />
        <result column="create_time" property="createTime" />
        <result column="update_time" property="updateTime" />
        <result column="is_delete" property="isDelete" />
    </resultMap>

    <!-- 通用查询结果列 -->
    <sql id="Base_Column_List">
        id, username, password, is_account_non_expired, is_account_non_locked,
is_credentials_non_expired, is_enabled, real_name, nick_name, department_id,
department_name, gender, phone, email,avatar, is_admin, create_time, update_time,
is_delete
    </sql>

</mapper>
```

(2) RoleMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

```xml
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.manong.dao.RoleMapper">

    <!-- 通用查询映射结果 -->
    <resultMap id="BaseResultMap" type="com.manong.entity.Role">
        <id column="id" property="id" />
        <result column="role_code" property="roleCode" />
        <result column="role_name" property="roleName" />
        <result column="create_user" property="createUser" />
        <result column="create_time" property="createTime" />
        <result column="update_time" property="updateTime" />
        <result column="remark" property="remark" />
        <result column="is_delete" property="isDelete" />
    </resultMap>

    <!-- 通用查询结果列 -->
    <sql id="Base_Column_List">
        id, role_code, role_name, create_user, create_time, update_time, remark,
is_delete
    </sql>

</mapper>
```

(3) PermissionMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.manong.dao.PermissionMapper">

    <!-- 通用查询映射结果 -->
    <resultMap id="BaseResultMap" type="com.manong.entity.Permission">
        <id column="id" property="id" />
        <result column="label" property="label" />
        <result column="parent_id" property="parentId" />
        <result column="parent_name" property="parentName" />
        <result column="code" property="code" />
        <result column="path" property="path" />
        <result column="name" property="name" />
        <result column="url" property="url" />
        <result column="type" property="type" />
        <result column="icon" property="icon" />
        <result column="create_time" property="createTime" />
        <result column="update_time" property="updateTime" />
        <result column="remark" property="remark" />
        <result column="is_delete" property="isDelete" />
        <result column="order_num" property="orderNum" />
    </resultMap>

    <!-- 通用查询结果列 -->
    <sql id="Base_Column_List">
        id, label, parent_id, parent_name, code, path, name, url, type, icon,
create_time, update_time, remark, is_delete,order_num
    </sql>

</mapper>
```

(4) DepartmentMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.manong.dao.DepartmentMapper">

    <!-- 通用查询映射结果 -->
    <resultMap id="BaseResultMap" type="com.manong.entity.Department">
        <id column="id" property="id" />
        <result column="department_name" property="departmentName" />
        <result column="phone" property="phone" />
        <result column="address" property="address" />
        <result column="pid" property="pid" />
        <result column="parent_name" property="parentName" />
        <result column="is_delete" property="isDelete" />
        <result column="order_num" property="orderNum" />
    </resultMap>

    <!-- 通用查询结果列 -->
    <sql id="Base_Column_List">
        id, department_name, phone, address, pid, parent_name,
is_delete,order_num
    </sql>

</mapper>
```

2.5.4 生成Service接口

(1) UserService

```java
package com.manong.service;

import com.manong.entity.User;
import com.baomidou.mybatisplus.extension.service.IService;

public interface UserService extends IService<User> {

}
```

(2) RoleService

```java
package com.manong.service;

import com.manong.entity.Role;
import com.baomidou.mybatisplus.extension.service.IService;

public interface RoleService extends IService<Role> {

}
```

(3) PermissionService

```java
package com.manong.service;

import com.manong.entity.Permission;
import com.baomidou.mybatisplus.extension.service.IService;

public interface PermissionService extends IService<Permission> {

}
```

(4) DepartmentService

```java
package com.manong.service;

import com.manong.entity.Department;
import com.baomidou.mybatisplus.extension.service.IService;

public interface DepartmentService extends IService<Department> {

}
```

2.5.5 生成Service实现类

注意：在每个Service实现类上加入@Transaction注解。

(1) UserServiceImpl

```java
package com.manong.service.impl;

import com.manong.entity.User;
import com.manong.dao.UserMapper;
import com.manong.service.UserService;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements UserService {

}
```

(2) RoleServiceImpl

```java
package com.manong.service.impl;

import com.manong.entity.Role;
import com.manong.dao.RoleMapper;
import com.manong.service.RoleService;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class RoleServiceImpl extends ServiceImpl<RoleMapper, Role> implements RoleService {

```

```
}
```

(3) PermissionServiceImpl

```
package com.manong.service.impl;

import com.manong.entity.Permission;
import com.manong.dao.PermissionMapper;
import com.manong.service.PermissionService;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class PermissionServiceImpl extends ServiceImpl<PermissionMapper,
Permission> implements PermissionService {

}
```

(4) DepartmentServiceImpl

```
package com.manong.service.impl;

import com.manong.entity.Department;
import com.manong.dao.DepartmentMapper;
import com.manong.service.DepartmentService;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class DepartmentServiceImpl extends ServiceImpl<DepartmentMapper,
Department> implements DepartmentService {

}
```

2.5.6 生成Controller控制器类

注意：将每个Controller类的@Controller注解修改为@RestController注解；修改@RequestMapping注解的访问地址。

(1) UserController

```
package com.manong.controller;


import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/user")
public class UserController {

}
```

(2) RoleController

```java
package com.manong.controller;


import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/role")
public class RoleController {

}
```

(3) PermissionController

```java
package com.manong.controller;


import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/permission")
public class PermissionController {

}
```

(4) DepartmentController

```java
package com.manong.controller;


import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/department")
public class DepartmentController {

}
```


2.6 封装统一返回值类

在com.manong.utils包下编写统一返回值类。

(1) ResultCode

```java
package com.manong.utils;

public class ResultCode {
    /**
     * 成功状态码
     */
    public static Integer SUCCESS = 200;

    /**
     * 失败状态码
     */
    public static Integer ERROR = 500;
```

```
13
14      /**
15       * 未登录状态码
16       */
17      public static final int NO_LOGIN = 600;
18
19      /**
20       * 没有权限状态码
21       */
22      public static final int NO_AUTH = 700;
23
24  }
```

(2) Result

```
1   package com.manong.utils;
2
3   import lombok.Data;
4
5   /**
6    * 全局统一返回结果类
7    */
8   @Data
9   public class Result<T> {
10      private Boolean success;//是否成功
11      private Integer code;//状态码
12      private String message;//返回消息
13      private T data;//返回数据
14
15      /**
16       * 私有化构造方法，禁止在其它类创建对象
17       */
18      private Result(){}
19
20      /**
21       * 成功执行，不返回数据
22       * @return
23       */
24      public static<T> Result<T> ok(){
25          Result<T> result = new Result<T>();
26          result.setSuccess(true);
27          result.setCode(ResultCode.SUCCESS);
28          result.setMessage("执行成功");
29          return result;
30      }
31
32      /**
33       * 成功执行，并返回数据
34       * @param data
35       * @param <T>
36       * @return
37       */
38      public static<T> Result<T> ok(T data){
39          Result<T> result = new Result<T>();
40          result.setSuccess(true);
41          result.setCode(ResultCode.SUCCESS);
42          result.setMessage("执行成功");
43          result.setData(data);
```

```
            return result;
        }

        /**
         * 失败
         * @return
         */
        public static<T> Result<T> error(){
            Result<T> result = new Result<T>();
            result.setSuccess(false);
            result.setCode(ResultCode.ERROR);
            result.setMessage("执行失败");
            return result;
        }
        /**
         * 设置是否成功
         * @param success
         * @return
         */
        public Result<T> success(Boolean success){
            this.setSuccess(success);
            return this;
        }

        /**
         * 设置状态码
         * @param code
         * @return
         */
        public Result<T> code(Integer code){
            this.setCode(code);
            return this;
        }

        /**
         * 设置返回消息
         * @param message
         * @return
         */
        public Result<T> message(String message){
            this.setMessage(message);
            return this;
        }

        /**
         * 是否存在
         * @return
         */
        public static<T> Result<T> exist(){
            Result<T> result = new Result<T>();
            result.setSuccess(true);
            result.setCode(ResultCode.SUCCESS);
            result.setMessage("执行成功");
            return result;
        }
    }
```

2.7 编写MyBatis Plus配置类

在com.manong.config.mybatis包下编写MyBatis相关配置类。

2.7.1 分页配置类

```
package com.manong.config.mybatis;

import com.baomidou.mybatisplus.annotation.DbType;
import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import
com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MyBatisPlusConfig {

    // 最新版
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        PaginationInnerInterceptor paginationInnerInterceptor =
            new PaginationInnerInterceptor(DbType.MYSQL);
        paginationInnerInterceptor.setOverflow(true);//溢出后从第1页开始
        //指定数据库类型
        interceptor.addInnerInterceptor(paginationInnerInterceptor);
        return interceptor;
    }

}
```

2.7.2 编写自动填充配置类

```
package com.manong.config.mybatis;

import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
import org.apache.ibatis.reflection.MetaObject;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
public class CommonMetaObjectHandler implements MetaObjectHandler {

    /**
     * 新增
     * @param metaObject
     */
    @Override
    public void insertFill(MetaObject metaObject) {
        //参数1：元数据对象
        //参数2：属性名称
        //参数3：类对象
        //参数4：当前系统时间
        this.strictInsertFill(metaObject, "createTime", Date.class, new Date());
        this.strictUpdateFill(metaObject, "updateTime", Date.class, new Date());
```

```
    }

    /**
     * 修改
     * @param metaObject
     */
    @Override
    public void updateFill(MetaObject metaObject) {
        this.strictUpdateFill(metaObject, "updateTime", Date.class, new Date());
    }
}
```

2.8 接口代码测试

2.8.1 编写UserContoller

在UserController控制器类中编写查询所有用户的方法。

```
@RestController
@RequestMapping("/api/user")
public class UserController {

    @Resource
    private UserService userService;

    /**
     * 查询所有用户列表
     * @return
     */
    @GetMapping("/listAll")
    public Result listAll(){
        return Result.ok(userService.list());
    }

}
```

2.8.2 接口调用测试

在浏览器或postman工具中输入 `http://localhost:9999/api/user/listAll` 进行测试，测试前需要在数据库中添加相应的数据。

2.9 编写跨域请求配置

在com.manong.config.web包下编写CORSConfig跨域请求配置类。

```
package com.manong.config.web;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CORSConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**").//允许所有的访问请求（访问路径）
```

```
                allowedMethods("*").//允许所有的请求方法访问该跨域资源服务器
                allowedOrigins("*").//允许所有的请求域名访问我们的跨域资源
                allowedHeaders("*");//允许所有的请求header访问
        }
    }
```

3.用户认证与授权

3.1 Spring Security介绍

Spring Security是基于Spring生态圈的，用于提供安全访问控制解决方案的框架。Spring Security的安全管理有两个重要概念，分别是Authentication（认证）和Authorization（授权）。

为了方便Spring Boot项目的安全管理，Spring Boot对Spring Security安全框架进行了整合支持，并提供了通用的自动化配置，从而实现了Spring Security安全框架中包含的多数安全管理功能。

Spring Security登录认证主要涉及两个重要的接口 UserDetailService和UserDetails接口。UserDetailService接口主要定义了一个方法 loadUserByUsername(String username)用于完成用户信息的查询，其中username就是登录时的登录名称，登录认证时，需要自定义一个实现类实现UserDetailService接口，完成数据库查询，该接口返回UserDetail。

UserDetail主要用于封装认证成功时的用户信息，即UserDetailService返回的用户信息，可以用Spring自己的User对象，但是最好是实现UserDetail接口，自定义用户对象。

3.2 Spring Security认证步骤

1. 自定UserDetails类：当实体对象字段不满足的需要自定义UserDetails，一般都要自定义UserDetails。
2. 自定义UserDetailsService类，主要用于从数据库查询用户信息。
3. 创建登录认证成功处理器，认证成功后需要返回JSON数据，菜单权限等。
4. 创建登录认证失败处理器，认证失败需要返回JSON数据，给前端判断。
5. 创建匿名用户访问无权限资源时处理器，匿名用户访问时，需要提示JSON。
6. 创建认证过的用户访问无权限资源时的处理器，无权限访问时，需要提示JSON。
7. 配置Spring Security配置类，把上面自定义的处理器交给Spring Security。

3.3 Spring Security认证实现

3.3.1 添加Spring Security依赖

在pom.xml文件中添加Spring Security核心依赖，代码如下所示：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

3.3.2 自定义UserDetails类

当实体对象字段不满足时Spring Security认证时，需要自定义UserDetails。

1. 将User类实现UserDetails接口
2. 将原有的isAccountNonExpired、isAccountNonLocked、isCredentialsNonExpired和isEnabled属性修改成boolean类型，同时添加authorities属性。

注意：上述4个属性只能是非包装类的boolean类型属性，且默认值设置为true。

```
@Data
@TableName("sys_user")
public class User implements Serializable, UserDetails {

    //省略原有的属性......

    /**
     * 帐户是否过期(1 未过期，0已过期)
     */
    private boolean isAccountNonExpired = true;

    /**
     * 帐户是否被锁定(1 未过期，0已过期)
     */
    private boolean isAccountNonLocked = true;

    /**
     * 密码是否过期(1 未过期，0已过期)
     */
    private boolean isCredentialsNonExpired = true;

    /**
     * 帐户是否可用(1 可用，0 删除用户)
     */
    private boolean isEnabled = true;

    /**
     * 权限列表
     */
    @TableField(exist = false)
    Collection<? extends GrantedAuthority> authorities;
}
```

3.3.3 编写UserService接口

在com.manong.service.UserService接口编写 根据用户名查询用户信息 的方法。

```
public interface UserService extends IService<User> {
    /**
     * 根据用户名查询用户信息
     * @param userName
     * @return
     */
    User findUserByUserName(String userName);
}
```

3.3.4 编写UserService接口实现类

在com.manong.service.impl.UserServiceImpl类中实现UserService接口。

```
@Service
@Transactional
public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements
UserService {

    /**
```

```
     * 根据用户名查询用户信息
     *
     * @param userName
     * @return
     */
    @Override
    public User findUserByUserName(String userName) {
        //创建条件构造器
        QueryWrapper<User> queryWrapper = new QueryWrapper<User>();
        //用户名
        queryWrapper.eq("username",userName);
        //返回查询记录
        return baseMapper.selectOne(queryWrapper);
    }
}
```

3.3.5 自定义UserDetailsService类

在com.manong.config.security.service包下创建CustomerUserDetailsService用户认证处理类，该类需要实现UserDetailsService接口。

```
package com.manong.config.security.service;

import com.manong.entity.User;
import com.manong.service.UserService;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

@Component
public class CustomerUserDetailsService implements UserDetailsService {

    @Resource
    private UserService userService;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        //调用根据用户名查询用户信息的方法
        User user = userService.findUserByUserName(username);
        //如果对象为空，则认证失败
        if (user == null) {
            throw new UsernameNotFoundException("用户名或密码错误!");
        }
        return user;
    }
}
```

3.3.6 编写自定义认证成功处理器

在com.manong.config.security.handler包下创建LoginSuccessHandler登录认证成功处理器类。

```
package com.manong.config.security.handler;

import com.alibaba.fastjson.JSON;
```

```
import com.alibaba.fastjson.serializer.SerializerFeature;
import com.manong.entity.User;
import org.springframework.security.core.Authentication;
import
org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 登录认证成功处理器类
*/
@Component
public class LoginSuccessHandler implements AuthenticationSuccessHandler {
    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) throws IOException,
ServletException {
        //设置客户端的响应的内容类型
        response.setContentType("application/json;charset=UTF-8");
        //获取当登录用户信息
        User user = (User) authentication.getPrincipal();
        //消除循环引用
        String result = JSON.toJSONString(user,
SerializerFeature.DisableCircularReferenceDetect);
        //获取输出流
        ServletOutputStream outputStream = response.getOutputStream();
        outputStream.write(result.getBytes(StandardCharsets.UTF_8));
        outputStream.flush();
        outputStream.close();
    }
}
```

3.3.7 编写自定义认证失败处理器

在com.manong.config.security.handler包下创建LoginFailureHandler用户认证失败处理类。

```
package com.manong.config.security.handler;

import com.alibaba.fastjson.JSON;
import com.manong.utils.Result;
import org.springframework.security.authentication.*;
import org.springframework.security.core.AuthenticationException;
import
org.springframework.security.web.authentication.AuthenticationFailureHandler;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
```

```
16
17  /**
18   * 用户认证失败处理类
19   */
20  @Component
21  public class LoginFailureHandler implements AuthenticationFailureHandler {
22      @Override
23      public void onAuthenticationFailure(HttpServletRequest request,
    HttpServletResponse response, AuthenticationException exception) throws
    IOException, ServletException {
24          //设置客户端响应编码格式
25          response.setContentType("application/json;charset=UTF-8");
26          //获取输出流
27          ServletOutputStream outputStream = response.getOutputStream();
28          String message = null;//提示信息
29          int code = 500;//错误编码
30          //判断异常类型
31          if(exception instanceof AccountExpiredException){
32              message = "账户过期,登录失败！";
33          }else if(exception instanceof BadCredentialsException){
34              message = "用户名或密码错误,登录失败！";
35          }else if(exception instanceof CredentialsExpiredException){
36              message = "密码过期,登录失败！";
37          }else if(exception instanceof DisabledException){
38              message = "账户被禁用,登录失败！";
39          }else if(exception instanceof LockedException){
40              message = "账户被锁,登录失败！";
41          }else if(exception instanceof InternalAuthenticationServiceException){
42              message = "账户不存在,登录失败！";
43          }else{
44              message = "登录失败！";
45          }
46          //将错误信息转换成JSON
47          String result =
    JSON.toJSONString(Result.error().code(code).message(message));
48          outputStream.write(result.getBytes(StandardCharsets.UTF_8));
49          outputStream.flush();
50          outputStream.close();
51      }
52  }
```


3.3.8 编写认证用户无权限访问处理器

在com.manong.config.security.handler包下创建CustomerAccessDeniedHandler认证用户访问无权限资源时处理器类。

```
1   package com.manong.config.security.handler;
2
3   import com.alibaba.fastjson.JSON;
4   import com.alibaba.fastjson.serializer.SerializerFeature;
5   import com.manong.utils.Result;
6   import org.springframework.security.access.AccessDeniedException;
7   import org.springframework.security.web.access.AccessDeniedHandler;
8   import org.springframework.stereotype.Component;
9
10  import javax.servlet.ServletException;
11  import javax.servlet.ServletOutputStream;
12  import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 认证用户访问无权限资源时处理器
 */
@Component
public class CustomerAccessDeniedHandler implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
AccessDeniedException accessDeniedException) throws IOException, ServletException
{
        //设置客户端的响应的内容类型
        response.setContentType("application/json;charset=UTF-8");
        //获取输出流
        ServletOutputStream outputStream = response.getOutputStream();
        //消除循环引用
        String result = JSON.toJSONString(Result.error().code(700).message("无权限
访问,请联系管理员！"), SerializerFeature.DisableCircularReferenceDetect);
        outputStream.write(result.getBytes(StandardCharsets.UTF_8));
        outputStream.flush();
        outputStream.close();
    }
}
```

3.3.9 编写匿名用户访问资源处理器

在com.manong.config.security.handler包下创建AnonymousAuthenticationHandler匿名用户访问资源处理器类。

```
package com.manong.config.security.handler;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.serializer.SerializerFeature;
import com.manong.utils.Result;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 匿名用户访问资源处理器
 */
@Component
public class AnonymousAuthenticationHandler implements AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse
response, AuthenticationException authException) throws IOException,
ServletException {
        //设置客户端的响应的内容类型
        response.setContentType("application/json;charset=UTF-8");
```

```
            //获取输出流
            ServletOutputStream outputStream = response.getOutputStream();
            //消除循环引用
            String result = JSON.toJSONString(Result.error().code(600).message("匿名用户无权限访问！"), SerializerFeature.DisableCircularReferenceDetect);
            outputStream.write(result.getBytes(StandardCharsets.UTF_8));
            outputStream.flush();
            outputStream.close();
        }
    }
```

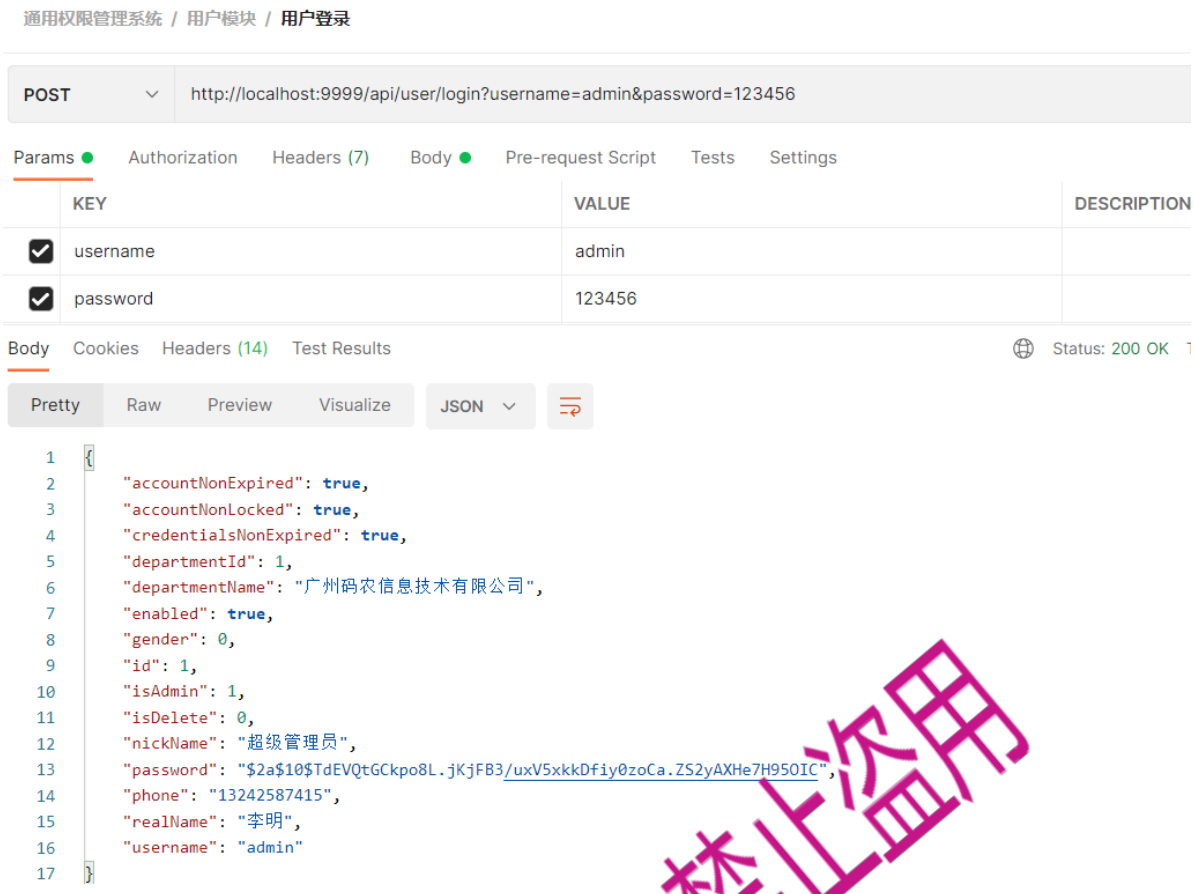
3.3.10 编写Spring Security配置类

在com.manong.config.security包下创建SpringSecurityConfig配置类。

```
package com.manong.config.security;

import com.manong.config.security.handler.AnonymousAuthenticationHandler;
import com.manong.config.security.handler.CustomerAccessDeniedHandler;
import com.manong.config.security.handler.LoginFailureHandler;
import com.manong.config.security.handler.LoginSuccessHandler;
import com.manong.config.security.service.CustomerUserDetailsService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import javax.annotation.Resource;

@Configuration
@EnableWebSecurity
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
    @Resource
    private CustomerUserDetailsService customerUserDetailsService;

    @Resource
    private LoginSuccessHandler loginSuccessHandler;

    @Resource
    private LoginFailureHandler loginFailureHandler;

    @Resource
    private AnonymousAuthenticationHandler anonymousAuthenticationHandler;

    @Resource
    private CustomerAccessDeniedHandler customerAccessDeniedHandler;

    /**
```

```
38          * 注入加密处理类
39          *
40          * @return
41          */
42         @Bean
43         public BCryptPasswordEncoder passwordEncoder() {
44             return new BCryptPasswordEncoder();
45         }
46
47         @Override
48         protected void configure(HttpSecurity http) throws Exception {
49             //登录前进行过滤
50             http.formLogin()
51                     .loginProcessingUrl("/api/user/login")
52                     //  设置登录验证成功或失败后的的跳转地址
53
    .successHandler(loginSuccessHandler).failureHandler(loginFailureHandler)
54                     // 禁用csrf防御机制
55                     .and().csrf().disable()
56
    .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
57                     .and()
58                     .authorizeRequests()
59                     .antMatchers("/api/user/login").permitAll()
60                     .anyRequest().authenticated()
61                     .and()
62                     .exceptionHandling()
63                     .authenticationEntryPoint(anonymousAuthenticationHandler)
64                     .accessDeniedHandler(customerAccessDeniedHandler)
65                     .and().cors();//开启跨域配置
66         }
67
68         /**
69          * 配置认证处理器
70          *
71          * @param auth
72          * @throws Exception
73          */
74         @Override
75         protected void configure(AuthenticationManagerBuilder auth) throws Exception
    {
76
     auth.userDetailsService(customerUserDetailsService).passwordEncoder(passwordEnco
    der());
77         }
78     }
```


3.3.11 测试登录认证接口

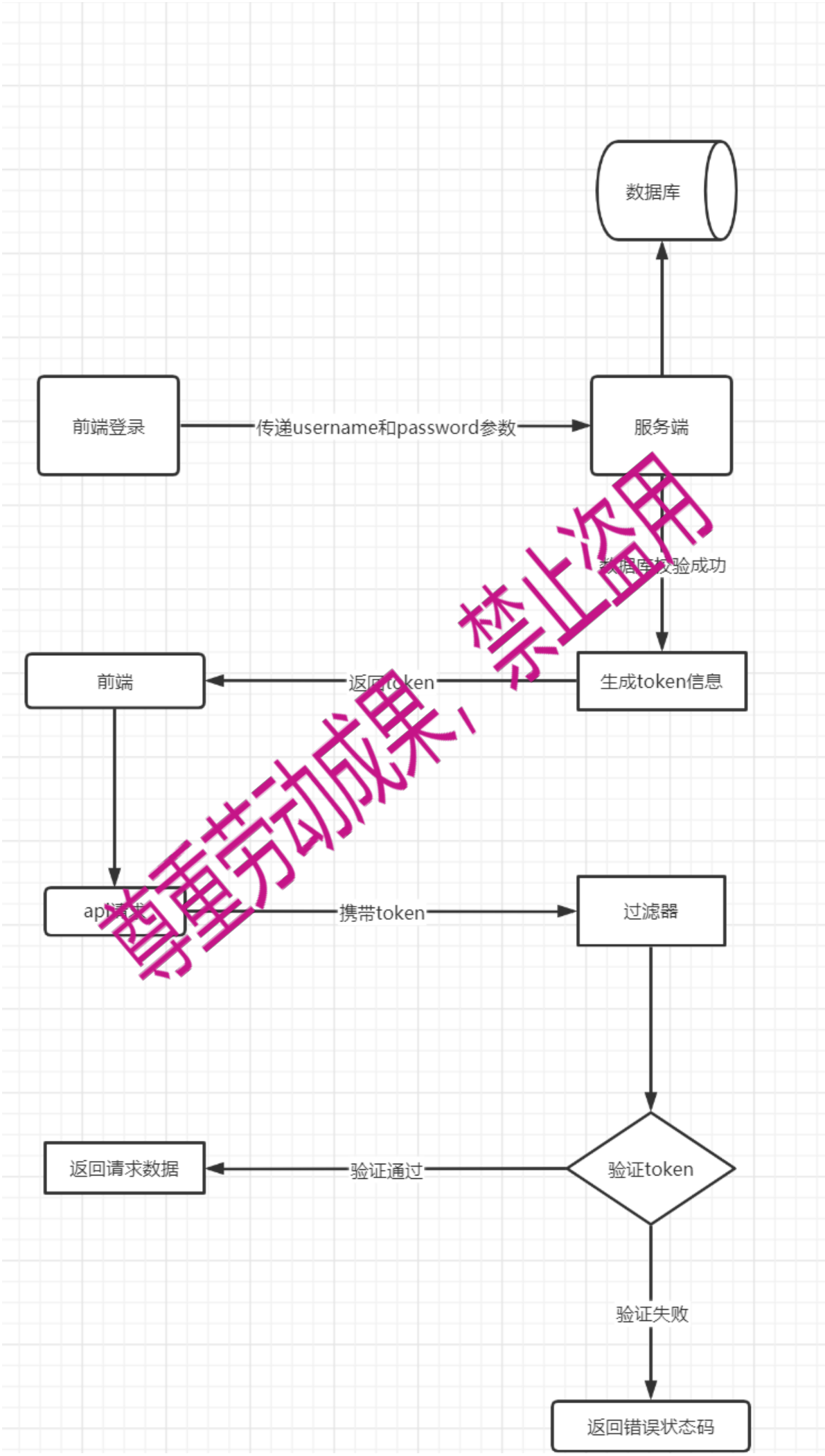


3.4 认证成功返回token

3.4.1 什么是token

Token是服务端生成的一串字符串，以作客户端进行请求的一个令牌，当第一次登录后，服务器生成一个Token便将此Token返回给客户端，以后客户端只需带上这个Token前来请求数据即可，无需再次带上用户名和密码。

3.4.2 token认证流程图


数据库
前端登录
传递username和password参数
服务端
数据库校验成功
前端
返回token
生成token信息
api请求
携带token
过滤器
返回请求数据
验证通过
验证token
验证失败
返回错误状态码

3.4.3 认证成功处理器返回token信息

（1）封装token返回的数据信息

```
package com.manong.utils;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class LoginResult {
    //用户编号
    private Long id;
    //状态码
    private int code;
    //token令牌
    private String token;
    //token过期时间
    private Long expireTime;
}
```

（2）编写token工具类

```
package com.manong.utils;

import com.manong.entity.User;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

@Data
@ConfigurationProperties(prefix = "jwt")
@Component
public class JwtUtils {
    //密钥
    private String secret;

    // 过期时间 毫秒
    private Long expiration;


    /**
     * 从数据声明生成令牌
     *
```

```
30          * @param claims 数据声明
31          * @return 令牌
32          */
33         private String generateToken(Map<String, Object> claims) {
34             Date expirationDate = new Date(System.currentTimeMillis() + expiration);
35             return
    Jwts.builder().setClaims(claims).setExpiration(expirationDate).signWith(Signatur
    eAlgorithm.HS512, secret).compact();
36         }
37
38         /**
39          * 从令牌中获取数据声明
40          *
41          * @param token 令牌
42          * @return 数据声明
43          */
44         public Claims getClaimsFromToken(String token) {
45             Claims claims;
46             try {
47                 claims =
    Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
48             } catch (Exception e) {
49                 claims = null;
50             }
51             return claims;
52         }
53
54         /**
55          * 生成令牌
56          *
57          * @param userDetails 用户
58          * @return 令牌
59          */
60         public String generateToken(UserDetails userDetails) {
61             Map<String, Object> claims = new HashMap<>(2);
62             claims.put(Claims.SUBJECT, userDetails.getUsername());
63             claims.put(Claims.ISSUED_AT, new Date());
64             return generateToken(claims);
65         }
66
67         /**
68          * 从令牌中获取用户名
69          *
70          * @param token 令牌
71          * @return 用户名
72          */
73         public String getUsernameFromToken(String token) {
74             String username;
75             try {
76                 Claims claims = getClaimsFromToken(token);
77                 username = claims.getSubject();
78             } catch (Exception e) {
79                 username = null;
80             }
81             return username;
82         }
83
84         /**
```

```
85          * 判断令牌是否过期
86          *
87          * @param token 令牌
88          * @return 是否过期
89          */
90         public Boolean isTokenExpired(String token) {
91             Claims claims = getClaimsFromToken(token);
92             Date expiration = claims.getExpiration();
93             return expiration.before(new Date());
94         }
95
96         /**
97          * 刷新令牌
98          *
99          * @param token 原令牌
100         * @return 新令牌
101         */
102        public String refreshToken(String token) {
103            String refreshedToken;
104            try {
105                Claims claims = getClaimsFromToken(token);
106                claims.put(Claims.ISSUED_AT, new Date());
107                refreshedToken = generateToken(claims);
108            } catch (Exception e) {
109                refreshedToken = null;
110            }
111            return refreshedToken;
112        }
113
114        /**
115         * 验证令牌
116         *
117         * @param token       令牌
118         * @param userDetails 用户
119         * @return 是否有效
120         */
121        public Boolean validateToken(String token, UserDetails userDetails) {
122            User user = (User) userDetails;
123            String username = getUsernameFromToken(token);
124            return (username.equals(user.getUsername()) && !isTokenExpired(token));
125        }
126    }
```


（3）编写全局配置文件

在application.properties全局配置文件中自定义jwt属性。

```
1   #jwt配置
2   #密钥
3   jwt.secret=com.manong
4   #过期时间
5   jwt.expiration=1800000
```

（4）认证成功处理器类返回token数据

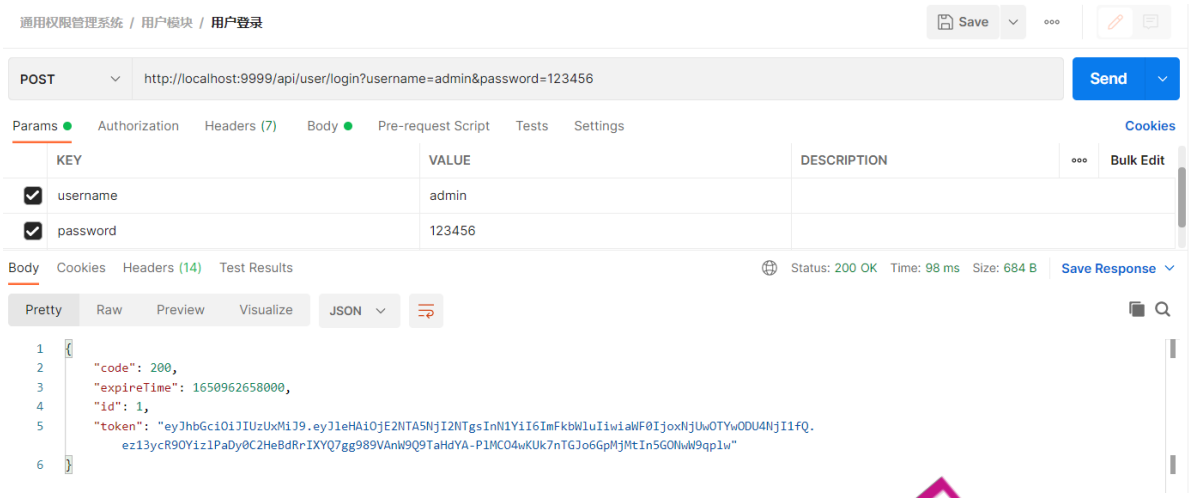
在原有的LoginSuccessHandler登录认证成功处理器类上加入jwt相关代码。

注意：在原有的基础上添加下面第37~48行的代码。

```java
package com.manong.config.security.handler;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.serializer.SerializerFeature;
import com.manong.entity.User;
import com.manong.utils.JwtUtils;
import com.manong.utils.LoginResult;
import com.manong.utils.ResultCode;
import io.jsonwebtoken.Jwts;
import org.springframework.security.core.Authentication;
import
org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 登录认证成功处理器类
*/
@Component
public class LoginSuccessHandler implements AuthenticationSuccessHandler {

    @Resource
    private JwtUtils jwtUtils;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) throws IOException,
ServletException {
        //设置客户端的响应的内容类型
        response.setContentType("application/json;charset=UTF-8");
        //获取当登录用户信息
        User user = (User) authentication.getPrincipal();
        //生成token
        String token = jwtUtils.generateToken(user);
        //设置token签名密钥及过期时间
        long expireTime = Jwts.parser() //获取DefaultJwtParser对象
                .setSigningKey(jwtUtils.getSecret())    //设置签名的密钥
                .parseClaimsJws(token.replace("jwt_", ""))
                .getBody().getExpiration().getTime();//获取token过期时间
        //创建登录结果对象
        LoginResult loginResult = new LoginResult(user.getId(),
ResultCode.SUCCESS,token,expireTime);
        //消除循环引用
        String result = JSON.toJSONString(loginResult,

 SerializerFeature.DisableCircularReferenceDetect);
        //获取输出流
        ServletOutputStream outputStream = response.getOutputStream();
        outputStream.write(result.getBytes(StandardCharsets.UTF_8));
        outputStream.flush();
        outputStream.close();
```

```
54          }
55      }
```

3.4.4 测试token



3.5 用户授权

当用户登录成功后，需要将用户及该用户对应的权限交给Spring Security进行管理，即用户的权限验证交给Spring Security进行统一校验。

3.5.1 修改Permission实体类

在Permission实体类中加入如下属性：

```java
/**
 * 子菜单列表
 */
@JsonInclude(JsonInclude.Include.NON_NULL)  //属性值为null不进行序列化操作
@TableField(exist = false)
private List<Permission> children = new ArrayList<Permission>();

/**
 * 用于前端判断是菜单、目录或按钮
 */
@TableField(exist = false)
private String value;

/**
 * 是否展开
 */
@TableField(exist = false)
private Boolean open;
```

3.5.2 编写PermissionMapper接口

在PermissionMapper接口中编写 根据用户ID查询权限列表 的方法。

```java
public interface PermissionMapper extends BaseMapper<Permission> {
    /**
     * 根据用户ID查询权限列表
     * @param userId
     * @return
     */
    List<Permission> findPermissionListByUserId(Long userId);
}
```

3.5.3 编写PermissionMapper映射文件

```xml
<select id="findPermissionListByUserId" resultType="com.manong.entity.Permission">
    select
        DISTINCT
p.id,p.parent_id,p.label,p.`code`,p.url,p.type,p.icon,p.remark,p.path,p.name
    from
        sys_user as u
            left join sys_user_role as ur  on u.id = ur.user_id
            left join sys_role  as r  on ur.role_id = r.id
            left join sys_role_permission  as rp on rp.role_id = r.id
            left join sys_permission as p on rp.permission_id = p.id

    where u.id =#{userId}
    order by p.id asc
</select>
```

3.5.4 编写PermissionService接口

在PermissionService接口中编写 `根据用户ID查询权限列表` 的方法。

```java
public interface PermissionService extends IService<Permission> {
    /**
     * 根据用户ID查询权限列表
     * @param userId
     * @return
     */
    List<Permission> findPermissionListByUserId(Long userId);
}
```

3.5.5 编写PermissionService接口实现类

在PermissionServiceImpl实现类中实现PermissionService接口的方法。

```java
@Service
@Transactional
public class PermissionServiceImpl extends ServiceImpl<PermissionMapper, Permission>
    implements PermissionService {

    /**
     * 根据用户ID查询权限列表
     *
     * @param userId
     * @return
     */
```

```
    @Override
    public List<Permission> findPermissionListByUserId(Long userId) {
        return baseMapper.findPermissionListByUserId(userId);
    }
}
```

3.5.6 User类添加权限菜单集合

在User类中添加权限菜单集合属性：

```
/**
 * 用户权限列表
 */
@TableField(exist = false)
private List<Permission> permissionList;
```

3.5.7 修改CustomerUserDetailsService类

在CustomerUserDetailsService类原有的基础上添加相应的用户授权代码。

```
package com.manong.config.security.service;

import com.manong.entity.Permission;
import com.manong.entity.User;
import com.manong.service.PermissionService;
import com.manong.service.UserService;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.List;
import java.util.stream.Collectors;

@Component
public class CustomerUserDetailsService implements UserDetailsService {

    @Resource
    private UserService userService;

    @Resource
    private PermissionService permissionService;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        //调用根据用户名查询用户信息的方法
        User user = userService.findUserByUserName(username);
        //如果对象为空，则认证失败
        if (user == null) {
            throw new UsernameNotFoundException("用户名或密码错误!");
        }
        //查询用户的拥有的权限列表
        List<Permission> permissionList =
permissionService.findPermissionListByUserId(user.getId());
```

```
            //获取权限编码
            List<String> collect = permissionList.stream()
                .filter(item -> item != null)
                .map(item -> item.getCode()).filter(item -> item != null)
                .collect(Collectors.toList());
            //转换成数组
            String[] strings = collect.toArray(new String[collect.size()]);
            //设置权限列表
            List<GrantedAuthority> authorityList =
    AuthorityUtils.createAuthorityList(strings);
            user.setAuthorities(authorityList);
            //设置菜单列表
            user.setPermissionList(permissionList);
            //返回用户信息
            return user;
        }
    }
```

3.6 token验证

3.6.1 添加Redis依赖

在pom.xml文件中添加Redis核心依赖。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

3.6.2 编写全局配置文件

在application.properties全局配置文件中加入Redis相关配置。

```
################################ Redis相关配置 ###################################
spring.redis.host=localhost
spring.redis.port=6379
spring.redis.database=0
spring.redis.timeout=10000
#自定义属性
spring.redis.expire=60000
```

3.6.3 编写Redis配置类

在com.manong.config.redis包下创建RedisConfig配置类和RedisService业务工具类。

（1）编写RedisConfig配置类。

```
package com.manong.config.redis;


import com.fasterxml.jackson.annotation.JsonAutoDetect;
import com.fasterxml.jackson.annotation.PropertyAccessor;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.data.redis.cache.RedisCacheConfiguration;
import org.springframework.data.redis.cache.RedisCacheManager;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.RedisSerializationContext;
import org.springframework.data.redis.serializer.RedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;

import java.time.Duration;

/**
 * redis配置类
 */
@Configuration
public class RedisConfig {
    //缓存过期时间
    @Value("${spring.redis.expire}")
    private Long expire;

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
factory) {
        RedisTemplate<String, Object> template = new RedisTemplate<String,
Object>();
        // 配置连接工厂
        template.setConnectionFactory(factory);
        //使用Jackson2JsonRedisSerialize 替换默认序列化(默认采用的是JDK序列化)
        Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer(Object.class);
        //解决查询缓存转换异常的问题
        ObjectMapper om = new ObjectMapper();
        //设置在生成json串时，对象中的成员的可见性
        om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        //存储到redis的json将是有类型的数据
        //指定序列化输入的类型，类必须是非final修饰的，final修饰的类，比如String,Integer等
会跑出异常
        om.activateDefaultTyping(om.getPolymorphicTypeValidator(),
ObjectMapper.DefaultTyping.NON_FINAL);
        jackson2JsonRedisSerializer.setObjectMapper(om);
        StringRedisSerializer stringRedisSerializer = new
StringRedisSerializer();
        // key采用String的序列化方式
        template.setKeySerializer(stringRedisSerializer);
        // hash的key也采用String的序列化方式
        template.setHashKeySerializer(stringRedisSerializer);
        // value序列化方式采用jackson
        template.setValueSerializer(jackson2JsonRedisSerializer);
        // hash的value序列化方式采用jackson
        template.setHashValueSerializer(jackson2JsonRedisSerializer);
        template.afterPropertiesSet();
        return template;
    }

    //@Cacheable注解字符集编码配置
    @Bean
    public RedisCacheManager cacheManager(RedisConnectionFactory factory) {
```

```
        RedisCacheConfiguration config =
RedisCacheConfiguration.defaultCacheConfig();
        config.entryTtl(Duration.ofMinutes(expire))//缓存过期时间

.serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(Red
isSerializer.string()))

.serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(R
edisSerializer.json()));

        return RedisCacheManager
                .builder(factory)
                .cacheDefaults(config)
                .build();
    }
}
```

（2）编写RedisService业务工具类

```
package com.manong.config.redis;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Component;

import java.util.concurrent.TimeUnit;

@Component
public class RedisService {
    @Autowired
    private RedisTemplate<String, Object> redisTemplate;
    //存缓存
    public void set(String key ,String value,Long timeOut){
        redisTemplate.opsForValue().set(key,value,timeOut, TimeUnit.SECONDS);
    }
    //取缓存
    public String get(String key){
        return (String) redisTemplate.opsForValue().get(key);
    }
    //清除缓存
    public void del(String key){
        redisTemplate.delete(key);
    }
}
```


3.6.4 设置登录认证请求地址

在application.properties全局配置文件中自定义登录验证的请求地址。

```
#登录请求地址(自定义)
request.login.url=/api/user/login
```

3.6.5 编写token验证过滤器类

在com.manong.config.security.filter包下创建CheckTokenFilter过滤器类。

```
package com.manong.config.security.filter;

import com.manong.config.redis.RedisService;
import com.manong.config.security.exception.CustomerAuthenticationException;
import com.manong.config.security.handler.LoginFailureHandler;
import com.manong.config.security.service.CustomerUserDetailsService;
import com.manong.utils.JwtUtils;
import lombok.Data;
import org.springframework.beans.factory.annotation.Value;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.util.ObjectUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.annotation.Resource;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * token验证过滤器
 */
@Data
@Component
public class CheckTokenFilter extends OncePerRequestFilter {

    @Resource
    private JwtUtils jwtUtils;
    @Resource
    private CustomerUserDetailsService customerUserDetailsService;
    @Resource
    private LoginFailureHandler loginFailureHandler;
    @Resource
    private RedisService redisService;
    //获取登录请求地址
    @Value("${request.login.url}")
    private String loginUrl;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws ServletException,
IOException {
        try {
            //获取当前请求的url地址
            String url = request.getRequestURI();
            //如果当前请求不是登录请求，则需要进行token验证
```

```
51              if (!url.equals(loginUrl)) {
52                  this.validateToken(request);
53              }
54          } catch (AuthenticationException e) {
55              loginFailureHandler.onAuthenticationFailure(request, response, e);
56          }
57          //登录请求不需要验证token
58          doFilter(request, response, filterChain);
59      }
60
61      /**
62       * 验证token
63       *
64       * @param request
65       */
66      private void validateToken(HttpServletRequest request) throws
    AuthenticationException {
67          //从头部获取token信息
68          String token = request.getHeader("token");
69          //如果请求头部没有获取到token，则从请求的参数中进行获取
70          if (ObjectUtils.isEmpty(token)) {
71              token = request.getParameter("token");
72          }
73          //如果请求参数中也不存在token信息，则抛出异常
74          if (ObjectUtils.isEmpty(token)) {
75              throw new CustomerAuthenticationException("token不存在");
76          }
77          //判断redis中是否存在该token
78          String tokenKey = "token_" + token;
79          String redisToken = redisService.get(tokenKey);
80          //如果redis里面没有token，证明该token失效
81          if (ObjectUtils.isEmpty(redisToken)) {
82              throw new CustomerAuthenticationException("token已过期");
83          }
84          //如果token和Redis中的token不一致，则验证失败
85          if (!token.equals(redisToken)) {
86              throw new CustomerAuthenticationException("token验证失败");
87          }
88          //如果存在token，则从token中解析出用户名
89          String username = jwtUtils.getUsernameFromToken(token);
90          //如果用户名为空，则解析失败
91          if (ObjectUtils.isEmpty(username)) {
92              throw new CustomerAuthenticationException("token解析失败");
93          }
94          //获取用户信息
95          UserDetails userDetails =
    customerUserDetailsService.loadUserByUsername(username);
96          //判断用户信息是否为空
97          if (userDetails == null) {
98              throw new CustomerAuthenticationException("token验证失败");
99          }
100         //创建身份验证对象
101         UsernamePasswordAuthenticationToken authenticationToken = new
    UsernamePasswordAuthenticationToken(userDetails, null,
    userDetails.getAuthorities());
102         authenticationToken.setDetails(new
    WebAuthenticationDetailsSource().buildDetails(request));
103         //设置到Spring Security上下文
```

```
104
        SecurityContextHolder.getContext().setAuthentication(authenticationToken);
105         }
106     }
```

3.6.6 配置token验证过滤器类

将CheckTokenFilter过滤器类交给Spring Security进行管理，在SpringSecurityConfig配置类中添加如下代码。

```
1   @Resource
2   private CheckTokenFilter checkTokenFilter;
3
4   @Override
5   protected void configure(HttpSecurity http) throws Exception {
6       //登录前进行过滤
7       http.addFilterBefore(checkTokenFilter,
    UsernamePasswordAuthenticationFilter.class);
8       http.formLogin()
9               //省略后续代码....
10  }
```

3.6.7 编写自定义异常类

在com.manong.config.security.exception包下编写CustomerAuthenticationException异常处理类。

```
1   package com.manong.config.security.exception;
2
3   import org.springframework.security.core.AuthenticationException;
4
5   /**
6    * 自定义验证异常类
7    */
8   public class CustomerAuthenticationException extends AuthenticationException {
9
10      public CustomerAuthenticationException(String message){
11          super(message);
12      }
13  }
```

3.6.8 token验证失败处理

在LoginFailureHandler用户认证失败处理类中加入判断。

```
1   /**
2    * 用户认证失败处理类
3    */
4   @Component
5   public class LoginFailureHandler implements AuthenticationFailureHandler {
6       @Override
7       public void onAuthenticationFailure(HttpServletRequest request,
8                                           HttpServletResponse response,
9                                           AuthenticationException exception) throws
    IOException, ServletException {
10          //设置客户端响应编码格式
11          response.setContentType("application/json;charset=UTF-8");
12          //获取输出流
13          ServletOutputStream outputStream = response.getOutputStream();
```

```
            String message = null;//提示信息
            int code = 500;//错误编码
            //判断异常类型
            if(exception instanceof AccountExpiredException){
                message = "账户过期,登录失败！";
            }else if(exception instanceof BadCredentialsException){
                message = "用户名或密码错误,登录失败！";
            }else if(exception instanceof CredentialsExpiredException){
                message = "密码过期,登录失败！";
            }else if(exception instanceof DisabledException){
                message = "账户被禁用,登录失败！";
            }else if(exception instanceof LockedException){
                message = "账户被锁,登录失败！";
            }else if(exception instanceof InternalAuthenticationServiceException){
                message = "账户不存在,登录失败！";
            }else if(exception instanceof CustomerAuthenticationException){
                message = exception.getMessage();
                code = 600;
            }else{
                message = "登录失败！";
            }
            //将错误信息转换成JSON
            String result =
    JSON.toJSONString(Result.error().code(code).message(message));
            outputStream.write(result.getBytes(StandardCharsets.UTF_8));
            outputStream.flush();
            outputStream.close();
        }
    }
```

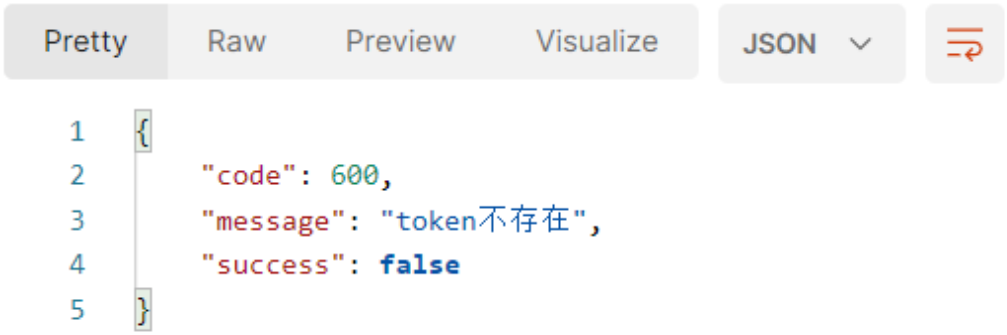
3.6.9 认证成功处理

修改LoginSuccessHandler登录认证成功处理类，将token保存到Redis缓存中。

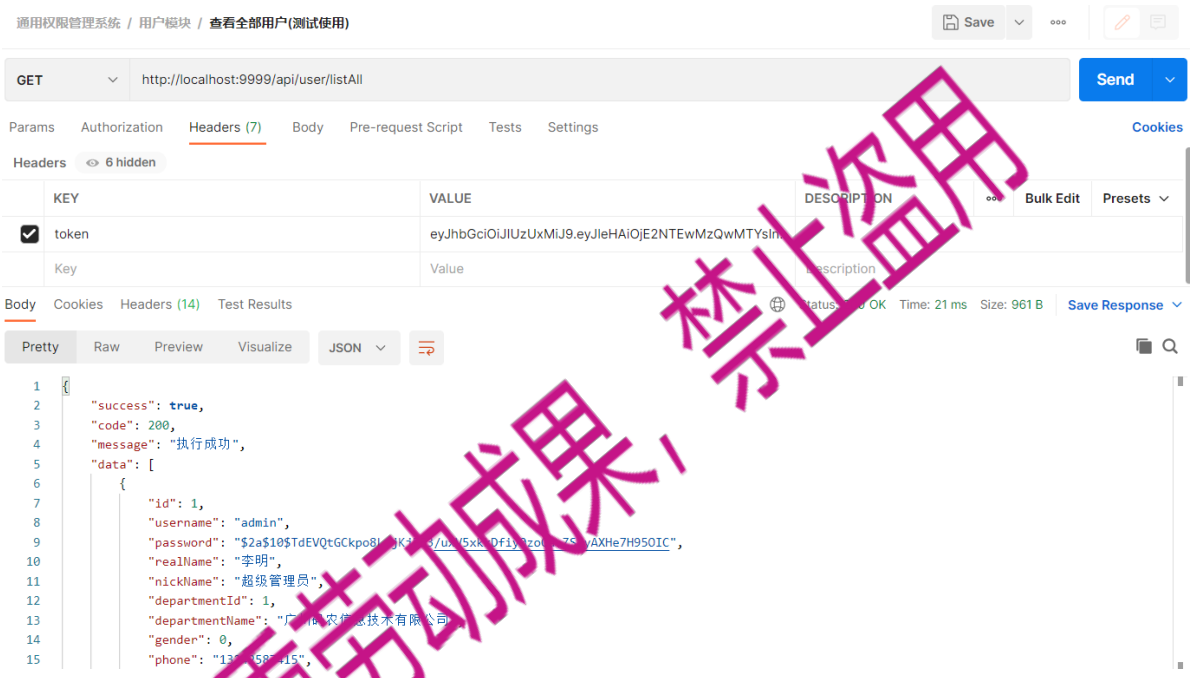
```
@Component
public class LoginSuccessHandler implements AuthenticationSuccessHandler {

    @Resource
    private JwtUtils jwtUtils;
    @Resource
    private RedisService redisService;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) throws IOException,
ServletException {
        //省略原有代码......

        //把生成的token存到redis
        String tokenKey = "token_"+token;
        redisService.set(tokenKey,token,jwtUtils.getExpiration() / 1000);
    }
}
```

3.6.10 token验证测试

测试`http://localhost:9999/api/user/listAll`，首次测试会出现如下图所示：



生成token后，将token与请求一并携带到后端接口中，如下图所示：



3.7 刷新token信息

3.7.1 创建TokenVo类

在com.manong.vo包下创建TokenVo类，该类用于保存Token信息。

```
package com.manong.vo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class TokenVo {
    //过期时间
    private Long expireTime;
    //token
    private String token;
```

```
}
```

3.7.2 编写刷新token方法

在com.manong.controller包下创建SysUserController控制器类，并在该类中编写refreshToken刷新token的方法。

```
package com.manong.controller;

import com.manong.config.redis.RedisService;
import com.manong.utils.JwtUtils;
import com.manong.utils.Result;
import com.manong.vo.TokenVo;
import io.jsonwebtoken.Jwts;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.util.ObjectUtils;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;

@RestController
@RequestMapping("/api/sysUser")
public class SysUserController {
    @Resource
    private RedisService redisService;
    @Resource
    private JwtUtils jwtUtils;

    /**
     * 刷新token
     *
     * @param request
     * @return
     */
    @PostMapping("/refreshToken")
    public Result refreshToken(HttpServletRequest request) {
        //从header中获取前端提交的token
        String token = request.getHeader("token");
        //如果header中没有token，则从参数中获取
        if (ObjectUtils.isEmpty(token)) {
            token = request.getParameter("token");
        }
        //从Spring Security上下文获取用户信息
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        //获取身份信息
        UserDetails details = (UserDetails) authentication.getPrincipal();
        //重新生成token
        String reToken = "";
        //验证原来的token是否合法
        if (jwtUtils.validateToken(token, details)) {
            //生成新的token
            reToken = jwtUtils.refreshToken(token);
```

```
            }
            //获取本次token的到期时间，交给前端做判断
            long expireTime = Jwts.parser().setSigningKey(jwtUtils.getSecret())
                    .parseClaimsJws(reToken.replace("jwt_", ""))
                    .getBody().getExpiration().getTime();
            //清除原来的token信息
            String oldTokenKey = "token_" + token;
            redisService.del(oldTokenKey);
            //存储新的token
            String newTokenKey = "token_" + reToken;
            redisService.set(newTokenKey, reToken, jwtUtils.getExpiration() / 1000);
            //创建TokenVo对象
            TokenVo tokenVo = new TokenVo(expireTime, reToken);
            //返回数据
            return Result.ok(tokenVo).message("token生成成功");
        }
    }
```

3.7.3 接口运行测试

1. 先运行用户登录请求，生成token信息
2. 测试查询全部用户信息，预期结果是查询成功
3. 运行刷新token接口，重新生成token信息
4. 再次测试查询全部用户信息，预期结果是token过期。

4. 获取用户信息及菜单信息

4.1 获取登录用户信息

4.1.1 封装用户信息类

在com.manong.entity包下创建UserInfo登录用户信息类，该类封装的属性需要和前端页面显示的数据属性一致。

```
package com.manong.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class UserInfo implements Serializable {
    private Long id;//用户ID
    private String name;//用户名称
    private String avatar;//头像
    private String introduction;//介绍
    private Object[] roles;//角色权限集合
}
```

4.1.2 编写获取用户信息接口方法

在SysUserController控制器类中编写获取用户信息的方法。

```
/**
 * 获取用户信息
 *
 * @return
 */
@GetMapping("/getInfo")
public Result getInfo() {
    //从Spring Security上下文获取用户信息
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    //判断authentication对象是否为空
    if (authentication == null) {
        return Result.error().message("用户信息查询失败");
    }
    //获取用户信息
    User user = (User) authentication.getPrincipal();
    //用户权限集合
    List<Permission> permissionList = user.getPermissionList();
    //获取角色权限编码字段
    Object[] roles =
        permissionList.stream()
        .filter(Objects::nonNull)
        .map(Permission::getCode).toArray();
    //创建用户信息对象
    UserInfo userInfo = new UserInfo(user.getId(), user.getNickName(),
user.getAvatar(), null, roles);
    //返回数据
    return Result.ok(userInfo).message("用户信息查询成功");
}
```

4.2 获取登录用户的菜单信息

4.2.1 封装前端路由菜单对象

在com.manong.vo包下编写RouterVo类，该类封装的属性需要和前端vue-element-admin的router数据格式一致。

```
package com.manong.vo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;

import java.util.ArrayList;
import java.util.List;

@Data
@JsonInclude(JsonInclude.Include.NON_EMPTY)
public class RouterVo {
    //路由地址
    private String path;
    //路由对应的组件
```

```
    private String component;
    //是否显示
    private boolean alwaysShow;
    //路由名称
    private String name;
    //路由meta信息
    private Meta meta;

    @Data
    @AllArgsConstructor
    public class Meta {
        private String title;//标题
        private String icon;//图标
        private Object[] roles;//角色列表
    }

    //子路由
    private List<RouterVo> children = new ArrayList<RouterVo>();

}
```

4.2.2 编写菜单树工具类

在com.manong.utils包下编写MenuTree菜单树工具类。

```
package com.manong.utils;

import com.manong.entity.Permission;
import com.manong.vo.RouterVo;
import org.springframework.beans.BeanUtils;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Optional;

/**
 * 生成菜单树
 */
public class MenuTree {

    /**
     * 生成路由
     *
     * @param meuList 菜单列表
     * @param pid     父级菜单
     * @return
     */
    public static List<RouterVo> makeRouter(List<Permission> meuList, Long pid) {
        //创建集合保存路由列表
        List<RouterVo> routerList = new ArrayList<RouterVo>();
        //如果menuList菜单列表不为空，则使用菜单列表，否则创建集合对象
        Optional.ofNullable(meuList).orElse(new ArrayList<Permission>())
                //筛选不为空的菜单及菜单父id相同的数据
                .stream().filter(item -> item != null && item.getParentId() ==
pid)
                .forEach(item -> {
                    //创建路由对象
```

```
                    RouterVo router = new RouterVo();
                    router.setName(item.getName());//路由名称
                    router.setPath(item.getPath());//路由地址
                    //判断是否是一级菜单
                    if (item.getParentId() == 0L) {
                        router.setComponent("Layout");//一级菜单组件
                        router.setAlwaysShow(true);//显示路由
                    } else {
                        router.setComponent(item.getUrl());//具体的组件
                        router.setAlwaysShow(false);//折叠路由
                    }
                    //设置meta信息
                    router.setMeta(router.new Meta(item.getLabel(),
                                                   item.getIcon(),
                                                   item.getCode().split(",")));
                    //递归生成路由
                    List<RouterVo> children = makeRouter(meuList, item.getId());
                    router.setChildren(children);//设置子路由到路由对象中
                    //将路由信息添加到集合中
                    routerList.add(router);
                });
        return routerList;
    }

    /**
     * 生成菜单树
     *
     * @param meuList
     * @param pid
     * @return
     */
    public static List<Permission> makeMenuTree(List<Permission> meuList, Long
pid) {
        //创建集合保存菜单
        List<Permission> permissionList = new ArrayList<Permission>();
        //如果meuList菜单列表不为空，则使用菜单列表，否则创建集合对象
        Optional.ofNullable(meuList).orElse(new ArrayList<Permission>())
                .stream().filter(item -> item != null &&
Objects.equals(item.getParentId(), pid))
                .forEach(item -> {
                    //创建菜单权限对象
                    Permission permission = new Permission();
                    //复制属性
                    BeanUtils.copyProperties(item, permission);
                    //获取每一个item的下级菜单,递归生成菜单树
                    List<Permission> children = makeMenuTree(meuList,
item.getId());
                    //设置子菜单
                    permission.setChildren(children);
                    //将菜单对象添加到集合
                    permissionList.add(permission);
                });
        return permissionList;
    }
}
```

4.2.3编写获取菜单数据接口方法

在SysUserController控制器类中编写getMenuList获取菜单数据的方法。

```
/**
 * 获取菜单数据
 *
 * @return
 */
@GetMapping("/getMenuList")
public Result getMenuList() {
    //从Spring Security上下文获取用户信息
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    //获取用户信息
    User user = (User) authentication.getPrincipal();
    //获取相应的权限
    List<Permission> permissionList = user.getPermissionList();
    //筛选目录和菜单
    List<Permission> collect = permissionList.stream()
        .filter(item -> item != null && item.getType() !=2)
        .collect(Collectors.toList());
    //生成路由数据
    List<RouterVo> routerVoList = MenuTree.makeRouter(collect, 0L);
    //返回数据
    return Result.ok(routerVoList).message("菜单数据获取成功");
}
```

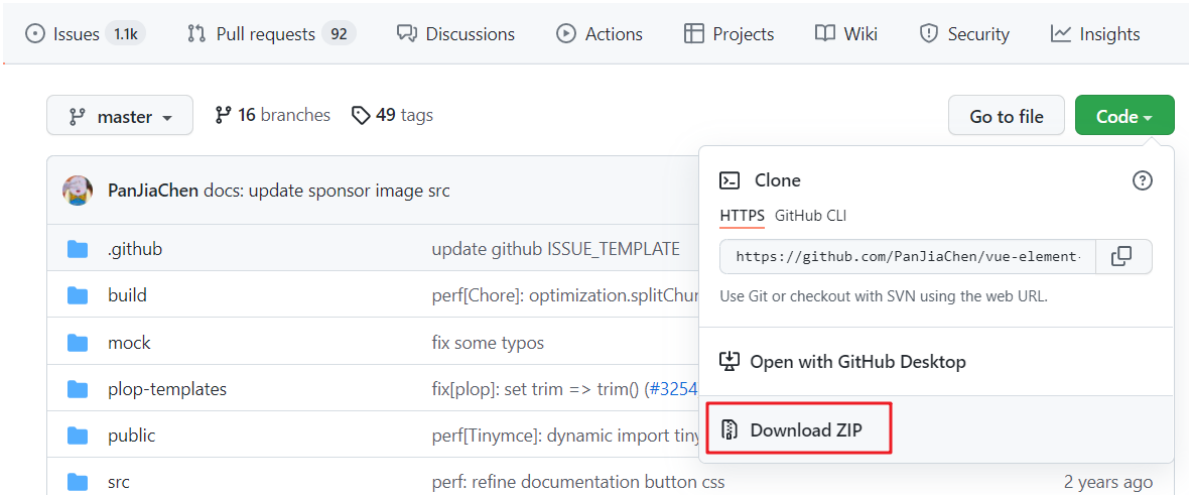

5. 搭建前端项目环境

5.1 vue-element-admin

5.1.1下载vue-element-admin

官网地址：https://panjiachen.gitee.io/vue-element-admin-site/zh/

下载地址：https://github.com/PanJiaChen/vue-element-admin

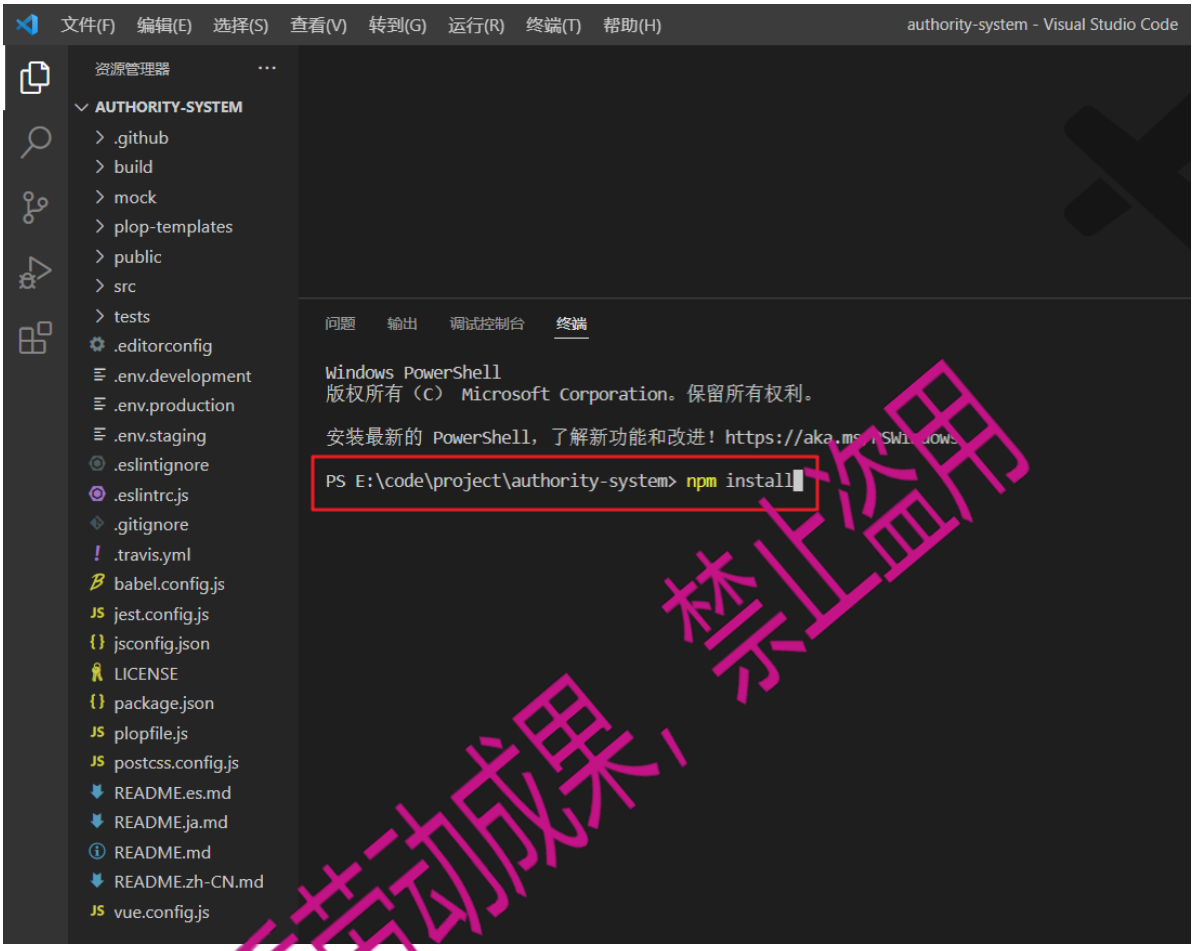


5.1.2 运行vue-element-admin

注意：在运行前必须具备node.js、git和python运行环境。

（1）将下载好的vue-element-admin解压，解压后放置到具体的某个目录下，可以对当前项目进行重命名，此处重命名为 `authority-system`。

（2）进入项目的根目录，执行 `npm install` 指令。

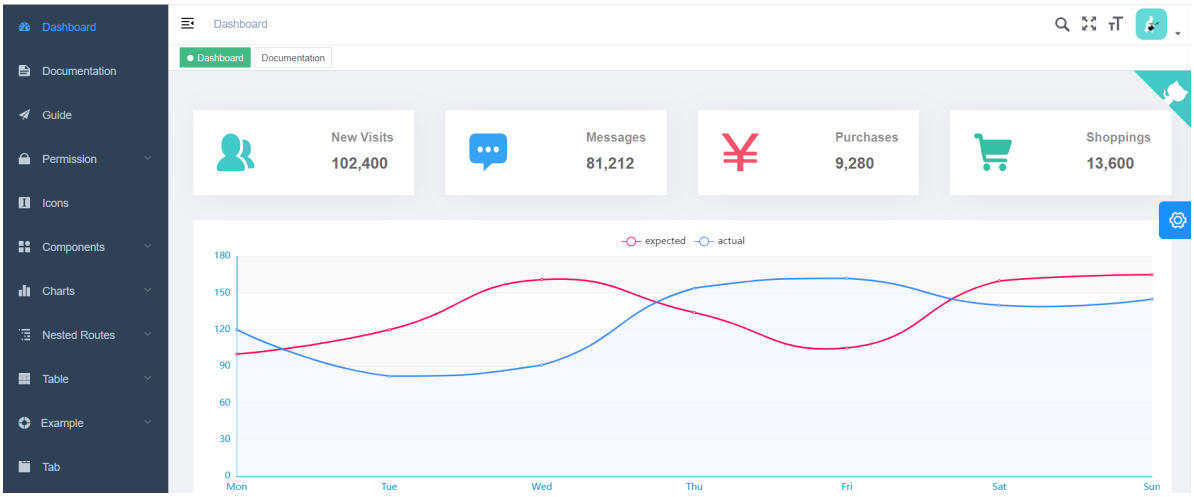


等待命令成功执行完成，此处需要具备node.js、git和python运行环境。

（3）在项目根目录下使用 `npm run dev` 指令运行项目。

```
PS E:\code\project\authority-system> npm run dev

> vue-element-admin@4.4.0 dev
> vue-cli-service serve

 INFO  Starting development server...
98% after emitting CopyPlugin

 DONE  Compiled successfully in 31271ms


  App running at:
  - Local:   http://localhost:9527/
  - Network: unavailable

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

启动完成后会自动打开浏览器访问 http://localhost:9527， 你看到下面的页面就代表操作成功了。



5.1.3 关闭ESLint检查

（1）vue.config.js

将vue.config.js脚本文件中的`lintOnSave: process.env.NODE_ENV === 'development'`改成`lintOnSave:false`

（2）package.json

将package.json文件中的下述代码删除。

```
"husky": {
    "hooks": {
      "pre-commit": "lint-staged"
    }
  }
```

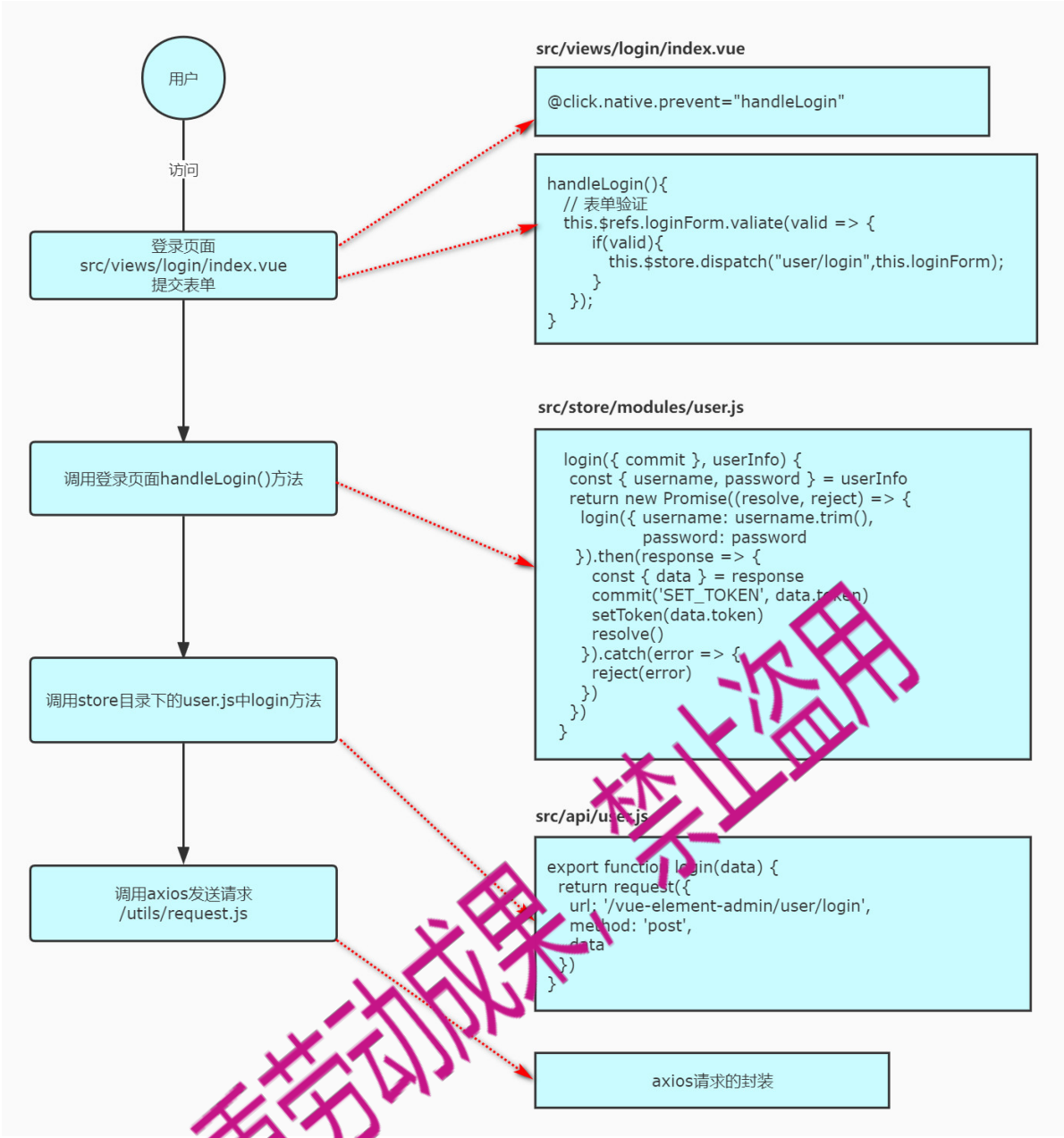
（3）.eslintignore

在.eslintignore文件最后一行加入`*`。

```
build/*.js
src/assets
public
dist
*
```


5.2 前端登录流程分析

5.2.1 登录流程图



5.2.2 登录代码分析

（1）用户进入登录页面，即工程的 src/views/login/index.vue ,点击 login按钮，调用handleLogin()方法，如下所示：

```
handleLogin() {
  //表单验证
  this.$refs.loginForm.validate((valid) => {
    //如果验证通过
    if (valid) {
      //开启进度条
      this.loading = true;
      //调用src/store/modules/user.js中的login方法
      this.$store
        .dispatch("user/login", this.loginForm)
        .then(() => {
          //路由转发到指定地址
          this.$router.push({
            path: this.redirect || "/",
            query: this.otherQuery,
          });
          //关闭进度条
```

```
            this.loading = false;
          })
          .catch(() => {
            //关闭进度条
            this.loading = false;
          });
      } else {
        console.log("error submit!!");
        return false;
      }
    });
  }
```

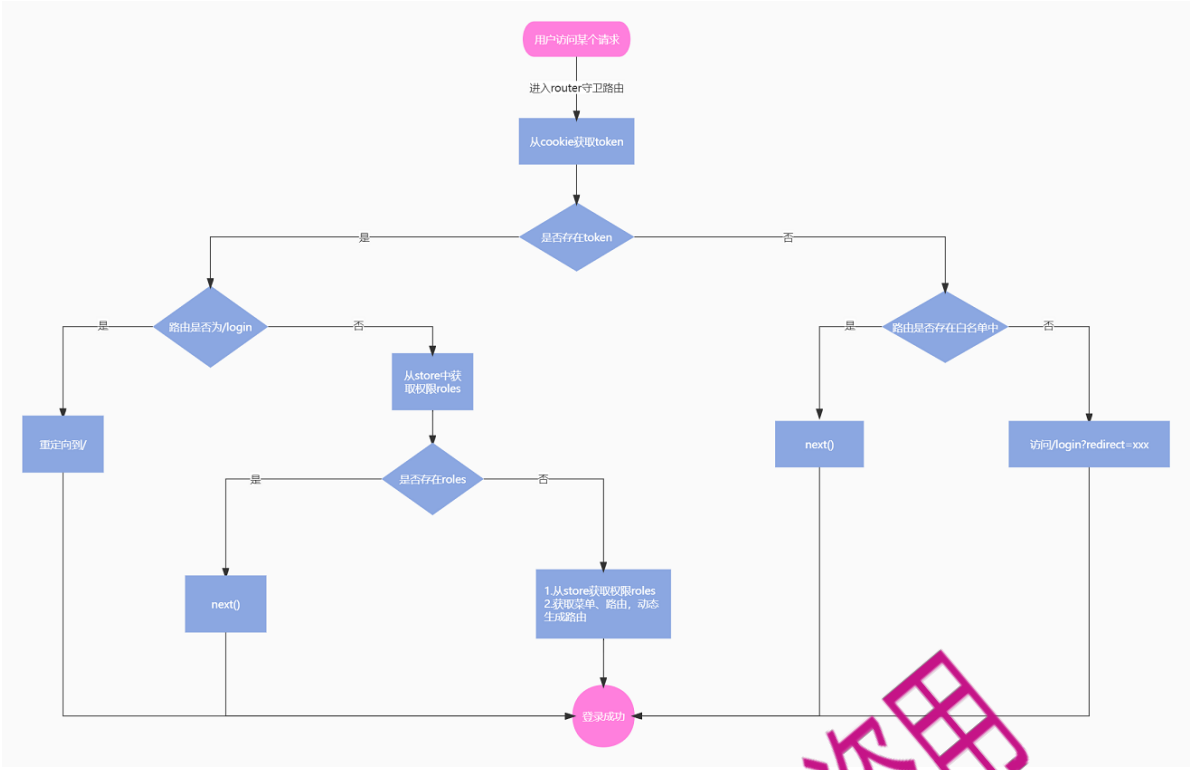
（2）登录调用的store中的方法， 即src/store/modules/user.js。

```
// 用户登录
login({ commit }, userInfo) {
  //从用户信息userInfo中解构出用户名和密码
  const { username, password } = userInfo
  return new Promise((resolve, reject) => {
    //调用src/api/user.js文件中的login()方法
    login({ username: username.trim(), password: password }).then(response => {
      //从response中解构出返回的data数据
      const { data } = response
      //将返回的token数据保存到store中，作为全局变量使用
      commit('SET_TOKEN', data.token)
      //将token信息保存到cookie中
      setToken(data.token)
      resolve()
    }).catch(error => {
      reject(error)
    })
  })
}
```

（3）store中调用的login，即工程中api/user中的login方法，此处主要是通过utils/request的axios发送请求到后端获取token。

```
//注意：此处调用的是mock数据，并非真实的后端接口数据
//mock数据在工程的 mock/user.js里面，此处的token是写死的
export function login(data) {
  return request({
    url: '/vue-element-admin/user/login',
    method: 'post',
    data
  })
}
```

5.2.3 权限验证流程图



6. 用户登录与退出

6.1 前端用户登录

6.1.1 登录页面效果图

6.1.2 改造前端登录页面

将src/views/login/index.vue页面进行修改，页面代码如下所示：

```
<template>
  <div class="login-container">
    <el-form
      ref="loginForm"
      :model="loginForm"
      :rules="loginRules"
      class="login-form"
      autocomplete="on"
      label-position="left"
    >
      <div class="title-container">
        <h3 class="title">通用权限管理系统</h3>
      </div>

      <el-form-item prop="username">
        <span class="svg-container">
          <svg-icon icon-class="user" />
        </span>
        <el-input
          ref="username"
          v-model="loginForm.username"
          placeholder="请输入用户名"
          name="username"
          type="text"
          tabindex="1"
          autocomplete="on"
        />
      </el-form-item>

      <el-tooltip
        v-model="capsTooltip"
        content="Caps lock is On"
        placement="right"
        manual
      >
        <el-form-item prop="password">
          <span class="svg-container">
            <svg-icon icon-class="password" />
          </span>
          <el-input
            :key="passwordType"
            ref="password"
            v-model="loginForm.password"
            :type="passwordType"
            placeholder="请输入密码"
            name="password"
            tabindex="2"
            autocomplete="on"
            @keyup.native="checkCapslock"
            @blur="capsTooltip = false"
            @keyup.enter.native="handleLogin"
          />
          <span class="show-pwd" @click="showPwd">
            <svg-icon
```

```
              :icon-class="passwordType === 'password' ? 'eye' : 'eye-open'"
            />
          </span>
        </el-form-item>
      </el-tooltip>

      <el-button
        :loading="loading"
        type="primary"
        style="width: 100%; margin-bottom: 30px"
        @click.native.prevent="handleLogin"
        >登录</el-button
      >
    </el-form>
  </div>
</template>
```

6.1.3 表单验证代码

src/views/login/index.vue页面的表单验证关键代码如下所示：

```
export default {
  name: "Login",
  data() {
    return {
      //登录表单对象
      loginForm: {
        username: "",//用户名
        password: "",//密码
      },
      //表单验证规则
      loginRules: {
        username: [
          { required: true, trigger: "blur", message:"用户名不能为空" },
        ],
        password: [
          { required: true, trigger: "blur", message:"密码不能为空" },
        ],
      }
    };
  },
  methods: {
    /**
     * 登录处理
     */
    handleLogin() {
      //表单验证
      this.$refs.loginForm.validate((valid) => {
        //如果验证通过
        if (valid) {
          //开启进度条
          this.loading = true;
          //调用src/store/modules/user.js中的login方法
          this.$store
            .dispatch("user/login", this.loginForm)
            .then(() => {
              //路由转发到指定地址
              this.$router.push({
```

```
                path: this.redirect || "/",
                query: this.otherQuery,
              });
              //关闭进度条
              this.loading = false;
            })
            .catch(() => {
              //关闭进度条
              this.loading = false;
            });
        } else {
          console.log("error submit!!");
          return false;
        }
      });
    }
  };
```

6.1.4 登录前后端接口联调

（1）修改vue.config.js文件

将vue.config.js文件中的mock请求注释掉，并设置代理。

```
devServer: {
  port: port,
  open: true,
  overlay: {
    warnings: false,
    errors: true
  },
  // before: require('./mock/mock-server.js')
  //代理配置
  proxy: {
    [process.env.VUE_APP_BASE_API]: {
      target: "http://localhost:8089/api",
      changeOrigin: true,
      pathRewrite: {
        '^/api': ''
      }
    }
  }
}
```

（2）修改main.js

将main.js文件中的如下代码注释。

```
if (process.env.NODE_ENV === 'production') {
  const { mockXHR } = require('../mock')
  mockXHR()
}
```

（3）修改src/utils目录下的request.js脚本文件。

注意：在修改该文件前需要安装qs依赖，该依赖用于将参数进行序列化，如：/user?username=xxx&password=&&&

```
npm install qs
```

随后将如下代码替换调用原有的request.js文件。

```
import axios from 'axios'
import { MessageBox, Message } from 'element-ui'
import store from '@/store'
import { getToken } from '@/utils/auth'
import qs from 'qs'
// create an axios instance
const service = axios.create({
  baseURL: process.env.VUE_APP_BASE_API, // url = base url + request url
  // withCredentials: true, // send cookies when cross-domain requests
  timeout: 5000 // request timeout
})

// request interceptor
service.interceptors.request.use(
  config => {
    console.log(config)
    // do something before request is sent

    if (store.getters.token) {
      // let each request carry token
      // ['X-Token'] is a custom headers key
      // please modify it according to the actual situation
      config.headers['token'] = getToken()
    }
    return config
  },
  error => {
    // do something with request error
    console.log(error) // for debug
    return Promise.reject(error)
  }
)

// response interceptor
service.interceptors.response.use(
  /**
   * If you want to get http information such as headers or status
   * Please return  response => response
  */

  /**
   * Determine the request status by custom code
   * Here is just an example
   * You can also judge the status by HTTP Status Code
   */
  response => {
    const res = response.data

    // if the custom code is not 20000, it is judged as an error.
    if (res.code !== 200) {
      Message({
        message: res.message || 'Error',
        type: 'error',
        duration: 5 * 1000
      })

```

```
57        // 50008: Illegal token; 50012: Other clients logged in; 50014: Token
  expired;
58        if (res.code === 50008 || res.code === 50012 || res.code === 50014) {
59          // to re-login
60          MessageBox.confirm('You have been logged out, you can cancel to stay on
  this page, or log in again', 'Confirm logout', {
61            confirmButtonText: 'Re-Login',
62            cancelButtonText: 'Cancel',
63            type: 'warning'
64          }).then(() => {
65            store.dispatch('user/resetToken').then(() => {
66              location.reload()
67            })
68          })
69        }
70        return Promise.reject(new Error(res.message || 'Error'))
71      } else {
72        return res
73      }
74    },
75    error => {
76      console.log('err' + error) // for debug
77      Message({
78        message: error.message,
79        type: 'error',
80        duration: 5 * 1000
81      })
82      return Promise.reject(error)
83    }
84  )
85
86  //请求方法
87  const http = {
88    post(url, params) {
89      return service.post(url, params, {
90        transformRequest: [(params) => {
91          return JSON.stringify(params)
92        }],
93        headers: {
94          'Content-Type': 'application/json'
95        }
96      })
97    },
98    put(url, params) {
99      return service.put(url, params, {
100       transformRequest: [(params) => {
101         return JSON.stringify(params)
102       }],
103       headers: {
104         'Content-Type': 'application/json'
105       }
106     })
107   },
108   get(url, params) {
109     return service.get(url, {
110       params: params,
111       paramsSerializer: (params) => {
112         return qs.stringify(params)
```

```
113         }
114       })
115     },
116     getRestApi(url, params) {
117       let _params
118       if (Object.is(params, undefined || null)) {
119         _params = ''
120       } else {
121         _params = '/'
122         for (const key in params) {
123           console.log(key)
124           console.log(params[key])
125           if (params.hasOwnProperty(key) && params[key] !== null && params[key]
    !== '') {
126             _params += `${params[key]}/`
127           }
128         }
129         _params = _params.substr(0, _params.length - 1)
130       }
131       console.log(_params)
132       if (_params) {
133         return service.get(`${url}${_params}`)
134       } else {
135         return service.get(url)
136       }
137     },
138     delete(url, params) {
139       let _params
140       if (Object.is(params, undefined || null)) {
141         _params = ''
142       } else {
143         _params = '/'
144         for (const key in params) {
145           // eslint-disable-next-line no-prototype-builtins
146           if (params.hasOwnProperty(key) && params[key] !== null && params[key]
    !== '') {
147             _params += `${params[key]}/`
148           }
149         }
150         _params = _params.substr(0, _params.length - 1)
151       }
152       if (_params) {
153         return service.delete(`${url}${_params}`).catch(err => {
154           message.error(err.msg)
155           return Promise.reject(err)
156         })
157       } else {
158         return service.delete(url).catch(err => {
159           message.error(err.msg)
160           return Promise.reject(err)
161         })
162       }
163     },
164     upload(url, params) {
165       return service.post(url, params, {
166         headers: {
167           'Content-Type': 'multipart/form-data'
168         }
```

```
      })
    },
    login(url, params) {
      return service.post(url, params, {
        transformRequest: [(params) => {
          return qs.stringify(params)
        }],
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded'
        }
      })
    }
  }
  export default http
```

（4）修改src/api/user.js文件

将原有的user.js文件代码替换成如下代码：

```
import http from '@/utils/request'

/**
 * 用户登录
 * @returns
 */
export async function login(data) {
  return await http.login("/api/user/login",data)
}

/**
 * 获取用户信息和权限信息
 * @returns
 */
export async function getInfo() {
  return await http.get("/api/sysUser/getInfo")
}

/**
 * 退出登录
 * @returns
 */
export async function logout(param){
  return await http.post("/api/sysUser/loginOut",param);
}
```

（5）修改.env.development和.env.production请求地址

注意：该请求地址

```
VUE_APP_BASE_API = 'http://localhost:9999/'
```

（6）修改src/store/modules/user.js中的登录方法。

```
const actions = {
  // 用户登录
  login({ commit }, userInfo) {
    //从用户信息userInfo中解构出用户名和密码
    const { username, password } = userInfo
    return new Promise((resolve, reject) => {
```

```
        //调用src/api/user.js文件中的login()方法
        login({ username: username.trim(), password: password }).then(response => {
          //从response中解构出返回的token数据
          const { token } = response
          //将返回的token数据保存到store中，作为全局变量使用
          commit('SET_TOKEN', token)
          //将token信息保存到cookie中
          setToken(token)
          resolve()
        }).catch(error => {
          reject(error)
        })
      })
  }
```

（7）运行vue项目，在浏览器中输入用户名和密码，登录成功后即可进入到前台首页。

6.2 用户退出登录接口

6.2.1 编写用户退出登录接口

在SysUserController控制器类中编写用户退出登录的方法。

```
/**
 * 用户退出
 * @param request
 * @param response
 * @return
 */
@PostMapping("/logout")
public Result logout(HttpServletRequest request, HttpServletResponse response) {
    //获取token
    String token = request.getParameter("token");
    //如果没有头部获取token，那么从参数里面获取
    if (ObjectUtils.isEmpty(token)) {
        token = request.getHeader("token");
    }
    //获取用户相关信息
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null) {
        //清空用户信息
        new SecurityContextLogoutHandler().logout(request, response,
authentication);
        //清空redis里面的token
        String key = "token_" + token;
        redisService.del(key);
    }
    return Result.ok().message("用户退出成功");
}
```

6.2.2 接口运行测试

1. 执行用户登录请求后，再查询所有用户信息。
2. 执行用户退出登录请求，再次查询所有用户信息，预期结果是token已过期。

6.3 用户退出登录前后端联调

6.3.1 编写退出登录方法

在src/api/user.js文件中编写logout()方法，代码如下：

```
/**
 * 退出登录
 * @returns
 */
export async function logout(param){
  return await http.post("/api/sysUser/logout",param);
}
```

6.3.2 编写清空sessionStorage方法

在src/utils/auth.js中添加如下方法：

```
/**
 * 清空sessionStorage
 */
export function clearStorage(){
  return sessionStorage.clear();
}
```

6.3.3 编写前端页面代码

在src/layout/components下找到NavBar.vue组件，进行如下修改。

（1）引入相应脚本

```
//导入脚本
import { logout } from "@/api/user";
import { getToken, removeToken, clearStorage } from "@/utils/auth";
```

（2）修改页面下拉菜单

```
<el-dropdown-menu slot="dropdown">
    <router-link to="/">
        <el-dropdown-item>首页</el-dropdown-item>
    </router-link>
    <router-link to="/profile/index">
        <el-dropdown-item>个人中心</el-dropdown-item>
    </router-link>
    <el-dropdown-item divided @click.native="logout">
        <span style="display: block">退出系统</span>
    </el-dropdown-item>
</el-dropdown-menu>
```

（3）在methods中添加loginOut()方法

```
logout() {
    this.$confirm("确定退出系统吗?", "提示", {
```

```
        confirmButtonText: "确定",
        cancelButtonText: "取消",
        type: "warning",
    }).then(async() => {
        //请求参数
        let params = {token:getToken()};
        //发送退出请求
        let res = await logout(params);
        //判断是否成功
        if(res.success){
            //清空token
            removeToken();
            clearStorage();
            //跳转到登录页面
            window.location.href="/login";
        }
    });
}
```

6.4 封装信息确认提示框

6.4.1 编写信息确认提示框代码

在src/utils目录下创建myconfirm.js文件，代码如下所示

```
import { MessageBox, Message } from 'element-ui'
//删除弹框
export default function myconfirm(text) {
    return new Promise((resolve, reject) => {
        MessageBox.confirm(text, '系统提示', {
            confirmButtonText: '确定',
            cancelButtonText: '取消',
            type: 'warning'
        }).then(() => {
            resolve(true)
        }).catch(() => {
            reject(false)
        })
    }).catch(()=>{

    })
}
```

6.4.2 挂载信息确认提示框

在main.js中将myconfirm挂载到vue上。

```
//导入封装信息确认提示框组件脚本
import myconfirm from '@/utils/myconfirm'
Vue.prototype.$myconfirm = myconfirm;
```

6.4.3 使用信息确认提示框组件

在src\layout\components\Navbar.vue页面组件中改造用户退出登录方法，代码如下所示：

```
async logout() {
    //提示是否退出系统
    let confirm = await this.$myconfirm("确定要退出系统吗?");
```

```
    if (confirm) {
        //请求参数
        let params = { token: getToken() };
        //发送退出请求
        let res = await logout(params);
        //判断是否成功
        if (res.success) {
            //清空token
            removeToken();
            clearStorage();
            //跳转到登录页面
            window.location.href = "/login";
        }
    }
}
```

7.动态路由菜单

7.1 动态获取菜单数据

通常情况下，系统菜单会随着功能模块的增加，菜单数据也会随之增加，其次，不同的用户，所拥有的菜单权限也是不同的，根据不同的用户，服务器动态的返回不同的菜单权限。

7.1.1 编写获取菜单数据方法

在api/user.js中添加getMenuList()方法，用于从服务器获取菜单数据。

```
/**
 * 获取菜单数据
 */
export async function getMenuList(){
  return await http.get("/api/sysUser/getMenuList");
}
```

7.1.2 改造permission.js

找到store/modules目录下的permission.js的generateRoutes方法，将原有的代码修改成如下代码：

```
//导入获取菜单数据的方法
import { getMenuList } from '@/api/user';

const actions = {
  generateRoutes({ commit }, roles) {
    return new Promise((resolve,reject) => {
      getMenuList().then(res=>{
        let accessedRoutes;//存放对应权限的路由信息
        //如果状态码为200，则表示成功
        if(res.code === 200){
          accessedRoutes = filterAsyncRoutes(res.data, roles)
        }
        //将路由信息保存到store中
        commit("SET_ROUTES",accessedRoutes);
        resolve(accessedRoutes);
      }).catch(error=>{
```

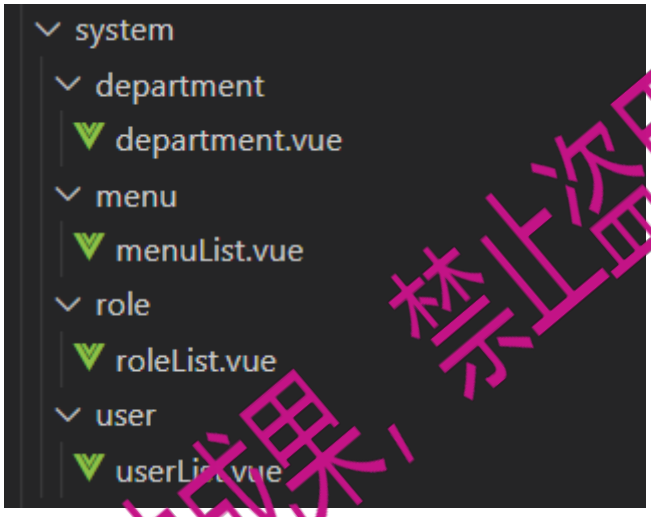
```
          reject(error);
        });
      })
    }
  }
```

7.2 生成动态路由

利用 vue-router 的 addRoutes 方法可以动态添加路由， router.addRoutes(routes: Array)，动态添加路由，参数必是一个符合 routes 选项要求的数组。

7.2.1 添加功能页面

在src/views目录下创建系统管理模块的文件夹及页面，如下图所示：



7.2.2 改造permission.js

修改store/modules下的permission.js中的 filterAsyncRoutes()方法如下所示：

```
import { constantRoutes } from '@/router'
//导入Layout组件
import Layout from '@/layout'
/**
 * 过滤出所有拥有权限的菜单
 * @param routes asyncRoutes
 * @param roles
 */
export function filterAsyncRoutes(routes, roles) {
  const res = []
  routes.forEach(route => {
    const tmp = { ...route }
    //判断是否有相应的权限
    if (hasPermission(roles, tmp)) {
      //获取组件
      const component = tmp.component;
      //判断该路由使用有组件
      if(route.component){
        //判断是否是根组件
        if(component === 'Layout'){
          tmp.component = Layout;
        }else{
```

```
            //获取对应的具体的组件信息
            tmp.component = (resolve) => require([`@/views${component}`],resolve)
          }
        }
        //判断是否有子菜单
        if (tmp.children) {
          tmp.children = filterAsyncRoutes(tmp.children, roles)
        }
        res.push(tmp)
      }
    })

    return res
  }
```

8. 部门管理模块

8.1 部门管理后端接口

8.1.1 Department

在原有的基础上添加如下属性：

```
/**
 * 是否展开
 */
@TableField(exist = false)
private Boolean open;

/**
 * 子部门
 */
@TableField(exist = false)
private List<Department> children = new ArrayList<Department>();
```


8.1.2 DepartmentQueryVo

在com.manong.vo.query包下创建DepartmentQueryVo查询条件类。

```
package com.manong.vo.query;

import com.manong.entity.Department;
import lombok.Data;

/**
 * 查询条件类
 */
@Data
public class DepartmentQueryVo extends Department {
}
```

8.1.3 DepartmentTree

在com.manong.utils包下创建DepartmentTree部门树工具类。

```
package com.manong.utils;

import com.manong.entity.Department;
import org.springframework.beans.BeanUtils;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class DepartmentTree {
    /**
     * 生成部门树
     *
     * @param deptList
     * @param pid
     * @return
     */
    public static List<Department> makeDepartmentTree(List<Department> deptList, Long pid) {
        //创建集合保存部门信息
        List<Department> list = new ArrayList<Department>();
        //如果deptList部门列表不为空，则使用部门列表，否则创建集合对象
        Optional.ofNullable(deptList).orElse(new ArrayList<Department>())
                .stream().filter(item -> item != null && item.getPid() == pid)
                .forEach(item -> {
                    //创建部门对象
                    Department dept = new Department();
                    //复制属性
                    BeanUtils.copyProperties(item, dept);
                    //获取每一个item的下级部门,递归生成部门树
                    List<Department> children = makeDepartmentTree(deptList, item.getId());
                    //设置子部门
                    dept.setChildren(children);
                    //将部门对象添加到集合
                    list.add(dept);
                });
        return list;

    }
}
```

8.1.4 DepartmentService

```
public interface DepartmentService extends IService<Department> {
    /**
     * 查询部门列表
     * @param departmentQueryVo
     * @return
     */
    List<Department> findDepartmentList(DepartmentQueryVo departmentQueryVo);

    /**
     * 查询上级部门列表
```

```
     * @return
     */
    List<Department> findParentDepartment();

    /**
     * 判断部门下是否有子部门
     * @param id
     * @return
     */
    boolean hasChildrenOfDepartment(Long id);

    /**
     * 判断部门下是否存在用户
     * @param id
     * @return
     */
    boolean hasUserOfDepartment(Long id);
}
```

8.1.5 DepartmentServiceImpl

```
@Service
@Transactional
public class DepartmentServiceImpl extends ServiceImpl<DepartmentMapper,
Department> implements DepartmentService {

    @Resource
    private UserMapper userMapper;

    /**
     * 查询部门列表
     *
     * @param departmentQueryVo
     * @return
     */
    @Override
    public List<Department> findDepartmentList(DepartmentQueryVo
departmentQueryVo) {
        //创建条件构造器对象
        QueryWrapper<Department> queryWrapper = new QueryWrapper<Department>();
        //部门名称

 queryWrapper.like(!ObjectUtils.isEmpty(departmentQueryVo.getDepartmentName()),"d
epartment_name",departmentQueryVo.getDepartmentName());
        //排序
        queryWrapper.orderByAsc("order_num");
        //查询部门列表
        List<Department> departmentList = baseMapper.selectList(queryWrapper);
        //生成部门树
        List<Department> departmentTree =
DepartmentTree.makeDepartmentTree(departmentList, 0L);
        return departmentTree;
    }

    /**
     * 查询上级部门列表
     *
```

```
     * @return
     */
    @Override
    public List<Department> findParentDepartment() {
        //创建条件构造器对象
        QueryWrapper<Department> queryWrapper = new QueryWrapper<Department>();
        //排序
        queryWrapper.orderByAsc("order_num");
        //查询部门列表
        List<Department> departmentList = baseMapper.selectList(queryWrapper);
        //创建部门对象
        Department department = new Department();
        department.setId(0L);
        department.setDepartmentName("顶级部门");
        department.setPid(-1L);
        departmentList.add(department);
        //生成部门树列表
        List<Department> departmentTree =
DepartmentTree.makeDepartmentTree(departmentList, -1L);
        //返回部门列表
        return departmentTree;
    }

    /**
     * 判断部门下是否有子部门
     *
     * @param id
     * @return
     */
    @Override
    public boolean hasChildrenOfDepartment(Long id) {
        //创建条件构造器对象
        QueryWrapper<Department> queryWrapper = new QueryWrapper<Department>();
        queryWrapper.eq("pid",id);
        //如果数量大于0，表示存在
        if(baseMapper.selectCount(queryWrapper)>0){
            return true;
        }
        return false;
    }

    /**
     * 判断部门下是否存在用户
     *
     * @param id
     * @return
     */
    @Override
    public boolean hasUserOfDepartment(Long id) {
        //创建条件构造器对象
        QueryWrapper<User> queryWrapper = new QueryWrapper<User>();
        queryWrapper.eq("department_id",id);
        //如果数量大于0，表示存在
        if(userMapper.selectCount(queryWrapper)>0){
            return true;
        }
        return false;
    }
```

```
89  }
```

8.1.6 DepartmentController

在DepartmentController控制器接口中完成部门的增删改查方法。

```
1   package com.manong.controller;
2
3
4   import com.manong.entity.Department;
5   import com.manong.service.DepartmentService;
6   import com.manong.utils.Result;
7   import com.manong.vo.query.DepartmentQueryVo;
8   import org.springframework.web.bind.annotation.*;
9
10  import javax.annotation.Resource;
11  import java.util.List;
12
13  @RestController
14  @RequestMapping("/api/department")
15  public class DepartmentController {
16
17      @Resource
18      private DepartmentService departmentService;
19
20      /**
21       * 查询部门列表
22       * @param departmentQueryVo
23       * @return
24       */
25      @GetMapping("/list")
26      public Result list(DepartmentQueryVo departmentQueryVo){
27          //调用查询部门列表方法
28          List<Department> departmentList =
    departmentService.findDepartmentList(departmentQueryVo);
29          //返回数据
30          return Result.ok(departmentList);
31      }
32
33      /**
34       * 查询上级部门列表
35       * @return
36       */
37      @GetMapping("/parent/list")
38      public Result getParentDepartment(){
39          //调用查询上级部门列表方法
40          List<Department> departmentList =
    departmentService.findParentDepartment();
41          //返回数据
42          return Result.ok(departmentList);
43      }
44
45      /**
46       * 添加部门
47       * @param department
48       * @return
49       */
50      @PostMapping("/add")
```

```
51      public Result add(@RequestBody Department department){
52          if(departmentService.save(department)){
53              return Result.ok().message("部门添加成功");
54          }
55          return Result.error().message("部门添加失败");
56      }
57
58      /**
59       * 修改部门
60       * @param department
61       * @return
62       */
63      @PutMapping("/update")
64      public Result update(@RequestBody Department department){
65          if(departmentService.updateById(department)){
66              return Result.ok().message("部门修改成功");
67          }
68          return Result.error().message("部门修改失败");
69      }
70
71      /**
72       * 查询某个部门下是否存在子部门
73       * @param id
74       * @return
75       */
76      @GetMapping("/check/{id}")
77      public Result check(@PathVariable Long id){
78          //调用查询部门下是否存在子部门的方法
79          if(departmentService.hasChildrenOfDepartment(id)){
80              return Result.exist().message("该部门下存在子部门，无法删除");
81          }
82          //调用查询部门下是否存在用户的方法
83          if(departmentService.hasUserOfDepartment(id)){
84              return Result.exist().message("该部门下存在用户，无法删除");
85          }
86          return Result.ok();
87      }
88
89      /**
90       * 删除部门
91       * @param id
92       * @return
93       */
94      @DeleteMapping("/delete/{id}")
95      public Result delete(@PathVariable Long id){
96          if(departmentService.removeById(id)){
97              return Result.ok().message("部门删除成功");
98          }
99          return Result.error().message("部门删除失败");
100     }
101 }
```

8.1.7 Result

注意：修改exist()方法。

```
/**
 * 是否存在
 * @return
 */
public static<T> Result<T> exist(){
    Result<T> result = new Result<T>();
    result.setSuccess(false);//存在该数据
    //由于vue-element-admin模板在响应时验证状态码是否是200，如果不是200，则报错
    result.setCode(ResultCode.SUCCESS);//执行成功，但存在该数据
    result.setMessage("该数据存在");
    return result;
}
```

8.1.8 逻辑删除配置

在application.properties添加Mybatis Plus的逻辑删除配置。

```
#全局逻辑删除的实体字段名
mybatis-plus.global-config.db-config.logic-delete-field=isDelete
#逻辑删除值，默认为1
mybatis-plus.global-config.db-config.logic-delete-value=1
#逻辑未删除值，默认为0
mybatis-plus.global-config.db-config.logic-not-delete-value=0
```

8.2 查询部门列表

8.2.1 效果图

8.2.2 页面原型代码

在src/views/system/department目录下创建department.vue组件。

```
<template>
  <el-main>
    <!-- 查询条件 -->
    <el-form
      ref="searchForm"
      label-width="80px"
      :inline="true"
      size="small"
    >
      <el-form-item>
```

```
          <el-input placeholder="请输入部门名称"></el-input>
        </el-form-item>
        <el-form-item>
          <el-button type="primary" icon="el-icon-search">查询</el-button>
          <el-button icon="el-icon-refresh-right">重置</el-button>
          <el-button type="success" icon="el-icon-plus">新增</el-button>
        </el-form-item>
      </el-form>
      <!-- 数据表格 -->
      <el-table
        :data="tableData"
        border
        stripe
        style="width: 100%; margin-bottom: 20px"
        row-key="id"
        default-expand-all
        :tree-props="{ children: 'children' }"
      >
        <el-table-column prop="name" label="部门名称"></el-table-column>
        <el-table-column prop="parentName" label="所属部门"></el-table-column>
        <el-table-column prop="deptCode" label="部门编码"></el-table-column>
        <el-table-column prop="deptPhone" label="部门电话"></el-table-column>
        <el-table-column label="操作" width="200" align="center">
          <template slot-scope="scope">
            <el-button
              icon="el-icon-edit-outline"
              type="primary"
              size="small"
              @click="handleEdit(scope.row)"
            >编辑
            </el-button
            >
            <el-button
              icon="el-icon-close"
              type="danger"
              size="small"
              @click="handleDelete(scope.row)"
            >删除
            </el-button
            >
          </template>
        </el-table-column>
      </el-table>
    </el-main>
  </template>

  <script>
  export default {
    name: 'department',
    data() {
      return {
        tableData: []//表格数据列表
      }
    },
    methods: {
      /**
       * 编辑部门
       * @param row
```

```
      */
      handleEdit(row) {

      },
      /**
       * 删除部门
       * @param row
       */
      handleDelete(row) {

      },
    },
    created() {

    }
  }
  </script>

  <style lang="scss" scoped></style>
```

8.2.3 编写前端api脚本

在src/api目录下创建department.js脚本文件。

```
import http from '@/utils/request';

export default {
  /**
   * 查询部门列表
   * @param param
   * @returns
   */
  async getDepartmentList(param) {
    return await http.get("/api/department/list", param);
  }
}
```

8.2.4 编写页面组件

（1）添加组件脚本代码

```
//导入api脚本
import departmentApi from "@/api/department";

export default {
  name: "department",
  data() {
    return {
      searchModel: {
        departmentName: "", //部门名称
      },
      tableData: [], //表格数据列表
    };
  },
  methods: {
    /**
     * 查询部门列表
     */
```

```
    async search() {
      //发送查询请求
      let res = await departmentApi.getDepartmentList(this.searchModel);
      //判断是否存在数据
      if (res.success) {
        //获取数据
        this.tableData = res.data;
      }
    },
    /**
     * 重置
     */
    resetValue(){
        this.searchModel = {};//清空数据
        this.search();//重新调用方法
    },
  },
  //初始化时执行
  created() {
    //调用查询部门列表方法
    this.search();
  },
};
```

(2) 编写页面组件代码

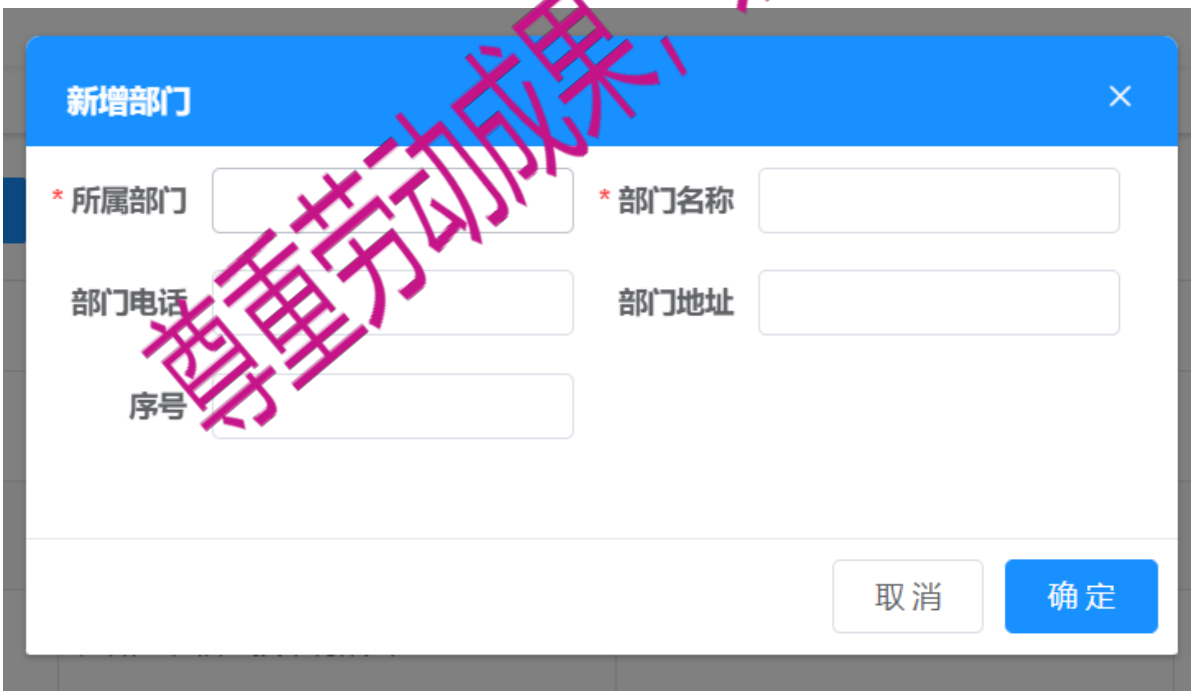
注意：表单属性设置及表格属性设置

```
<template>
  <el-main>
    <!-- 查询条件 -->
    <el-form :model="searchModel" ref="searchForm" label-width="80px" :inline="true" size="small">
      <el-form-item>
        <el-input placeholder="请输入部门名称" v-model="searchModel.departmentName"></el-input>
      </el-form-item>
      <el-form-item>
        <el-button type="primary" icon="el-icon-search" @click="search()">查询</el-button>
        <el-button icon="el-icon-refresh-right" @click="resetValue()">重置</el-button>
        <el-button type="success" icon="el-icon-plus">新增</el-button>
      </el-form-item>
    </el-form>
    <!-- 数据表格 -->
    <el-table
      :data="tableData"
      border
      stripe
      style="width: 100%; margin-bottom: 20px"
      row-key="id"
      default-expand-all
      :tree-props="{ children: 'children' }"
    >
      <el-table-column prop="departmentName" label="部门名称"></el-table-column>
      <el-table-column prop="parentName" label="所属部门"></el-table-column>
      <el-table-column prop="phone" label="部门电话"></el-table-column>
```

```
        <el-table-column prop="address" label="部门位置"></el-table-column>
        <el-table-column label="操作" width="200" align="center">
          <template slot-scope="scope">
            <el-button
              icon="el-icon-edit-outline"
              type="primary"
              size="small"
              @click="handleEdit(scope.row)"
              >编辑
            </el-button>
            <el-button
              icon="el-icon-close"
              type="danger"
              size="small"
              @click="handleDelete(scope.row)"
              >删除
            </el-button>
          </template>
        </el-table-column>
      </el-table>
    </el-main>
  </template>
```

8.3 新增部门

8.3.1 效果图



8.3.2 封装通用窗口组件

在components下创建system目录，并在system目录下创建SystemDialog.vue组件，代码如下所示：

```
<template>
  <div>
    <el-dialog
      top="5vh"
      :title="title"
      :visible.sync="visible"
      :width="width + 'px'"
```

```
        :before-close="onClose"
        :close-on-click-modal="false"
      >
        <div class="container" :style="{height:height+'px'}">
          <slot name="content"></slot>
        </div>
        <span slot="footer" class="dialog-footer">
          <el-button @click="onClose">取 消</el-button>
          <el-button type="primary" @click="onConfirm">确 定</el-button>
        </span>
      </el-dialog>
    </div>
  </template>

  <script>
  export default {
    props: {
      title: {
        type: String,
        default: "标题",
      },
      visible: {
        type: Boolean,
        default: false,
      },
      width: {
        type: Number,
        default: 600,
      },
      height: {
        type: Number,
        default: 250,
      },
    },
    data() {
      return {};
    },
    methods: {
      onClose() {
        this.$emit("onClose");
      },
      onConfirm() {
        this.$emit("onConfirm");
      },
    },
  };
  </script>

  <style lang="scss" scoped>
  .container{
    overflow-x: initial;
    overflow-y: auto;
  }
  .el-dialog__wrapper {
    ::v-deep .el-dialog {
      border-top-left-radius: 7px !important;
      border-top-right-radius: 7px !important;
      .el-dialog__header {
```

```
        border-top-left-radius: 7px !important;
        border-top-right-radius: 7px !important;
        background-color: #1890ff;
        .el-dialog__title {
          color: #fff;
          font-size: 15px;
          font-weight: 700;
        }
        .el-dialog__close {
          color: #fff;
        }
      }
      .el-dialog__body {
        padding: 10px 10px !important;
      }
      .el-dialog__footer {
        border-top: 1px solid #e8eaec !important;
        padding: 10px !important;
      }
    }
  }
  </style>
```

8.3.2 编写新增部门弹窗代码

(1) 在src/views/system/department/department.vue组件中导入SystemDialog.vue组件。

```
//导入对话框组件
import SystemDialog from '@/components/system/SystemDialog.vue';
```

(2) 在department.vue组件中注册SystemDialog.vue组件。

```
export default {
  name: "department"
  //注册组件
  components:{
      SystemDialog
  },
}
```

(3) 编写新增部门窗口代码

注意：需要添加窗口属性及事件，表单属性等

```
<!-- 添加和编辑部门窗口 -->
<system-dialog
  :title="deptDialog.title"
  :visible="deptDialog.visible"
  :width="deptDialog.width"
  :height="deptDialog.height"
  @onClose="onClose"
  @onConfirm="onConfirm"
>
  <div slot="content">
    <el-form
      :model="dept"
      ref="deptForm"
      :rules="rules"
```

```
        label-width="80px"
        :inline="true"
        size="small"
      >
        <el-form-item label="所属部门" prop="parentName">
          <el-input
            v-model="dept.parentName"
            @click.native="selectDepartment()"
            :readonly="true"
          ></el-input>
        </el-form-item>
        <el-form-item label="部门名称" prop="departmentName">
          <el-input v-model="dept.departmentName"></el-input>
        </el-form-item>
        <el-form-item label="部门电话">
          <el-input v-model="dept.phone"></el-input>
        </el-form-item>
        <el-form-item label="部门地址">
          <el-input v-model="dept.address"></el-input>
        </el-form-item>
        <el-form-item label="序号">
          <el-input v-model="dept.orderNum"></el-input>
        </el-form-item>
      </el-form>
    </div>
  </system-dialog>
```

(4) 编写新增部门窗口脚本代码

注意：以下是新增部门窗口核心代码，原有的属性和方法已省略。

```
export default {
  name: "department",
  components:{
      SystemDialog
  },
  data() {
    return {
      //部门窗口属性
      deptDialog: {
        title: "新增部门",
        visible: false, //是否显示
        width: 560,
        height: 170,
      },
      //部门对象
      dept: {
        id: "",
        pid: "",
        parentName: "",
        departmentName: "",
        address: "",
        phone: "",
        orderNum: "",
      },
      //表单验证规则
      rules: {
```

```
          parentName: [{ required: true, trigger: "change", message: "请选择所属部门" }],
          departmentName: [{ required: true, trigger: "blur", message: "请输入部门名称" }],
        },
      };
    },
    methods: {
      /**
       * 弹窗取消事件
       */
      onClose() {
        //关闭窗口
        this.deptDialog.visible = false;
      },
      /**
       * 弹窗确认事件
       */
      onConfirm() {
        //表单验证
        this.$refs.deptForm.validate((valid) => {
          //如果验证通过
          if (valid) {
            //关闭窗口
            this.deptDialog.visible = false;
          }
        });
      },
      /**
       * 打开添加部门窗口
       */
      openAddWindow() {
        //设置窗口标题
        this.deptDialog.title = "新增部门";
        //显示添加部门窗口
        this.deptDialog.visible = true;
      },
      /**
       * 选择所属部门
       */
      selectDepartment() {},
    },
  };
```

8.3.4 编写选择所属部门代码

（1）在src/api/department.js脚本文件编写获取所属部门列表的api脚本代码

```
/**
 * 获取所属部门列表
 */
async getParentTreeList() {
  return await http.get("/api/department/parent/list");
}
```

（2）编写选择所属部门窗口代码

注意：需要添加选择所属窗口属性及事件。

```vue
<!-- 选择所属部门窗口 -->
<system-dialog
  :title="parentDialog.title"
  :visible="parentDialog.visible"
  :width="parentDialog.width"
  :height="parentDialog.height"
  @onClose="parentOnClose"
  @onConfirm="parentOnConfirm"
>
  <div slot="content">
    <el-tree
      ref="parentTree"
      :data="treeList"
      node-key="id"
      :props="defaultProps"
      :default-expand-all="true"
      :highlight-current="true"
      :expand-on-click-node="false"
      @node-click="handleNodeClick"
    >
      <div class="customer-tree-node" slot-scope="{ node, data }">
        <span v-if="data.children.length === 0">
          <i class="el-icon-document"></i>
        </span>
        <span v-else @click="openBtn(data)">
          <svg-icon v-if="data.open" icon-class="add-s" />
          <svg-icon v-else icon-class="sub-s" />
        </span>
        <span style="margin-left: 5px;">{{ node.label }}</span>
      </div>
    </el-tree>
  </div>
</system-dialog>
```

(3) 编写选择所属部门窗口脚本代码

注意：以下是选择所属部门窗口核心代码，原有的属性和方法已省略。

```vue
export default {
  name: "department",
  components: {
    SystemDialog,
  },
  data() {
    return {
      //选择所属部门属性
      parentDialog: {
        title: "选择所属部门",
        visible: false, //是否显示
        width: 300,
        height: 400,
      },
      //树形菜单列表
      treeList: [],
      defaultProps: {
        children: "children",
        label: "departmentName",
      },
```

```
    };
  },
  methods: {
    /**
     * 选择所属部门
     */
    async selectDepartment() {
      //显示窗口
      this.parentDialog.visible = true;
      //设置窗口标题
      this.parentDialog.title = "选择所属部门";
      //查询部门列表
      let res = await departmentApi.getParentTreeList();
      //判断是否成功
      if (res.success) {
        //赋值
        this.treeList = res.data;
      }
    },
    /**
     * 选择所属部门取消事件
     */
    parentOnClose() {
      this.parentDialog.visible = false;
    },
    /**
     * 选择所属部门确认事件
     */
    parentOnConfirm() {
      this.parentDialog.visible = false;
    },
    /**
     * 树节点点击事件
     */
    handleNodeClick(data) {
      //当点击树的值时，赋予选中的值
      this.dept.pid = data.id;
      this.dept.parentName = data.departmentName;
    },
    /**
     * 点击树节点+-号折叠展开事件
     */
    openBtn(data) {
      //修改折叠展开状态
      data.open = !data.open;
      this.$refs.parentTree.store.nodesMap[data.id].expanded = !data.open;
    },
  },
};
```


（4）选择所属部门树菜单样式代码

```
<style lang="scss" scoped>
::v-deep .el-tree {
  .el-tree-node {
    position: relative;
    padding-left: 10px;
  }
```

```
    .el-tree-node__children {
      padding-left: 20px;
    }
    .el-tree-node :last-child:before {
      height: 40px;
    }
    .el-tree > .el-tree-node:before {
      border-left: none;
    }
    .el-tree > .el-tree-node:after {
      border-top: none;
    }
    .el-tree-node:before,
    .el-tree-node:after {
      content: "";
      left: -4px;
      position: absolute;
      right: auto;
      border-width: 1px;
    }
    .tree :first-child .el-tree-node:before {
      border-left: none;
    }
    .el-tree-node:before {
      border-left: 1px dotted #d9d9d9;
      bottom: 0px;
      height: 100%;
      top: -25px;
      width: 1px;
    }
    .el-tree-node:after {
      border-top: 1px dotted #d9d9d9;
      height: 20px;
      top: 14px;
      width: 24px;
    }
    .el-tree-node__expand-icon.is-leaf {
      width: 0px;
    }
    .el-tree-node__content > .el-tree-node__expand-icon {
      display: none;
    }
    .el-tree-node__content {
      line-height: 30px;
      height: 30px;
      padding-left: 10px !important;
    }
  }
  ::v-deep .el-tree > div {
    &::before {
      display: none;
    }
    &::after {
      display: none;
    }
  }
  </style>
```

（5）调用选择所属部门的方法

```
<el-form-item label="所属部门" prop="parentName">
    <el-input v-model="dept.parentName" :readonly="true"
              @click.native="selectDepartment()"></el-input>
</el-form-item>
```

8.3.5 表单数据快速清空

（1）封装清空表单数据方法

在src/utils目录下创建resetForm.js脚本文件，代码如下所示：

```
//重置表单和表单数据
export default function resetForm(fromName,obj){
    //清空表单
    if(this.$refs[fromName]){
        this.$refs[fromName].resetFields();
    }
    //清空数据域
    Object.keys(obj).forEach(key =>{
        obj[key] = '';
    })
}
```

（2）全局引入resetForm.js脚本文件

在src/main.js脚本文件中引入resetForm.js文件。

```
//导入清空表单工具
import resetForm from '@/utils/resetForm'
Vue.prototype.$resetForm = resetForm;
```

（3）打开新增部门窗口时清空表单数据

使用方式：this.$resetForm("表单ref属性值",数据对象);

```
/**
 * 打开添加部门窗口
 */
openAddWindow() {
  //清空表单数据
  this.$resetForm("deptForm", this.dept);
  //设置窗口标题
  this.deptDialog.title = "新增部门";
  //显示添加部门窗口
  this.deptDialog.visible = true;
},
```

8.3.6 新增部门

（1）编写前端api脚本

在src/api/department.js文件中加入新增部门的方法，代码如下：

```
/**
 * 添加部门
 * @returns
 */
async addDept(params) {
  return await http.post("/api/department/add", params);
},
```

(2) 编写department.vue的窗口确认事件方法

```
/**
 * 弹窗确认事件
 */
onConfirm() {
  //表单验证
  this.$refs.deptForm.validate( async (valid) => {
    //如果验证通过
    if (valid) {
      let res = null;//后端返回的数据
      //判断部门ID是否有数据，如果部门ID为空，则表示新增，否则就是修改
      if(this.dept.id===''){//新增
        //发送新增请求
        res =  await departmentApi.addDept(this.dept);
      }else{
        //发送修改请求

      }
      //判断是否成功
      if (res.success) {
        this.$message.success(res.message);
        //刷新
        this.search();
        //关闭窗口
        this.deptDialog.visible = false;
      } else {
        this.$message.error(res.message);
      }
    }
  });
},
```


8.4 编辑部门

8.4.1 效果图

8.4.2 封装数据回显脚本

（1）在src/utils目录下创建objCopy.js脚本文件。

```
//快速把obj1里面对应的数据复制到obj2
export default async function objCopy(obj1,obj2){
    Object.keys(obj2).forEach(key =>{
        obj2[key] = obj1[key]
    })
}
```

（2）全局引入objCopy.js脚本文件

在src/main.js脚本文件中引入objCopy.js文件。

```
//导入快速复制对象工具
import objCopy from '@/utils/objCopy'
Vue.prototype.$objCopy = objCopy;
```

8.4.3 编辑按钮代码

```
<el-table-column label="操作" width="200" align="center">
  <template slot-scope="scope">
    <el-button
      icon="el-icon-edit-outline"
      type="primary"
      size="small"
      @click="handleEdit(scope.row)"
      >编辑
    </el-button>
    <el-button
      icon="el-icon-close"
      type="danger"
      size="small"
      @click="handleDelete(scope.row)"
      >删除
    </el-button>
  </template>
</el-table-column>
```

8.4.4 编辑按钮点击事件

```
/**
 * 编辑部门
 * @param row
 */
handleEdit(row) {
  //数据回显
  this.$objCopy(row, this.dept);
  //设置窗口标题
  this.deptDialog.title = "编辑部门";
  //显示编辑部门窗口
  this.deptDialog.visible = true;
},
```

8.4.5 编写前端api脚本

在src/api/department.js文件中加入编辑部门的方法，代码如下：

```
/**
 * 修改部门
 * @returns
 */
async updateDept(params) {
  return await http.put("/api/department/update", params);
},
```

8.4.6 确认按钮点击事件

```
/**
 * 弹窗确认事件
 */
onConfirm() {
  this.$refs.deptForm.validate(async (valid) => {
    if (valid) {
      let res = null; //后端返回的数据
      //判断部门ID是否有数据，如果部门ID为空，则表示新增，否则就是修改
      if(this.dept.id===''){//新增
        //发送新增请求
        res =  await departmentApi.addDept(this.dept);
      }else{
        //发送修改请求
        res = await departmentApi.updateDept(this.dept);
      }
      //判断是否成功
      if (res.success) {
        this.$message.success(res.message);
        //刷新
        this.search();
        //关闭窗口
        this.deptDialog.visible = false;
      } else {
        this.$message.error(res.message);
      }
    }
  });
},
```

8.5 删除部门

8.5.1 需求分析

删除某个部门前，判断该部门是否存在子部门，如果存在子部门，则提示不能删除；判断该部门下是否存在用户，如果存在则不能删除。

8.5.2 编写前端api脚本

在src/api/department.js文件中加入 `检查部门下是否存在子部门` 和 `删除部门` 的方法，代码如下：

```
/**
 * 检查部门下是否存在子部门
 */
async checkDepartment(params){
  return await http.getRestApi("/api/department/check",params);
},
/**
 * 删除部门
 * @returns
 */
async deleteById(params) {
  return await http.delete("/api/department/delete", params);
}
```

8.5.3 删除按钮代码

```
<el-table-column label="操作" width="220" align="center">
  <template slot-scope="scope">
    <el-button
      icon="el-icon-edit-outline"
      type="primary"
      size="small"
      @click="handleEdit(scope.row)"
      >编辑
    </el-button>
    <el-button
      icon="el-icon-close"
      type="danger"
      size="small"
      @click="handleDelete(scope.row)"
      >删除
    </el-button>
  </template>
</el-table-column>
```

8.5.4 删除按钮点击事件

```
/**
 * 删除部门
*/
async handleDelete(row) {
    //查询部门下是否存在子部门或用户
    let result = await departmentApi.checkDepartment({ id: row.id });
    //判断是否可以删除
```

```
        if (!result.success) {
            //提示不能删除
            this.$message.warning(result.message);
        } else {
            //确认是否删除
            let confirm =await this.$myconfirm("确定要删除该数据吗?");
            if (confirm) {
                //发送删除请求
                let res = await departmentApi.deleteById({ id: row.id });
                //判断是否成功
                if (res.success) {
                    //成功提示
                    this.$message.success(res.message);
                    //刷新
                    this.search();
                } else {
                    //失败提示
                    this.$message.error(res.message);
                }
            }
        }
    }
```

9. 权限菜单管理

9.1 权限菜单管理后端接口

9.1.1 PermissionQueryVo

在com.manong.vo.query包下创建PermissionQueryVo查询条件类。

```
package com.manong.vo.query;

import com.manong.entity.Permission;
import lombok.Data;

@Data
public class PermissionQueryVo extends Permission {
}
```

9.1.2 PermissionService

```
/**
 * 查询菜单列表
 * @param permissionQueryVo
 * @return
 */
List<Permission> findPermissionList(PermissionQueryVo permissionQueryVo);

/**
 * 查询上级菜单列表
 * @return
```

```
     */
    List<Permission> findParentPermissionList();

    /**
     * 检查菜单是否有子菜单
     * @param id
     * @return
     */
    boolean hasChildrenOfPermission(Long id);
```

9.1.3 PermissionServiceImpl

```
    /**
     * 查询菜单列表
     *
     * @return
     */
    @Override
    public List<Permission> findPermissionList(PermissionQueryVo permissionQueryVo) {
        //创建条件构造器对象
        QueryWrapper<Permission> queryWrapper = new QueryWrapper<Permission>();
        //排序
        queryWrapper.orderByAsc("order_num");
        //调用查询菜单列表的方法
        List<Permission> permissionList = baseMapper.selectList(queryWrapper);
        //生成菜单树
        List<Permission> menuTree = MenuTree.makeMenuTree(permissionList, 0L);
        //返回数据
        return menuTree;
    }

    /**
     * 查询上级菜单列表
     *
     * @return
     */
    @Override
    public List<Permission> findParentPermissionList() {
        QueryWrapper<Permission> queryWrapper = new QueryWrapper<Permission>();
        //只查询type为目录和菜单的数据(type=0或type=1)
        queryWrapper.in("type", Arrays.asList(0,1));
        //排序
        queryWrapper.orderByAsc("order_num");
        //查询菜单数据
        List<Permission> permissionList = baseMapper.selectList(queryWrapper);
        //构造顶级菜单信息，如果数据库中的菜单表没有数据，选择上级菜单时则显示顶级菜单
        Permission permission = new Permission();
        permission.setId(0L);
        permission.setParentId(-1L);
        permission.setLabel("顶级菜单");
        permissionList.add(permission);//将顶级菜单添加到集合
        //生成菜单数据
        List<Permission> menuTree = MenuTree.makeMenuTree(permissionList, -1L);
        //返回数据
        return menuTree;
    }

```

```
/**
 * 检查菜单是否有子菜单
 *
 * @param id
 * @return
*/
@Override
public boolean hasChildrenOfPermission(Long id) {
    //创建条件构造器对象
    QueryWrapper<Permission> queryWrapper = new QueryWrapper<Permission>();
    //查询父级ID
    queryWrapper.eq("parent_id", id);
    //判断数量是否大于0，如果大于0则表示存在
    if(baseMapper.selectCount(queryWrapper)>0){
        return true;
    }
    return false;
}
```

9.1.4 PermissionController

```
package com.manong.controller;


import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.manong.entity.Permission;
import com.manong.service.PermissionService;
import com.manong.utils.Result;
import com.manong.vo.query.PermissionQueryVo;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;
import java.util.List;

@RestController
@RequestMapping("/api/permission")
public class PermissionController {
    @Resource
    private PermissionService permissionService;

    /**
     * 查询菜单列表
     * @return
     */
    @GetMapping("/list")
    public Result getMenuList(PermissionQueryVo permissionQueryVo){
        //查询菜单列表
        List<Permission> permissionList =
            permissionService.findPermissionList(permissionQueryVo);
        //返回数据
        return Result.ok(permissionList);
    }

    /**
     * 查询上级菜单列表
     * @return
     */
```

```
37      @GetMapping("/parent/list")
38      public Result getParentList(){
39          //查询上级菜单列表
40          List<Permission> permissionList =
    permissionService.findParentPermissionList();
41          //返回数据
42          return Result.ok(permissionList);
43      }
44
45      /**
46       * 根据id查询菜单信息
47       * @param id
48       * @return
49       */
50      @GetMapping("/{id}")
51      public Result getMenuById(@PathVariable Long id){
52          return Result.ok(permissionService.getById(id));
53      }
54
55      /**
56       * 添加菜单
57       * @param permission
58       * @return
59       */
60      @PostMapping("/add")
61      public Result add(@RequestBody Permission permission){
62          if(permissionService.save(permission)){
63              return Result.ok().message("菜单添加成功");
64          }
65          return Result.error().message("菜单添加失败");
66      }
67
68      /**
69       * 修改菜单
70       * @param permission
71       * @return
72       */
73      @PutMapping("/update")
74      public Result update(@RequestBody Permission permission){
75          if(permissionService.updateById(permission)){
76              return Result.ok().message("菜单修改成功");
77          }
78          return Result.error().message("菜单修改失败");
79      }
80
81      /**
82       * 删除菜单
83       * @param id
84       * @return
85       */
86      @DeleteMapping("/delete/{id}")
87      public Result delete(@PathVariable Long id){
88          if(permissionService.removeById(id)){
89              return Result.ok().message("菜单删除成功");
90          }
91          return Result.error().message("菜单删除失败");
92      }
93
```

```
94        /**
95         * 检查菜单下是否有子菜单
96         * @param id
97         * @return
98         */
99        @GetMapping("/check/{id}")
100       public Result check(@PathVariable Long id){
101           //判断菜单是否有子菜单
102           if(permissionService.hasChildrenOfPermission(id)){
103               return Result.exist().message("该菜单下有子菜单，无法删除");
104           }
105           return Result.ok();
106       }
107   }
```

9.2 查询权限菜单列表

9.2.1 效果图

9.2.2 页面原型代码

在src/views/system/role目录下创建role.vue组件。

```
1   <template>
2     <el-main>
3       <!-- 新增按钮 -->
4       <el-button type="success" icon="el-icon-plus" @click="openAddWindow">新增</el-
    button>
5       <!-- 数据表格 -->
6       <el-table
7         style="margin-top: 10px"
8         :height="tableHeight"
9         :data="menuList"
10        row-key="id"
11        default-expand-all
12        :tree-props="{ children: 'children' }"
13        :border="true"
14        stripe
15      >
16        <el-table-column prop="label" label="菜单名称"></el-table-column>
17        <el-table-column prop="type" label="菜单类型" align="center">
18          <template slot-scope="scope">
```

```
          <el-tag v-if="scope.row.type == '0'" size="normal">目录</el-tag>
          <el-tag type="success" v-else-if="scope.row.type == '1'" size="normal">菜单</el-tag>
          <el-tag type="warning" v-else-if="scope.row.type == '2'" size="normal">按钮</el-tag>
        </template>
      </el-table-column>
      <el-table-column prop="icon" label="图标" align="center">
        <template slot-scope="scope">
          <i :class="scope.row.icon" v-if="scope.row.icon.includes('el-icon')" ></i>
          <svg-icon v-else :icon-class="scope.row.icon"></svg-icon>
        </template>
      </el-table-column>
      <el-table-column prop="name" label="路由名称"></el-table-column>
      <el-table-column prop="path" label="路由地址"></el-table-column>
      <el-table-column prop="url" label="组件路径"></el-table-column>
      <el-table-column align="center">
        <template slot-scope="scope">
          <el-button
            type="primary"
            icon="el-icon-edit"
            size="small"
            @click="editMenu(scope.row)"
          >编辑
          </el-button
          >
          <el-button
            type="danger"
            size="small"
            icon="el-icon-delete"
            @click="deleteMenu(scope.row)"
          >删除
          </el-button
          >
        </template>
      </el-table-column>
    </el-table>
  </el-main>
</template>
```


9.2.3 编写前端api脚本

在src/api目录下创建menu.js脚本文件。

```
import http from '@/utils/request';

export default {
  /**
   * 查询权限菜单列表
   * @param params
   */
  async getMenuList(params){
    return await http.get("/api/permission/list",params);
  }
}
```

9.2.4 编写页面组件代码

```
//导入menu.js脚本代码
import menuApi from '@/api/menu';

export default {

  name: 'menuList',

  data() {
    return {
      menuList: [], //菜单列表
      tableHeight: 0 //表格高度
    }
  },
  methods: {
    /**
     * 查询菜单列表
     */
    async search() {
      //发送查询请求
      let res = await menuApi.getMenuList();
      //判断是否成功
      if(res.success){
        //赋值
        this.menuList = res.data;
      }
    },
    /**
     * 打开添加窗口
     */
    openAddWindow() {

    },
    /**
     * 编辑菜单
     * @param row
     */
    editMenu(row){

    },
    /**
     * 删除菜单
     * @param row
     */
    deleteMenu(row){

    },
  },
  //初始化时调用
  created(){
    //调用查询菜单列表的方法
    this.search();
  },
  mounted() {
    this.$nextTick(() => {
      this.tableHeight = window.innerHeight - 180
```

```
      })
    }
  }
```

9.2.5 优化滚动条

在public目录下的index.html添加如下样式代码：

```
<style>
  ::-webkit-scrollbar{
    width: 5px;
    height: 5px;
    background-color: #F5F5F5;
  }

  ::-webkit-scrollbar-track {
    box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.3);
    -webkit-box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.3);
    border-radius: 8px;
    background-color: #F5F5F5;
  }

  ::-webkit-scrollbar-thumb{
    border-radius: 8px;
    box-shadow: inset 0 0 6px rgba(0, 0, 0, .1);
    -webkit-box-shadow: inset 0 0 6px rgba(0, 0, 0, .1);
    background-color: #c8c8c8;
  }
</style>
```

9.3 新增权限菜单

9.3.1 效果图

（1）新增目录

（2）新增菜单

（3）新增按钮

9.3.2 编写新增权限菜单窗口代码

（1）在src/views/system/menu/menuList.vue组件中引入SystemDialog.vue组件。

```
1   //导入对话框组件
2   import SystemDialog from '@/components/system/SystemDialog.vue';
```

（2）在menuList.vue组件中注册SystemDialog.vue组件。

```
export default {
  name: "menuList",
  //注册组件
  components:{
      SystemDialog
  },
}
```

(3) 编写新增权限菜单窗口代码

注意：需要添加窗口属性及事件，表单属性等

```
<!-- 新增或编辑弹框 -->
<system-dialog
  :title="menuDialog.title"
  :width="menuDialog.width"
  :height="menuDialog.height"
  :visible="menuDialog.visible"
  @onClose="onClose"
  @onConfirm="onConfirm"
>
  <div slot="content">
    <el-form
      :model="menu"
      ref="menuForm"
      :rules="rules"
      label-width="80px"
      :inline="true"
      size="small"
    >
      <el-row>
        <el-col :span="24">
          <el-form-item prop="type" label="菜单类型">
            <el-radio-group v-model="menu.type">
              <el-radio :label="0">目录</el-radio>
              <el-radio :label="1">菜单</el-radio>
              <el-radio :label="2">按钮</el-radio>
            </el-radio-group>
          </el-form-item>
        </el-col>
      </el-row>
      <el-form-item prop="parentName" size="small" label="所属菜单">
        <el-input
          @click.native="selectParentMenu"
          v-model="menu.parentName"
          :readonly="true"
        ></el-input>
      </el-form-item>
      <el-form-item prop="label" size="small" label="菜单名称">
        <el-input v-model="menu.label"></el-input>
      </el-form-item>
      <el-form-item size="small" label="菜单图标">
        <el-input v-model="menu.icon"></el-input>
      </el-form-item>
      <el-form-item
        prop="name"
        v-if="menu.type == 1"
        size="small"
```

```
          label="路由名称"
        >
          <el-input v-model="menu.name"></el-input>
        </el-form-item>
        <el-form-item
          prop="path"
          v-if="menu.type != 2"
          size="small"
          label="路由地址"
        >
          <el-input v-model="menu.path"></el-input>
        </el-form-item>
        <el-form-item
          prop="url"
          v-if="menu.type == 1"
          size="small"
          label="组件路径"
        >
          <el-input v-model="menu.url"></el-input>
        </el-form-item>
        <el-form-item prop="code" size="small" label="权限字符">
          <el-input v-model="menu.code"></el-input>
        </el-form-item>
        <el-form-item size="small" label="菜单序号">
          <el-input v-model="menu.orderNum"></el-input>
        </el-form-item>
      </el-form>
    </div>
  </system-dialog>
```

(4) 编写新增权限菜单窗口脚本代码

注意：以下是新增权限窗口核心代码，原有的属性和方法已省略。

```
export default {
  data() {
    return {
      //新增或编辑对话框属性
      menuDialog: {
        title: "",
        width: 630,
        height: 270,
        visible: false,
      },
      //菜单属性
      menu: {
        id: "",
        type: "",
        parentId: "",
        parentName: "",
        label: "",
        icon: "",
        name: "",
        path: "",
        url: "",
        code: "",
        orderNum: "",
      },
```

```
        rules: {
          type: [{ required: true, trigger: "change", message: "请选择菜单类型" }],
          parentName: [{ required: true, trigger: "change", message: "请选择所属菜单"
}],
          label: [{ required: true, trigger: "blur", message: "请输入菜单名称" }],
          name: [{ required: true, trigger: "blur", message: "请输入路由名称" }],
          path: [{ required: true, trigger: "blur", message: "请输入路由路径" }],
          url: [{ required: true, trigger: "blur", message: "请输入组件路径" }],
          code: [{ required: true, trigger: "blur", message: "请输入权限字段" }],
        },
      };
    },
    methods: {
      /**
       * 打开添加窗口
       */
      openAddWindow() {
        this.$resetForm("menuForm", this.menu); //清空表单数据
        this.menuDialog.title = "新增菜单"; //设置窗口标题
        this.menuDialog.visible = true; //显示窗口
      },
      /**
       * 添加和修改菜单窗口关闭事件
       */
      onClose() {
        this.menuDialog.visible = false; //关闭窗口
      },
      /**
       * 添加和修改菜单窗口确认事件
       */
      onConfirm() {
        //表单验证
        this.$refs.menuForm.validate((valid) => {
          if (valid) {

          }
        });
      },
      /**
       * 选择所属菜单
       */
      selectParentMenu() {

      },
    },
  };
```


9.3.3 编写选择所属权限菜单代码

（1）在src/api/menu.js脚本文件编写获取所属菜单列表的api脚本代码

```
/**
 * 获取上级菜单
 * @returns
 */
async getParentMenuList(params) {
  return await http.get("/api/permission/parent/list", params)
}
```

(2) 编写选择所属权限菜单窗口代码

注意：需要添加选择所属权限菜单窗口属性及事件。

```
<!-- 选择所属菜单弹框 -->
<system-dialog
  :title="parentDialog.title"
  :width="parentDialog.width"
  :height="parentDialog.height"
  :visible="parentDialog.visible"
  @onClose="onParentClose"
  @onConfirm="onParentConfirm"
>
  <div slot="content">
    <el-tree
      style="font-size: 14px"
      ref="parentTree"
      :data="parentMenuList"
      node-key="id"
      :props="defaultProps"
      empty-text="暂无数据"
      :show-checkbox="false"
      default-expand-all
      :highlight-current="true"
      :expand-on-click-node="false"
      @node-click="handleNodeClick"
    >
      <div class="customer-tree-node" slot-scope="{ node, data }">
        <!-- 长度为0说明没有下级 -->
        <span v-if="data.children.length == 0">
          <i class="el-icon-document" style="color: #8c8c8c; font-size: 18px"/>
        </span>
        <span v-else @click.stop="openBtn(data)">
          <svg-icon v-if="data.open" icon-class="add-s"/>
          <svg-icon v-else icon-class="sub-s"/>
        </span>
        <span style="margin-left: 3px">{{ node.label }}</span>
      </div>
    </el-tree>
  </div>
</system-dialog>
```

(3) 编写选择所属权限菜单窗口脚本代码

注意：以下是选择所属权限菜单窗口核心代码，原有的属性和方法已省略。

```
export default {
  name: 'menuList',
  data() {
    return {
      //上级菜单弹框属性
```

```
      parentDialog: {
        title: '选择所属菜单',
        width: 280,
        height: 450,
        visible: false
      },
      //树属性定义
      defaultProps: {
        children: 'children',
        label: 'label'
      },
      parentMenuList: [] //所属菜单列表
    }
  },
  methods: {
    /**
     * 选择所属菜单
     */
    async selectParentMenu() {
      //显示窗口
      this.parentDialog.visible = true;
      //发送查询请求
      let res = await menuApi.getParentMenuList();
      //判断是否成功
      if (res.success) {
        //赋值
        this.parentMenuList = res.data;
      }
    },
    /**
     * 选择所属菜单取消事件
     */
    onParentClose() {
      this.parentDialog.visible = false //关闭窗口
    },
    /**
     * 选择所属菜单确认事件
     */
    onParentConfirm() {
      this.parentDialog.visible = false //关闭窗口
    },
    /**
     * 切换图标
     * @param data
     */
    openBtn(data) {
      data.open = !data.open
      this.$refs.parentTree.store.nodesMap[data.id].expanded = !data.open
    },
    /**
     * 所属菜单节点点击事件
     */
    handleNodeClick(data) {
      //所属父级菜单ID
      this.menu.parentId = data.id;
      //所属父级菜单名称
      this.menu.parentName = data.label;
    },
```

```
64      }
65    }
```

9.3.4 优化图标选择器

（1）创建图标选择器脚本

在src/utils目录 下创建icons.js脚步文件。

```
export const elementIcons = ["platform-eleme", "eleme", "delete-solid", "delete", "s-tools", "setting", "user-solid", "user", "phone", "phone-outline", "more", "more-outline", "star-on", "star-off", "s-goods", "goods", "warning", "warning-outline", "question", "info", "remove", "circle-plus", "success", "error", "zoom-in", "zoom-out", "remove-outline", "circle-plus-outline", "circle-check", "circle-close", "s-help", "help", "minus", "plus", "check", "close", "picture", "picture-outline", "picture-outline-round", "upload", "upload2", "download", "camera-solid", "camera", "video-camera-solid", "video-camera", "message-solid", "bell", "s-cooperation", "s-order", "s-platform", "s-fold", "s-unfold", "s-operation", "s-promotion", "s-home", "s-release", "s-ticket", "s-management", "s-open", "s-shop", "s-marketing", "s-flag", "s-comment", "s-finance", "s-claim", "s-custom", "s-opportunity", "s-data", "s-check", "s-grid", "menu", "share", "d-caret", "caret-left", "caret-right", "caret-bottom", "caret-top", "bottom-left", "bottom-right", "back", "right", "bottom", "top", "top-left", "top-right", "arrow-left", "arrow-right", "arrow-down", "arrow-up", "d-arrow-left", "d-arrow-right", "video-pause", "video-play", "refresh", "refresh-right", "refresh-left", "finished", "sort", "sort-up", "sort-down", "rank", "loading", "view", "c-scale-to-original", "date", "edit", "edit-outline", "folder", "folder-opened", "folder-add", "folder-remove", "folder-delete", "folder-checked", "tickets", "document-remove", "document-delete", "document-copy", "document-checked", "document", "document-add", "printer", "paperclip", "takeaway-box", "search", "monitor", "attract", "mobile", "scissors", "umbrella", "headset", "brush", "mouse", "coordinate", "magic-stick", "reading", "data-line", "data-board", "pie-chart", "data-analysis", "collection-tag", "film", "suitcase", "suitcase-1", "receiving", "collection", "files", "notebook-1", "notebook-2", "toilet-paper", "office-building", "school", "table-lamp", "house", "no-smoking", "smoking", "shopping-cart-full", "shopping-cart-1", "shopping-cart-2", "shopping-bag-1", "shopping-bag-2", "sold-out", "sell", "present", "box", "bank-card", "money", "coin", "wallet", "discount", "price-tag", "news", "guide", "male", "female", "thumb", "cpu", "link", "connection", "open", "turn-off", "set-up", "chat-round", "chat-line-round", "chat-square", "chat-dot-round", "chat-dot-square", "chat-line-square", "message", "postcard", "position", "turn-off-microphone", "microphone", "close-notification", "bangzhu", "time", "odometer", "crop", "aim", "switch-button", "full-screen", "copy-document", "mic", "stopwatch", "medal-1", "medal", "trophy", "trophy-1", "first-aid-kit", "discover", "place", "location", "location-outline", "location-information", "add-location", "delete-location", "map-location", "alarm-clock", "timer", "watch-1", "watch", "lock", "unlock", "key", "service", "mobile-phone", "bicycle", "truck", "ship", "basketball", "football", "soccer", "baseball", "wind-power", "light-rain", "lightning", "heavy-rain", "sunrise", "sunrise-1", "sunset", "sunny", "cloudy", "partly-cloudy", "cloudy-and-sunny", "moon", "moon-night", "dish", "dish-1", "food", "chicken", "fork-spoon", "knife-fork", "burger", "tableware", "sugar", "dessert", "ice-cream", "hot-water", "water-cup", "coffee-cup", "cold-drink", "goblet", "goblet-full", "goblet-square", "goblet-square-full", "refrigerator", "grape", "watermelon", "cherry", "apple", "pear", "orange", "coffee", "ice-tea", "ice-drink", "milk-tea", "potato-strips", "lollipop", "ice-cream-square", "ice-cream-round"].map(s => "el-icon-" + s);
```

（2）引用图标脚本文件

在menuList.vue组件中引入icons.js脚本文件。

```
//导入自定义的icon图标库
import { elementIcons } from "@/utils/icons";
```

（3）定义图标选择器所需要的属性及事件。

```
export default {
  name: 'menuList',
  data() {
    return {

      iconList: [],//图标列表
      userChooseIcon: "",//用户选中的图标
    }
  },
  methods: {

    /**
     * 打开添加窗口
     */
    openAddWindow() {
      this.$resetForm('menuForm', this.menu) //清空表单数据
      this.menuDialog.title = '新增菜单' //设置窗口标题
      this.menuDialog.visible = true //显示窗口
      this.userChooseIcon = "";//清空菜单图标
    },
    /**
     * 获取图标列表
     */
    getIconList(){
      this.iconList = elementIcons;
    },
    //给icon绑定的点击事件
    setIcon(icon) {
      this.userChooseIcon = icon; //将i的样式设为选中的样式el-icon-xxx
      this.menu.icon = icon;
    },
  },
  //初始化时调用
  created() {
    //调用查询菜单列表的方法
    this.search()
    //调用获取图标列表
    this.getIconList();
  },

}
```

（4）编写样式代码

```
.iconList {
  width: 400px;
  height: 300px;
  overflow-y: scroll;
  overflow-x: hidden;
  display: flex;
  justify-content: space-around;
```

```
    flex-wrap: wrap;
    i {
      display: inline-block;
      width: 60px;
      height: 45px;
      color: #000000;
      font-size: 20px;
      border: 1px solid #e6e6e6;
      border-radius: 4px;
      cursor: pointer;
      text-align: center;
      line-height: 45px;
      margin: 5px;
      &:hover {
        color: orange;
        border-color:orange;
      }
    }
  }

  .chooseIcons{
    width: 175px;
    background-color: #FFFFFF;
    background-image: none;
    border-radius: 4px;
    border: 1px solid #DCDFE6;
    box-sizing: border-box;
    color: #606266;
    display: inline-block;
    font-size: inherit;
    height: 33px;
    line-height: 25px;
    outline: none;
    padding: 0 15px;
    transition: border-color 0.2s cubic-bezier(0.645, 0.045, 0.355, 1);
  }
```


(5) 替换图标选择器输入框

将以下图标选择器的html代码替换原有的 `<el-input>` 标签。

```
<div class="chooseIcons">
  <el-popover placement="bottom" width="450" trigger="click">
    <span
      slot="reference"
      style="
        display: inline-block;
        width: 200px;
        height: 33px;
        line-height: 33px;
      "
    >
      <i :class="userChooseIcon"></i>
      {{ userChooseIcon }}
    </span>
    <div class="iconList">
      <i
        v-for="item in iconList"
```

```
          :key="item"
          :class="item"
          @click="setIcon(item)"
          style="font-size: 20px"
        ></i>
      </div>
    </el-popover>
  </div>
```

9.3.5 封装图标选择器组件

（1）在src/components/system目录下创建MyIcon.vue组件。

```
<template>
  <div class="chooseIcons">
    <el-popover placement="bottom" width="450" trigger="click">
                <span
                  slot="reference"
                  style="
                    display: inline-block;
                    width: 200px;
                    height: 33px;
                    line-height: 33px;
                  "
                >
                  <i :class="userChooseIcon"></i>
                  {{ userChooseIcon }}
                </span>
      <div class="iconList">
        <i
          v-for="item in iconList"
          :key="item"
          :class="item"
          @click="setIcon(item)"
          style="font-size: 20px"
        ></i>
      </div>
    </el-popover>
  </div>
</template>

<script>
//导入自定义的icon图标库
import { elementIcons } from "@/utils/icons";
export default {
  name: 'MyIcon',
  data(){
    return{
      userChooseIcon:"",//用户选中的图标
      iconList:[],//图标列表
    }
  },
  methods:{
    /**
     * 获取图标列表
     */
```

```
    getIconList(){
      this.iconList = elementIcons;
    },
    //给icon绑定的点击事件
    setIcon(icon) {
      //将i的样式设为选中的样式el-icon-xxx
      this.userChooseIcon = icon;
      //将选中的图标传递给父组件
      this.$emit("selecticon",icon)
    },
  },
  created() {
    //获取图标列表
    this.getIconList();
  }
}
</script>

<style scoped lang="scss">
.iconList {
  width: 400px;
  height: 300px;
  overflow-y: scroll;
  overflow-x: hidden;
  display: flex;
  justify-content: space-around;
  flex-wrap: wrap;
  i {
    display: inline-block;
    width: 60px;
    height: 45px;
    color: #000000;
    font-size: 20px;
    border: 1px solid #e6e6eb;
    border-radius: 4px;
    cursor: pointer;
    text-align: center;
    line-height: 45px;
    margin: 5px;
    &:hover {
      color: orange;
      border-color:orange;
    }
  }
}

.chooseIcons{
  width: 175px;
  background-color: #FFFFFF;
  background-image: none;
  border-radius: 4px;
  border: 1px solid #DCDFE6;
  box-sizing: border-box;
  color: #606266;
  display: inline-block;
  font-size: inherit;
  height: 33px;
  line-height: 25px;
```

```
    outline: none;
    padding: 0 15px;
    transition: border-color 0.2s cubic-bezier(0.645, 0.045, 0.355, 1);
  }
  </style>
```

（2）编写menuList.vue脚本代码

```
//导入自定义图标组件
import MyIcon from '@/components/system/MyIcon.vue'


export default {
  name: 'menuList',
  //注册组件
  components: {
    SystemDialog,
    MyIcon
  },
  methods: {
    /**
     * 打开添加窗口
     */
    openAddWindow() {
      this.$resetForm('menuForm', this.menu); //清空表单数据
      this.menuDialog.title = '新增菜单' //设置窗口标题
      this.menuDialog.visible = true //显示窗口
      this.$nextTick(() => {
        this.$refs["child"].userChooseIcon = "";//清空菜单图标
      })
    },
    /**
     * 给icon绑定的点击事件
     * @param icon
     */
    setIcon(icon) {
      this.menu.icon = icon;
    },
  }
}
```

（3）编写menuList.vue页面代码

```
<el-form-item size="small" label="菜单图标">
  <my-icon @selecticon="setIcon" ref="child"></my-icon>
</el-form-item>
```

9.3.6 新增权限菜单

（1）编写前端api脚本

在src/api/menu.js文件中加入新增权限的方法，代码如下：

```
/**
 * 添加菜单
 * @returns
 */
async addMenu(params){
  return await http.post("/api/permission/add",params)
}
```

(2) 编写menuList.vue的窗口确认事件方法

```
/**
 * 添加和修改菜单窗口确认事件
 */
onConfirm() {
  //表单验证
  this.$refs.menuForm.validate(async (valid) => {
    if (valid) {
      let res = null;
      //判断菜单ID是否为空
      if (this.menu.id === "") {
        //发送添加请求
        res = await menuApi.addMenu(this.menu);
      } else {
        //发送修改请求

      }
      //判断是否成功
      if (res.success) {
        this.$message.success(res.message);
        //刷新
        this.search();
        //关闭窗口
        this.menuDialog.visible = false;
      } else {
        this.$message.error(res.message);
      }
    }
  })
}
```


9.4 编辑权限菜单

9.4.1 效果图

9.4.2 编写前端api脚本

在src/api/menu.js文件中加入编辑权限菜单的方法，代码如下：

```
/**
 * 修改菜单
 * @returns
 */
async updateMenu(params){
  return await http.put("/api/permission/update",params)
}
```

9.4.3 编写修改权限菜单脚本

在menuList.vue组件中编写编辑菜单及窗口确认事件。

```
/**
 * 编辑菜单
 * @param row
 */
editMenu(row) {
  //把当前要编辑的数据复制到数据域，给表单回显
  this.$objCopy(row, this.menu);
  //设置弹框属性
  this.menuDialog.title = "编辑菜单";
  this.menuDialog.visible = true;
  this.$nextTick(() => {
    this.$refs["child"].userChooseIcon = row.icon;//菜单图标回显
  })
},
/**
 * 添加和修改菜单窗口确认事件
 */
onConfirm() {
  //表单验证
  this.$refs.menuForm.validate(async (valid) => {
```

```
      if (valid) {
        let res = null;
        //判断菜单ID是否为空
        if (this.menu.id === "") {
          //发送添加请求
          res = await menuApi.addMenu(this.menu);
        } else {
          //发送修改请求
          res = await menuApi.updateMenu(this.menu);
        }
        //判断是否成功
        if (res.success) {
          this.$message.success(res.message);
          //刷新
          //this.search();
          window.location.reload();
          //关闭窗口
          this.menuDialog.visible = false;
        } else {
          this.$message.error(res.message);
        }
      }
    })
  },
```

9.5 删除权限菜单

9.5.1 需求分析

删除某个菜单前，判断该菜单是否存在子菜单，如果存在子菜单，则提示不能删除；如果该菜单下不存在子菜单，则提示是否确认删除。

9.5.2 编写前端api脚本

在src/api/menu.js文件中加入 `检查菜单下是否存在子菜单` 和 `删除菜单` 的方法，代码如下：

```
/**
 * 检查菜单下是否存在子菜单
 */
async checkPermission(param){
  return await http.getRestApi("/api/permission/check",param);
},
/**
 * 删除菜单
 * @returns
 */
async deleteById(params) {
  return await http.delete("/api/permission/delete", params);
}
```

9.5.3 删除按钮代码

```
<el-table-column align="center">
  <template slot-scope="scope">
    <el-button
      type="primary"
      icon="el-icon-edit"
```

```
        size="small"
        @click="editMenu(scope.row)"
      >编辑
      </el-button>
      <el-button
        type="danger"
        size="small"
        icon="el-icon-delete"
        @click="deleteMenu(scope.row)"
      >删除
      </el-button>
    </template>
  </el-table-column>
```

9.5.4 删除按钮点击事件

```
/**
 * 删除菜单
 * @param row
 */
async deleteMenu(row) {
  //判断是否存在子菜单
  let result = await menuApi.checkPermission({ id: row.id });
  //判断是否可以删除
    if (!result.success) {
      //提示不能删除
      this.$message.warning(result.message);
    } else {
      //确认是否删除
      let confirm =await this.$myconfirm("确定要删除该数据吗?");
      if (confirm) {
        //发送删除请求
        let res = await menuApi.deleteById({ id: row.id });
        //判断是否成功
        if (res.success) {
          //成功提示
          this.$message.success(res.message);
          //刷新
          this.search();
        } else {
          //失败提示
          this.$message.error(res.message);
        }
      }
  }
}
```


10. 角色管理

10.1 角色管理后端接口

10.1.1 RoleQueryVo

在com.manong.vo.query包下创建RoleQueryVo查询条件类。

```
package com.manong.vo.query;

import com.manong.entity.Role;
import lombok.Data;

@Data
public class RoleQueryVo extends Role {
    private Long pageNo = 1L;//当前页码
    private Long pageSize = 10L;//每页显示数量
    private Long userId;//用户ID
}
```

10.1.2 RoleService

在RoleService接口中编写 根据用户ID查询角色列表 的方法。

注意：非管理员只能查询自己创建的角色信息。

```
public interface RoleService extends IService<Role> {
    /**
     * 根据用户查询角色列表
     * @param page
     * @param roleQueryVo
     * @return
     */
    IPage<Role> findRoleListByUserId(IPage<Role> page, RoleQueryVo roleQueryVo);
}
```

10.1.3 RoleServiceImpl

在RoleServiceImpl实现类中实现RoleService接口的方法。

```
@Service
@Transactional
public class RoleServiceImpl extends ServiceImpl<RoleMapper, Role> implements
RoleService {

    @Resource
    private UserMapper userMapper;

    /**
     * 根据用户查询角色列表
     *
     * @param page
     * @param roleQueryVo
     * @return
     */
    @Override
    public IPage<Role> findRoleListByUserId(IPage<Role> page, RoleQueryVo
roleQueryVo) {
        //创建条件构造器
        QueryWrapper<Role> queryWrapper = new QueryWrapper<Role>();
```

```
        //角色名称
        queryWrapper.like(!ObjectUtils.isEmpty(roleQueryVo.getRoleName()),
                          "role_name",roleQueryVo.getRoleName());
        //排序
        queryWrapper.orderByAsc("id");
        //根据用户ID查询用户信息
        User user = userMapper.selectById(roleQueryVo.getUserId());
        //如果用户不为空、且不是管理员，则只能查询自己创建的角色
        if(user!=null && !ObjectUtils.isEmpty(user.getIsAdmin()) &&
user.getIsAdmin() !=1){
            queryWrapper.eq("create_user",roleQueryVo.getUserId());
        }
        return baseMapper.selectPage(page,queryWrapper);
    }
}
```

10.1.4 RoleController

```
package com.manong.controller;


import com.baomidou.mybatisplus.core.metadata.IPage;
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
import com.manong.entity.Role;
import com.manong.service.RoleService;
import com.manong.utils.Result;
import com.manong.vo.query.RoleQueryVo;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;

@RestController
@RequestMapping("/api/role")
public class RoleController {
    @Resource
    private RoleService roleService;

    /**
     * 分页查询角色列表
     * @param roleQueryVo
     * @return
     */
    @GetMapping("/list")
    public Result list(RoleQueryVo roleQueryVo){
        //创建分页对象
        IPage<Role> page = new Page<Role>
(roleQueryVo.getPageNo(),roleQueryVo.getPageSize());
        //调用分页查询方法
        roleService.findRoleListByUserId(page,roleQueryVo);
        //返回数据
        return Result.ok(page);
    }

    /**
     * 添加角色
     * @param role
     * @return
```

```
     */
    @PostMapping("/add")
    public Result add(@RequestBody Role role){
        if(roleService.save(role)){
            return Result.ok().message("角色添加成功");
        }
        return Result.error().message("角色添加失败");
    }

    /**
     * 修改角色
     * @param role
     * @return
     */
    @PutMapping("/update")
    public Result update(@RequestBody Role role){
        if(roleService.updateById(role)){
            return Result.ok().message("角色修改成功");
        }
        return Result.error().message("角色修改失败");
    }

    /**
     * 删除角色
     * @param id
     * @return
     */
    @DeleteMapping("/delete/{id}")
    public Result delete(@PathVariable Long id){
        if(roleService.removeById(id)){
            return Result.ok().message("角色删除成功");
        }
        return Result.error().message("角色删除失败");
    }

}
```


10.2 查询角色列表

10.2.1 效果图

10.2.2 页面原型代码

在src/views/system/role目录下创建roleList.vue组件。

```
<template>
  <el-main>
    <!-- 查询条件 -->
    <el-form
      :model="searchModel"
      ref="searchForm"
      label-width="80px"
      :inline="true"
      size="small"
    >
      <el-form-item>
        <el-input v-model="searchModel.roleName" placeholder="请输入角色名称"/>
      </el-form-item>
      <el-form-item>
        <el-button type="primary" icon="el-icon-search">查询</el-button>
        <el-button icon="el-icon-refresh-right">重置</el-button>
        <el-button type="success" icon="el-icon-plus">新增</el-button>
      </el-form-item>
    </el-form>
    <!-- 数据表格 -->
    <el-table
      :data="roleList"
      :height="tableHeight"
      border
      stripe
      style="width: 100%; margin-bottom: 10px"
    >
      <el-table-column
        prop="id"
        label="角色编号"
        width="100"
        align="center"
```

```
        ></el-table-column>
        <el-table-column prop="name" label="角色名称"></el-table-column>
        <el-table-column prop="remark" label="角色备注"></el-table-column>
        <el-table-column label="操作" align="center" width="290">
          <template slot-scope="scope">
            <el-button
              icon="el-icon-edit"
              type="primary"
              size="small"
              @click="handleEdit(scope.row)"
            >编辑
            </el-button
            >
            <el-button
              icon="el-icon-delete"
              type="danger"
              size="small"
              @click="handleDelete(scope.row)"
            >删除
            </el-button
            >
            <el-button
              icon="el-icon-setting"
              type="primary"
              size="small"
              @click="assignRole(scope.row)"
            >分配权限
            </el-button
            >
          </template>
        </el-table-column>
      </el-table>
      <!-- 分页工具栏 -->
      <el-pagination
        background
        @size-change="handleSizeChange"
        @current-change="handleCurrentChange"
        :current-page="pageNo"
        :page-sizes="[10, 20, 30, 40, 50]"
        :page-size="10"
        layout="total, sizes, prev, pager, next, jumper"
        :total="total"
      >
      </el-pagination>
    </el-main>
  </template>

  <script>

  export default {
    name: 'roleList',
    data() {
      return {
        //查询条件
        searchModel: {
          roleName: ''
        },
        roleList: [], //数据列表
```

```
        tableHeight: 0, //表格高度
        pageNo: 1, //当前页码
        pageSize: 10, //每页显示数量
        total: 0, //总数量
      }
    },
    methods: {
      /**
       * 查询
       */
      search() {
      },
      /**
       * 重置查询条件
       */
      resetValue() {
      },
      /**
       * 当每页数量发生变化时触发该事件
       */
      handleSizeChange(size) {
      },
      /**
       * 当页码发生变化时触发该事件
       */
      handleCurrentChange(page) {
      },
      /**
       * 打开编辑窗口
       */
      handleEdit(row) {
      },
      /**
       * 删除
       */
      handleDelete(row) {
      },
    },
    //初始化时调用
    created() {
    },
    //挂载后调用
    mounted() {
      this.$nextTick(() => {
        this.tableHeight = window.innerHeight - 220
      })
    }
  }
  </script>
```

10.2.3 编写前端api脚本

在src/api目录下创建role.js。

```
import http from '@/utils/request'

export default {
  /**
   * 查询角色列表
   * @returns
   */
  async getRoleListApi(params) {
    return await http.get("/api/role/list", params);
  }
}
```

10.2.4 编写页面组件代码

注意：脚本代码中原有的属性和方法已省略，此处不再赘述。

```
<template>
  <el-main>
    <!-- 查询条件 -->
    <el-form
      :model="searchModel"
      ref="searchForm"
      label-width="80px"
      :inline="true"
      size="small"
    >
      <el-form-item>
        <el-input v-model="searchModel.roleName" placeholder="请输入角色名称"/>
      </el-form-item>
      <el-form-item>
        <el-button type="primary" icon="el-icon-search"
@click="search(pageNo,pageSize)">查询
        </el-button>
        <el-button icon="el-icon-refresh-right" @click="resetValue">重置</el-
button>
        <el-button type="success" icon="el-icon-plus">新增</el-button>
      </el-form-item>
    </el-form>
    <!-- 数据表格 -->
    <el-table
      :data="roleList"
      :height="tableHeight"
      border
      stripe
      style="width: 100%; margin-bottom: 10px"
    >
      <el-table-column
        prop="id"
        label="角色编号"
        width="100"
        align="center"
      ></el-table-column>
      <el-table-column prop="roleName" label="角色名称"></el-table-column>
      <el-table-column prop="remark" label="角色备注"></el-table-column>
      <el-table-column label="操作" align="center" width="290">
        <template slot-scope="scope">
          <el-button
```

```
              icon="el-icon-edit"
              type="primary"
              size="small"
              @click="handleEdit(scope.row)"
            >编辑
            </el-button>
            <el-button
              icon="el-icon-delete"
              type="danger"
              size="small"
              @click="handleDelete(scope.row)"
            >删除
            </el-button>
            <el-button
              icon="el-icon-setting"
              type="primary"
              size="small"
              @click="assignRole(scope.row)"
            >分配权限
            </el-button>
          </template>
        </el-table-column>
      </el-table>
      <!-- 分页工具栏 -->
      <el-pagination
        background
        @size-change="handleSizeChange"
        @current-change="handleCurrentChange"
        :current-page="pageNo"
        :page-sizes="[10, 20, 30, 40, 50]"
        :page-size="10"
        layout="total, sizes, prev, pager, next, jumper"
        :total="total"
      >
      </el-pagination>
    </el-main>
  </template>

  <script>
  //导入role.js脚本
  import roleApi from '@/api/role';

  export default {
    name: 'roleList',
    data() {
      return {
        //查询条件
        searchModel: {
          roleName: '',
          pageNo:1,
          pageSize:10,
          userId:this.$store.getters.userId //用户ID
        },
        roleList: [], //数据列表
        tableHeight: 0, //表格高度
        pageNo: 1, //当前页码
        pageSize: 10, //每页显示数量
        total: 0, //总数量
```

```
    }
  },
  methods: {
    /**
     * 查询
     */
    async search(pageNo=1,pageSize=10) {
      this.searchModel.pageNo = pageNo;//当前页码
      this.searchModel.pageSize = pageSize;//每页显示数量
      //发送查询请求
      let res = await roleApi.getRoleListApi(this.searchModel);
      //执行成功
      if (res.success) {
        //角色列表
        this.roleList = res.data.records;
        //总数量
        this.total = res.data.total;
      }
    },
    /**
     * 重置查询条件
     */
    resetValue() {
      //清空查询条件
      this.searchModel.roleName="";
      //重新查询
      this.search();
    },
    /**
     * 当每页数量发生变化时触发该事件
     */
    handleSizeChange(size) {
      this.pageSize = size; //将每页显示的数量交给成员变量
      this.search(this.pageNo, size);
    },
    /**
     * 当页码发生变化时触发该事件
     */
    handleCurrentChange(page) {
      this.pageNo = page;
      //调用查询方法
      this.search(page, this.pageSize);
    },
  },
  created() {
    //调用查询角色列表的方法
    this.search();
  },
  mounted() {
    this.$nextTick(() => {
      this.tableHeight = window.innerHeight - 220
    })
  }
}
</script>
```

注意：当前登录用户ID从Vuex中获取，需要进行添加如下代码。

（1）在src/store/modules/user.js文件中保存用户ID。

注意：改动第10行、第22行代码即可。

```
1     getInfo({ commit, state }) {
2       return new Promise((resolve, reject) => {
3         //调用api/user里面的getInfo方法获取用户信息和权限信息
4         getInfo(state.token).then(response => {
5           const { data } = response
6           if (!data) {
7             reject('Verification failed, please Login again.')
8           }
9           //此处解构出用户ID
10          const { roles, name, avatar, introduction,id } = data
11          // roles must be a non-empty array
12          if (!roles || roles.length <= 0) {
13            reject('用户未分配角色权限，请联系管理员！')
14          }
15          //将权限字段保存到sessionStorage中
16          sessionStorage.setItem("codeList",JSON.stringify(roles));
17          commit('SET_ROLES', roles)
18          commit('SET_NAME', name)
19          commit('SET_AVATAR', avatar)
20          commit('SET_INTRODUCTION', introduction)
21          //将用户ID保存到Vuex中
22          commit('SET_USERUID', id);//用户ID
23          resolve(data)
24        }).catch(error => {
25          console.log(error)
26          reject(error)
27        })
28      })
29    }
```

同理，src/store/modules/user.js文件mutations中设置用户ID。

```
1   //设置用户ID
2   SET_USERUID: (state, userId) => {
3     state.userId = userId
4   },
```

（2）在src/store/getters.js文件中保存用户ID。

```
1   userId: state => state.user.userId,
```

10.3 新增角色

10.3.1 效果图

10.3.2 编写新增角色窗口代码

（1）在src/views/system/role/roleList.vue组件中引入SystemDialog.vue组件。

```
//导入对话框组件
import SystemDialog from '@/components/system/SystemDialog.vue';
```

（2）在roleList.vue组件中注册SystemDialog.vue组件。

```
export default {
  name: "roleList",
  //注册组件
  components: {
      SystemDialog
  },
}
```

（3）编写新增角色窗口代码

注意：需要添加窗口属性及事件，表单属性等

```
<!-- 添加和修改角色窗口 -->
<system-dialog
  :title="roleDialog.title"
  :visible="roleDialog.visible"
  :width="roleDialog.width"
  :height="roleDialog.height"
  @onClose="onClose"
  @onConfirm="onConfirm"
>
  <div slot="content">
    <el-form
      :model="role"
```

```
        ref="roleForm"
        :rules="rules"
        label-width="80px"
        :inline="false"
        size="small"
      >
        <el-form-item label="角色编码" prop="roleCode">
          <el-input v-model="role.roleCode"></el-input>
        </el-form-item>
        <el-form-item label="角色名称" prop="roleName">
          <el-input v-model="role.roleName"></el-input>
        </el-form-item>
        <el-form-item label="角色描述">
          <el-input type="textarea" v-model="role.remark" :rows="5"></el-input>
        </el-form-item>
      </el-form>
    </div>
  </system-dialog>
```

（4）编写新增角色窗口脚本代码

注意：以下是新增角色窗口核心代码，原有的属性和方法已省略。

```
export default {
  components: { SystemDialog },
  name: 'roleList',
  data() {
    return {
      rules: {
        roleCode: [{ required: true, trigger: 'blur', message: '请输入角色编码' }],
        roleName: [{ required: true, trigger: 'blur', message: '请输入角色名称' }]
      },
      //添加和修改角色窗口属性
      roleDialog: {
        title: '',
        visible: false,
        height: 230,
        width: 500
      },
      //角色对象
      role: {
        id:"",
        roleCode:"",
        roleName:"",
        remark:"",
        createUser:this.$store.getters.userId
      }
    }
  },
  methods: {
    /**
     * 打开添加窗口
     */
    openAddWindow() {
      //清空表单数据
      this.$resetForm("roleForm",this.role);
      this.roleDialog.title = '新增角色'//设置窗口标题
      this.roleDialog.visible = true//显示窗口
```

```
    },
    /**
     * 窗口取消事件
     */
    onClose() {
      this.roleDialog.visible = false
    },
    /**
     * 窗口确认事件
     */
    onConfirm() {
      //表单验证
      this.$refs.roleForm.validate(async (valid)=>{
        if(valid){

        }
      });
    },
  }
}
```

（5）在store中保存当前登录用户id

在src/store/modules/user.js中保存当前登录用户id。

```
const mutations = {
  //省略其他代码......
  //设置用户ID
  SET_USERUID: (state, userId) => {
    state.userId = userId
  },
}

const actions = {
  // get user info
  getInfo({ commit, state }) {
    return new Promise((resolve, reject) => {
      //调用api/user里面的getInfo方法获取用户信息和权限信息
      getInfo(state.token).then(response => {
        const { data } = response
        if (!data) {
          reject('Verification failed, please Login again.')
        }
        //解构出用户id
        const { roles, name, avatar, introduction,id } = data

        if (!roles || roles.length <= 0) {
          reject('getInfo: roles must be a non-null array!')
        }
        //省略其他代码......
        commit('SET_USERUID', id);//用户ID
        resolve(data)
      }).catch(error => {
        reject(error)
      })
    })
  },
}
```

```
34
```

完成上述操作后，在在src/store/getters.js中保存用户id。

```
const getters = {
  //省略其它代码

  //用户ID
  userId: state => state.user.userId,

}
export default getters
```

10.3.3 编写前端api脚本

在src/api/role.js文件下编写添加角色的方法。

```
/**
 * 添加角色
 * @returns
 */
async addRoleApi(params) {
  return await http.post("/api/role/add", params);
}
```

10.3.4 编写新增角色确认事件

编写roleList.vue的窗口确认事件方法。

```
/**
 * 窗口确认事件
 */
onConfirm() {
  //表单验证
  this.$refs.roleForm.validate(async (valid)=>{
    if(valid){
      let res = null;
      //判断角色ID是否为空
      if (this.role.id === "") {
        //新增
        //发送添加请求
        res = await roleApi.addRoleApi(this.role);
      } else {
        //发送修改请求

      }
      //判断是否成功
      if (res.success) {
        this.$message.success(res.message);
        //刷新
        this.search(this.pageNo,this.pageSize);
        //关闭窗口
        this.roleDialog.visible = false;
      } else {
        this.$message.error(res.message);
      }
    }
  });
```

```
30    }
```

10.4 编辑角色

10.4.1 效果图

10.4.2 编写前端api脚本

在src/api/role.js文件中编写编辑角色的方法。

```
1   /**
2    * 编辑角色
3    * @returns
4    */
5   async updateRoleApi(params) {
6     return await http.put("/api/role/update", params);
7   }
```

10.4.3 编写修改角色脚本代码

```
1   /**
2    * 窗口确认事件
3    */
4   onConfirm() {
5     //表单验证
6     this.$refs.roleForm.validate(async (valid)=>{
7       if(valid){
8         let res = null;
9         //判断角色ID是否为空
10        if (this.role.id === "") {
```

```
          //新增
          //发送添加请求
          res = await roleApi.addRoleApi(this.role);
        } else {
          //发送修改请求
          res = await roleApi.updateRoleApi(this.role);
        }
        //判断是否成功
        if (res.success) {
          this.$message.success(res.message);
          //刷新
          this.search(this.pageNo,this.pageSize);
          //关闭窗口
          this.roleDialog.visible = false;
        } else {
          this.$message.error(res.message);
        }
      }
    });
  },
  /**
   * 打开编辑窗口
   */
  handleEdit(row) {
    //数据回显
    this.$objCopy(row, this.role); //将当前编辑的数据复制到role对象中
    //设置窗口标题
    this.roleDialog.title = "编辑角色";
    //显示编辑角色窗口
    this.roleDialog.visible = true;
  }
```


10.5 分配权限

10.5.1 分配权限思路

分配权限前需要回显当前角色拥有的权限信息。 被分配角色的权限信息，不能超出当前用户所拥有的权限信息。

1. 查询当前登录用户的信息。
2. 判断当前登录用户是否是管理员，如果是管理员，则查询所有的权限信息；如果不是管理员，则需要根据当前用户Id查询出当前用户所拥有的权限信息。
3. 将当前登录用户所拥有的权限封装成菜单树。
4. 查询要分配角色拥有的权限列表，进行数据回显。

保存角色权限关系时，需要删除该角色原有的权限信息，再保存新的权限信息。

10.5.2 分配权限回显接口实现

10.5.2.1 PermissionMapper

```
/**
 * 根据角色ID查询权限列表
 * @param roleId
 * @return
 */
List<Permission> findPermissionListByRoleId(Long roleId);
```

10.5.2.2 PermissionMapper.xml

```
<select id="findPermissionListByRoleId" resultType="com.manong.entity.Permission">
    select t1.* from sys_permission t1 inner join sys_role_permission t2 on t1.id = t2.permission_id
    where t2.role_id = #{roleId}
</select>
```

10.5.2.3 RolePermissionVo

在com.manong.vo包下创建RolePermissionVo类，该类用于封装权限菜单数据回显。

```
package com.manong.vo;

import com.manong.entity.Permission;
import lombok.Data;

import java.util.ArrayList;
import java.util.List;

@Data
public class RolePermissionVo {
    /**
     * 菜单数据
     */
    private List<Permission> permissionList = new ArrayList<Permission>();

    /**
     * 该角色原有分配的菜单数据
     */
    private Object [] checkList;
}
```

10.5.2.4 PermissionService

```
/**
 * 查询分配权限树列表
 * @param userId
 * @param roleId
 * @return
 */
RolePermissionVo findPermissionTree(Long userId, Long roleId);
```

10.5.2.5 PermissionServiceImpl

```
/**
 * 查询分配权限树列表
 *
 * @param userId
 * @param roleId
 * @return
 */
@Override
public RolePermissionVo findPermissionTree(Long userId, Long roleId) {
    //1.查询当前用户信息
    User user = userMapper.selectById(userId);
    List<Permission> list = null;
    //2.判断当前用户角色，如果是管理员，则查询所有权限；如果不是管理员，则只查询自己所拥有的的权限
    if(!ObjectUtils.isEmpty(user.getIsAdmin()) && user.getIsAdmin() == 1){
        //查询所有权限
        list = baseMapper.selectList(null);
    }else{
        //根据用户ID查询
        list = baseMapper.findPermissionListByUserId(userId);
    }
    //3.组装成树数据
    List<Permission> permissionList = MenuTree.makeMenuTree(list, 0L);
    //4.查询要分配角色的原有权限
    List<Permission> rolePermissions = baseMapper.findPermissionListByRoleId(roleId);
    //5.找出该角色存在的数据
    List<Long> listIds = new ArrayList<Long>();
    Optional.ofNullable(list).orElse(new ArrayList<>())
            .stream()
            .filter(Objects::nonNull) //等同于 obj -> obj!=null
            .forEach(item -> {
                Optional.ofNullable(rolePermissions).orElse(new ArrayList<>())
                        .stream()
                        .filter(Objects::nonNull)
                        .forEach(obj ->{
                            if(item.getId().equals(obj.getId())){
                                listIds.add(obj.getId());
                                return;
                            }
                        });
            });
    //创建
    RolePermissionVo vo = new RolePermissionVo();
    vo.setPermissionList(permissionList);
    vo.setCheckList(listIds.toArray());
    return vo;
}
```

10.5.2.6 RoleController

在RoleController编写查询权限树数据的方法。

```
/**
 * 分配权限-查询权限树数据
 *
```

```
     * @param userId
     * @param roleId
     * @return
     */
    @GetMapping("/getAssignPermissionTree")
    public Result getAssignPermissionTree(Long userId, Long roleId) {
        //调用查询权限树数据的方法
        RolePermissionVo permissionTree =
    permissionService.findPermissionTree(userId, roleId);
        //返回数据
        return Result.ok(permissionTree);
    }
```

10.5.3 分配权限回显前端实现

10.5.3.1 编写前端api脚本

在src/api/role.js文件中编写查询分配权限树列表的方法。

```
/**
 * 查询分配权限树列表
 * @returns
 */
async getAssignTreeApi(params){
  return await http.get("/api/role/getAssignPermissionTree", params);
}
```

10.5.3.2 编写分配权限窗口代码

在roleList.vue组件中添加分配权限窗口的代码。

```
<!-- 分配权限树窗口 -->
<system-dialog
  :title="assignDialog.title"
  :visible="assignDialog.visible"
  :width="assignDialog.width"
  :height="assignDialog.height"
  @onClose="onAssignClose"
  @onConfirm="onAssignConfirm"
>
  <div slot="content">
    <el-tree
      ref="assignTree"
      :data="assignTreeData"
      node-key="id"
      :props="defaultProps"
      empty-text="暂无数据"
      :show-checkbox="true"
      :highlight-current="true"
      default-expand-all
    ></el-tree>
  </div>
</system-dialog>
```

10.5.3.3 编写判断菜单节点是否是末级节点的方法

在src/utils目录下创建leaf.js脚本文件，代码如下：

```
//判断菜单节点是否是末级节点
export default function leafUtils(){
  const setLeaf =(arr) => {
    if(arr && arr.length >0){
      for(let i =0; i < arr.length;i++){
        if(arr[i].children && arr[i].children.length > 0){
          arr[i].open = false;
          setLeaf(arr[i].children)
        }else{
          arr[i].open = true;
        }
      }
    }
    return arr
  }
  return{
    setLeaf
  }
}
```

10.5.3.4 编写分配权限回显脚本代

注意：原有属性及事件已省略

```
//导入末级节点脚本
import leafUtils from '@/utils/leaf'

export default {
  components: { SystemDialog },
  name: 'roleList',
  data() {
    return {
      //分配权限窗口属性
      assignDialog: {
        title: '',
        visible: false,
        height: 450,
        width: 300
      },
      roleId: '', //角色ID
      assignTreeData: [], //树节点数据
      //树节点属性
      defaultProps: {
        children: 'children',
        label: 'label'
      }
    }
  },
  methods: {
    /**
     * 分配权限
     */
    async assignRole(row) {
      //设置角色ID
```

```
        this.roleId = row.id
        //构建参数
        let params = {
          roleId: row.id,
          userId: this.$store.getters.userId
        }
        //发送查询请求
        let res = await roleApi.getAssignTreeApi(params)
        //判断是否成功
        if (res.success) {
          //获取当前登录用户拥有的所有权限
          let permissionList = res.data.permissionList
          //获取当前被分配的角色已经拥有的权限信息
          let checkList = res.data.checkList
          //判断当前菜单是否是末级
          let { setLeaf } = leafUtils()
          //设置权限菜单列表
          let newPermissionList = setLeaf(permissionList)
          //设置树节点菜单数据
          this.assignTreeData = newPermissionList
          //将回调延迟到下次DOM更新循环之后执行,在修改数据之后立即使用它，然后等待DOM更新。
          this.$nextTick(() => {
            //获取树菜单的节点数据
            let nodes = this.$refs.assignTree.children
            //设置子节点
            this.setChild(nodes, checkList)
          })
        }
        //显示窗口
        this.assignDialog.visible = true
        //设置窗口标题
        this.assignDialog.title = `给【${row.roleName}】分配权限`
      },
      /**
       * 设置子节点
       */
      setChild(childNodes, checkList) {
        //判断是否存在子节点
        if (childNodes && childNodes.length > 0) {
          //循环所有权限
          for (let i = 0; i < childNodes.length; i++) {
            //根据 data 或者 key 拿到 Tree 组件中的 node
            let node = this.$refs.assignTree.getNode(childNodes[i])
            //判断是否已经拥有对应的角色数据
            if (checkList && checkList.length > 0) {
              //循环遍历已有的权限集合
              for (let j = 0; j < checkList.length; j++) {
                //找到已经存在的菜单权限节点
                if (checkList[j] == childNodes[i].id) {
                  //如果节点是展开状态，则将树节点选中
                  if (childNodes[i].open) {
                    this.$refs.assignTree.setChecked(node, true)
                    break
                  }
                }
              }
            }
            //如果存在子节点，则递归选中
```

```
            if (childNodes[i].children) {
              this.setChild(childNodes[i].children, checkList)
            }
          }
        }
      },
      /**
       * 分配权限窗口取消事件
       */
      onAssignClose() {
        this.assignDialog.visible = false
      },
      /**
       * 分配权限窗口确认事件
       */
      async onAssignConfirm() {

      }
    },
  }
```

10.5.4 分配权限接口实现

10.5.4.1 RolePermissionDTO

在com.manong.dto包下创建RolePermissionDTO类

```
package com.manong.dto;

import lombok.Data;

import java.util.List;

/**
 * 用于给角色分配权限时保存 选中的权限数据
 */
@Data
public class RolePermissionDTO {
    private Long roleId;//角色编号
    private List<Long> list;//权限菜单ID集合
}
```

10.5.4.2 RoleMapper

```
/**
 * 删除角色权限关系
 * @param roleId
 */
@Delete("delete from sys_role_permission where role_id = #{roleId}")
void deleteRolePermission(Long roleId);

/**
 * 保存角色权限关系
 * @param roleId
 * @param permissionIds
 * @return
```

```
    */
    int saveRolePermission(Long roleId, List<Long> permissionIds);
```

10.5.4.3 RoleMapper.xml

```
<insert id="saveRolePermission">
    insert into sys_role_permission(role_id,permission_id) values
    <foreach collection="permissionIds" item="item" index="index" separator=",">
        (#{roleId},#{item})
    </foreach>
</insert>
```

10.5.4.4 RoleService

```
/**
 * 保存角色权限关系
 * @param roleId
 * @param permissionIds
 * @return
 */
boolean saveRolePermission(Long roleId, List<Long> permissionIds);
```

10.5.4.5 RoleServiceImpl

```
/**
 * 保存角色权限关系
 *
 * @param roleId
 * @param permissionIds
 * @return
 */
@Override
public boolean saveRolePermission(Long roleId, List<Long> permissionIds) {
    //删除该角色对应的权限信息
    baseMapper.deleteRolePermission(roleId);
    //保存角色权限
    return baseMapper.saveRolePermission(roleId,permissionIds)>0;
}
```

10.5.4.6 RoleController

```
/**
 * 分配权限-保存权限数据
 *
 * @param rolePermissionDTO
 * @return
 */
@PostMapping("/saveRoleAssign")
public Result saveRoleAssign(@RequestBody RolePermissionDTO rolePermissionDTO) {
    if (roleService.saveRolePermission(rolePermissionDTO.getRoleId(),
rolePermissionDTO.getList())) {
      return Result.ok().message("权限分配成功");
    } else {
      return Result.error().message("权限分配失败");
    }
}
```

10.5.5 分配权限前端实现

10.5.5.1 编写前端api脚本

在src/api/role.js文件中编写分配权限的方法。

```
/**
 * 分配权限
 * @returns
 */
async assignSaveApi(params){
  return await http.post("/api/role/saveRoleAssign",params);
}
```

10.5.5.2 编写分配权限窗口确认事件

```
/**
 * 分配权限窗口确认事件
 */
async onAssignConfirm() {
  //获取选中的节点key
  let ids = this.$refs.assignTree.getCheckedKeys()
  //获取选中节点的父节点id
  let pids = this.$refs.assignTree.getHalfCheckedKeys()
  //组装选中的节点ID数据
  let listId = ids.concat(pids)
  //组装参数
  let params = {
    roleId: this.roleId,
    list: listId
  }
  //发送请求
  let res = await roleApi.assignSaveApi(params)
  //判断是否成功
  if (res.success) {
    //关闭窗口
    this.assignDialog.visible = false
    //提示成功
    this.$message.success(res.message)
  } else {
    //提示失败
    this.$message.error(res.data)
  }
}
```


10.6 删除角色

10.6.1 需求分析

在删除角色前需要判断该角色是否被分配出去，如果被分配给用户使用，则提示无法删除，否则提示确认是否删除，确认删除的同时需要将sys_role_permission角色权限关系表对应的数据一起删除。

10.6.2 编写前端api脚本

在src/api/role.js中编写检查该角色是否被使用和删除角色的脚本方法。

11.用户管理

11.1 查询用户列表

11.1.1 查询用户列表接口实现

11.1.1.1 UserQueryVo

在com.manong.vo.query包下创建UserQueryVo条件类。

```
package com.manong.vo.query;

import com.manong.entity.User;
import lombok.Data;

@Data
public class UserQueryVo extends User {
    private Long pageNo = 1;//当前页码
    private Long pageSize = 10L;//每页显示数量
}
```

11.1.1.2 UserService

在UserService接口中编写`分页查询用户信息`的方法。

```
/**
 * 分页查询用户信息
 * @param page
 * @param userQueryVo
 * @return
 */
IPage<User> findUserListByPage(IPage<User> page, UserQueryVo userQueryVo);
```

11.1.1.3 UserServiceImpl

```
/**
 * 分页查询用户信息
 *
 * @param page
 * @param userQueryVo
 * @return
 */
@Override
public IPage<User> findUserListByPage(IPage<User> page, UserQueryVo userQueryVo)
{
```

```
        //创建条件构造器对象
        QueryWrapper<User> queryWrapper = new QueryWrapper<User>();
        //部门编号
        queryWrapper.eq(!ObjectUtils.isEmpty(userQueryVo.getDepartmentId()),
                        "department_id",userQueryVo.getDepartmentId());
        //用户名
        queryWrapper.like(!ObjectUtils.isEmpty(userQueryVo.getUsername()),
                          "username",userQueryVo.getUsername());
        //真实姓名
        queryWrapper.like(!ObjectUtils.isEmpty(userQueryVo.getRealName()),
                          "real_name",userQueryVo.getRealName());
        //电话
        queryWrapper.like(!ObjectUtils.isEmpty(userQueryVo.getPhone()),
                          "phone",userQueryVo.getPhone());
        //查询并返回数据
        return baseMapper.selectPage(page,queryWrapper);
    }
```

11.1.1.4 UserController

```
    @Resource
    private PasswordEncoder passwordEncoder;

    /**
     * 查询用户列表
     * @param userQueryVo
     * @return
     */
    @GetMapping("/list")
    public Result list(UserQueryVo userQueryVo) {
        //创建分页信息
        IPage<User> page = new Page<User>(userQueryVo.getPageNo(),
    userQueryVo.getPageSize());
        //调用分页查询方法
        userService.findUserListByPage(page, userQueryVo);
        //返回数据
        return Result.ok(page);
    }
```


11.1.2 查询用户列表前端实现

11.1.2.1 编写前端api脚本代码

在src/api/user.js文件中添加查询用户列表的方法。

```
export default {
  /**
   * 查询用户列表
   * @param params
   * @returns
   */
  async getUserList(params){
    return await http.get("/api/user/list",params);
  }
}
```

11.1.2.2 加载左侧部门树菜单

在src/views/system/user下创建userList.vue组件。 树菜单样式代码参考department.vue组件。

```
<template>
  <el-container :style="{ height: containerHeight + 'px' }">
    <!-- 左侧部门树形菜单列表 -->
    <el-aside
      style="
        padding: 10px 0px 0px 0px;
        background: #fff;
        border-right: 1px solid #dfe6ec;
      "
      width="200px"
    >
      <el-tree
        style="font-size: 14px"
        ref="leftTree"
        :data="deptList"
        node-key="id"
        :props="defaultProps"
        default-expand-all
        empty-text="暂无数据"
        :show-checkbox="false"
        :highlight-current="true"
        :expand-on-click-node="false"
        @node-click="handleNodeClick"
      >
        <div class="custom-tree-node" slot-scope="{ node, data }">
          <div>
            <span v-if="data.children.length == 0">
              <i class="el-icon-document"></i>
            </span>
            <span v-else @click.stop="openBtn(data)">
              <svg-icon v-if="data.open" icon-class="add-s"/>
              <svg-icon v-else icon-class="sub-s"/>
            </span>
            <!-- 名称 -->
            <span style="margin-left: 3px">{{ node.label }}</span>
          </div>
        </div>
      </el-tree>
    </el-aside>
  </el-container>
</template>

<script>
//导入部门api脚本
import departmentApi from '@/api/department';

export default {
  name: 'userList',
  data() {
    return {
      containerHeight: 0, //容器高度
      deptList: [], //左侧部门树形菜单列表
      //树节点属性
      defaultProps: {
```

```
          children: 'children',
          label: 'departmentName'
        }
      }
    },
    methods: {
      /**
       * 查询部门列表
       */
      async getDeptList() {
        //发送查询请求
        let res = await departmentApi.getDepartmentList(null);
        //判断是否成功
        if (res.success) {
          this.deptList = res.data;
        }
      },
      /**
       * 左侧树节点点击事件
       */
      handleNodeClick(data) {

      },
      //加减号图标点击事件
      openBtn(data) {
        //修改折叠展开状态
        data.open = !data.open
        this.$refs.leftTree.store.nodesMap[data.id].expanded = !data.open;
      },

    },
    created() {
      //查询部门列表
      this.getDeptList();
    },
    mounted() {
      this.$nextTick(() => {
        //内容高度
        this.containerHeight = window.innerHeight - 85;
      })
    }
  }
  </script>
```

11.1.2.3 查询用户列表

注意：省略原有的属性和事件，加载左侧部门菜单部分事件代码有更改。

```
  <template>
    <el-container :style="{ height: containerHeight + 'px' }">
      <!-- 左侧部门树形菜单列表 -->

      <!-- 右侧用户数据 -->
      <!-- 表格数据 -->
      <el-main>
        <!-- 查询条件 -->
        <el-form
          :model="searchModel"
```

```
11          ref="searchForm"
12          label-width="80px"
13          :inline="true"
14          size="small"
15        >
16          <el-form-item>
17            <el-input v-model="searchModel.username" placeholder="请输入用户名"/>
18          </el-form-item>
19          <el-form-item>
20            <el-input v-model="searchModel.realName" placeholder="请输入真实姓名"/>
21          </el-form-item>
22          <el-form-item>
23            <el-input v-model="searchModel.phone" placeholder="请输入电话"/>
24          </el-form-item>
25          <el-form-item>
26            <el-button icon="el-icon-search" type="primary"
27                       @click="search(departmentId, pageNo, pageSize)">查询</el-
    button>
28            <el-button icon="el-icon-delete" @click="resetValue()">重置</el-
    button>
29            <el-button icon="el-icon-plus" size="small" type="success"
30                       @click="openAddWindow()">新增</el-button>
31          </el-form-item>
32        </el-form>
33        <!-- 用户表格数据 -->
34        <el-table
35          :height="tableHeight"
36          :data="userList"
37          border
38          stripe
39          style="width: 100%; margin-bottom: 10px"
40        >
41          <el-table-column prop="username" label="用户名"></el-table-column>
42          <el-table-column prop="realName" label="姓名"></el-table-column>
43          <el-table-column prop="departmentName" label="所属部门"></el-table-
    column>
44          <el-table-column prop="phone" label="电话"></el-table-column>
45          <el-table-column prop="email" label="邮箱"></el-table-column>
46          <el-table-column align="center" width="290" label="操作">
47            <template slot-scope="scope">
48              <el-button icon="el-icon-edit" type="primary"
49                         size="mini" @click="handleEdit(scope.row)" >编辑</el-
    button>
50              <el-button icon="el-icon-delete" type="danger"
51                         size="mini" @click="handleDelete(scope.row)">删除</el-
    button>
52              <el-button icon="el-icon-setting" type="primary"
53                         size="mini" @click="assignRole(scope.row)">分配角色</el-
    button>
54            </template>
55          </el-table-column>
56        </el-table>
57        <!-- 分页工具栏 -->
58        <el-pagination
59          background
60          @size-change="handleSizeChange"
61          @current-change="handleCurrentChange"
62          :current-page="pageNo"
```

```
          :page-sizes="[10, 20, 30, 40, 50]"
          :page-size="10"
          layout="total, sizes, prev, pager, next, jumper"
          :total="total"
        >
        </el-pagination>
      </el-main>
    </el-container>
</template>

<script>
//导入部门api脚本
import departmentApi from '@/api/department';
//导入用户api脚本
import userApi from '@/api/user';

export default {
  name: 'UserList',
  data() {
    return {
      //查询条件对象
      searchModel: {
        username: "",
        realName:"",
        phone: "",
        departmentId: "",
        pageNo: 1,
        pageSize: 10,
      },
      userList: [], //用户列表
      tableHeight: 0, //表格高度
      pageNo: 1, //当前页码
      pageSize: 10, //每页显示数量
      total: 0, //总数量
      departmentId: "", //部门编号
    }
  },
  methods: {
    /**
     * 查询部门列表
     */
    async getDeptList() {
      //发送查询请求
      let res = await departmentApi.getDepartmentList(null)
      //判断是否成功
      if (res.success) {
        this.deptList = res.data;
        //树加载完成后，默认点击第一个节点
        this.$nextTick(() => {
          const firstNode = document.querySelector(".el-tree-node");
          firstNode.click();
        });
      }
    },
    /**
     * 左侧树节点点击事件
     */
    handleNodeClick(data) {
```

```
      //给部门编号赋值
      this.departmentId = data.id;
      //查询数据
      this.search(this.departmentId);
    },
    /**
     * 查询用户列表
     */
    async search(departmentId, pageNo = 1, pageSize = 10) {
      this.searchModel.pageNo = pageNo;
      this.searchModel.pageSize = pageSize;
      this.searchModel.departmentId = departmentId;
      //发送查询请求
      let res = await userApi.getUserList(this.searchModel);
      if (res.success) {
        this.userList = res.data.records;
        this.total = res.data.total;
      }
    },
    /**
     * 当每页数量发生变化时触发该事件
     */
    handleSizeChange(size) {
      this.pageSize = size; //将每页显示的数量交给成员变量
      this.search(this.departmentId, this.pageNo, size);
    },
    /**
     * 当页码发生变化时触发该事件
     */
    handleCurrentChange(page) {
      this.pageNo = page;
      //调用查询方法
      this.search(this.departmentId, page, this.pageSize);
    },
    /**
     * 重置查询条件
     */
    resetValue() {
      //清空查询条件
      this.searchModel = {};
      //重新查询
      this.search(this.departmentId);
    },
  },
  created() {
    //查询部门列表
    this.getDeptList();
    //调用查询用户列表
    this.search(this.departmentId);
  },
  mounted() {
    this.$nextTick(() => {
      //内容高度
      this.containerHeight = window.innerHeight - 85;
      //表格高度
      this.tableHeight = window.innerHeight - 220;
    })
  }
```

```
179    }
180    </script>
```

11.2 新增用户

11.2.1 效果图

11.2.2 新增用户后端接口实现

在UserController中编写新增用户的请求方法。

```
1    @Resource
2    private PasswordEncoder passwordEncoder;
3
4    /**
5     * 添加用户
6     *
7     * @param user
8     * @return
9     */
10   @PostMapping("/add")
11   public Result add(@RequestBody User user) {
12       //查询用户
13       User item = userService.findUserByUserName(user.getUsername());
14       //判断对象是否为空
15       if (item != null) {
```

```
            return Result.error().message("该登录名称已被使用，请重新输入！");
        }
        //密码加密
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        //调用保存用户信息的方法
        if(userService.save(user)){
            return Result.ok().message("用户添加成功");
        }
        return Result.error().message("用户添加失败");
    }
```

11.2.3 前端页面实现

11.2.3.1 编写新增用户窗口代码

在userList.vue页面组件中添加新增用户窗口代码。

```
<!-- 添加和编辑用户窗口 -->
<system-dialog
  :title="userDialog.title"
  :height="userDialog.height"
  :width="userDialog.width"
  :visible="userDialog.visible"
  @onClose="onClose"
  @onConfirm="onConfirm"
>
  <div slot="content">
    <el-form
      :model="user"
      ref="userForm"
      :rules="rules"
      label-width="80px"
      :inline="true"
      size="small"
    >
      <el-form-item prop="username" label="用户名">
        <el-input v-model="user.username"></el-input>
      </el-form-item>
      <el-form-item prop="password" v-if="user.id === ''" label="密码">
        <el-input type="password" v-model="user.password"></el-input>
      </el-form-item>
      <el-form-item prop="departmentName" label="所属部门">
        <el-input
          v-model="user.departmentName"
          :readonly="true"
          @click.native="selectDepartment()"
        ></el-input>
      </el-form-item>
      <el-form-item prop="realName" label="姓名">
        <el-input v-model="user.realName"></el-input>
      </el-form-item>
      <el-form-item prop="phone" label="电话">
        <el-input v-model="user.phone"></el-input>
      </el-form-item>
      <el-form-item label="昵称">
        <el-input v-model="user.nickName"></el-input>
```

```
        </el-form-item>
        <el-form-item label="邮箱">
          <el-input v-model="user.email"></el-input>
        </el-form-item>
        <el-form-item prop="gender" label="性别">
          <el-radio-group v-model="user.gender">
            <el-radio :label="0">男</el-radio>
            <el-radio :label="1">女</el-radio>
          </el-radio-group>
        </el-form-item>
        <br>
        <!-- 用户头像：待补充 -->
      </el-form>
    </div>
  </system-dialog>
```

11.2.3.2 编写新增用户脚本代码

在userList.vue组件脚本中添加新增用户所需要的属性和方法。

```
//导入对话框组件
import SystemDialog from '@/components/system/SystemDialog.vue'

export default {
  name: 'UserList',
  components: {
    SystemDialog
  },
  data() {
    //自定义验证规则
    let validatePhone = (rule, value, callback) => {
      if (!value) {
        callback(new Error("请输入手机号码"));
      //使用正则表达式进行验证手机号码
      }else if (!/^1[3456789]\d{9}$/.test(value)) {
        callback(new Error("手机号格式不正确"));
      }else{
        callback();
      }
    };
    return {
      //添加和修改用户窗口属性
      userDialog: {
        title: '',
        height: 410,
        width: 610,
        visible: false
      },
      //用户对象
      user: {
        id: '',
        departmentId: '',
        departmentName: '',
        email: '',
        realName: '',
        phone: '',
        nickName: '',
        password: '',
```

```
          username: '',
          gender: '',
          avatar:''
        },
        rules: {
          departmentName: [{ required: true, trigger: 'change', message: '请选择所属部门' }],
          realName: [{ required: true, trigger: 'blur', message: '请填写姓名' }],
          phone: [{ trigger: 'blur', validator: validatePhone }],
          username: [{ required: true, trigger: 'blur', message: '请填写登录名' }],
          password: [{ required: true, trigger: 'blur', message: '请填写登录密码' }],
          gender: [{ required: true, trigger: 'change', message: '请选择性别' }]
        },
      }
    },
    methods: {
      /**
       * 打开添加窗口
       */
      openAddWindow() {
        this.$resetForm('userForm', this.user) //清空表单
        this.userDialog.visible = true //显示窗口
        this.userDialog.title = '新增用户' //设置标题
      },
      /**
       * 新增或编辑取消事件
       */
      onClose() {
        this.userDialog.visible = false //关闭窗口
      },
      /**
       * 新增或编辑确认事件
       */
      onConfirm() {
        //表单验证
        this.$refs.userForm.validate((valid) => {
          if (valid) {

          }
        })
      },
      /**
       * 打开选择所属部门窗口
       */
      selectDepartment() {

      },
    }
  }
  </script>
```

11.2.3.3 编写选择所属部门窗口代码

在userList.vue页面组件中添加选择所属部门窗口代码。

```
<!-- 所属部门弹框 -->
<system-dialog
  :title="parentDialog.title"
```

```
    :width="parentDialog.width"
    :height="parentDialog.height"
    :visible="parentDialog.visible"
    @onClose="onParentClose"
    @onConfirm="onParentConfirm"
  >
    <div slot="content">
      <el-tree
        ref="parentTree"
        :data="parentList"
        default-expand-all
        node-key="id"
        :props="parentProps"
        :show-checkbox="false"
        :highlight-current="true"
        :expand-on-click-node="false"
        @node-click="parentClick"
      >
        <div class="customer-tree-node" slot-scope="{ node, data }">
          <span v-if="data.children.length == 0">
            <i class="el-icon-document"/>
          </span>
          <span v-else @click.stop="openParentBtn(data)">
            <svg-icon v-if="data.open"icon-class="add-s" />
            <svg-icon v-else icon-class="sub-s" />
          </span>
          <span style="margin-left: 3px">{{ node.label }}</span>
        </div>
      </el-tree>
    </div>
  </system-dialog>
```

11.2.3.4 编写选择所属部门脚本代码

在userList.vue组件脚本中添加选择属性部门所需要的属性和方法。

```
export default {
  name: 'userList',
  components: {
    SystemDialog
  },
  data() {
    return {
      //选择所属部门窗口属性
      parentDialog: {
        title: '选择所属部门',
        width: 300,
        height: 450,
        visible: false
      },
      //树节点属性
      parentProps: {
        children: 'children',
        label: 'departmentName'
      },
      parentList: [], //所属部门节点数据
```

```
      }
    },
    methods: {
      /**
       * 打开选择上级部门窗口
       */
      async selectDepartment() {
        //查询上级部门数据
        let res = await departmentApi.getDepartmentList(null)
        //判断是否成功
        if (res.success) {
          this.parentList = res.data
        }
        //显示窗口
        this.parentDialog.visible = true
      },
      /**
       * 选择上级部门取消事件
       */
      onParentClose() {
        this.parentDialog.visible = false //关闭窗口
      },
      /**
       * 选择上级部门确认事件
       */
      onParentConfirm() {
        this.parentDialog.visible = false
      },
      //上级部门树节点点击事件
      parentClick(data) {
        this.user.departmentId = data.id
        this.user.departmentName = data.departmentName
      },
      //上级部门树展号 减号 图标点击事件
      openParentBtn(data) {
        data.open = !data.open
        this.$refs.parentTree.store.nodesMap[data.id].expanded = !data.open
      },
    }
  }
  </script>
```

11.2.3.5 编写前端api脚本代码

在src/api/user.js文件中编写添加用户的方法。

```
/**
 * 添加用户
 * @param params
 * @returns
 */
async addUser(params){
  return await http.post("/api/user/add",params);
}
```

11.2.3.6 编写窗口确认事件

```
/**
 * 新增或编辑确认事件
 */
onConfirm() {
  this.$refs.userForm.validate(async(valid) => {
    if (valid) {
      let res = null
      //判断用户ID是否为空
      if (this.user.id === '') {
        //新增
        //发送添加请求
        res = await userApi.addUser(this.user)
      } else {
        //发送修改请求

      }
      //判断是否成功
      if (res.success) {
        this.$message.success(res.message)
        //刷新
        this.search(this.departmentId, this.pageNo, this.pageSize);
        //关闭窗口
        this.userDialog.visible = false
      } else {
        this.$message.error(res.message)
      }
    }
  })
}
```


11.3 阿里云OSS对象存储服务

11.3.1 开通阿里云对象存储OSS服务

- 第1步：申请/注册阿里云账号
- 第2步：实名认证
- 第3步：开通"对象存储OSS"服务

11.3.1.1 申请阿里云账号

去到阿里云官方网站进行注册即可，如下图所示：

11.3.1.2 实名认证

使用上面注册的阿里云账号进行登录，登录成功后去个人资料处进行实名认证，自行实现实名认证

11.3.1.3 开通阿里云OSS对象存储服务

（1）在阿里云首页找到 产品 ，在产品下拉菜单中找到 对象存储OSS

（2）在OSS页面中点击 立即开通

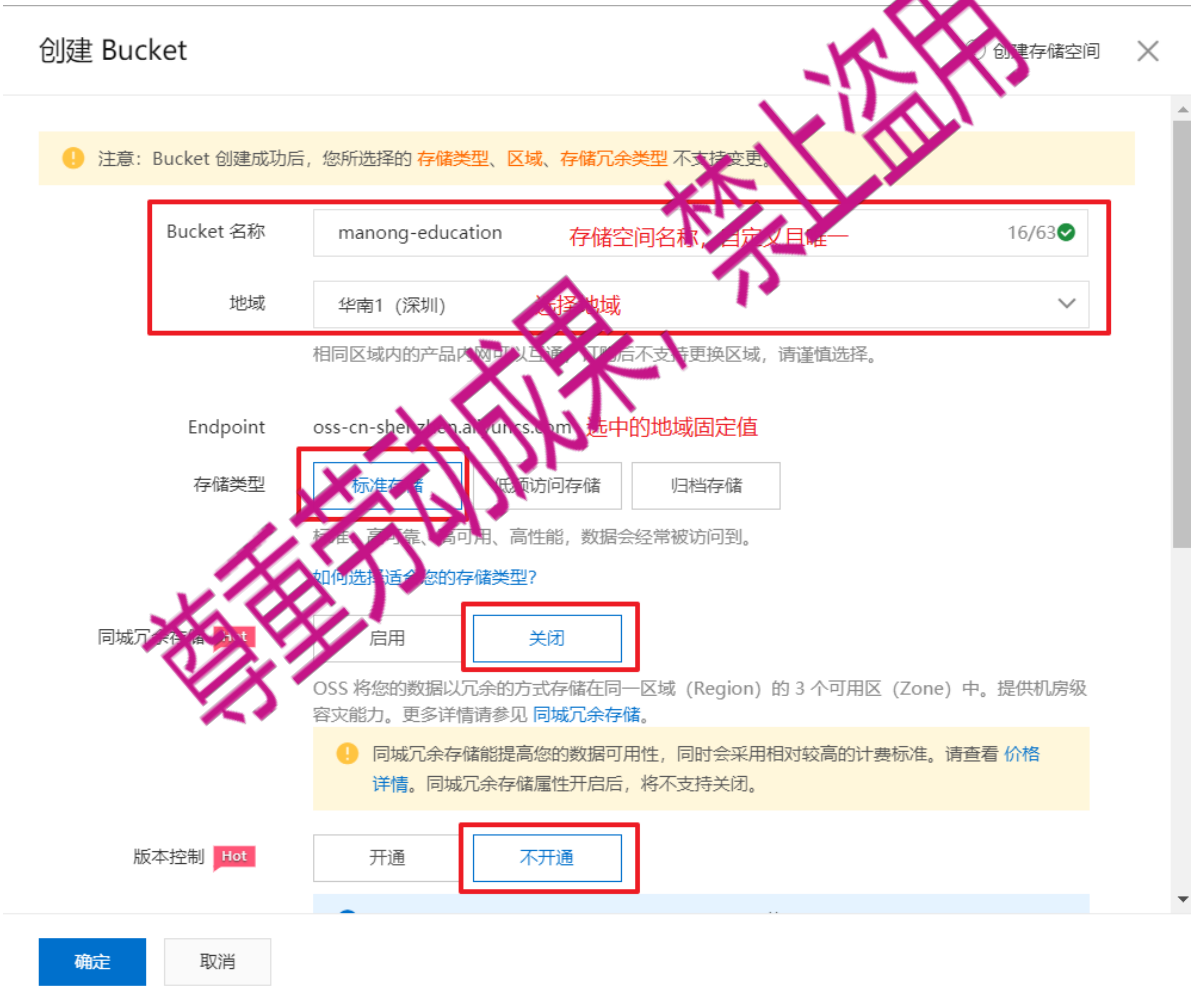
11.3.2 使用阿里云OSS对象存储

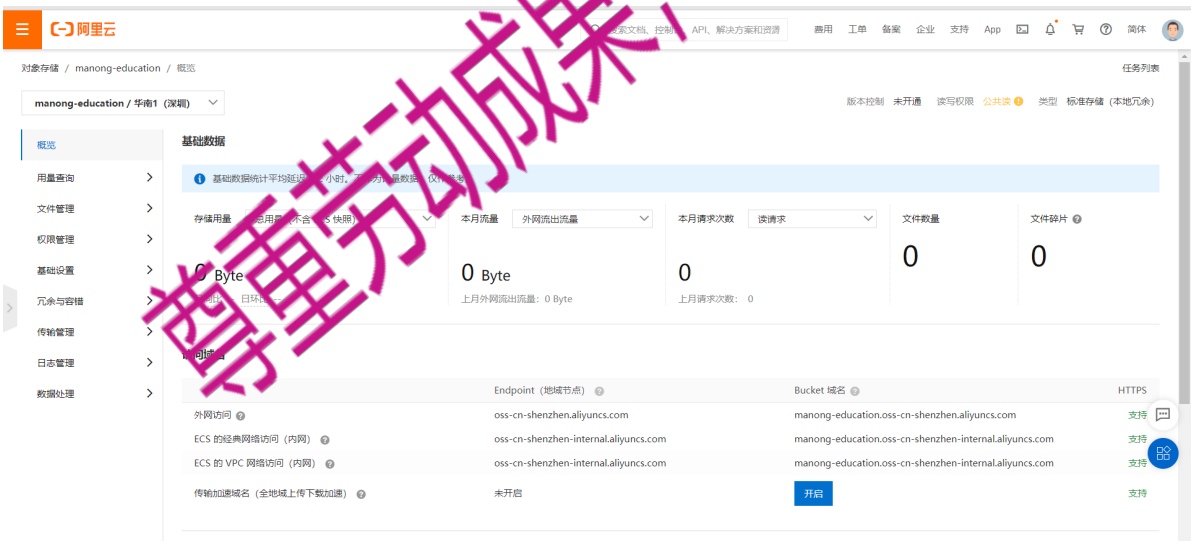
11.3.2.1 创建Bucket存储空间

（1）进入OSS控制台，点击Bucket列表

（2）创建Bucket

注意：地域请根据自己所在地进行选择。



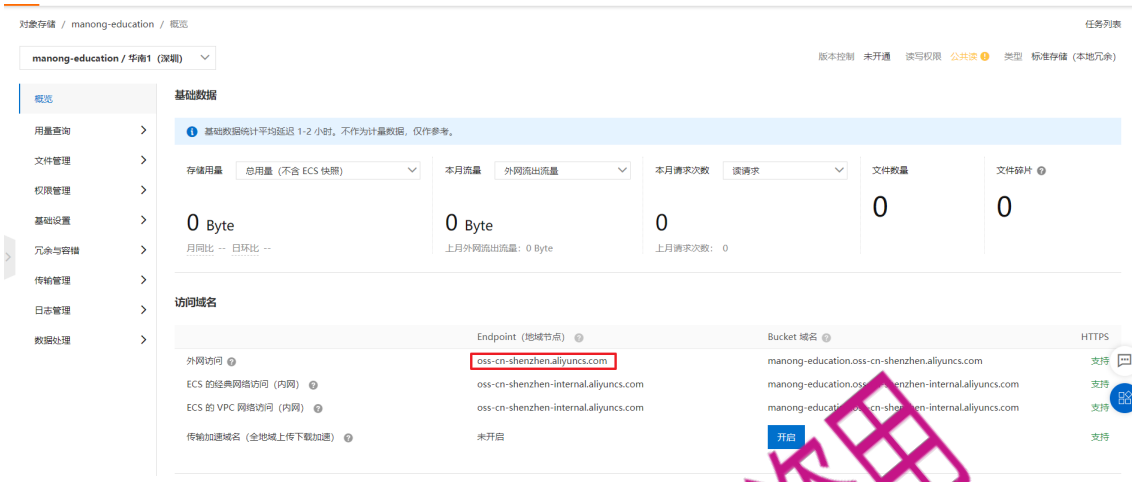


11.3.2.2 文件管理

11.3.2.3 使用Java SDK操作OSS

使用Java SDK操作OSS对象存储服务，需要明确4个固定值，分别是EndPoint(地域节点)、AccessKeyId、AccessKeySecret 、 BucketName。

- EndPoint地域节点：必须使用当前Bucket所在的地域

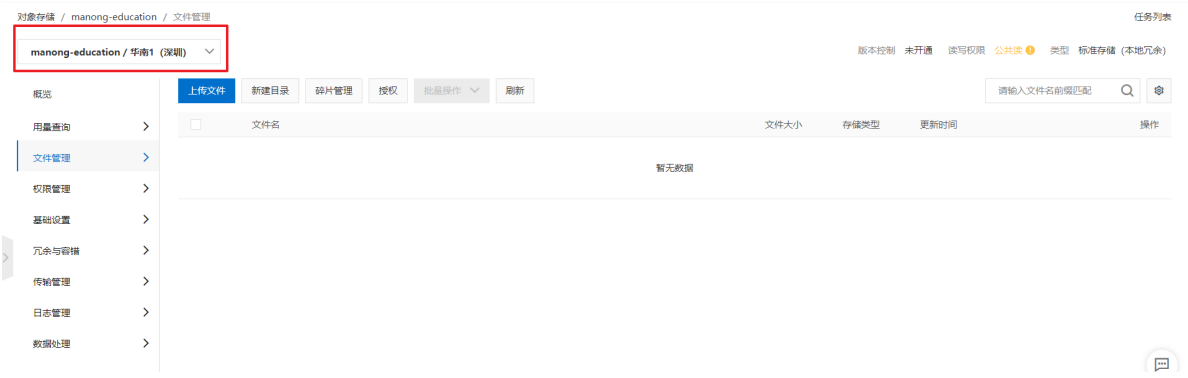


- AccessKeyId：账号ID
- AccessKeySecret ： 账号密码

注意：AccessKeyId和AccessKeySecret 不是阿里云的账号密码，是由下图创建出来的ID和密码。

点击 创建AccessKey 按钮后，会下载一份AccessKeyID和AccessKeySecret密码的文件，该秘钥不能轻易暴露，否则会造成安全隐患。

- BucketName：存储空间的名称



11.4 使用OSS实现文件上传

11.4.1 添加OSS依赖

在pom.xml文件中添加阿里云OSS依赖。

```
<!-- 阿里云OSS文件上传开始 -->
<!-- 阿里云 OSS -->
<dependency>
    <groupId>com.aliyun.oss</groupId>
    <artifactId>aliyun-sdk-oss</artifactId>
    <version>3.14.1</version>
</dependency>

<!--日期时间工具-->
<dependency>
    <groupId>joda-time</groupId>
    <artifactId>joda-time</artifactId>
    <version>2.10.14</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
</dependency>
<!-- 阿里云OSS文件上传结束 -->
```


11.4.2 编写application.properties配置文件

```
#阿里云OSS配置
aliyun.oss.file.endpoint=填写自己的地域节点
aliyun.oss.file.keyid=填写自己的keyId
aliyun.oss.file.keysecret=填写自己的AccessKeySecuret
#bucket名称
aliyun.oss.file.bucketname=填写自己的Bucket名称
```

11.4.3 读取配置文件信息

在com.manong.config.oss包下创建OssProperties属性配置类。

```
package com.manong.config.oss;

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
```

```
import org.springframework.stereotype.Component;

@Data
@Component
@ConfigurationProperties(prefix = "aliyun.oss.file")
public class OssProperties {
    private String endPoint;
    private String keyId;
    private String keySecret;
    private String bucketName;
}
```

11.4.4 编写文件上传业务代码

11.4.4.1 编写FileService接口

```
package com.manong.service;

import org.springframework.web.multipart.MultipartFile;

public interface FileService {

    /**
     * 文件上传
     * @param file      文件上传对象
     * @param module    文件夹名称
     * @return
     */
    String upload(MultipartFile file, String module);

    /**
     * 删除文件
     * @param url
     */
    void deleteFile(String url);

}
```

11.4.4.2 编写FileServiceImpl实现类

```
package com.manong.service.impl;

import com.aliyun.oss.OSS;
import com.aliyun.oss.OSSClientBuilder;
import com.manong.config.oss.OssProperties;
import com.manong.service.FileService;
import org.joda.time.DateTime;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.multipart.MultipartFile;

import javax.annotation.Resource;
import java.io.IOException;
import java.io.InputStream;
import java.util.UUID;

@Service
@Transactional
```

```java
public class FileServiceImpl implements FileService {
    /**
     * 文件上传
     *
     * @param file   文件上传对象
     * @param module 文件夹名称
     * @return
     */
    @Resource
    private OssProperties ossProperties;

    /**
     * 文件上传
     *
     * @param file   文件上传对象
     * @param module 文件夹名称
     * @return
     */
    @Override
    public String upload(MultipartFile file, String module) {
        //获取地域节点
        String endPoint = ossProperties.getEndPoint();
        //获取AccessKeyId
        String keyId = ossProperties.getKeyId();
        //获取AccessKeySecret
        String keySecret = ossProperties.getKeySecret();
        //获取BucketName
        String bucketName = ossProperties.getBucketName();

        try {
            //创建OSSClient实例
            OSS ossClient = new OSSClientBuilder().build(endPoint, keyId,
keySecret);
            //上传文件流
            InputStream inputStream = file.getInputStream();
            //获取旧名称
            String originalFilename = file.getOriginalFilename();
            //获取文件后缀名
            String extension = FilenameUtils.getExtension(originalFilename);
            //将文件名重命名
            String newFileName = UUID.randomUUID().toString()
                .replace("-", "")+"."+extension;
            //使用当前日期进行分类管理
            String datePath = new DateTime().toString("yyyy/MM/dd");
            //构建文件名
            newFileName = module + "/" + datePath + "/" + newFileName;
            //调用OSS文件上传的方法
            ossClient.putObject(bucketName, newFileName, inputStream);
            //关闭OSSClient
            ossClient.shutdown();
            //返回文件地址
            return "https://"+bucketName+"."+endPoint+"/"+newFileName;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

```

```
    @Override
    public void deleteFile(String url) {
        //获取地域节点
        String endPoint = ossProperties.getEndPoint();
        //获取AccessKeyId
        String keyId = ossProperties.getKeyId();
        //获取AccessKeySecret
        String keySecret = ossProperties.getKeySecret();
        //获取BucketName
        String bucketName = ossProperties.getBucketName();
        try {
            //创建OSSClient实例
            OSS ossClient = new OSSClientBuilder().build(endPoint, keyId,
keySecret);
            //组装文件地址
            String host = "https://"+bucketName+"."+endPoint+"/";
            //获取文件名称
            String objectName = url.substring(host.length());
            //删除文件
            ossClient.deleteObject(bucketName,objectName);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

11.4.5 编写文件上传控制器类

在com.manong.controller包下创建OSSController控制器类。

```
package com.manong.controller;

import com.manong.service.FileService;
import com.manong.utils.Result;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

import javax.annotation.Resource;

@RestController
@RequestMapping("/api/oss/file")
public class OSSController {

    @Resource
    private FileService fileService;

    /**
     * 文件上传
     * @param file
     * @param module
     * @return
     */
    @PostMapping("/upload")
    public Result upload(MultipartFile file, String module){
        //返回上传到oss的路径
        String url = fileService.upload(file,module);
```

```
            return Result.ok(url).message("文件上传成功");
        }

    }
```

11.4.6 上传用户头像

11.4.6.1 定义页面组件模板

（1）编写页面代码

```
<!-- 用户头像 -->
<el-form-item label="头像">
  <el-upload
    :show-file-list="false"
    :on-success="handleAvatarSuccess"
    :before-upload="beforeAvatarUpload"
    class="avatar-uploader"
    :data="uploadHeader"
    action="http://localhost:9999/api/oss/file/upload?module=avatar"
  >
    <img v-if="user.avatar" :src="user.avatar">
    <i v-else class="el-icon-plus avatar-uploader-icon"/>
  </el-upload>
</el-form-item>
```

（2）使用CSS美化文件上传组件

注意：<style>标签需要去掉scoped属性。

```
<style lang="scss">
//用户头像
.avatar-uploader .el-upload {
  border: 1px dashed #d9d9d9 !important;
  border-radius: 6px;
  cursor: pointer;
  position: relative;
  overflow: hidden;
}

.avatar-uploader .el-upload:hover {
  border-color: #409EFF;
}

.avatar-uploader .avatar-uploader-icon {
  font-size: 28px;
  color: #8c939d;
  width: 178px;
  height: 178px;
  line-height: 178px;
  text-align: center;
}

.avatar-uploader img {
  width: 178px;
  height: 178px;
  display: block;
```

```
28  }
29  </style>
```

11.4.6.2 定义页面组件脚本代码

(1) 导入token信息

```
1   //导入token
2   import {getToken} from '@/utils/auth'
```

(2) 定义上传需要携带的token数据

```
1   export default {
2     name: 'UserList',
3     components: {
4       SystemDialog
5     },
6     data() {
7       return {
8         //上传需要携带的数据
9         uploadHeader:{"token":getToken()}
10      }
11    },
```

(3) 上传的事件方法

```
1   /**
2    * 上传成功回调
3    * @param res
4    * @param file
5    */
6   handleAvatarSuccess(res, file) {
7     this.user.avatar = res.data
8     // 强制重新渲染
9     this.$forceUpdate()
10  },
11  /**
12   * 上传之前
13   * @param file
14   * @returns {boolean}
15   */
16  beforeAvatarUpload(file) {
17    const isJPG = file.type === 'image/jpeg'
18    const isLt2M = file.size / 1024 / 1024 < 2
19    if (!isJPG) {
20      this.$message.error('上传头像图片只能是 JPG 格式!')
21    }
22    if (!isLt2M) {
23      this.$message.error('上传头像图片大小不能超过 2MB!')
24    }
25    return isJPG && isLt2M
26  }
```

11.5 编辑用户

11.5.1 编辑用户后端接口实现

在UserController中添加编辑用户的方法。

```
/**
 * 修改用户
 *
 * @param user
 * @return
 */
@PutMapping("/update")
public Result update(@RequestBody User user) {
    //查询用户
    User item = userService.findUserByUserName(user.getUsername());
    //判断对象是否为空,且查询到的用户ID不等于当前编辑的用户ID，表示该名称被占用
    if (item != null && item.getId() != user.getId()) {
        return Result.error().message("登录名称已被占用！");
    }
    //调用修改用户信息的方法
    if(userService.updateById(user)){
        return Result.ok().message("用户修改成功");
    }
    return Result.error().message("用户修改失败");
}
```

11.5.2 编辑用户前端实现

11.5.2.1 效果图

11.5.2.2 编写前端api脚本代码

在src/api/user.js文件中编写修改用户的方法。

```
/**
 * 编辑用户
 * @param params
 * @returns
 */
async updateUser(params){
  return await http.put("/api/user/update",params);
}
```

11.5.2.3 编写修改用户脚本代码

```
/**
 * 编辑用户
 */
handleEdit(row) {
  //设置弹框属性
  this.userDialog.title = '编辑用户'
  this.userDialog.visible = true
  //把当前编辑的数据复制到表单数据域，供回显使用
  this.$objCopy(row, this.user)
},
/**
 * 新增或编辑确认事件
 */
```

```
onConfirm() {
  this.$refs.userForm.validate(async(valid) => {
    if (valid) {
      let res = null
      //判断用户ID是否为空
      if (this.user.id === '') {
        //新增
        //发送添加请求
        res = await userApi.addUser(this.user)
      } else {
        //发送修改请求
        res = await userApi.updateUser(this.user)
      }
      //判断是否成功
      if (res.success) {
        this.$message.success(res.message)
        //刷新
        this.search(this.departmentId, this.pageNo, this.pageSize);
        //关闭窗口
        this.userDialog.visible = false
      } else {
        this.$message.error(res.message)
      }
    }
  })
}
```

11.6 删除用户

11.6.1 删除用户后端接口实现

11.6.1.1 UserMapper

```
public interface UserMapper extends BaseMapper<User> {

    /**
     * 删除用户角色关系
     * @param userId
     * @return
     */
    @Delete("delete from sys_user_role where user_id=#{userId}")
    int deleteUserRole(Long userId);
}
```

11.6.1.2 SystemConstants

```
package com.manong.utils;

public class SystemConstants {

    /**
     * 默认头像
     */
    public static final String DEFAULT_AVATAR =
        "https://manong-authority.oss-cn-guangzhou.aliyuncs.com/avatar/male.jpg";

}
```

1.6.1.3 UserService

```
/**
 * 删除用户信息
 * @param id
 * @return
 */
boolean deleteById(Long id);
```

11.6.1.4 UserServiceImpl

```
@Override
@Transactional(rollbackFor = RuntimeException.class)
public boolean deleteById(Long id) {
    //查询
    User user = baseMapper.selectById(id);
    //删除用户角色关系
    baseMapper.deleteUserRole(id);
    //删除用户
    if (baseMapper.deleteById(id) > 0) {
        //判断用户是否存在
        if (user != null && !ObjectUtils.isEmpty(user.getAvatar())
            && !user.getAvatar().equals(SystemConstants.DEFAULT_AVATAR)) {
            //删除文件
            fileService.deleteFile(user.getAvatar());
        }
        return true;
    }
    return false;
}
```

11.6.1.5 UserController

```
/**
 * 删除用户
 * @param id
 * @return
 */
@DeleteMapping("/delete/{id}")
public Result delete(@PathVariable Long id) {
    //调用删除用户信息的方法
    if(userService.deleteById(id)){
        return Result.ok().message("用户删除成功");
    }
    return Result.error().message("用户删除失败");
}
```

11.6.2 删除用户前端实现

11.6.2.1 编写前端api脚本

在src/api/user.js文件中编写删除用户的方法。

```
/**
 * 删除用户
 * @param params
 * @returns
 */
async deleteUser(params){
  return await http.delete("/api/user/delete",params);
}
```

11.6.2.2 编写删除确认事件

```
/**
 * 删除
 */
async handleDelete(row) {
  let confirm = await this.$myconfirm('确定要删除该数据吗?')
  if (confirm) {
    //封装条件
    let params = { id: row.id }
    //发送删除请求
    let res = await userApi.deleteUser(params)
    //判断是否成功
    if (res.success) {
      this.$message.success(res.message)
      //刷新
      this.search(this.departmentId, this.pageNo, this.pageSize);
    } else {
      this.$message.error(res.message)
    }
  }
}
```

11.7 分配角色

11.7.1 分配角色思路

1. 根据登录用户id查询出该用户的所有角色信息。
2. 点击要分配角色的用户，进行数据回显，即需要查询出被分配角色用户原有的角色信息。
3. 保存选中的角色，在保存前删除用户角色关系数据。

11.7.2 分配角色回显接口实现

11.7.2.1 RoleMapper

```
/**
 * 根据用户ID查询该用户拥有的角色ID
 * @param userId
 * @return
 */
@Select("select role_id from `sys_user_role` where user_id = #{userId}")
List<Long> findRoleIdByUserId(Long userId);
```

11.7.2.2 RoleService

```
/**
 * 根据用户ID查询该用户拥有的角色ID
 * @param userId
 * @return
 */
List<Long> findRoleIdByUserId(Long userId);
```

11.7.2.3 RoleServiceImpl

```
/**
 * 根据用户ID查询该用户拥有的角色ID
 *
 * @param userId
 * @return
 */
@Override
public List<Long> findRoleIdByUserId(Long userId) {
    return baseMapper.findRoleIdByUserId(userId);
}
```

11.7.2.4 UserController

```
/**
 * 获取分配角色列表
 * @param roleQueryVo
 * @return
 */
@GetMapping("/getRoleListForAssign")
public Result getRoleListForAssign(RoleQueryVo roleQueryVo){
    //创建分页对象
    IPage<Role> page = new Page<Role>(roleQueryVo.getPageNo(),
roleQueryVo.getPageSize());
    //调用查询方法
    roleService.findRoleListByUserId(page,roleQueryVo);
    //返回数据
    return Result.ok(page);
}

/**
 * 根据用户ID查询该用户拥有的角色列表
 * @param userId
 * @return
 */
@GetMapping("/getRoleByUserId/{userId}")
public Result getRoleByUserId(@PathVariable Long userId){
    //调用根据用户ID查询该用户拥有的角色ID的方法
    List<Long> roleIds = roleService.findRoleIdByUserId(userId);
    return Result.ok(roleIds);
}
```

11.7.3 分配角色回显前端实现

11.7.3.1 编写前端api脚本

在src/api/user.js文件中编写方法。

```
/**
 * 查询用户角色列表
 * @param params
 * @returns
 */
async getAssignRoleList(params){
  return await http.get("/api/user/getRoleListForAssign",params);
},
/**
 * 获取分配角色列表数据
 * @param params
 * @returns
 */
async getRoleIdByUserId(params){
  return await http.getRestApi("/api/user/getRoleByUserId",params);
}
```

11.7.3.2 编写分配角色窗口代码

```
<!-- 分配角色窗口 -->
<system-dialog
  :title="assignDialog.title"
  :height="assignDialog.height"
  :width="assignDialog.width"
  :visible="assignDialog.visible"
  @onClose="onAssignClose"
  @onConfirm="onAssignConfirm"
>
  <div slot="content">
    <!-- 分配角色数据列表 -->
    <el-table
      ref="assignRoleTable"
      :data="assignRoleList"
      border
      stripe
      :height="assignHeight"
      style="width: 100%; margin-bottom: 10px"
      @selection-change="handleSelectionChange"
    >
      <el-table-column
        type="selection"
        width="55"
        align="center"
      ></el-table-column>
      <el-table-column prop="roleCode" label="角色编码"/>
      <el-table-column prop="roleName" label="角色名称"/>
      <el-table-column prop="remark" label="角色备注"/>
    </el-table>
    <!-- 分页工具栏 -->
    <el-pagination
      @size-change="assignSizeChange"
      @current-change="assignCurrentChange"
```

```
        :current-page.sync="roleVo.pageNo"
        :page-sizes="[10, 20, 30, 40, 50]"
        :page-size="roleVo.pageSize"
        layout="total, sizes, prev, pager, next, jumper"
        :total="roleVo.total"
        background
      >
      </el-pagination>
    </div>
  </system-dialog>
```

11.7.3.3 编写分配角色回显脚本代码

```
export default {
  name: 'UserList',
  components: {
    SystemDialog
  },
  data() {
    return {
      //分配角色窗口属性
      assignDialog: {
        title: "",
        visible: false,
        width: 800,
        height: 410,
      },
      //角色对象
      roleVo: {
        pageNo: 1,
        pageSize: 10,
        userId: "",
        total: 0
      },
      assignRoleList: [], //角色列表
      assignHeight: 0, //分配角色表格高度
      selectedIds: [], //被选中的角色id
      selectedUserId: "", //被分配角色的用户ID
    }
  },
  methods: {
    /**
     * 打开分配角色
     */
    async assignRole(row){
      //防止回显出现问题
      this.selectedIds = [];
      this.selectedUserId = "";
      //被分配用户的id
      this.selectedUserId = row.id;
      //显示窗口
      this.assignDialog.visible = true;
      //设置标题
      this.assignDialog.title = `给【${row.realName}】分配角色`;
      //查询当前登录用户的所有角色信息
      await this.getAssignRoleList();
      //获取当前被分配用户的ID
      let params = {
```

```
          userId: row.id,
        };
        //发送根据用户ID查询角色列表的请求
        let res = await userApi.getRoleIdByUserId(params);
        //如果存在数据
        if (res.success && res.data){
          //将查询到的角色ID列表交给选中角色数组
          this.selectedIds = res.data;
          //循环遍历
          this.selectedIds.forEach((key) => {
            this.assignRoleList.forEach((item) => {
              if (item.id === key) {
                this.$refs.assignRoleTable.toggleRowSelection(item, true);
              }
            });
          });
        }
      },
      /**
       * 查询当前登录用户的所有角色信息
       */
      async getAssignRoleList(pageNo = 1, pageSize = 10) {
        //给用户ID赋值
        this.roleVo.userId = this.$store.getters.userId;
        this.roleVo.pageNo = pageNo;
        this.roleVo.pageSize = pageSize;
        //发送查询请求
        let res = await userApi.getAssignRoleList(this.roleVo);
        //判断是否成功
        if (res.success) {
          //角色列表赋值
          this.assignRoleList = res.data.records;
          //角色总数量赋值
          this.roleVo.total = res.data.total;
        }
      },
      /**
       * 分配角色取消事件
       */
      onAssignClose(){
        //隐藏窗口
        this.assignDialog.visible = false;
      },
      /**
       * 分配角色确认事件
       */
      onAssignConfirm(){
        //隐藏窗口
        this.assignDialog.visible = false;
      },
      /**
       * 当每页数量发生变化时触发该事件
       */
      assignSizeChange(size){
        this.roleVo.pageSize = size; //将每页显示的数量交给成员变量
        this.getAssignRoleList(this.roleVo.pageNo, size);
      },
      /**
```

```
104         * 当页码发生变化时触发该事件
105         */
106       assignCurrentChange(page){
107         this.roleVo.pageNo = page;
108         //调用查询方法
109         this.getAssignRoleList(page, this.roleVo.pageSize);
110       },
111       /**
112        * 当点击多选框时触发该事件
113        */
114       handleSelectionChange(rows){
115         let roleIds = [];
116         //循环遍历选中的角色ID
117         for (let i = 0; i < rows.length; i++) {
118           //将当前选中的角色ID放到数组中
119           roleIds.push(rows[i].id);
120         }
121         //将选中的角色ID交给成员变量
122         this.selectedIds = roleIds;
123         // this.selectedIds = rows.map(item => item.id);//等同于上述代码
124       }
125     },
126
127     mounted() {
128       this.$nextTick(() => {
129         //内容高度
130         this.containerHeight = window.innerHeight - 85
131         //表格高度
132         this.tableHeight = window.innerHeight - 220
133         //角色表格高度
134         this.assignHeight = window.innerHeight - 350;
135       })
136     }
137   }
```


11.7.4 分配角色接口实现

11.7.4.1 UserRoleDTO

在com.manong.dto包下创建UserRoleDTO类。

```
1   package com.manong.dto;
2
3   import lombok.Data;
4
5   import java.util.List;
6
7   /**
8    * 用于给用户分配角色时保存选中的角色数据
9    */
10  @Data
11  public class UserRoleDTO {
12      private Long userId;
13      private List<Long> roleIds;
14  }
```

11.7.4.2 UserMapper

```
/**
 * 保存用户角色关系
 * @param userId
 * @param roleIds
 * @return
 */
int saveUserRole(Long userId, List<Long> roleIds);
```

11.7.4.3 UserMapper.xml

```
<insert id="saveUserRole">
    insert into sys_user_role(user_id,role_id) values
    <foreach collection="roleIds" item="item" index="index" separator=",">
        (#{userId},#{item})
    </foreach>
</insert>
```

11.7.4.4 UserService

```
/**
 * 分配角色
 * @param userId
 * @param roleIds
 * @return
 */
boolean saveUserRole(Long userId, List<Long> roleIds);
```

11.7.4.5 UserServiceImpl

```
@Override
@Transactional(rollbackFor = RuntimeException.class)
public boolean saveUserRole(Long userId, List<Long> roleIds) {
    //删除该用户对应的角色信息
    baseMapper.deleteUserRole(userId);
    //保存用户角色信息
    return baseMapper.saveUserRole(userId,roleIds)>0;
}
```

11.7.4.6 UserController

```
/**
 * 分配角色
 * @param userRoleDTO
 * @return
 */
@PostMapping("/saveUserRole")
public Result saveUserRole(@RequestBody UserRoleDTO userRoleDTO){
    if (userService.saveUserRole(userRoleDTO.getUserId(),
userRoleDTO.getRoleIds())) {
        return Result.ok().message("角色分配成功");
    }
    return Result.error().message("角色分配失败");
}
```

11.7.5 分配角色前端实现

11.7.5.1 编写前端api脚本

```
/**
 * 分配角色
 */
async assignRoleSave(params){
  return await http.post("/api/user/saveUserRole",params)
}
```

11.7.5.2 编写分配角色窗口确认事件

```
/**
 * 分配角色确认事件
 */
async onAssignConfirm(){
  //判断用户是否有选中角色
  if (this.selectedIds.length === 0) {
    this.$message.warning("请选择要分配的角色！");
    return;
  }
  let params = {
    userId: this.selectedUserId,
    roleIds: this.selectedIds,
  };
  //发送请求
  let res = await userApi.assignRoleSave(params);
  //判断是否成功
  if (res.success) {
    this.$message.success(res.message);
    //关闭窗口
    this.assignDialog.visible = false;
  } else {
    this.$message.error(res.message);
  }
}
```


12.权限处理

12.1 后端接口处理

12.1.1 SpringSecurityConfig

修改SpringSecurityConfig配置类，加入`@EnableGlobalMethodSecurity(prePostEnabled = true)`注解权限控制。

```
@Configuration
@EnableWebSecurity
//开启权限注解控制
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
}
```

12.1.2 在控制器中加入权限控制

以UserController控制器为例，除查询方法外，其余加入`@PreAuthorize`注解控制。

```
@RestController
@RequestMapping("/api/user")
public class UserController {

    @Resource
    private UserService userService;

    @Resource
    private RoleService roleService;

    @Resource
    private PasswordEncoder passwordEncoder;

    /**
     * 查询所有用户列表
     * @return
     */
    @GetMapping("/listAll")
    public Result listAll(){
        return Result.ok(userService.list());
    }

    /**
     * 查询用户列表
     *
     * @param userQueryVo
     * @return
     */
    @GetMapping("/list")
    public Result list(UserQueryVo userQueryVo) {
        //创建分页信息
        Page<User> page = new Page<User>(userQueryVo.getPageNo(),
userQueryVo.getPageSize());
        //调用分页查询方法
        userService.findUserListByPage(page, userQueryVo);
        //返回数据
        return Result.ok(page);
    }

    /**
     * 添加用户
     *
     * @param user
     * @return
     */
    @PostMapping("/add")
    @PreAuthorize("hasAuthority('sys:user:add')")
    public Result add(@RequestBody User user) {
        //查询用户
        User item = userService.findUserByUserName(user.getUsername());
        //判断对象是否为空
        if (item != null) {
            return Result.error().message("该登录名称已被使用，请重新输入！");
        }
```

```
54            //密码加密
55            user.setPassword(passwordEncoder.encode(user.getPassword()));
56            //调用保存用户信息的方法
57            if(userService.save(user)){
58                return Result.ok().message("用户添加成功");
59            }
60            return Result.error().message("用户添加失败");
61        }
62
63        /**
64         * 修改用户
65         *
66         * @param user
67         * @return
68         */
69        @PutMapping("/update")
70        @PreAuthorize("hasAuthority('sys:user:edit')")
71        public Result update(@RequestBody User user) {
72            //查询用户
73            User item = userService.findUserByUserName(user.getUsername());
74            //判断对象是否为空,且查询到的用户ID不等于当前编辑的用户ID,表示该登录名称被占用
75            if (item != null && item.getId() != user.getId()) {
76                return Result.error().message("登录名称已被占用");
77            }
78            //调用修改用户信息的方法
79            if(userService.updateById(user)){
80                return Result.ok().message("用户修改成功");
81            }
82            return Result.error().message("用户修改失败");
83        }
84
85        /**
86         * 删除用户
87         * @param id
88         * @return
89         */
90        @DeleteMapping("/delete/{id}")
91        @PreAuthorize("hasAuthority('sys:user:delete')")
92        public Result delete(@PathVariable Long id) {
93            //调用删除用户信息的方法
94            if(userService.deleteById(id)){
95                return Result.ok().message("用户删除成功");
96            }
97            return Result.error().message("用户删除失败");
98        }
99
100       /**
101        * 获取分配角色列表
102        * @param roleQueryVo
103        * @return
104        */
105       @GetMapping("/getRoleListForAssign")
106       @PreAuthorize("hasAuthority('sys:user:assign')")
107       public Result getRoleListForAssign(RoleQueryVo roleQueryVo){
108           //创建分页对象
109           IPage<Role> page = new Page<Role>(roleQueryVo.getPageNo(),
    roleQueryVo.getPageSize());
110           //调用查询方法
```

```
111             roleService.findRoleListByUserId(page,roleQueryVo);
112             //返回数据
113             return Result.ok(page);
114         }
115 
116         /**
117          * 根据用户ID查询该用户拥有的角色列表
118          * @param userId
119          * @return
120          */
121         @GetMapping("/getRoleByUserId/{userId}")
122         @PreAuthorize("hasAuthority('sys:user:assign')")
123         public Result getRoleByUserId(@PathVariable Long userId){
124             //调用根据用户ID查询该用户拥有的角色ID的方法
125             List<Long> roleIds = roleService.findRoleIdByUserId(userId);
126             return Result.ok(roleIds);
127         }
128 
129         /**
130          * 分配角色
131          * @param userRoleDTO
132          * @return
133          */
134         @PostMapping("/saveUserRole")
135         @PreAuthorize("hasAuthority('sys:user:assign')")
136         public Result saveUserRole(@RequestBody UserRoleDTO userRoleDTO){
137             if (userService.saveUserRole(userRoleDTO.getUserId(),
    userRoleDTO.getRoleIds())) {
138                 return Result.ok().message("角色分配成功");
139             }
140             return Result.error().message("角色分配失败");
141         }
142     }
```

12.2 前端页面按钮权限判断

12.2.1 保存权限字段

在src/store/modules的user.js中的getInfo()方法添加如下代码。

```
1   //将权限字段保存到sessionStorage中
2   sessionStorage.setItem("codeList",JSON.stringify(roles));
```

12.2.2 编写按钮权限判断

在src下新建permission文件夹，然后新建index.js。

```
1   export default function hasPermission(params){
2     let tag = false;//标识是否拥有权限
3     //从sessionStorage中获取codeList权限字段列表
4     let codeList = JSON.parse(sessionStorage.getItem("codeList"));
5     //循环遍历权限字段列表
6     for (let i = 0; i < codeList.length; i++) {
7       //判断当前权限字段是否与参数传递过来的字段一致
8       if(codeList[i] === params){
9         tag = true;
10        break;
```

```
      }
    }
    return tag;
  }
```

12.2.3 引入按钮权限判断脚本

在main.js中添加如下代码。

```
//导入按钮权限判断
import hasPermission from '@/permission/index'
Vue.prototype.hasPermission = hasPermission;
```

12.2.4 按钮权限判断使用方式

以userList.vue举例，在删除按钮上使用权限判断。

```
<el-button
  icon="el-icon-close"
  type="danger"
  size="small"
  @click="handleDelete(scope.row)"
  v-if="hasPermission('sys:user:delete')"
  >删除
</el-button>
```


13.token过期处理

当token过期后，系统会提示`"用户登录信息过期，请重新登录！"`。

13.1 编写store代码

修改src/store/modules/user.js脚本代码

```
import { login, logout, getInfo } from '@/api/user'
// 导入auth脚本
import { getToken, setToken, removeToken,setTokenTime } from '@/utils/auth'
import router, { resetRouter } from '@/router'

const state = {
  token: getToken(),//获取token信息
  name: '',
  avatar: '',
  introduction: '',
  roles: []
}

```

```
const mutations = {
  //设置token信息
  SET_TOKEN: (state, token) => {
    state.token = token
  },
  //设置个人介绍
  SET_INTRODUCTION: (state, introduction) => {
    state.introduction = introduction
  },
  //设置用户姓名
  SET_NAME: (state, name) => {
    state.name = name
  },
  //设置用户头像
  SET_AVATAR: (state, avatar) => {
    state.avatar = avatar
  },
  //设置用户对应的角色
  SET_ROLES: (state, roles) => {
    state.roles = roles
  }
}

const actions = {
  // 用户登录
  login({ commit }, userInfo) {
    //从用户信息userInfo中解构出用户名和密码
    const { username, password } = userInfo
    return new Promise((resolve, reject) => {
      //调用src/api/user.js文件中的login()方法
      login({ username: username.trim(), password: password }).then(response => {
        //从response中解构出返回的token数据
        const { token, expireTime } = response
        //将返回的token数据保存到store中，作为全局变量使用
        commit('SET_TOKEN', token)
        //将token信息保存到cookie中
        setToken(token)
        //设置token过期时间
        setTokenTime(expireTime);
        resolve()
      }).catch(error => {
        reject(error)
      })
    })
  },
}
```


13.2 编写刷新token方法

在src/api/user.js脚本中编写刷新token的方法。

```
/**
 * 刷新token
 * @returns
 */
export async function refreshTokenApi(params){
  return await http.post("/api/sysUser/refreshToken",params);
}
```

13.3 编写token过期方法

修改src/utils下的auth.js代码。

```
import Cookies from 'js-cookie'

const TokenKey = 'Admin-Token'
//定义token过期时间的key
const timeKey = "expireTime";

/**
 * 获取token
 * @returns
 */
export function getToken() {
  return Cookies.get(TokenKey)
}

/**
 * 设置token
 * @returns
 */
export function setToken(token) {
  return Cookies.set(TokenKey, token)
}
/**
 * 删除token
 * @returns
 */
export function removeToken() {
  return Cookies.remove(TokenKey)
}

/**
 * 清空sessionStorage
 */
export function clearStorage(){
  return sessionStorage.clear();
}

/**
 * 设置token过期时间
 * @returns
 */
export function setTokenTime(time){
  return sessionStorage.setItem(timeKey,time);
}

/**
```

```
 * 获取token过期时间
 * @returns
 */
export function getTokenTime(){
 return sessionStorage.getItem(timeKey);
}

/**
 * 清空token过期时间
 * @returns
 */
export function removeTokenTime(){
 return sessionStorage.setItem(timeKey,0);
}
```

13.4 编写请求拦截

修改src/utils下的request.js代码。

```
import axios from 'axios'
import { MessageBox, Message } from 'element-ui'
import store from '@/store'
//导入auth脚本
import {
getToken,setToken,clearStorage,getTokenTime,setTokenTime,removeTokenTime } from
'@/utils/auth'
import qs from 'qs'
//导入刷新token的api脚本
import { refreshTokenApi } from '@/api/user'


// create an axios instance
const service = axios.create({
  baseURL: process.env.VUE_APP_BASE_API, // url = base url + request url
  // withCredentials: true, // send cookies when cross-domain requests
  timeout: 15000 // 请求超时时间
})

/**
 * 刷新token
 */
function refreshTokenInfo(){
  //设置请求参数
  let param = {
    token:getToken()
  }
  return refreshTokenApi(param).then(res=>res);
}


//定义变量，标识是否刷新token
let isRefresh = false;
// 发送请求之前进行拦截
service.interceptors.request.use(
  config => {
    //获取当前系统时间
    let currentTime = new Date().getTime();
```

```
      //获取token过期时间
      let expireTime = getTokenTime();
      //判断token是否过期
      if(expireTime>0){
        //计算时间
        let min = (expireTime - currentTime) / 1000 / 60;
        //如果token离过期时间相差10分钟，则刷新token
        if(min<10){
          //判断是否刷新
          if(!isRefresh){
            //标识刷新
            isRefresh = true;
            //调用刷新token的方法
            return refreshTokenInfo().then(res=>{
              //判断是否成功
              if(res.success){
                //设置新的token和过期时间
                setToken(res.data.token);
                setTokenTime(res.data.expireTime);
                //将新的token添加到header头部
                config.headers.token = getToken();
              }
              return config;
            }).catch(error=>{
            }).finally(()=>{
              //修改是否刷新token的状态
              isRefresh = false;
            });
          }
        }
      }
      // 从store里面获取token，如果token存在，则将token添加到请求的头部headers中
      if (store.getters.token) {
        // 将token添加到请求的头部
        config.headers['token'] = getToken()
      }
      return config
    },
    error => {
      //清空sessionStorage
      clearStorage();
      //清空token过期时间
      removeTokenTime();
      // do something with request error
      return Promise.reject(error)
    }
  )

  // response interceptor
  service.interceptors.response.use(
    response => {
      const res = response.data
      // if the custom code is not 20000, it is judged as an error.
      if (res.code !== 200) {
        Message({
          message: res.message || 'Error',
          type: 'error',
          duration: 5 * 1000
```

```
95          })
96
97          // 50008: Illegal token; 50012: Other clients logged in; 50014: Token
    expired;
98          if (res.code === 50008 || res.code === 50012 || res.code === 50014) {
99            MessageBox.confirm('用户登录信息过期，请重新登录！', '系统提示', {
100             confirmButtonText: '登录',
101             cancelButtonText: '取消',
102             type: 'warning'
103           }).then(() => {
104             store.dispatch('user/resetToken').then(() => {
105               //清空sessionStorage
106               clearStorage();
107               //清空token过期时间
108               removeTokenTime();
109               location.reload()
110             })
111           })
112         }
113         return Promise.reject(new Error(res.message || 'Error'))
114       } else {
115         return res
116       }
117     },
118     error => {
119       //清空sessionStorage
120       clearStorage();
121       //清空token过期时间
122       removeTokenTime();
123       Message({
124         message: error.message
125         type: 'error'
126         duration: 5 * 1000
127       })
128       return Promise.reject(error)
129     }
130   )
131
132   //请求方法
133   const http = {
134     post(url, params) {
135       return service.post(url, params, {
136         transformRequest: [(params) => {
137           return JSON.stringify(params)
138         }],
139         headers: {
140           'Content-Type': 'application/json'
141         }
142       })
143     },
144     put(url, params) {
145       return service.put(url, params, {
146         transformRequest: [(params) => {
147           return JSON.stringify(params)
148         }],
149         headers: {
150           'Content-Type': 'application/json'
151         }
```

```
152       })
153     },
154     get(url, params) {
155       return service.get(url, {
156         params: params,
157         paramsSerializer: (params) => {
158           return qs.stringify(params)
159         }
160       })
161     },
162     getRestApi(url, params) {
163       let _params
164       if (Object.is(params, undefined || null)) {
165         _params = ''
166       } else {
167         _params = '/'
168         for (const key in params) {
169           // eslint-disable-next-line no-prototype-builtins
170           if (params.hasOwnProperty(key) && params[key] !== null && params[key]
    !== '') {
171             _params += `${params[key]}/`
172           }
173         }
174         //去掉参数最后一位?
175         _params = _params.substr(0, _params.length - 1)
176       }
177       if (_params) {
178         return service.get(`${url}${_params}`)
179       } else {
180         return service.get(url)
181       }
182     },
183     delete(url, params) {
184       let _params
185       if (Object.is(params, undefined || null)) {
186         params = ''
187       } else {
188         params = '/'
189         for (const key in params) {
190           // eslint-disable-next-line no-prototype-builtins
191           if (params.hasOwnProperty(key) && params[key] !== null && params[key]
    !== '') {
192             _params += `${params[key]}/`
193           }
194         }
195         //去掉参数最后一位?
196         _params = _params.substr(0, _params.length - 1)
197       }
198       if (_params) {
199         return service.delete(`${url}${_params}`).catch(err => {
200           message.error(err.msg)
201           return Promise.reject(err)
202         })
203       } else {
204         return service.delete(url).catch(err => {
205           message.error(err.msg)
206           return Promise.reject(err)
207         })
```

```
208       }
209     },
210     upload(url, params) {
211       return service.post(url, params, {
212         headers: {
213           'Content-Type': 'multipart/form-data'
214         }
215       })
216     },
217     login(url, params) {
218       return service.post(url, params, {
219         transformRequest: [(params) => {
220           return qs.stringify(params)
221         }],
222         headers: {
223           'Content-Type': 'application/x-www-form-urlencoded'
224         }
225       })
226     }
227   }
228   export default http
```