

# 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

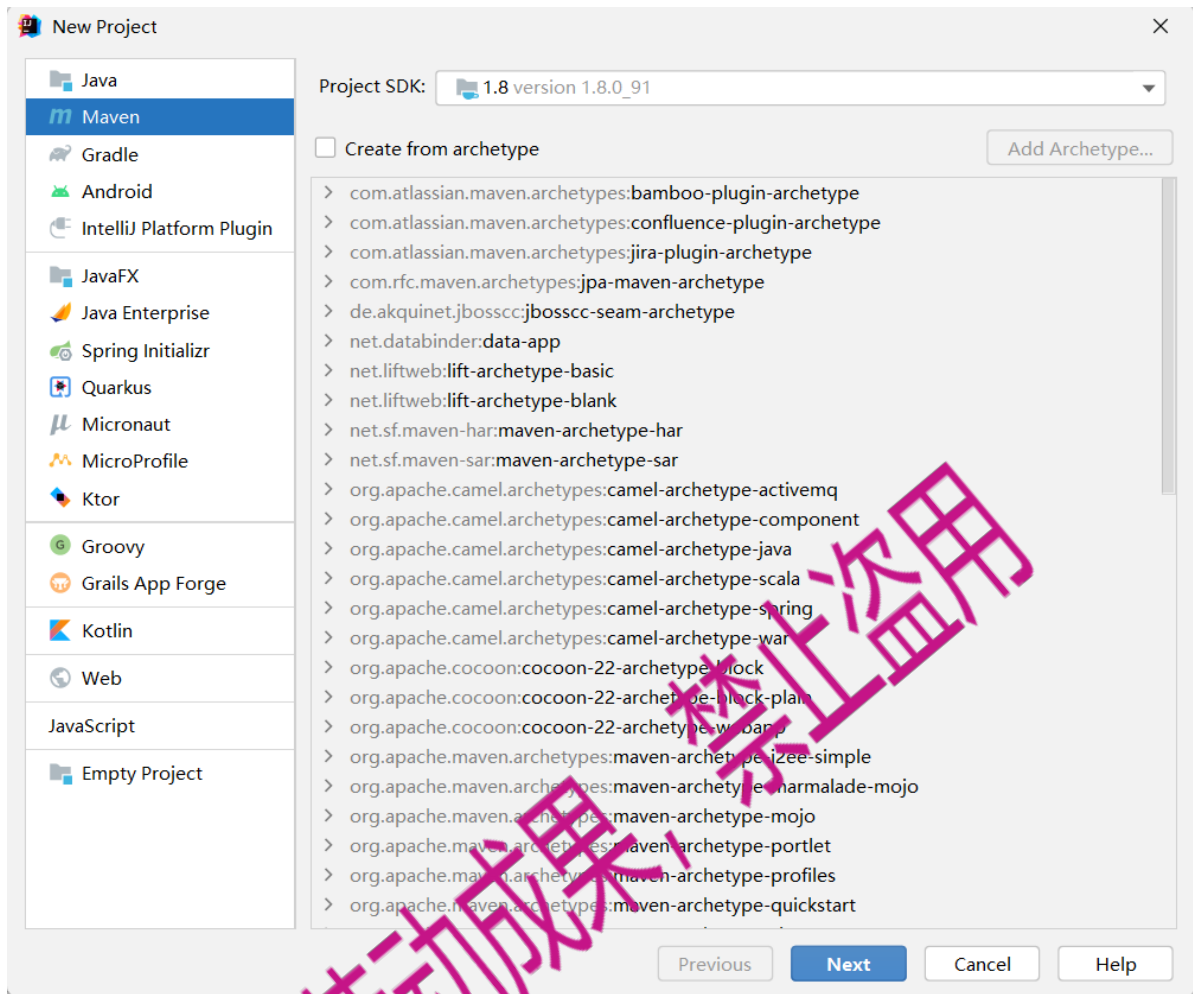
## 2. 搭建后端项目环境

---

## 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

New Project

Name:

authority-system 项目名称

Location:

E:\code\java\project\authority-system 项目保存地址

Artifact Coordinates

GroupId:

com.manong 包结构信息

The name of the artifact group, usually a company domain

ArtifactId:

authority-system 坐标名称

The name of the artifact within the group, usually a project name

Version:

1.0 项目版本号

Previous

Finish

Cancel

Help

## 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：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5 http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <!-- Spring Boot版本 -->
8     <parent>
9         <groupId>org.springframework.boot</groupId>
10        <artifactId>spring-boot-starter-parent</artifactId>
11        <version>2.6.6</version>
12    </parent>
13
14    <groupId>com.manong</groupId>
15    <artifactId>authority-system</artifactId>
16    <version>1.0</version>
17
18    <properties>
19        <java.version>1.8</java.version>
20        <jwt.version>0.9.1</jwt.version>
21        <mybatis-plus.version>3.5.1</mybatis-plus.version>
22        <fastjson.version>1.2.80</fastjson.version>
23    </properties>
24
25    <dependencies>
```

```
26     <!-- web -->
27     <dependency>
28         <groupId>org.springframework.boot</groupId>
29         <artifactId>spring-boot-starter-web</artifactId>
30     </dependency>
31     <!-- mysql -->
32     <dependency>
33         <groupId>mysql</groupId>
34         <artifactId>mysql-connector-java</artifactId>
35         <scope>runtime</scope>
36     </dependency>
37     <!-- mybatis plus -->
38     <dependency>
39         <groupId>com.baomidou</groupId>
40         <artifactId>mybatis-plus-boot-starter</artifactId>
41         <version>${mybatis-plus.version}</version>
42     </dependency>
43     <!-- lombok -->
44     <dependency>
45         <groupId>org.projectlombok</groupId>
46         <artifactId>lombok</artifactId>
47         <optional>true</optional>
48     </dependency>
49     <!-- 热部署 -->
50     <dependency>
51         <groupId>org.springframework.boot</groupId>
52         <artifactId>spring-boot-devtools</artifactId>
53         <scope>runtime</scope>
54         <optional>true</optional>
55     </dependency>
56     <!-- 单元测试 -->
57     <dependency>
58         <groupId>org.springframework.boot</groupId>
59         <artifactId>spring-boot-starter-test</artifactId>
60         <scope>test</scope>
61     </dependency>
62     <!-- fastjson -->
63     <dependency>
64         <groupId>com.alibaba</groupId>
65         <artifactId>fastjson</artifactId>
66         <version>${fastjson.version}</version>
67     </dependency>
68     <!-- jwt -->
69     <dependency>
70         <groupId>io.jsonwebtoken</groupId>
71         <artifactId>jjwt</artifactId>
72         <version>${jwt.version}</version>
73     </dependency>
74     <!-- redis -->
75     <dependency>
76         <groupId>org.springframework.boot</groupId>
77         <artifactId>spring-boot-starter-data-redis</artifactId>
78     </dependency>
79 </dependencies>
80
81 <build>
82     <plugins>
83         <plugin>
```

```
84         <groupId>org.springframework.boot</groupId>
85         <artifactId>spring-boot-maven-plugin</artifactId>
86     </plugin>
87 </plugins>
88 </build>
89
90
91 </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：

```
1  #设置端口号
2  server.port=9999
3  #数据库驱动
4  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5  #数据库连接地址
6  spring.datasource.url=jdbc:mysql://localhost:3306/db_authority_system?
   useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
7  #数据库用户名
8  spring.datasource.username=root
9  #数据库密码
10 spring.datasource.password=root
11
12 #加载映射文件
13 mybatis-plus.mapper-locations=classpath*:/mapper/**/*.xml
14 #设置别名
15 mybatis-plus.type-aliases-package=com.manong.entity
16 #关闭驼峰命名映射
17 #mybatis-plus.configuration.map-underscore-to-camel-case=false
18
19 #显示日志
20 logging.level.com.manong.dao=debug
21
22 #JSON日期格式化
23 spring.jackson.date-format= yyyy-MM-dd
24 #JSON日期格式设置时区为上海
25 spring.jackson.time-zone=Asia/Shanghai
26
27 #日期格式化
28 spring.mvc.format.date=yyyy-MM-dd
29 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。

```

1 package com.manong;
2
3 import org.mybatis.spring.annotation.MapperScan;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
6
7 @MapperScan("com.manong.dao") //加载Mapper接口
8 @SpringBootApplication
9 public class AuthorityApplication {
10     public static void main(String[] args) {
11         SpringApplication.run(AuthorityApplication.class, args);
12     }
13 }

```

## 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

```

1 package com.manong.entity;
2
3 import com.baomidou.mybatisplus.annotation.IdType;
4 import com.baomidou.mybatisplus.annotation.TableField;
5 import com.baomidou.mybatisplus.annotation.TableId;
6 import com.baomidou.mybatisplus.annotation.TableName;
7 import lombok.Data;
8 import org.springframework.security.core.GrantedAuthority;
9 import org.springframework.security.core.userdetails.UserDetails;
10
11 import java.io.Serializable;
12 import java.util.Collection;
13 import java.util.Date;
14 import java.util.List;
15
16 @Data
17 @TableName("sys_user")
18 public class User implements Serializable, UserDetails {
19
20     private static final long serialVersionUID = 1L;
21
22     /**
23      * 用户编号
24      */
25     @TableId(value = "id", type = IdType.AUTO)
26     private Long id;
27
28     /**
29      * 登录名称(用户名)
30      */
31     private String username;
32
33     /**
34      * 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

```



```

19     */
20     @TableId(value = "id", type = IdType.AUTO)
21     private Long id;
22
23     /**
24      * 角色编码
25      */
26     private String roleCode;
27
28     /**
29      * 角色名称
30      */
31     private String roleName;
32
33     /**
34      * 创建人
35      */
36     private Long createUser;
37
38     /**
39      * 创建时间
40      */
41     private Date createTime;
42
43     /**
44      * 修改时间
45      */
46     private Date updateTime;
47
48     /**
49      * 备注
50      */
51     private String remark;
52
53     /**
54      * 是否删除(0-未删除, 1-已删除)
55      */
56     private Integer isDelete;
57
58
59 }

```

### (3) Permission

```

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_permission")
13     public class Permission implements Serializable {
14

```

```
15     private static final long serialVersionUID = 1L;
16
17     /**
18      * 权限编号
19      */
20     @TableId(value = "id", type = IdType.AUTO)
21     private Long id;
22
23     /**
24      * 权限名称
25      */
26     private String label;
27
28     /**
29      * 父权限ID
30      */
31     private Long parentId;
32
33     /**
34      * 父权限名称
35      */
36     private String parentName;
37
38     /**
39      * 授权标识符
40      */
41     private String code;
42
43     /**
44      * 路由地址
45      */
46     private String path;
47
48     /**
49      * 路由名称
50      */
51     private String name;
52
53     /**
54      * 授权路径
55      */
56     private String url;
57
58     /**
59      * 权限类型(0-目录 1-菜单 2-按钮)
60      */
61     private Integer type;
62
63     /**
64      * 图标
65      */
66     private String icon;
67
68     /**
69      * 创建时间
70      */
71     private Date createTime;
72
```

```

73     /**
74      * 修改时间
75      */
76     private Date updateTime;
77
78     /**
79      * 备注
80      */
81     private String remark;
82
83     /**
84      * 是否删除(0-未删除, 1-已删除)
85      */
86     private Integer isDelete;
87
88     /**
89      * 排序
90      */
91     private Integer orderNum;
92 }

```

#### (4) Department

```

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
10 @Data
11 @TableName("sys_department")
12 public class Department implements Serializable {
13
14     private static final long serialVersionUID = 1L;
15
16     /**
17      * 部门编号
18      */
19     @TableId(value = "id", type = IdType.AUTO)
20     private Long id;
21
22     /**
23      * 部门名称
24      */
25     private String departmentName;
26
27     /**
28      * 部门电话
29      */
30     private String phone;
31
32     /**
33      * 部门地址
34      */
35     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

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

### 2.5.3 生成Mapper 映射文件

#### (1) UserMapper.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4
5 <mapper namespace="com.manong.dao.UserMapper">
6
7     <!-- 通用查询映射结果 -->
8     <resultMap id="BaseResultMap" type="com.manong.entity.User">
9         <id column="id" property="id" />
10        <result column="username" property="username" />
11        <result column="password" property="password" />
12        <result column="is_account_non_expired" property="isAccountNonExpired" />
13        <result column="is_account_non_locked" property="isAccountNonLocked" />
14        <result column="is_credentials_non_expired"
15        property="isCredentialsNonExpired" />
16        <result column="is_enabled" property="isEnabled" />
17        <result column="real_name" property="realName" />
18        <result column="nick_name" property="nickName" />
19        <result column="department_id" property="departmentId" />
20        <result column="department_name" property="departmentName" />
21        <result column="gender" property="gender" />
22        <result column="phone" property="phone" />
23        <result column="avatar" property="avatar" />
24        <result column="email" property="email" />
25        <result column="is_admin" property="isAdmin" />
26        <result column="create_time" property="createTime" />
27        <result column="update_time" property="updateTime" />
28        <result column="is_delete" property="isDelete" />
29    </resultMap>
30
31    <!-- 通用查询结果列 -->
32    <sql id="Base_Column_List">
33        id, username, password, is_account_non_expired, is_account_non_locked,
34        is_credentials_non_expired, is_enabled, real_name, nick_name, department_id,
35        department_name, gender, phone, email, avatar, is_admin, create_time, update_time,
36        is_delete
37    </sql>
38
39 </mapper>
```

#### (2) RoleMapper.xml

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

```

2  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
3  <mapper namespace="com.manong.dao.RoleMapper">
4
5      <!-- 通用查询映射结果 -->
6      <resultMap id="BaseResultMap" type="com.manong.entity.Role">
7          <id column="id" property="id" />
8          <result column="role_code" property="roleCode" />
9          <result column="role_name" property="roleName" />
10         <result column="create_user" property="createUser" />
11         <result column="create_time" property="createTime" />
12         <result column="update_time" property="updateTime" />
13         <result column="remark" property="remark" />
14         <result column="is_delete" property="isDelete" />
15     </resultMap>
16
17     <!-- 通用查询结果列 -->
18     <sql id="Base_Column_List">
19         id, role_code, role_name, create_user, create_time, update_time, remark,
is_delete
20     </sql>
21
22 </mapper>

```

### (3) PermissionMapper.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
3  <mapper namespace="com.manong.dao.PermissionMapper">
4
5      <!-- 通用查询映射结果 -->
6      <resultMap id="BaseResultMap" type="com.manong.entity.Permission">
7          <id column="id" property="id" />
8          <result column="label" property="label" />
9          <result column="parent_id" property="parentId" />
10         <result column="parent_name" property="parentName" />
11         <result column="code" property="code" />
12         <result column="path" property="path" />
13         <result column="name" property="name" />
14         <result column="url" property="url" />
15         <result column="type" property="type" />
16         <result column="icon" property="icon" />
17         <result column="create_time" property="createTime" />
18         <result column="update_time" property="updateTime" />
19         <result column="remark" property="remark" />
20         <result column="is_delete" property="isDelete" />
21         <result column="order_num" property="orderNum" />
22     </resultMap>
23
24     <!-- 通用查询结果列 -->
25     <sql id="Base_Column_List">
26         id, label, parent_id, parent_name, code, path, name, url, type, icon,
create_time, update_time, remark, is_delete, order_num
27     </sql>
28
29 </mapper>

```

#### (4) DepartmentMapper.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
3  <mapper namespace="com.manong.dao.DepartmentMapper">
4
5      <!-- 通用查询映射结果 -->
6      <resultMap id="BaseResultMap" type="com.manong.entity.Department">
7          <id column="id" property="id" />
8          <result column="department_name" property="departmentName" />
9          <result column="phone" property="phone" />
10         <result column="address" property="address" />
11         <result column="pid" property="pid" />
12         <result column="parent_name" property="parentName" />
13         <result column="is_delete" property="isDelete" />
14         <result column="order_num" property="orderNum" />
15     </resultMap>
16
17     <!-- 通用查询结果列 -->
18     <sql id="Base_Column_List">
19         id, department_name, phone, address, pid, parent_name,
20         is_delete, order_num
21     </sql>
22 </mapper>
```

### 2.5.4 生成Service接口

#### (1) UserService

```
1  package com.manong.service;
2
3  import com.manong.entity.User;
4  import com.baomidou.mybatisplus.extension.service.IService;
5
6  public interface UserService extends IService<User> {
7
8  }
```

#### (2) RoleService

```
1  package com.manong.service;
2
3  import com.manong.entity.Role;
4  import com.baomidou.mybatisplus.extension.service.IService;
5
6  public interface RoleService extends IService<Role> {
7
8  }
```

#### (3) PermissionService

```

1 package com.manong.service;
2
3 import com.manong.entity.Permission;
4 import com.baomidou.mybatisplus.extension.service.IService;
5
6 public interface PermissionService extends IService<Permission> {
7
8 }

```

#### (4) DepartmentService

```

1 package com.manong.service;
2
3 import com.manong.entity.Department;
4 import com.baomidou.mybatisplus.extension.service.IService;
5
6 public interface DepartmentService extends IService<Department> {
7
8 }

```

### 2.5.5 生成Service实现类

注意：在每个Service实现类上加入@Transactional注解。

#### (1) UserServiceImpl

```

1 package com.manong.service.impl;
2
3 import com.manong.entity.User;
4 import com.manong.dao.UserMapper;
5 import com.manong.service.UserService;
6 import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
7 import org.springframework.stereotype.Service;
8 import org.springframework.transaction.annotation.Transactional;
9
10 @Service
11 @Transactional
12 public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements
    UserService {
13
14 }

```

#### (2) RoleServiceImpl

```

1 package com.manong.service.impl;
2
3 import com.manong.entity.Role;
4 import com.manong.dao.RoleMapper;
5 import com.manong.service.RoleService;
6 import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
7 import org.springframework.stereotype.Service;
8 import org.springframework.transaction.annotation.Transactional;
9
10 @Service
11 @Transactional
12 public class RoleServiceImpl extends ServiceImpl<RoleMapper, Role> implements
    RoleService {
13

```



```
14 }
```

### (3) PermissionServiceImpl

```
1 package com.manong.service.impl;
2
3 import com.manong.entity.Permission;
4 import com.manong.dao.PermissionMapper;
5 import com.manong.service.PermissionService;
6 import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
7 import org.springframework.stereotype.Service;
8 import org.springframework.transaction.annotation.Transactional;
9
10 @Service
11 @Transactional
12 public class PermissionServiceImpl extends ServiceImpl<PermissionMapper,
13     Permission> implements PermissionService {
14 }
```

### (4) DepartmentServiceImpl

```
1 package com.manong.service.impl;
2
3 import com.manong.entity.Department;
4 import com.manong.dao.DepartmentMapper;
5 import com.manong.service.DepartmentService;
6 import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
7 import org.springframework.stereotype.Service;
8 import org.springframework.transaction.annotation.Transactional;
9
10 @Service
11 @Transactional
12 public class DepartmentServiceImpl extends ServiceImpl<DepartmentMapper,
13     Department> implements DepartmentService {
14 }
```

## 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

```
1 package com.manong.controller;
2
3
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController
8 @RequestMapping("/api/user")
9 public class UserController {
10
11 }
```

### (2) RoleController

```

1  package com.manong.controller;
2
3
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.RestController;
6
7  @RestController
8  @RequestMapping("/api/role")
9  public class RoleController {
10
11  }

```

### (3) PermissionController

```

1  package com.manong.controller;
2
3
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.RestController;
6
7  @RestController
8  @RequestMapping("/api/permission")
9  public class PermissionController {
10
11  }

```

### (4) DepartmentController

```

1  package com.manong.controller;
2
3
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.RestController;
6
7  @RestController
8  @RequestMapping("/api/department")
9  public class DepartmentController {
10
11  }

```

## 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

```

1  package com.manong.utils;
2
3  public class ResultCode {
4      /**
5       * 成功状态码
6       */
7      public static Integer SUCCESS = 200;
8
9      /**
10     * 失败状态码
11     */
12     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，禁止在其子类创建对象
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);

```

```
44         return result;
45     }
46
47     /**
48      * 失败
49      * @return
50      */
51     public static<T> Result<T> error(){
52         Result<T> result = new Result<T>();
53         result.setSuccess(false);
54         result.setCode(ResultCode.ERROR);
55         result.setMessage("执行失败");
56         return result;
57     }
58
59     /**
60      * 设置是否成功
61      * @param success
62      * @return
63      */
64     public Result<T> success(Boolean success){
65         this.setSuccess(success);
66         return this;
67     }
68
69     /**
70      * 设置状态码
71      * @param code
72      * @return
73      */
74     public Result<T> code(Integer code){
75         this.setCode(code);
76         return this;
77     }
78
79     /**
80      * 设置返回消息
81      * @param message
82      * @return
83      */
84     public Result<T> message(String message){
85         this.setMessage(message);
86         return this;
87     }
88
89     /**
90      * 是否存在
91      * @return
92      */
93     public static<T> Result<T> exist(){
94         Result<T> result = new Result<T>();
95         result.setSuccess(true);
96         result.setCode(ResultCode.SUCCESS);
97         result.setMessage("执行成功");
98         return result;
99     }
100 }
```

## 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

```
1 package com.manong.config.mybatis;
2
3 import com.baomidou.mybatisplus.annotation.DbType;
4 import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
5 import
    com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 @Configuration
10 public class MybatisPlusConfig {
11
12     // 最新版
13     @Bean
14     public MybatisPlusInterceptor mybatisPlusInterceptor() {
15         MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
16         PaginationInnerInterceptor paginationInnerInterceptor =
17             new PaginationInnerInterceptor(DbType.MYSQL);
18         paginationInnerInterceptor.setOverflow(true); //溢出后从第1页开始
19         //指定数据库类型
20         interceptor.addInnerInterceptor(paginationInnerInterceptor);
21         return interceptor;
22     }
23
24 }
```

### 2.7.2 编写自动填充配置类

```
1 package com.manong.config.mybatis;
2
3 import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
4 import org.apache.ibatis.reflection.MetaObject;
5 import org.springframework.stereotype.Component;
6
7 import java.util.Date;
8
9 @Component
10 public class CommonMetaObjectHandler implements MetaObjectHandler {
11
12     /**
13      * 新增
14      * @param metaObject
15      */
16     @Override
17     public void insertFill(MetaObject metaObject) {
18         //参数1: 元数据对象
19         //参数2: 属性名称
20         //参数3: 类对象
21         //参数4: 当前系统时间
22         this.strictInsertFill(metaObject, "createTime", Date.class, new Date());
23         this.strictUpdateFill(metaObject, "updateTime", Date.class, new Date());
24     }
25 }
```

```

24     }
25
26     /**
27      * 修改
28      * @param metaObject
29      */
30     @Override
31     public void updateFill(MetaObject metaObject) {
32         this.strictUpdateFill(metaObject, "updateTime", Date.class, new Date());
33     }
34 }

```

## 2.8 接口代码测试

### 2.8.1 编写UserController

在UserController控制器类中编写查询所有用户的方法。

```

1  @RestController
2  @RequestMapping("/api/user")
3  public class UserController {
4
5      @Resource
6      private UserService userService;
7
8      /**
9       * 查询所有用户列表
10      * @return
11      */
12      @GetMapping("/listAll")
13      public Result listAll(){
14          return Result.ok(userService.list());
15      }
16
17 }

```

### 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。

```

1  package com.manong.config.web;
2
3  import org.springframework.context.annotation.Configuration;
4  import org.springframework.web.servlet.config.annotation.CorsRegistry;
5  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
6
7  @Configuration
8  public class CORSConfig implements WebMvcConfigurer {
9
10     @Override
11     public void addCorsMappings(CorsRegistry registry) {
12         registry.addMapping("/*").//允许所有的访问请求（访问路径）

```

```
13         allowedMethods("*").//允许所有的请求方法访问该跨域资源服务器
14         allowedOrigins("*").//允许所有的请求域名访问我们的跨域资源
15         allowedHeaders("*");//允许所有的请求header访问
16     }
17 }
```

## 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 UserDetailsService和UserDetailsService接口。UserDetailsService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UserDetailsService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UserDetailsService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. 自定义UserDetails类：当实体对象字段不满足时需要自定义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的处理器，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：

```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-security</artifactId>
4 </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。

```
1  @Data
2  @TableName("sys_user")
3  public class User implements Serializable, UserDetails {
4
5      //省略原有的属性.....
6
7      /**
8       * 帐户是否过期(1 未过期, 0已过期)
9       */
10     private boolean isAccountNonExpired = true;
11
12     /**
13      * 帐户是否被锁定(1 未过期, 0已过期)
14      */
15     private boolean isAccountNonLocked = true;
16
17     /**
18      * 密码是否过期(1 未过期, 0已过期)
19      */
20     private boolean isCredentialsNonExpired = true;
21
22     /**
23      * 帐户是否可用(1 可用, 0 删除用户)
24      */
25     private boolean isEnabled = true;
26
27     /**
28      * 权限列表
29      */
30     @TableField(exist = false)
31     Collection<? extends GrantedAuthority> authorities;
32 }
```

### 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。

```
1  public interface UserService extends IService<User> {
2      /**
3       * 根据用户名查询用户信息
4       * @param userName
5       * @return
6       */
7      User findUserByUserName(String userName);
8  }
```

### 3.3.4 编写UserService接口实现类

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

```
1  @Service
2  @Transactional
3  public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements
4      UserService {
5      /**
```



```

6      * 根据用户名查询用户信息
7      *
8      * @param userName
9      * @return
10     */
11     @Override
12     public User findUserByUserName(String userName) {
13         //创建条件构造器
14         QueryWrapper<User> queryWrapper = new QueryWrapper<User>();
15         //用户名
16         queryWrapper.eq("username", userName);
17         //返回查询记录
18         return baseMapper.selectOne(queryWrapper);
19     }
20 }

```

### 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。

```

1  package com.manong.config.security.service;
2
3  import com.manong.entity.User;
4  import com.manong.service.UserService;
5  import org.springframework.security.core.userdetails.UserDetails;
6  import org.springframework.security.core.userdetails.UserDetailsService;
7  import org.springframework.security.core.userdetails.UsernameNotFoundException;
8  import org.springframework.stereotype.Component;
9
10 import javax.annotation.Resource;
11
12 @Component
13 public class CustomerUserDetailsService implements UserDetailsService {
14
15     @Resource
16     private UserService userService;
17
18     @Override
19     public UserDetails loadUserByUsername(String username) throws
20     UsernameNotFoundException {
21         //调用根据用户名查询用户信息的方法
22         User user = userService.findUserByUserName(username);
23         //如果对象为空，则认证失败
24         if (user == null) {
25             throw new UsernameNotFoundException("用户名或密码错误!");
26         }
27         return user;
28     }
29 }

```

### 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。

```

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.entity.User;
6 import org.springframework.security.core.Authentication;
7 import
  org.springframework.security.web.authentication.AuthenticationSuccessHandler;
8 import org.springframework.stereotype.Component;
9
10 import javax.servlet.ServletException;
11 import javax.servlet.ServletOutputStream;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14 import java.io.IOException;
15 import java.nio.charset.StandardCharsets;
16
17 /**
18  * 登录认证成功处理器类
19  */
20 @Component
21 public class LoginSuccessHandler implements AuthenticationSuccessHandler {
22     @Override
23     public void onAuthenticationSuccess(HttpServletRequest request,
  HttpServletResponse response, Authentication authentication) throws IOException,
  ServletException {
24         //设置客户端的响应的内容类型
25         response.setContentType("application/json;charset=UTF-8");
26         //获取当登录用户信息
27         User user = (User) authentication.getPrincipal();
28         //消除循环引用
29         String result = JSON.toJSONString(user,
  SerializerFeature.DisableCircularReferenceDetect);
30         //获取输出流
31         ServletOutputStream outputStream = response.getOutputStream();
32         outputStream.write(result.getBytes(StandardCharsets.UTF_8));
33         outputStream.flush();
34         outputStream.close();
35     }
36 }

```

### 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。

```

1 package com.manong.config.security.handler;
2
3 import com.alibaba.fastjson.JSON;
4 import com.manong.utils.Result;
5 import org.springframework.security.authentication.*;
6 import org.springframework.security.core.AuthenticationException;
7 import
  org.springframework.security.web.authentication.AuthenticationFailureHandler;
8 import org.springframework.stereotype.Component;
9
10 import javax.servlet.ServletException;
11 import javax.servlet.ServletOutputStream;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14 import java.io.IOException;
15 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,
24      HttpServletResponse response, AuthenticationException exception) throws
25      IOException, ServletException {
26          //设置客户端响应编码格式
27          response.setContentType("application/json;charset=UTF-8");
28          //获取输出流
29          ServletOutputStream outputStream = response.getOutputStream();
30          String message = null; //提示信息
31          int code = 500; //错误编码
32          //判断异常类型
33          if(exception instanceof AccountExpiredException){
34              message = "账户过期, 登录失败! ";
35          }else if(exception instanceof BadCredentialsException){
36              message = "用户名或密码错误, 登录失败! ";
37          }else if(exception instanceof CredentialsExpiredException){
38              message = "密码过期, 登录失败! ";
39          }else if(exception instanceof DisabledException){
40              message = "账户被禁用, 登录失败! ";
41          }else if(exception instanceof LockedException){
42              message = "账户被锁, 登录失败! ";
43          }else if(exception instanceof InternalAuthenticationServiceException){
44              message = "账户不存在, 登录失败! ";
45          }else{
46              message = "登录失败! ";
47          }
48          //将错误信息转换成JSON
49          String result =
50          JSON.toJSONString(Result.error().code(code).message(message));
51          outputStream.write(result.getBytes(StandardCharsets.UTF_8));
52          outputStream.flush();
53          outputStream.close();
54      }
55  }

```

### 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;

```

```

13 import javax.servlet.http.HttpServletResponse;
14 import java.io.IOException;
15 import java.nio.charset.StandardCharsets;
16
17 /**
18  * 认证用户访问无权限资源时处理器
19  */
20 @Component
21 public class CustomerAccessDeniedHandler implements AccessDeniedHandler {
22     @Override
23     public void handle(HttpServletRequest request, HttpServletResponse response,
24         AccessDeniedException accessDeniedException) throws IOException, ServletException {
25
26         //设置客户端的响应的内容类型
27         response.setContentType("application/json;charset=UTF-8");
28         //获取输出流
29         ServletOutputStream outputStream = response.getOutputStream();
30         //消除循环引用
31         String result = JSON.toJSONString(Result.error().code(700).message("无权限访问,请联系管理员!"), SerializerFeature.DisableCircularReferenceDetect);
32         outputStream.write(result.getBytes(StandardCharsets.UTF_8));
33         outputStream.flush();
34         outputStream.close();
35     }
36 }

```

### 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。

```

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.util.Result;
6 import org.springframework.security.core.AuthenticationException;
7 import org.springframework.security.web.AuthenticationEntryPoint;
8 import org.springframework.stereotype.Component;
9
10 import javax.servlet.ServletException;
11 import javax.servlet.ServletOutputStream;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14 import java.io.IOException;
15 import java.nio.charset.StandardCharsets;
16
17 /**
18  * 匿名用户访问资源处理器
19  */
20 @Component
21 public class AnonymousAuthenticationHandler implements AuthenticationEntryPoint {
22     @Override
23     public void commence(HttpServletRequest request, HttpServletResponse
24         response, AuthenticationException authException) throws IOException,
25         ServletException {
26
27         //设置客户端的响应的内容类型
28         response.setContentType("application/json;charset=UTF-8");
29
30     }
31 }

```

```

26         //获取输出流
27         ServletOutputStream outputStream = response.getOutputStream();
28         //消除循环引用
29         String result = JSON.toJSONString(Result.error().code(600).message("匿名用户无权限访问!"), SerializerFeature.DisableCircularReferenceDetect);
30         outputStream.write(result.getBytes(StandardCharsets.UTF_8));
31         outputStream.flush();
32         outputStream.close();
33     }
34 }

```

### 3.3.10 编写Spring Security配置类

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

```

1  package com.manong.config.security;
2
3  import com.manong.config.security.handler.AnonymousAuthenticationHandler;
4  import com.manong.config.security.handler.CustomerAccessDeniedHandler;
5  import com.manong.config.security.handler.LoginFailureHandler;
6  import com.manong.config.security.handler.LoginSuccessHandler;
7  import com.manong.config.security.service.CustomerUserDetailsService;
8  import org.springframework.context.annotation.Bean;
9  import org.springframework.context.annotation.Configuration;
10 import
11 org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
12 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
13 import
14 org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
15 import
16 org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
17 import org.springframework.security.config.http.SessionCreationPolicy;
18 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
19
20 import java.util.Resource;
21
22 @Configuration
23 @EnableWebSecurity
24 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
25     @Resource
26     private CustomerUserDetailsService customerUserDetailsService;
27
28     @Resource
29     private LoginSuccessHandler loginSuccessHandler;
30
31     @Resource
32     private LoginFailureHandler loginFailureHandler;
33
34     @Resource
35     private AnonymousAuthenticationHandler anonymousAuthenticationHandler;
36
37     @Resource
38     private CustomerAccessDeniedHandler customerAccessDeniedHandler;
39
40     /**

```

```

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
54             .successHandler(loginSuccessHandler).failureHandler(loginFailureHandler)
55             // 禁用csrf防御机制
56             .and().csrf().disable()
57
58             .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
59             .and()
60             .authorizeRequests()
61             .antMatchers("/api/user/login").permitAll()
62             .anyRequest().authenticated()
63             .and()
64             .exceptionHandling()
65             .authenticationEntryPoint(new AnonymousAuthenticationHandler())
66             .accessDeniedHandler(new CustomerAccessDeniedHandler())
67             .and().cors().configure() //开启跨域配置
68         }
69
70     /**
71     * 配置认证处理器
72     *
73     * @param auth
74     * @throws Exception
75     */
76     @Override
77     protected void configure(AuthenticationManagerBuilder auth) throws Exception
78     {
79
80         auth.userDetailsService(customerUserDetailsService).passwordEncoder(passwordEncoder());
81     }
82 }

```

### 3.3.11 测试登录认证接口

POST http://localhost:9999/api/user/login?username=admin&password=123456

Params Authorization Headers (7) Body Pre-request Script Tests Settings

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> username	admin	
<input checked="" type="checkbox"/> password	123456	

Body Cookies Headers (14) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "accountNonExpired": true,
3   "accountNonLocked": true,
4   "credentialsNonExpired": true,
5   "departmentId": 1,
6   "departmentName": "广州码农信息技术有限公司",
7   "enabled": true,
8   "gender": 0,
9   "id": 1,
10  "isAdmin": 1,
11  "isDelete": 0,
12  "nickName": "超级管理员",
13  "password": "$2a$10$TdEVQtGCkpo8L.jKjFB3/uxV5xkkDfiy0zoCa.ZS2yAXHe7H950IC",
14  "phone": "13242587415",
15  "realName": "李明",
16  "username": "admin"
17 }
```

禁止盗用

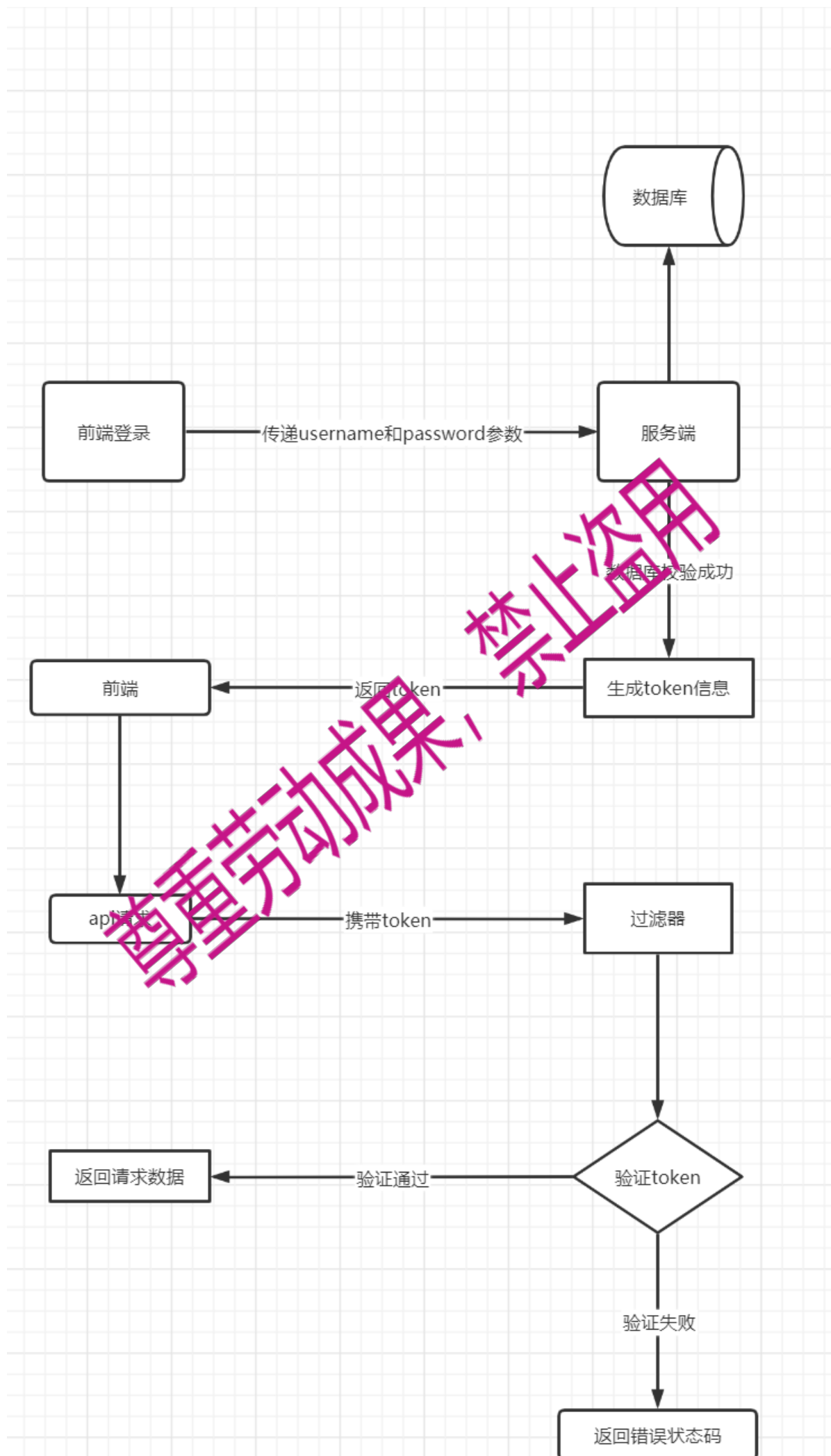
## 3.4 认证成功返回token

### 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







### 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

```
1 package com.manong.utils;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @NoArgsConstructor
9 @AllArgsConstructor
10 public class LoginResult {
11     //用户编号
12     private Long id;
13     //状态码
14     private int code;
15     //token令牌
16     private String token;
17     //token过期时间
18     private Long expireTime;
19 }
```

#### (2) 编写token工具类

```
1 package com.manong.utils;
2
3 import com.manong.entity.User;
4 import io.jsonwebtoken.Claims;
5 import io.jsonwebtoken.Jws;
6 import io.jsonwebtoken.SignatureAlgorithm;
7 import lombok.Data;
8 import org.springframework.boot.context.properties.ConfigurationProperties;
9 import org.springframework.security.core.userdetails.UserDetails;
10 import org.springframework.stereotype.Component;
11
12 import java.util.Date;
13 import java.util.HashMap;
14 import java.util.Map;
15
16 @Data
17 @ConfigurationProperties(prefix = "jwt")
18 @Component
19 public class JwtUtils {
20     //密钥
21     private String secret;
22
23     // 过期时间 毫秒
24     private Long expiration;
25
26
27     /**
28      * 从数据声明生成令牌
29      *
30      */
31 }
```

```

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。

```

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.entity.User;
6  import com.manong.utils.JwtUtils;
7  import com.manong.utils.LoginResult;
8  import com.manong.utils.ResultCode;
9  import io.jsonwebtoken.Jwts;
10 import org.springframework.security.core.Authentication;
11 import
    org.springframework.security.web.authentication.AuthenticationSuccessHandler;
12 import org.springframework.stereotype.Component;
13
14 import javax.annotation.Resource;
15 import javax.servlet.ServletException;
16 import javax.servlet.ServletOutputStream;
17 import javax.servlet.http.HttpServletRequest;
18 import javax.servlet.http.HttpServletResponse;
19 import java.io.IOException;
20 import java.nio.charset.StandardCharsets;
21
22 /**
23  * 登录认证成功处理器类
24  */
25 @Component
26 public class LoginSuccessHandler implements AuthenticationSuccessHandler {
27
28     @Resource
29     private JwtUtils jwtUtils;
30
31     @Override
32     public void onAuthenticationSuccess(HttpServletRequest request,
    HttpServletResponse response, Authentication authentication) throws IOException,
    ServletException {
33         //设置客户端的响应的内容类型
34         response.setContentType("application/json;charset=UTF-8");
35         //获取当登录用户信息
36         User user = (User) authentication.getPrincipal();
37         //生成token
38         String token = jwtUtils.generateToken(user);
39         //设置token签名密钥及过期时间
40         long expireTime = Jwts.parser() //获取DefaultJwtParser对象
41             .setSigningKey(jwtUtils.getSecret()) //设置签名的密钥
42             .parseClaimsJws(token.replace("jwt_", ""))
43             .getBody().getExpiration().getTime(); //获取token过期时间
44         //创建登录结果对象
45         LoginResult loginResult = new LoginResult(user.getId(),
    ResultCode.SUCCESS, token, expireTime);
46         //消除循环引用
47         String result = JSON.toJSONString(loginResult,
48
    SerializerFeature.DisableCircularReferenceDetect);
49         //获取输出流
50         ServletOutputStream outputStream = response.getOutputStream();
51         outputStream.write(result.getBytes(StandardCharsets.UTF_8));
52         outputStream.flush();
53         outputStream.close();

```

```
54     }
55 }
```

### 3.4.4 测试token

通用权限管理系统 / 用户模块 / 用户登录

POST http://localhost:9999/api/user/login?username=admin&password=123456

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

KEY	VALUE	DESCRIPTION
username	admin	
password	123456	

Body Cookies Headers (14) Test Results Status: 200 OK Time: 98 ms Size: 684 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "code": 200,
3   "expireTime": 1650962658000,
4   "id": 1,
5   "token": "eyJhbGciOiJIUzUxMiJ9.eyJleHAiOjE2NTA5MjI2NzgsInN1YiI6ImFkbWluIiwiaWF0IjoxNjUwOTYwODU4MjI1fQ.
6   ez13ycR90Yiz1PaDy8C2HeBdRrIXYQ7gg989VAnW9Q9TaHdYA-P1MC04wKuk7nTGJo6GpMjMtIn5GONw9qp1w"
```

## 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：

```
1  /**
2   * 子菜单列表
3   */
4  @JsonInclude(JsonInclude.Include.NON_NULL) //属性值为null不进行序列化操作
5  @TableField(exist = false)
6  private List<Permission> children = new ArrayList<Permission>();
7
8  /**
9   * 用于前端判断是菜单、目录或按钮
10  */
11  @TableField(exist = false)
12  private String value;
13
14  /**
15   * 是否展开
16   */
17  @TableField(exist = false)
18  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。

```

1 public interface PermissionMapper extends BaseMapper<Permission> {
2     /**
3      * 根据用户ID查询权限列表
4      * @param userId
5      * @return
6      */
7     List<Permission> findPermissionListByUserId(Long userId);
8 }

```

### 3.5.3 编写PermissionMapper映射文件

```

1 <select id="findPermissionListByUserId"
2     resultType="com.manong.entity.Permission">
3     select
4         DISTINCT
5         p.id,p.parent_id,p.label,p.`code`,p.url,p.type,p.icon,p.remark,p.path,p.name
6     from
7         sys_user as u
8         left join sys_user_role as ur on u.id = ur.user_id
9         left join sys_role as r on ur.role_id = r.id
10        left join sys_role_permission as rp on rp.role_id = r.id
11        left join sys_permission as p on rp.permission_id = p.id
12
13     where u.id =#{userId}
14     order by p.id asc
15 </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。

```

1 public interface PermissionService extends IService<Permission> {
2     /**
3      * 根据用户ID查询权限列表
4      * @param userId
5      * @return
6      */
7     List<Permission> findPermissionListByUserId(Long userId);
8 }

```

### 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。

```

1 @Service
2 @Transactional
3 public class PermissionServiceImpl extends ServiceImpl<PermissionMapper,
4     Permission>
5     implements PermissionService {
6     /**
7      * 根据用户ID查询权限列表
8      *
9      * @param userId
10     * @return
11     */

```

```

12     @Override
13     public List<Permission> findPermissionListByUserId(Long userId) {
14         return baseMapper.findPermissionListByUserId(userId);
15     }
16 }

```

### 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：

```

1  /**
2   * 用户权限列表
3   */
4  @TableField(exist = false)
5  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。

```

1  package com.manong.config.security.service;
2
3  import com.manong.entity.Permission;
4  import com.manong.entity.User;
5  import com.manong.service.PermissionService;
6  import com.manong.service.UserService;
7  import org.springframework.security.core.GrantedAuthority;
8  import org.springframework.security.core.authority.AuthorityUtils;
9  import org.springframework.security.core.userdetails.UserDetails;
10 import org.springframework.security.core.userdetails.UserDetailsService;
11 import org.springframework.security.core.userdetails.UsernameNotFoundException;
12 import org.springframework.stereotype.Component;
13
14 import javax.annotation.Resource;
15 import java.util.List;
16 import java.util.stream.Collectors;
17
18 @Component
19 public class CustomerUserDetailsService implements UserDetailsService {
20
21     @Resource
22     private UserService userService;
23
24     @Resource
25     private PermissionService permissionService;
26
27     @Override
28     public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
29         //调用根据用户名查询用户信息的方法
30         User user = userService.findUserByUserName(username);
31         //如果对象为空，则认证失败
32         if (user == null) {
33             throw new UsernameNotFoundException("用户名或密码错误!");
34         }
35         //查询用户的拥有的权限列表
36         List<Permission> permissionList =
permissionService.findPermissionListByUserId(user.getId());

```



```

37         //获取权限编码
38         List<String> collect = permissionList.stream()
39             .filter(item -> item != null)
40             .map(item -> item.getCode()).filter(item -> item != null)
41             .collect(Collectors.toList());
42         //转换成数组
43         String[] strings = collect.toArray(new String[collect.size()]);
44         //设置权限列表
45         List<GrantedAuthority> authorityList =
AuthorityUtils.createAuthorityList(strings);
46         user.setAuthorities(authorityList);
47         //设置菜单列表
48         user.setPermissionList(permissionList);
49         //返回用户信息
50         return user;
51     }
52 }

```

## 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。

```

1     <dependency>
2         <groupId>org.springframework.boot</groupId>
3         <artifactId>spring-boot-starter-data-redis</artifactId>
4     </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。

```

1     ##### Redis相关配置 #####
2     spring.redis.host=localhost
3     spring.redis.port=6379
4     spring.redis.database=0
5     spring.redis.timeout=10000
6     #自定义属性
7     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。

```

1     package com.manong.config.redis;
2
3
4     import com.fasterxml.jackson.annotation.JsonAutoDetect;
5     import com.fasterxml.jackson.annotation.PropertyAccessor;
6     import com.fasterxml.jackson.databind.ObjectMapper;
7     import org.springframework.beans.factory.annotation.Value;
8     import org.springframework.context.annotation.Bean;
9     import org.springframework.context.annotation.Configuration;

```



```

10 import org.springframework.data.redis.cache.RedisCacheConfiguration;
11 import org.springframework.data.redis.cache.RedisCacheManager;
12 import org.springframework.data.redis.connection.RedisConnectionFactory;
13 import org.springframework.data.redis.core.RedisTemplate;
14 import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
15 import org.springframework.data.redis.serializer.RedisSerializationContext;
16 import org.springframework.data.redis.serializer.RedisSerializer;
17 import org.springframework.data.redis.serializer.StringRedisSerializer;
18
19 import java.time.Duration;
20
21 /**
22  * redis配置类
23  */
24 @Configuration
25 public class RedisConfig {
26     //缓存过期时间
27     @Value("${spring.redis.expire}")
28     private Long expire;
29
30     @Bean
31     public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
factory) {
32         RedisTemplate<String, Object> template = new RedisTemplate<String,
Object>();
33         // 配置连接工厂
34         template.setConnectionFactory(factory);
35         //使用Jackson2JsonRedisSerialize 替换默认序列化(默认采用的是JDK序列化)
36         Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer(Object.class);
37         //解决查询缓存转换异常的问题
38         ObjectMapper om = new ObjectMapper();
39         //设置在生成 json 串时，对象中的成员的可见性
40         om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
41         //存储到redis的json将是有类型的数据
42         //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
43         om.activateDefaultTyping(om.getPolymorphicTypeValidator(),
ObjectMapper.DefaultTyping.NON_FINAL);
44         jackson2JsonRedisSerializer.setObjectMapper(om);
45         StringRedisSerializer stringRedisSerializer = new
StringRedisSerializer();
46         // key采用String的序列化方式
47         template.setKeySerializer(stringRedisSerializer);
48         // hash的key也采用String的序列化方式
49         template.setHashKeySerializer(stringRedisSerializer);
50         // value序列化方式采用jackson
51         template.setValueSerializer(jackson2JsonRedisSerializer);
52         // hash的value序列化方式采用jackson
53         template.setHashValueSerializer(jackson2JsonRedisSerializer);
54         template.afterPropertiesSet();
55         return template;
56     }
57
58     //@Cacheable注解字符集编码配置
59     @Bean
60     public RedisCacheManager cacheManager(RedisConnectionFactory factory) {

```

```

61         RedisCacheConfiguration config =
RedisCacheConfiguration.defaultCacheConfig();
62         config.entryTtl(Duration.ofMinutes(expire))//缓存过期时间
63
        .serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(Red
isSerializer.string()))
64
        .serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(R
edisSerializer.json()));
65
66         return RedisCacheManager
67             .builder(factory)
68             .cacheDefaults(config)
69             .build();
70     }
71 }

```

## (2) 编写RedisService业务工具类

```

1     package com.manong.config.redis;
2
3     import org.springframework.beans.factory.annotation.Autowired;
4     import org.springframework.data.redis.core.RedisTemplate;
5     import org.springframework.stereotype.Component;
6
7     import java.util.concurrent.TimeUnit;
8
9     @Component
10    public class RedisService {
11        @Autowired
12        private RedisTemplate<String, Object> redisTemplate;
13        //存缓存
14        public void set(String key, String value, Long timeOut){
15            redisTemplate.opsForValue().set(key, value, timeOut, TimeUnit.SECONDS);
16        }
17        //取缓存
18        public String get(String key){
19            return (String) redisTemplate.opsForValue().get(key);
20        }
21        //清除缓存
22        public void del(String key){
23            redisTemplate.delete(key);
24        }
25    }

```

### 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。

```

1     #登录请求地址(自定义)
2     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。

```
1  package com.manong.config.security.filter;
2
3  import com.manong.config.redis.RedisService;
4  import com.manong.config.security.exception.CustomerAuthenticationException;
5  import com.manong.config.security.handler.LoginFailureHandler;
6  import com.manong.config.security.service.CustomerUserDetailsService;
7  import com.manong.utils.JwtUtils;
8  import lombok.Data;
9  import org.springframework.beans.factory.annotation.Value;
10 import
    org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
11 import org.springframework.security.core.AuthenticationException;
12 import org.springframework.security.core.context.SecurityContextHolder;
13 import org.springframework.security.core.userdetails.UserDetails;
14 import
    org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
15 import org.springframework.stereotype.Component;
16 import org.springframework.util.ObjectUtils;
17 import org.springframework.web.filter.OncePerRequestFilter;
18
19 import javax.annotation.Resource;
20 import javax.servlet.FilterChain;
21 import javax.servlet.ServletException;
22 import javax.servlet.http.HttpServletRequest;
23 import javax.servlet.http.HttpServletResponse;
24 import java.io.IOException;
25
26 /**
27  * token验证过滤器
28  */
29 @Data
30 @Component
31 public class CheckTokenFilter extends OncePerRequestFilter {
32
33     @Resource
34     private JwtUtils jwtUtils;
35     @Resource
36     private CustomerUserDetailsService customerUserDetailsService;
37     @Resource
38     private LoginFailureHandler loginFailureHandler;
39     @Resource
40     private RedisService redisService;
41     //获取登录请求地址
42     @Value("${request.login.url}")
43     private String loginUrl;
44
45     @Override
46     protected void doFilterInternal(HttpServletRequest request,
47                                     HttpServletResponse response, FilterChain filterChain) throws ServletException,
48                                     IOException {
49         try {
50             //获取当前请求的url地址
51             String url = request.getRequestURI();
52             //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，说明该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,
8          UsernamePasswordAuthenticationFilter.class);
9      http.formLogin()
10         //省略后续代码....

```

### 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
10         IOException, ServletException {
11         //设置客户端响应编码格式
12         response.setContentType("application/json;charset=UTF-8");
13         //获取输出流
14         ServletOutputStream outputStream = response.getOutputStream();

```

```

14     String message = null; //提示信息
15     int code = 500; //错误编码
16     //判断异常类型
17     if(exception instanceof AccountExpiredException){
18         message = "账户过期, 登录失败! ";
19     }else if(exception instanceof BadCredentialsException){
20         message = "用户名或密码错误, 登录失败! ";
21     }else if(exception instanceof CredentialsExpiredException){
22         message = "密码过期, 登录失败! ";
23     }else if(exception instanceof DisabledException){
24         message = "账户被禁用, 登录失败! ";
25     }else if(exception instanceof LockedException){
26         message = "账户被锁, 登录失败! ";
27     }else if(exception instanceof InternalAuthenticationServiceException){
28         message = "账户不存在, 登录失败! ";
29     }else if(exception instanceof CustomerAuthenticationException){
30         message = exception.getMessage();
31         code = 600;
32     }else{
33         message = "登录失败! ";
34     }
35     //将错误信息转换成JSON
36     String result =
JSON.toJSONString(Result.error().code(code).message(message));
37     outputStream.write(result.getBytes(StandardCharsets.UTF_8));
38     outputStream.flush();
39     outputStream.close();
40 }
41 }

```

### 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。

```

1  @Component
2  public class LoginSuccessHandler implements AuthenticationSuccessHandler {
3
4      @Resource
5      private JwtUtils jwtUtils;
6      @Resource
7      private RedisService redisService;
8
9      @Override
10     public void onAuthenticationSuccess(HttpServletRequest request,
        HttpServletResponse response, Authentication authentication) throws IOException,
        ServletException {
11         //省略原有代码.....
12
13         //把生成的token存到redis
14         String tokenKey = "token_"+token;
15         redisService.set(tokenKey, token, jwtUtils.getExpiration() / 1000);
16     }
17 }

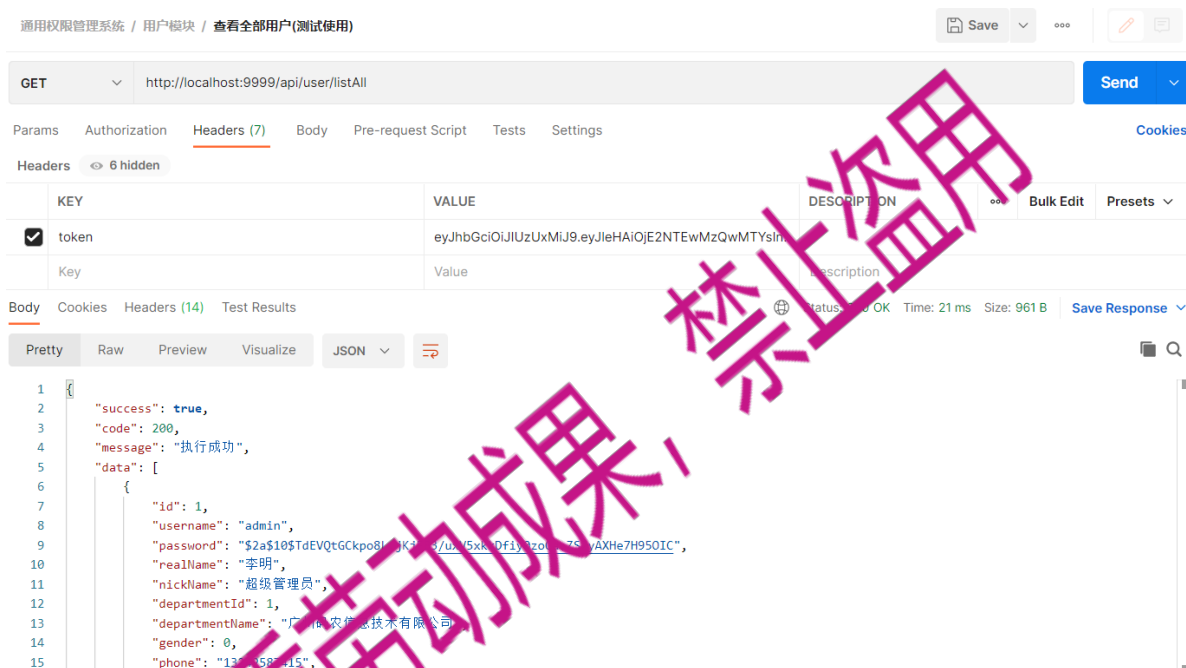
```

### 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。

```
1 package com.manong.vo;  
2  
3 import lombok.AllArgsConstructor;  
4 import lombok.Data;  
5 import lombok.NoArgsConstructor;  
6  
7 @Data  
8 @NoArgsConstructor  
9 @AllArgsConstructor  
10 public class TokenVo {  
11     //过期时间  
12     private Long expireTime;  
13     //token  
14     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。

```
1  package com.manong.controller;
2
3  import com.manong.config.redis.RedisService;
4  import com.manong.utils.JwtUtils;
5  import com.manong.utils.Result;
6  import com.manong.vo.TokenVo;
7  import io.jsonwebtoken.Jwts;
8  import org.springframework.security.core.Authentication;
9  import org.springframework.security.core.context.SecurityContextHolder;
10 import org.springframework.security.core.userdetails.UserDetails;
11 import org.springframework.util.ObjectUtils;
12 import org.springframework.web.bind.annotation.PostMapping;
13 import org.springframework.web.bind.annotation.RequestMapping;
14 import org.springframework.web.bind.annotation.RestController;
15
16 import javax.annotation.Resource;
17 import javax.servlet.http.HttpServletRequest;
18
19 @RestController
20 @RequestMapping("/api/sysUser")
21 public class SysUserController {
22     @Resource
23     private RedisService redisService;
24     @Resource
25     private JwtUtils jwtUtils;
26
27     /**
28      * 刷新token
29      *
30      * @param request
31      * @return
32      */
33     @PostMapping("/refreshToken")
34     public Result refreshToken(HttpServletRequest request) {
35         //从header中获取前端提交的token
36         String token = request.getHeader("token");
37         //如果header中没有token，则从参数中获取
38         if (ObjectUtils.isEmpty(token)) {
39             token = request.getParameter("token");
40         }
41         //从Spring Security上下文获取用户信息
42         Authentication authentication =
43             SecurityContextHolder.getContext().getAuthentication();
44         //获取身份信息
45         UserDetails details = (UserDetails) authentication.getPrincipal();
46         //重新生成token
47         String reToken = "";
48         //验证原来的token是否合法
49         if (jwtUtils.validateToken(token, details)) {
50             //生成新的token
51             reToken = jwtUtils.refreshToken(token);
```



```

51     }
52     //获取本次token的到期时间，交给前端做判断
53     long expireTime = Jwts.parser().setSigningKey(jwtUtils.getSecret())
54         .parseClaimsJws(reToken.replace("jwt_", ""))
55         .getBody().getExpiration().getTime();
56     //清除原来的token信息
57     String oldTokenKey = "token_" + token;
58     redisService.del(oldTokenKey);
59     //存储新的token
60     String newTokenKey = "token_" + reToken;
61     redisService.set(newTokenKey, reToken, jwtUtils.getExpiration() / 1000);
62     //创建TokenVo对象
63     TokenVo tokenVo = new TokenVo(expireTime, reToken);
64     //返回数据
65     return Result.ok(tokenVo).message("token生成成功");
66 }
67 }

```

### 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。

```

1  package com.manong.entity;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6
7  import java.io.Serializable;
8
9  @Data
10 @NoArgsConstructor
11 @AllArgsConstructor
12 public class UserInfo implements Serializable {
13     private Long id;//用户ID
14     private String name;//用户名称
15     private String avatar;//头像
16     private String introduction;//介绍
17     private Object[] roles;//角色权限集合
18 }

```

### 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。

```
1  /**
2   * 获取用户信息
3   *
4   * @return
5   */
6  @GetMapping("/getInfo")
7  public Result getInfo() {
8      //从Spring Security上下文获取用户信息
9      Authentication authentication =
10      SecurityContextHolder.getContext().getAuthentication();
11      //判断authentication对象是否为空
12      if (authentication == null) {
13          return Result.error().message("用户信息查询失败");
14      }
15      //获取用户信息
16      User user = (User) authentication.getPrincipal();
17      //用户权限集合
18      List<Permission> permissionList = user.getPermissionList();
19      //获取角色权限编码字段
20      Object[] roles =
21      permissionList.stream()
22      .filter(Objects::nonNull)
23      .map(Permission::getCode).toArray();
24      //创建用户信息对象
25      UserInfo userInfo = new UserInfo(user.getId(), user.getNickName(),
26      user.getAvatar(), null, roles);
27      //返回数据
28      return Result.ok(userInfo).message("用户信息查询成功");
29  }
```

## 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。

```
1  package com.manong.vo;
2
3  import com.fasterxml.jackson.annotation.JsonInclude;
4  import lombok.AllArgsConstructor;
5  import lombok.Data;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 @Data
11 @JsonInclude(JsonInclude.Include.NON_EMPTY)
12 public class RouterVo {
13     //路由地址
14     private String path;
15     //路由对应的组件
```

```

16     private String component;
17     //是否显示
18     private boolean alwaysShow;
19     //路由名称
20     private String name;
21     //路由meta信息
22     private Meta meta;
23
24     @Data
25     @AllArgsConstructor
26     public class Meta {
27         private String title;//标题
28         private String icon;//图标
29         private Object[] roles;//角色列表
30     }
31
32     //子路由
33     private List<RouterVo> children = new ArrayList<RouterVo>();
34
35 }

```

#### 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。

```

1     package com.manong.utils;
2
3     import com.manong.entity.Permission;
4     import com.manong.vo.RouterVo;
5     import org.springframework.beans.BeanUtil;
6
7     import java.util.ArrayList;
8     import java.util.List;
9     import java.util.Objects;
10    import java.util.Optional;
11
12    /**
13     * 生成菜单树
14     */
15    public class MenuTree {
16
17        /**
18         * 生成路由
19         *
20         * @param meulist 菜单列表
21         * @param pid     父级菜单
22         * @return
23         */
24        public static List<RouterVo> makeRouter(List<Permission> meulist, Long pid) {
25            //创建集合保存路由列表
26            List<RouterVo> routerList = new ArrayList<RouterVo>();
27            //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
28            Optional.ofNullable(meulist).orElse(new ArrayList<Permission>())
29                //筛选不为空的菜单及菜单父id相同的数据
30                .stream().filter(item -> item != null && item.getParentId() ==
pid)
31                .forEach(item -> {
32                    //创建路由对象

```

```

33         RouterVo router = new RouterVo();
34         router.setName(item.getName()); //路由名称
35         router.setPath(item.getPath()); //路由地址
36         //判断是否是一级菜单
37         if (item.getParentId() == 0L) {
38             router.setComponent("Layout"); //一级菜单组件
39             router.setAlwaysShow(true); //显示路由
40         } else {
41             router.setComponent(item.getUrl()); //具体的组件
42             router.setAlwaysShow(false); //折叠路由
43         }
44         //设置meta信息
45         router.setMeta(router.new Meta(item.getLabel(),
46                                         item.getIcon(),
47                                         item.getCode().split(",")));
48         //递归生成路由
49         List<RouterVo> children = makeRouter(meulist, item.getId());
50         router.setChildren(children); //设置子路由到路由对象中
51         //将路由信息添加到集合中
52         routerList.add(router);
53     });
54     return routerList;
55 }
56
57 /**
58  * 生成菜单树
59  *
60  * @param meulist
61  * @param pid
62  * @return
63  */
64 public static List<Permission> makeMenuTree(List<Permission> meulist, Long
pid) {
65     //创建集合保存菜单
66     List<Permission> permissionList = new ArrayList<Permission>();
67     //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
68     Optional.ofNullable(meulist).orElse(new ArrayList<Permission>())
69     .stream().filter(item -> item != null &&
Objects.equals(item.getParentId(), pid))
70     .forEach(item -> {
71         //创建菜单权限对象
72         Permission permission = new Permission();
73         //复制属性
74         BeanUtils.copyProperties(item, permission);
75         //获取每一个item的下级菜单, 递归生成菜单树
76         List<Permission> children = makeMenuTree(meulist,
item.getId());
77         //设置子菜单
78         permission.setChildren(children);
79         //将菜单对象添加到集合
80         permissionList.add(permission);
81     });
82     return permissionList;
83 }
84 }

```

## 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。

```
1  /**
2   * 获取菜单数据
3   *
4   * @return
5   */
6  @GetMapping("/getMenuList")
7  public Result getMenuList() {
8      //从Spring Security上下文获取用户信息
9      Authentication authentication =
10      SecurityContextHolder.getContext().getAuthentication();
11      //获取用户信息
12      User user = (User) authentication.getPrincipal();
13      //获取相应的权限
14      List<Permission> permissionList = user.getPermissionList();
15      //筛选目录和菜单
16      List<Permission> collect = permissionList.stream()
17          .filter(item -> item != null && item.getType() != 0)
18          .collect(Collectors.toList());
19      //生成路由数据
20      List<RouterVo> routerVoList = MenuTree.makeRouter(collect, 0L);
21      //返回数据
22      return Result.ok(routerVoList).message("菜单数据获取成功");
23  }
```

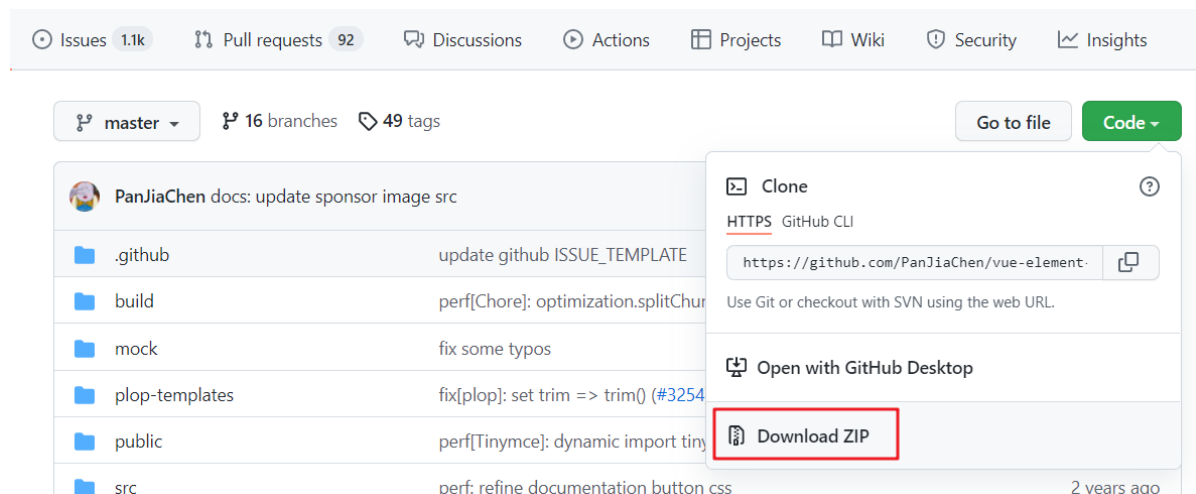
## 5. 搭建前端项目环境

### 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>

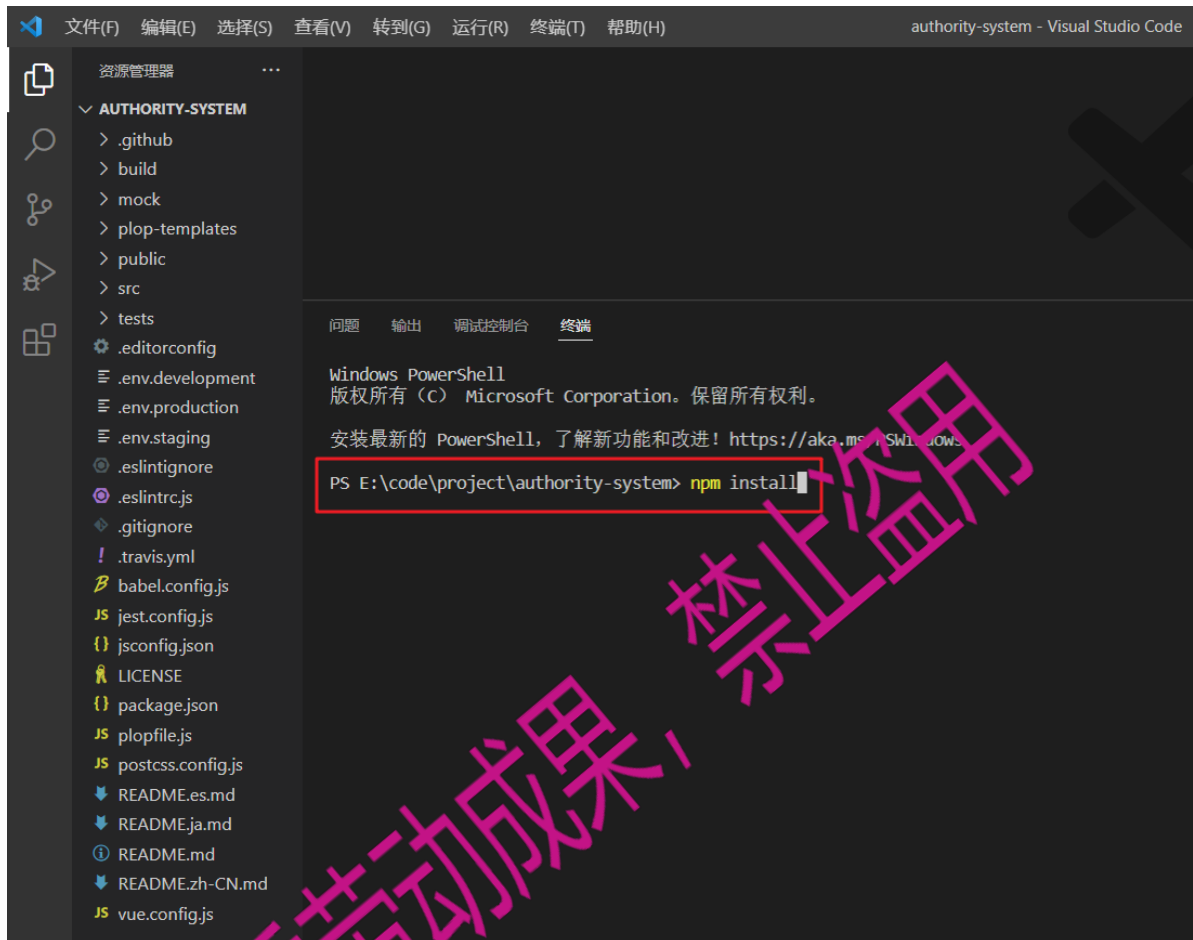


### 5.1.2 运行vue-element-admin

注意：在运行前必须具备node.js、git和python运行环境。

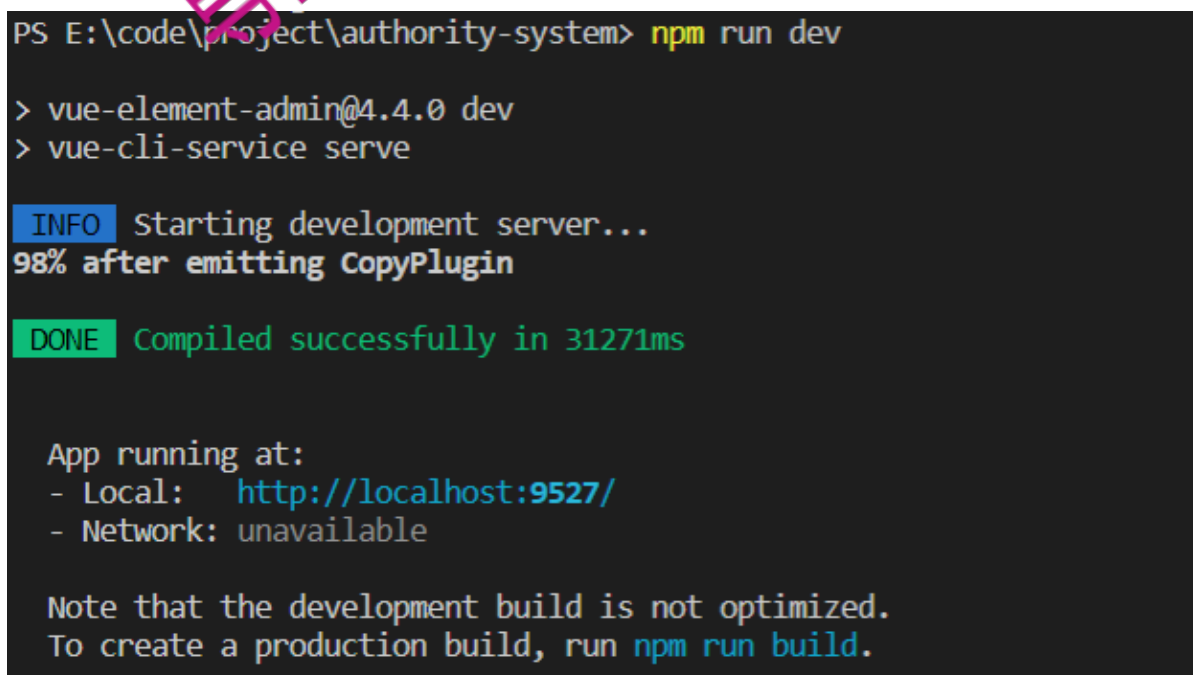
(1) 将下载好的vue-element-admin解压，解压后放置到具体的某个目录下，可以对当前项目进行重命名，此处重命名为 **authority-system**。

(2) 进入项目的根目录，执行 **npm install** 指令。

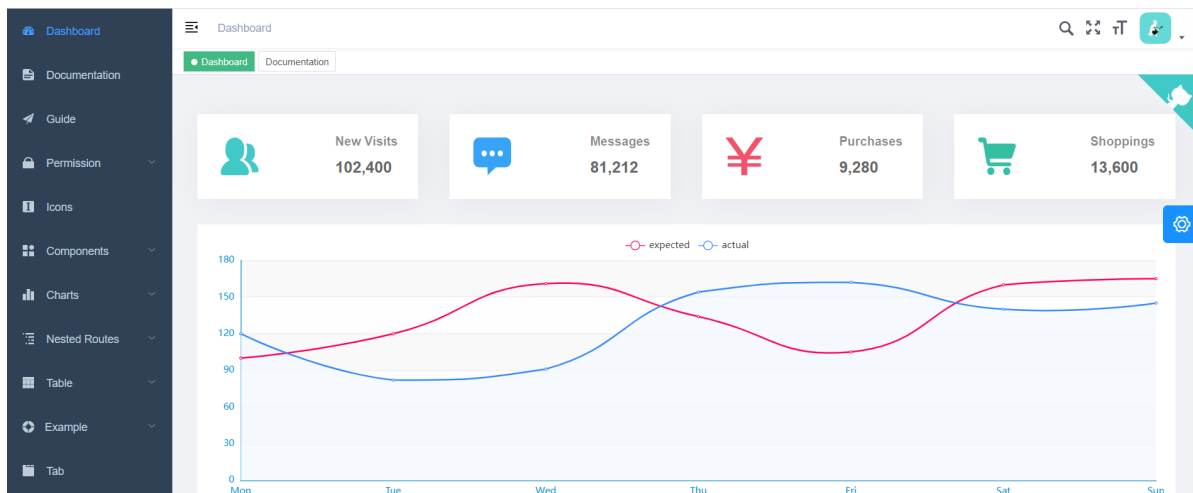


等待命令成功执行完成，此处需要具备node.js、git和python运行环境。

(3) 在项目根目录下使用 **npm run dev** 指令运行项目。



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。

```
1  "husky": {
2    "hooks": {
3      "pre-commit": "lint-staged"
4    }
5  }
```

(3) .eslintignore

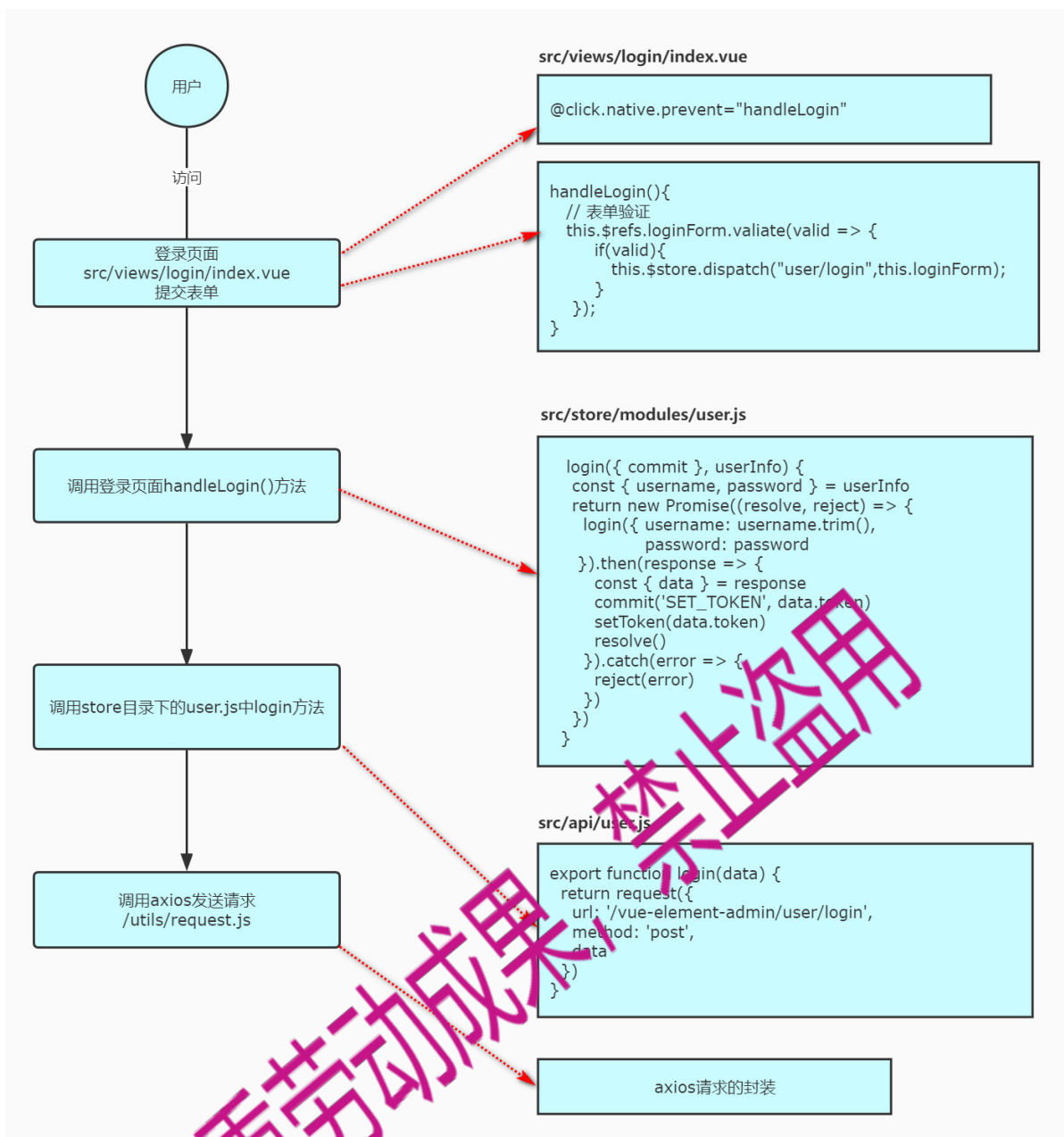
在.eslintignore文件最后一行加入 `*`。

```
1  build/*.js
2  src/assets
3  public
4  dist
5  *
```

## 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：

```
1  handleLogin() {
2    //表单验证
3    this.$refs.loginForm.validate((valid) => {
4      //如果验证通过
5      if (valid) {
6        //开启进度条
7        this.loading = true;
8        //调用src/store/modules/user.js中的login方法
9        this.$store
10       .dispatch("user/login", this.loginForm)
11       .then(() => {
12         //路由转发到指定地址
13         this.$router.push({
14           path: this.redirect || "/",
15           query: this.otherQuery,
16         });
17         //关闭进度条
```



```

18         this.loading = false;
19     })
20     .catch(() => {
21         //关闭进度条
22         this.loading = false;
23     });
24 } else {
25     console.log("error submit!!");
26     return false;
27 }
28 });
29 }

```

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

```

1  // 用户登录
2  login({ commit }, userInfo) {
3      //从用户信息userInfo中解构出用户名和密码
4      const { username, password } = userInfo
5      return new Promise((resolve, reject) => {
6          //调用src/api/user.js文件中的login()方法
7          login({ username: username.trim(), password: password }).then(response => {
8              //从response中解构出返回的data数据
9              const { data } = response
10             //将返回的token数据保存到store中，作为全局变量使用
11             commit('SET_TOKEN', data.token)
12             //将token信息保存到cookie中
13             setToken(data.token)
14             resolve()
15         }).catch(error => {
16             reject(error)
17         })
18     })
19 }

```

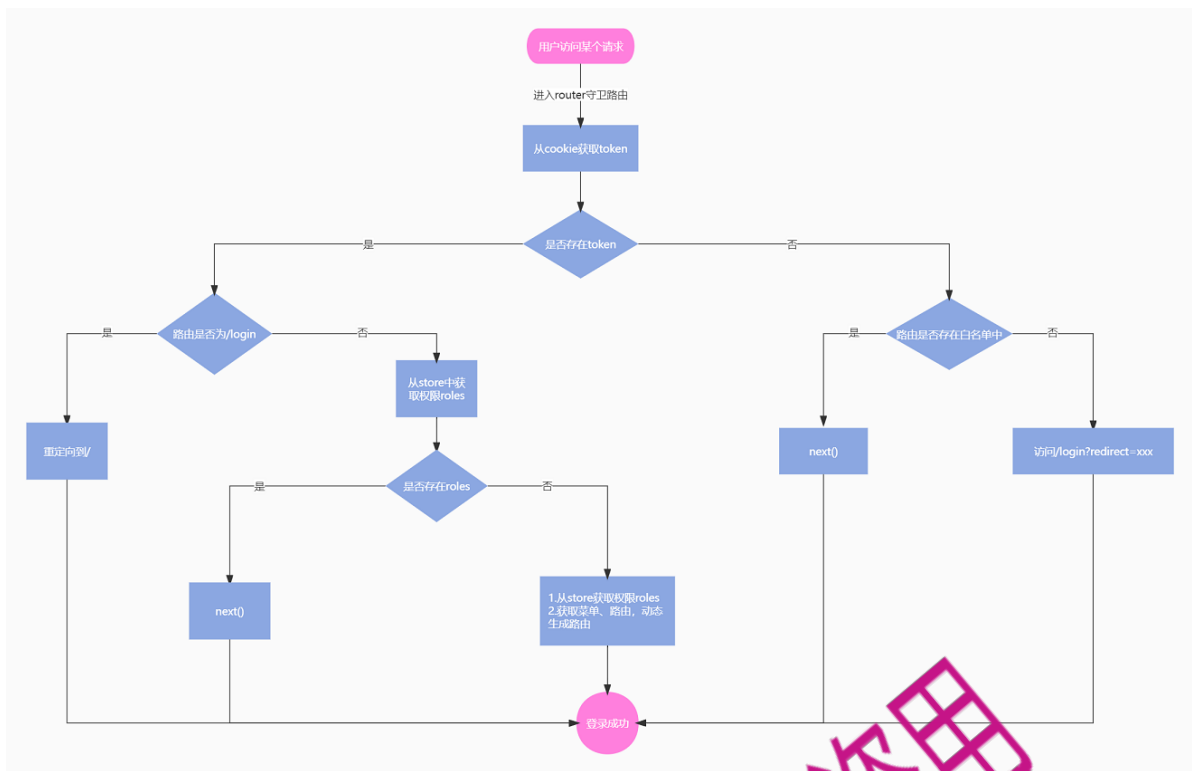
(3) store中调用的login，即工程中api/user中的login方法，此处主要是通过utils/request的axios发送请求到后端获取token。

```

1  //注意：此处调用的是mock数据，并非真实的后端接口数据
2  //mock数据在工程的 mock/user.js里面，此处的token是写死的
3  export function login(data){
4      return request({
5          url: '/vue-element-admin/user/login',
6          method: 'post',
7          data
8      })
9  }

```

### 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

通用权限管理系统

请输入用户名

用户名不能为空

请输入密码

密码不能为空

登录

## 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：

```
1  <template>
2    <div class="login-container">
3      <el-form
4        ref="loginForm"
5        :model="loginForm"
6        :rules="loginRules"
7        class="login-form"
8        autocomplete="on"
9        label-position="left"
10     >
11      <div class="title-container">
12        <h3 class="title">通用权限管理系统</h3>
13      </div>
14
15      <el-form-item prop="username">
16        <span class="svg-container">
17          <svg-icon icon-class="user" />
18        </span>
19        <el-input
20          ref="username"
21          v-model="loginForm.username"
22          placeholder="请输入用户名"
23          name="username"
24          type="text"
25          tabindex="1"
26          autocomplete="on"
27        />
28      </el-form-item>
29
30      <el-tooltip
31        v-model="capsTooltip"
32        content="Caps lock is On"
33        placement="right"
34        manual
35      >
36      <el-form-item prop="password">
37        <span class="svg-container">
38          <svg-icon icon-class="password" />
39        </span>
40        <el-input
41          :key="passwordType"
42          ref="password"
43          v-model="loginForm.password"
44          :type="passwordType"
45          placeholder="请输入密码"
46          name="password"
47          tabindex="2"
48          autocomplete="on"
49          @keyup.native="checkCapslock"
50          @blur="capsTooltip = false"
51          @keyup.enter.native="handleLogin"
52        />
53        <span class="show-pwd" @click="showPwd">
54          <svg-icon
```

```

55         :icon-class="passwordType === 'password' ? 'eye' : 'eye-open'"
56     />
57 </span>
58 </el-form-item>
59 </el-tooltip>
60
61 <el-button
62   :loading="loading"
63   type="primary"
64   style="width: 100%; margin-bottom: 30px"
65   @click.native.prevent="handleLogin"
66   >登录</el-button>
67 >
68 </el-form>
69 </div>
70 </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：

```

1  export default {
2    name: "Login",
3    data() {
4      return {
5        // 登录表单对象
6        loginForm: {
7          username: "", // 用户名
8          password: "", // 密码
9        },
10       // 表单验证规则
11       loginRules: {
12         username: [
13           { required: true, trigger: "blur", message: "用户名不能为空" },
14         ],
15         password: [
16           { required: true, trigger: "blur", message: "密码不能为空" },
17         ],
18       },
19     };
20   },
21   methods: {
22     /**
23      * 登录处理
24      */
25     handleLogin() {
26       // 表单验证
27       this.$refs.loginForm.validate((valid) => {
28         // 如果验证通过
29         if (valid) {
30           // 开启进度条
31           this.loading = true;
32           // 调用src/store/modules/user.js中的login方法
33           this.$store
34             .dispatch("user/login", this.loginForm)
35             .then(() => {
36               // 路由转发到指定地址
37               this.$router.push({

```

```

38         path: this.redirect || "/",
39         query: this.otherQuery,
40     });
41     //关闭进度条
42     this.loading = false;
43 }
44 .catch(() => {
45     //关闭进度条
46     this.loading = false;
47 });
48 } else {
49     console.log("error submit!!");
50     return false;
51 }
52 });
53 }
54 };

```

### 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。

```

1  devServer: {
2    port: port,
3    open: true,
4    overlay: {
5      warnings: false,
6      errors: true
7    },
8    // before: require('./mock/mock-server.js')
9    //代理配置
10   proxy: {
11     [process.env.VUE_APP_BASE_API]: {
12       target: "http://localhost:8089/api",
13       changeOrigin: true,
14       pathRewrite: {
15         '/api': ''
16       }
17     }
18   }
19 }

```

#### (2) 修改main.js

将main.js文件中的如下代码注释。

```

1  if (process.env.NODE_ENV === 'production') {
2    const { mockXHR } = require('../mock')
3    mockXHR()
4  }

```

#### (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=&&&

```

1  npm install qs

```

随后将如下代码替换调用原有的request.js文件。

```
1  import axios from 'axios'
2  import { MessageBox, Message } from 'element-ui'
3  import store from '@/store'
4  import { getToken } from '@/utils/auth'
5  import qs from 'qs'
6  // create an axios instance
7  const service = axios.create({
8    baseURL: process.env.VUE_APP_BASE_API, // url = base url + request url
9    // withCredentials: true, // send cookies when cross-domain requests
10   timeout: 5000 // request timeout
11 })
12
13 // request interceptor
14 service.interceptors.request.use(
15   config => {
16     console.log(config)
17     // do something before request is sent
18
19     if (store.getters.token) {
20       // let each request carry token
21       // ['X-Token'] is a custom headers key
22       // please modify it according to the actual situation
23       config.headers['token'] = getToken()
24     }
25     return config
26   },
27   error => {
28     // do something with request error
29     console.log(error) // for debug
30     return Promise.reject(error)
31   }
32 )
33
34 // response interceptor
35 service.interceptors.response.use(
36   /**
37    * If you want to get http information such as headers or status
38    * Please return response => response
39    */
40
41   /**
42    * Determine the request status by custom code
43    * Here is just an example
44    * You can also judge the status by HTTP Status Code
45    */
46   response => {
47     const res = response.data
48
49     // if the custom code is not 20000, it is judged as an error.
50     if (res.code !== 200) {
51       Message({
52         message: res.message || 'Error',
53         type: 'error',
54         duration: 5 * 1000
55       })
56     }
```

```

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 !== '') {
127         _params += `${params[key]}/`
128       }
129     }
130     _params = _params.substr(0, _params.length - 1)
131   }
132   console.log(_params)
133   if (_params) {
134     return service.get(`${url}${_params}`)
135   } else {
136     return service.get(url)
137   }
138 },
139 delete(url, params) {
140   let _params
141   if (Object.is(params, undefined || null)) {
142     _params = ''
143   } else {
144     _params = '/'
145     for (const key in params) {
146       // eslint-disable-next-line no-prototype-builtins
147       if (params.hasOwnProperty(key) && params[key] !== null && params[key]
148 !== '') {
149         _params += `${params[key]}/`
150       }
151     }
152     _params = _params.substr(0, _params.length - 1)
153   }
154   if (_params) {
155     return service.delete(`${url}${_params}`).catch(err => {
156       message.error(err.msg)
157       return Promise.reject(err)
158     })
159   } else {
160     return service.delete(url).catch(err => {
161       message.error(err.msg)
162       return Promise.reject(err)
163     })
164   }
165 },
166 upload(url, params) {
167   return service.post(url, params, {
168     headers: {
169       'Content-Type': 'multipart/form-data'
170     }
171   })

```



```

169     })
170   },
171   login(url, params) {
172     return service.post(url, params, {
173       transformRequest: [(params) => {
174         return qs.stringify(params)
175       }],
176       headers: {
177         'Content-Type': 'application/x-www-form-urlencoded'
178       }
179     })
180   }
181 }
182 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：

```

1  import http from '@utils/request'
2
3  /**
4   * 用户登录
5   * @returns
6   */
7  export async function login(data) {
8    return await http.login("/api/user/login", data)
9  }
10
11  /**
12   * 获取用户信息和权限信息
13   * @returns
14   */
15  export async function getInfo({
16    return await http.get("/api/sysUser/getInfo")
17  })
18
19  /**
20   * 退出登录
21   * @returns
22   */
23  export async function logout(param){
24    return await http.post("/api/sysUser/loginOut", param);
25  }

```

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

注意：该请求地址

```

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

```

#### (6) 修改src/store/modules/user.js中的登录方法。

```

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

```

```

7      //调用src/api/user.js文件中的login()方法
8      login({ username: username.trim(), password: password }).then(response => {
9          //从response中解构出返回的token数据
10         const { token } = response
11         //将返回的token数据保存到store中，作为全局变量使用
12         commit('SET_TOKEN', token)
13         //将token信息保存到cookie中
14         setToken(token)
15         resolve()
16     }).catch(error => {
17         reject(error)
18     })
19 })
20 }

```

(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。

```

1  /**
2   * 用户退出
3   * @param request
4   * @param response
5   * @return
6   */
7  @PostMapping("/logout")
8  public Result logout(HttpServletRequest request, HttpServletResponse response) {
9      //获取token
10     String token = request.getParameter("token");
11     //如果没有从头部获取token，那么从参数里面获取
12     if (ObjectUtils.isEmpty(token)) {
13         token = request.getHeader("token");
14     }
15     //获取用户相关信息
16     Authentication authentication =
17     SecurityContextHolder.getContext().getAuthentication();
18     if (authentication != null) {
19         //清空用户信息
20         new SecurityContextLogoutHandler().logout(request, response,
21         authentication);
22         //清空redis里面的token
23         String key = "token_" + token;
24         redisService.del(key);
25     }
26     return Result.ok().message("用户退出成功");
27 }

```

## 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：

```
1  /**
2   * 退出登录
3   * @returns
4   */
5  export async function logout(param){
6    return await http.post("/api/sysUser/logout",param);
7  }
```

### 6.3.2 编写清空sessionStorage方法

在src/utils/auth.js中添加如下方法：

```
1  /**
2   * 清空sessionStorage
3   */
4  export function clearStorage(){
5    return sessionStorage.clear();
6  }
```

### 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

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

(2) 修改页面下拉菜单

```
1  <el-dropdown-menu slot="dropdown">
2    <router-link to="/">
3      <el-dropdown-item>首页</el-dropdown-item>
4    </router-link>
5    <router-link to="/profile/index">
6      <el-dropdown-item>个人中心</el-dropdown-item>
7    </router-link>
8    <el-dropdown-item divided @click.native="logout">
9      <span style="display: block">退出系统</span>
10   </el-dropdown-item>
11 </el-dropdown-menu>
```

(3) 在methods中添加loginOut()方法

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

```

3      confirmButtonText: "确定",
4      cancelButtonText: "取消",
5      type: "warning",
6    }).then(async() => {
7      //请求参数
8      let params = {token:getToken()};
9      //发送退出请求
10     let res = await logout(params);
11     //判断是否成功
12     if(res.success){
13       //清空token
14       removeToken();
15       clearStorage();
16       //跳转到登录页面
17       window.location.href="/login";
18     }
19   });
20 }

```

## 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/utls目录下创建myconfirm.js文件，代码如下所示

```

1  import { MessageBox, Message } from 'element-ui'
2  //删除弹框
3  export default function myconfirm(text) {
4    return new Promise((resolve, reject) => {
5      MessageBox.confirm(text, '系统提示', {
6        confirmButtonText: '确定',
7        cancelButtonText: '取消',
8        type: 'warning'
9      }).then(() => {
10        resolve(true)
11      }).catch(() => {
12        reject(false)
13      })
14    }).catch(()=>{
15    })
16  })
17 }

```

### 6.4.2 挂载信息确认提示框

在main.js中将myconfirm挂载到vue上。

```

1  //导入封装信息确认提示框组件脚本
2  import myconfirm from '@/utils/myconfirm'
3  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：

```

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

```

```

4     if (confirm) {
5         //请求参数
6         let params = { token: getToken() };
7         //发送退出请求
8         let res = await logout(params);
9         //判断是否成功
10        if (res.success) {
11            //清空token
12            removeToken();
13            clearStorage();
14            //跳转到登录页面
15            window.location.href = "/login";
16        }
17    }
18 }

```

## 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。

```

1  /**
2   * 获取菜单数据
3   */
4  export async function getMenuList(){
5      return await http.get('/api/sysUser/getMenuList');
6  }

```

#### 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：

```

1  //导入获取菜单数据的方法
2  import { getMenuList } from '@api/user';
3
4  const actions = {
5      generateRoutes({ commit }, roles) {
6          return new Promise((resolve, reject) => {
7              getMenuList().then(res=>{
8                  let accessedRoutes;//存放对应权限的路由信息
9                  //如果状态码为200，则表示成功
10                 if(res.code === 200){
11                     accessedRoutes = filterAsyncRoutes(res.data, roles)
12                 }
13                 //将路由信息保存到store中
14                 commit("SET_ROUTES", accessedRoutes);
15                 resolve(accessedRoutes);
16             }).catch(error=>{

```

```

17     reject(error);
18   });
19 })
20 }
21 }

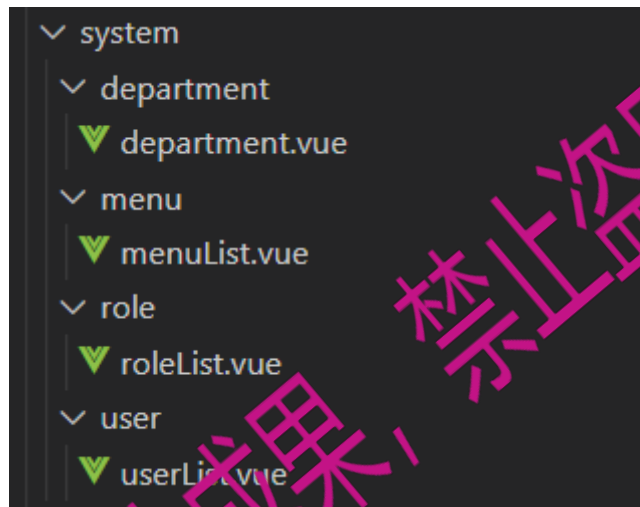
```

## 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：

```

1  import { constantRoutes } from '@/router'
2  // 导入Layout组件
3  import Layout from '@/layout'
4  /**
5   * 过滤出所有拥有权限的菜单
6   * @param routes asyncRoutes
7   * @param roles
8   */
9  export function filterAsyncRoutes(routes, roles) {
10    const res = []
11    routes.forEach(route => {
12      const tmp = { ...route }
13      // 判断是否有相应的权限
14      if (hasPermission(roles, tmp)) {
15        // 获取组件
16        const component = tmp.component;
17        // 判断该路由使用有组件
18        if (route.component) {
19          // 判断是否是根组件
20          if (component === 'Layout') {
21            tmp.component = Layout;
22          } else {

```

```

23         //获取对应的具体的组件信息
24         tmp.component = (resolve) => require(['@/views${component}'], resolve)
25     }
26 }
27 //判断是否有子菜单
28 if (tmp.children) {
29     tmp.children = filterAsyncRoutes(tmp.children, roles)
30 }
31 res.push(tmp)
32 }
33 })
34
35 return res
36 }

```

## 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：

```

1  /**
2   * 是否展开
3   */
4  @TableField(exist = false)
5  private Boolean open,
6
7  /**
8   * 子部门
9   */
10 @TableField(exist = false)
11 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。

```

1  package com.manong.vo.query;
2
3  import com.manong.entity.Department;
4  import lombok.Data;
5
6  /**
7   * 查询条件类
8   */
9  @Data
10 public class DepartmentQueryVo extends Department {
11 }

```

### 8.1.3 DepartmentTree

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

```
1  package com.manong.utils;
2
3  import com.manong.entity.Department;
4  import org.springframework.beans.BeanUtils;
5
6  import java.util.ArrayList;
7  import java.util.List;
8  import java.util.Optional;
9
10 public class DepartmentTree {
11     /**
12      * 生成部门树
13      *
14      * @param deptList
15      * @param pid
16      * @return
17      */
18     public static List<Department> makeDepartmentTree(List<Department> deptList,
19 Long pid) {
20         //创建集合保存部门信息
21         List<Department> list = new ArrayList<Department>();
22         //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
23         Optional.ofNullable(deptList).orElse(new ArrayList<Department>())
24             .stream().filter(item -> item != null && item.getPid() == pid)
25             .forEach(item -> {
26                 //创建部门对象
27                 Department dept = new Department();
28                 //复制属性
29                 BeanUtils.copyProperties(item, dept);
30                 //获取每个item的下级部门，递归生成部门树
31                 List<Department> children = makeDepartmentTree(deptList,
32 item.getId());
33                 //设置子部门
34                 dept.setChildren(children);
35                 //将部门对象添加到集合
36                 list.add(dept);
37             });
38     }
39 }
```

### 8.1.4 DepartmentService

```
1  public interface DepartmentService extends IService<Department> {
2      /**
3       * 查询部门列表
4       * @param departmentQueryVo
5       * @return
6       */
7      List<Department> findDepartmentList(DepartmentQueryVo departmentQueryVo);
8
9      /**
10     * 查询上级部门列表
```



```

11     * @return
12     */
13     List<Department> findParentDepartment();
14
15     /**
16     * 判断部门下是否有子部门
17     * @param id
18     * @return
19     */
20     boolean hasChildrenOfDepartment(Long id);
21
22     /**
23     * 判断部门下是否存在用户
24     * @param id
25     * @return
26     */
27     boolean hasUserOfDepartment(Long id);
28 }

```

### 8.1.5 DepartmentServiceImpl

```

1  @Service
2  @Transactional
3  public class DepartmentServiceImpl extends ServiceImpl<DepartmentMapper,
4      Department> implements DepartmentService {
5
6      @Resource
7      private UserMapper userMapper;
8
9      /**
10     * 查询部门列表
11     *
12     * @param departmentQueryVo
13     * @return
14     */
15     @Override
16     public List<Department> findDepartmentList(DepartmentQueryVo
17         departmentQueryVo) {
18         // 创建条件构造器对象
19         QueryWrapper<Department> queryWrapper = new QueryWrapper<Department>();
20         // 部门名称
21
22         queryWrapper.like(!ObjectUtils.isEmpty(departmentQueryVo.getDepartmentName()), "d
23 epartment_name", departmentQueryVo.getDepartmentName());
24         // 排序
25         queryWrapper.orderByAsc("order_num");
26         // 查询部门列表
27         List<Department> departmentList = baseMapper.selectList(queryWrapper);
28         // 生成部门树
29         List<Department> departmentTree =
30             DepartmentTree.makeDepartmentTree(departmentList, 0L);
31         return departmentTree;
32     }
33
34     /**
35     * 查询上级部门列表
36     *

```

```

32     * @return
33     */
34     @Override
35     public List<Department> findParentDepartment() {
36         //创建条件构造器对象
37         QueryWrapper<Department> queryWrapper = new QueryWrapper<Department>();
38         //排序
39         queryWrapper.orderByAsc("order_num");
40         //查询部门列表
41         List<Department> departmentList = baseMapper.selectList(queryWrapper);
42         //创建部门对象
43         Department department = new Department();
44         department.setId(0L);
45         department.setDepartmentName("顶级部门");
46         department.setPid(-1L);
47         departmentList.add(department);
48         //生成部门树列表
49         List<Department> departmentTree =
DepartmentTree.makeDepartmentTree(departmentList, -1L);
50         //返回部门列表
51         return departmentTree;
52     }
53
54     /**
55      * 判断部门下是否有子部门
56      *
57      * @param id
58      * @return
59      */
60     @Override
61     public boolean hasChildrenOfDepartment(Long id) {
62         //创建条件构造器对象
63         QueryWrapper<Department> queryWrapper = new QueryWrapper<Department>();
64         queryWrapper.eq("pid", id);
65         //如果数量大于0, 表示存在
66         if(baseMapper.selectCount(queryWrapper)>0){
67             return true;
68         }
69         return false;
70     }
71
72     /**
73      * 判断部门下是否存在用户
74      *
75      * @param id
76      * @return
77      */
78     @Override
79     public boolean hasUserOfDepartment(Long id) {
80         //创建条件构造器对象
81         QueryWrapper<User> queryWrapper = new QueryWrapper<User>();
82         queryWrapper.eq("department_id", id);
83         //如果数量大于0, 表示存在
84         if(userMapper.selectCount(queryWrapper)>0){
85             return true;
86         }
87         return false;
88     }

```

### 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 =
29 departmentService.findDepartmentList(departmentQueryVo);
30         //返回数据
31         return Result.ok(departmentList);
32     }
33
34     /**
35      * 查询上级部门列表
36      * @return
37      */
38     @GetMapping("/parent/list")
39     public Result getParentDepartment(){
40         //调用查询上级部门列表方法
41         List<Department> departmentList =
42 departmentService.findParentDepartment();
43         //返回数据
44         return Result.ok(departmentList);
45     }
46
47     /**
48      * 添加部门
49      * @param department
50      * @return
51      */
52     @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.hasChildronOfDepartment(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。

```
1  /**
2   * 是否存在
3   * @return
4   */
5  public static<T> Result<T> exist(){
6      Result<T> result = new Result<T>();
7      result.setSuccess(false); //存在该数据
8      //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
9      result.setCode(ResultCode.SUCCESS); //执行成功，但存在该数据
10     result.setMessage("该数据存在");
11     return result;
12 }
```

## 8.1.8 逻辑删除配置

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

```
1  #全局逻辑删除的实体字段名
2  mybatis-plus.global-config.db-config.logic-delete-field=isDelete
3  #逻辑删除值，默认为1
4  mybatis-plus.global-config.db-config.logic-delete-value=1
5  #逻辑未删除值，默认为0
6  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



部门名称	所属部门	部门电话	部门位置	操作
广州市天河区	顶级部门	020-8888888	广州市天河区	<a href="#">编辑</a> <a href="#">删除</a>
广州码农信息技术有限公司	广州码农信息技术有限公司	020-88881001	广州市天河区	<a href="#">编辑</a> <a href="#">删除</a>
人事管理部	广州码农信息技术有限公司	020-88881002	广州市天河区	<a href="#">编辑</a> <a href="#">删除</a>
市场管理部	广州码农信息技术有限公司	020-88881003	广州市天河区	<a href="#">编辑</a> <a href="#">删除</a>
软件研发部	广州码农信息技术有限公司	020-88881234	广州市天河区	<a href="#">编辑</a> <a href="#">删除</a>

### 8.2.2 页面原型代码

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

```
1  <template>
2    <el-main>
3      <!-- 查询条件 -->
4      <el-form
5        ref="searchForm"
6        label-width="80px"
7        :inline="true"
8        size="small"
9      >
10     <el-form-item>
```

```

11     <el-input placeholder="请输入部门名称"></el-input>
12   </el-form-item>
13   <el-form-item>
14     <el-button type="primary" icon="el-icon-search">查询</el-button>
15     <el-button icon="el-icon-refresh-right">重置</el-button>
16     <el-button type="success" icon="el-icon-plus">新增</el-button>
17   </el-form-item>
18 </el-form>
19 <!-- 数据表格 -->
20 <el-table
21   :data="tableData"
22   border
23   stripe
24   style="width: 100%; margin-bottom: 20px"
25   row-key="id"
26   default-expand-all
27   :tree-props="{ children: 'children' }"
28 >
29   <el-table-column prop="name" label="部门名称"></el-table-column>
30   <el-table-column prop="parentName" label="所属部门"></el-table-column>
31   <el-table-column prop="deptCode" label="部门编码"></el-table-column>
32   <el-table-column prop="deptPhone" label="部门电话"></el-table-column>
33   <el-table-column label="操作" width="200" align="center">
34     <template slot-scope="scope">
35       <el-button
36         icon="el-icon-edit-outline"
37         type="primary"
38         size="small"
39         @click="handleEdit(scope.row)"
40       >编辑
41     </el-button>
42     >
43     <el-button
44       icon="el-icon-close"
45       type="danger"
46       size="small"
47       @click="handleDelete(scope.row)"
48     >删除
49   </el-button>
50   >
51   </template>
52 </el-table-column>
53 </el-table>
54 </el-main>
55 </template>
56
57 <script>
58 export default {
59   name: 'department',
60   data() {
61     return {
62       tableData: [] // 表格数据列表
63     }
64   },
65   methods: {
66     /**
67      * 编辑部门
68      * @param row

```

```

69      */
70      handleEdit(row) {
71
72      },
73      /**
74       * 删除部门
75       * @param row
76       */
77      handleDelete(row) {
78
79      },
80    },
81    created() {
82
83    }
84  }
85 </script>
86
87 <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。

```

1  import http from '@utils/request';
2
3  export default {
4    /**
5     * 查询部门列表
6     * @param param
7     * @returns
8     */
9    async getDepartmentList(param) {
10      return await http.get('/api/department/list', param);
11    }
12  }

```

### 8.2.4 编写页面组件

(1) 添加组件脚本代码

```

1  //导入api脚本
2  import departmentApi from "@api/department";
3
4  export default {
5    name: "department",
6    data() {
7      return {
8        searchModel: {
9          departmentName: "", //部门名称
10        },
11        tableData: [], //表格数据列表
12      };
13    },
14    methods: {
15      /**
16       * 查询部门列表
17       */

```

```

18   async search() {
19       //发送查询请求
20       let res = await departmentApi.getDepartmentList(this.searchModel);
21       //判断是否存在数据
22       if (res.success) {
23           //获取数据
24           this.tableData = res.data;
25       }
26   },
27   /**
28    * 重置
29    */
30   resetValue(){
31       this.searchModel = {}; //清空数据
32       this.search(); //重新调用方法
33   },
34   },
35   //初始化时执行
36   created() {
37       //调用查询部门列表方法
38       this.search();
39   },
40   };

```

## (2) 编写页面组件代码

注意：表单属性设置及表格属性设置

```

1   <template>
2       <el-main>
3           <!-- 查询条件 -->
4           <el-form :model="searchModel" ref="searchForm" label-width="80px"
5               :inline="true" size="small">
6               <el-form-item>
7                   <el-input placeholder="请输入部门名称" v-model="searchModel.departmentName">
8               </el-input>
9               <el-form-item>
10                  <el-form-item>
11                      <el-button type="primary" icon="el-icon-search" @click="search()">查询
12                  </el-button>
13                  <el-button icon="el-icon-refresh-right" @click="resetValue()">重置</el-
14                  button>
15                  <el-button type="success" icon="el-icon-plus">新增</el-button>
16              </el-form-item>
17          </el-form>
18          <!-- 数据表格 -->
19          <el-table
20              :data="tableData"
21              border
22              stripe
23              style="width: 100%; margin-bottom: 20px"
24              row-key="id"
25              default-expand-all
26              :tree-props="{ children: 'children' }">
27              <el-table-column prop="departmentName" label="部门名称"></el-table-column>
28              <el-table-column prop="parentName" label="所属部门"></el-table-column>
29              <el-table-column prop="phone" label="部门电话"></el-table-column>

```



```

27     <el-table-column prop="address" label="部门位置"></el-table-column>
28     <el-table-column label="操作" width="200" align="center">
29       <template slot-scope="scope">
30         <el-button
31           icon="el-icon-edit-outline"
32           type="primary"
33           size="small"
34           @click="handleEdit(scope.row)"
35         >编辑
36       </el-button>
37       <el-button
38         icon="el-icon-close"
39         type="danger"
40         size="small"
41         @click="handleDelete(scope.row)"
42       >删除
43     </el-button>
44   </template>
45 </el-table-column>
46 </el-table>
47 </el-main>
48 </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：

```

1   <template>
2     <div>
3       <el-dialog
4         top="5vh"
5         :title="title"
6         :visible.sync="visible"
7         :width="width + 'px'"

```

```

8       :before-close="onClose"
9       :close-on-click-modal="false"
10    >
11      <div class="container" :style="{height:height+'px'}">
12        <slot name="content"></slot>
13      </div>
14      <span slot="footer" class="dialog-footer">
15        <el-button @click="onClose">取 消</el-button>
16        <el-button type="primary" @click="onConfirm">确 定</el-button>
17      </span>
18    </el-dialog>
19  </div>
20 </template>
21
22 <script>
23 export default {
24   props: {
25     title: {
26       type: String,
27       default: "标题",
28     },
29     visible: {
30       type: Boolean,
31       default: false,
32     },
33     width: {
34       type: Number,
35       default: 600,
36     },
37     height: {
38       type: Number,
39       default: 250,
40     },
41   },
42   data() {
43     return {}
44   },
45   methods: {
46     onClose() {
47       this.$emit("onClose");
48     },
49     onConfirm() {
50       this.$emit("onConfirm");
51     },
52   },
53 };
54 </script>
55
56 <style lang="scss" scoped>
57 .container{
58   overflow-x: initial;
59   overflow-y: auto;
60 }
61 .el-dialog__wrapper {
62   ::v-deep .el-dialog {
63     border-top-left-radius: 7px !important;
64     border-top-right-radius: 7px !important;
65     .el-dialog__header {

```

```

66     border-top-left-radius: 7px !important;
67     border-top-right-radius: 7px !important;
68     background-color: #1890ff;
69     .el-dialog__title {
70       color: #fff;
71       font-size: 15px;
72       font-weight: 700;
73     }
74     .el-dialog__close {
75       color: #fff;
76     }
77   }
78   .el-dialog__body {
79     padding: 10px 10px !important;
80   }
81   .el-dialog__footer {
82     border-top: 1px solid #e8eae6 !important;
83     padding: 10px !important;
84   }
85 }
86 }
87 </style>

```

### 8.3.2 编写新增部门弹窗代码

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

```

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

```

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

```

1  export default {
2    name: "department",
3    //注册组件
4    components: {
5      SystemDialog
6    },
7  }

```

(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

```

1  <!-- 添加和编辑部门窗口 -->
2  <system-dialog
3    :title="deptDialog.title"
4    :visible="deptDialog.visible"
5    :width="deptDialog.width"
6    :height="deptDialog.height"
7    @onClose="onClose"
8    @onConfirm="onConfirm"
9  >
10   <div slot="content">
11     <el-form
12       :model="dept"
13       ref="deptForm"
14       :rules="rules"

```

```

15     label-width="80px"
16     :inline="true"
17     size="small"
18   >
19     <el-form-item label="所属部门" prop="parentName">
20       <el-input
21         v-model="dept.parentName"
22         @click.native="selectDepartment()"
23         :readonly="true"
24       ></el-input>
25     </el-form-item>
26     <el-form-item label="部门名称" prop="departmentName">
27       <el-input v-model="dept.departmentName"></el-input>
28     </el-form-item>
29     <el-form-item label="部门电话">
30       <el-input v-model="dept.phone"></el-input>
31     </el-form-item>
32     <el-form-item label="部门地址">
33       <el-input v-model="dept.address"></el-input>
34     </el-form-item>
35     <el-form-item label="序号">
36       <el-input v-model="dept.orderNum"></el-input>
37     </el-form-item>
38   </el-form>
39 </div>
40 </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。

```

1   export default {
2     name: "department",
3     components: {
4       SystemDialog
5     },
6     data() {
7       return {
8         // 部门窗口属性
9         deptDialog: {
10           title: "新增部门",
11           visible: false, // 是否显示
12           width: 560,
13           height: 170,
14         },
15         // 部门对象
16         dept: {
17           id: "",
18           pid: "",
19           parentName: "",
20           departmentName: "",
21           address: "",
22           phone: "",
23           orderNum: "",
24         },
25         // 表单验证规则
26         rules: {

```

```

27         parentName: [{ required: true, trigger: "change", message: "请选择所属部门"
28     }],
29     departmentName: [{ required: true, trigger: "blur", message: "请输入部门名
30     称" }],
31     },
32     methods: {
33         /**
34         * 弹窗取消事件
35         */
36         onClose() {
37             //关闭窗口
38             this.deptDialog.visible = false;
39         },
40         /**
41         * 弹窗确认事件
42         */
43         onConfirm() {
44             //表单验证
45             this.$refs.deptForm.validate((valid) => {
46                 //如果验证通过
47                 if (valid) {
48                     //关闭窗口
49                     this.deptDialog.visible = false;
50                 }
51             });
52         },
53         /**
54         * 打开添加部门窗口
55         */
56         openAddWindow() {
57             //设置窗口标题
58             this.deptDialog.title = "新增部门";
59             //显示添加部门窗口
60             this.deptDialog.visible = true;
61         },
62         /**
63         * 选择所属部门
64         */
65         selectDepartment() {},
66     },
67 };

```

### 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

```

1  /**
2   * 获取所属部门列表
3   */
4  async getParentTreeList() {
5      return await http.get("/api/department/parent/list");
6  }

```

(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。

```

1  <!-- 选择所属部门窗口 -->
2  <system-dialog
3    :title="parentDialog.title"
4    :visible="parentDialog.visible"
5    :width="parentDialog.width"
6    :height="parentDialog.height"
7    @onClose="parentOnClose"
8    @onConfirm="parentOnConfirm"
9  >
10   <div slot="content">
11     <el-tree
12       ref="parentTree"
13       :data="treeList"
14       node-key="id"
15       :props="defaultProps"
16       :default-expand-all="true"
17       :highlight-current="true"
18       :expand-on-click-node="false"
19       @node-click="handleNodeClick"
20     >
21       <div class="customer-tree-node" slot-scope="{ node, data }">
22         <span v-if="data.children.length === 0">
23           <i class="el-icon-document"></i>
24         </span>
25         <span v-else @click="openBtn(data)">
26           <svg-icon v-if="data.open" icon-class="add-s" />
27           <svg-icon v-else icon-class="sub-s" />
28         </span>
29         <span style="margin-left: 5px">{{ node.label }}</span>
30       </div>
31     </el-tree>
32   </div>
33 </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。

```

1  export default {
2    name: "department",
3    components: {
4      SystemDialog,
5    },
6    data() {
7      return {
8        //选择所属部门属性
9        parentDialog: {
10          title: "选择所属部门",
11          visible: false, //是否显示
12          width: 300,
13          height: 400,
14        },
15        //树形菜单列表
16        treeList: [],
17        defaultProps: {
18          children: "children",
19          label: "departmentName",
20        },

```

```

21     };
22   },
23   methods: {
24     /**
25      * 选择所属部门
26      */
27     async selectDepartment() {
28       //显示窗口
29       this.parentDialog.visible = true;
30       //设置窗口标题
31       this.parentDialog.title = "选择所属部门";
32       //查询部门列表
33       let res = await departmentApi.getParentTreeList();
34       //判断是否成功
35       if (res.success) {
36         //赋值
37         this.treeList = res.data;
38       }
39     },
40     /**
41      * 选择所属部门取消事件
42      */
43     parentOnClose() {
44       this.parentDialog.visible = false;
45     },
46     /**
47      * 选择所属部门确认事件
48      */
49     parentOnConfirm() {
50       this.parentDialog.visible = false;
51     },
52     /**
53      * 树节点点击事件
54      */
55     handleNodeClick(data) {
56       //当点击树节点时，赋予选中的值
57       this.dept.id = data.id;
58       this.dept.parentName = data.departmentName;
59     },
60     /**
61      * 点击树节点+-号折叠展开事件
62      */
63     openBtn(data) {
64       //修改折叠展开状态
65       data.open = !data.open;
66       this.$refs.parentTree.store.nodesMap[data.id].expanded = !data.open;
67     },
68   },
69 };

```

#### (4) 选择所属部门树菜单样式代码

```

1  <style lang="scss" scoped>
2  ::v-deep .el-tree {
3    .el-tree-node {
4      position: relative;
5      padding-left: 10px;
6    }

```

```
7   .el-tree-node__children {
8     padding-left: 20px;
9   }
10  .el-tree-node :last-child:before {
11    height: 40px;
12  }
13  .el-tree > .el-tree-node:before {
14    border-left: none;
15  }
16  .el-tree > .el-tree-node:after {
17    border-top: none;
18  }
19  .el-tree-node:before,
20  .el-tree-node:after {
21    content: "";
22    left: -4px;
23    position: absolute;
24    right: auto;
25    border-width: 1px;
26  }
27  .tree :first-child .el-tree-node:before {
28    border-left: none;
29  }
30  .el-tree-node:before {
31    border-left: 1px dotted #d9d9d9;
32    bottom: 0px;
33    height: 100%;
34    top: -25px;
35    width: 1px;
36  }
37  .el-tree-node:after {
38    border-top: 1px dotted #d9d9d9;
39    height: 20px;
40    top: 14px;
41    width: 24px;
42  }
43  .el-tree-node__expand-icon.is-leaf {
44    width: 8px;
45  }
46  .el-tree-node__content > .el-tree-node__expand-icon {
47    display: none;
48  }
49  .el-tree-node__content {
50    line-height: 30px;
51    height: 30px;
52    padding-left: 10px !important;
53  }
54 }
55 ::v-deep .el-tree > div {
56   &::before {
57     display: none;
58   }
59   &::after {
60     display: none;
61   }
62 }
63 </style>
```



#### (5) 调用选择所属部门的方法

```
1 <el-form-item label="所属部门" prop="parentName">
2   <el-input v-model="dept.parentName" :readonly="true"
3     @click.native="selectDepartment()"></el-input>
4 </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：

```
1 //重置表单和表单数据
2 export default function resetForm(fromName,obj){
3   //清空表单
4   if(this.$refs[fromName]){
5     this.$refs[fromName].resetFields();
6   }
7   //清空数据域
8   Object.keys(obj).forEach(key =>{
9     obj[key] = '';
10  })
11 }
```

#### (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。

```
1 //导入清空表单工具
2 import resetForm from '@/utils/resetForm'
3 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);

```
1 /**
2  * 打开添加部门窗口
3  */
4 openAddWindow() {
5   //清空表单数据
6   this.$resetForm("deptForm", this.dept);
7   //设置窗口标题
8   this.deptDialog.title = "新增部门";
9   //显示添加部门窗口
10  this.deptDialog.visible = true;
11 },
```

### 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：

```

1  /**
2   * 添加部门
3   * @returns
4   */
5  async addDept(params) {
6      return await http.post("/api/department/add", params);
7  },

```

## (2) 编写department.vue的窗口确认事件方法

```

1  /**
2   * 弹窗确认事件
3   */
4  onConfirm() {
5      //表单验证
6      this.$refs.deptForm.validate( async (valid) => {
7          //如果验证通过
8          if (valid) {
9              let res = null; //后端返回的数据
10             //判断部门ID是否有数据，如果部门ID为空，则表示新增，否则就是修改
11             if(this.dept.id===''){//新增
12                 //发送新增请求
13                 res = await departmentApi.addDept(this.dept);
14             }else{
15                 //发送修改请求
16             }
17         }
18         //判断是否成功
19         if (res.success) {
20             this.$message.success(res.message);
21             //刷新
22             this.search();
23             //关闭窗口
24             this.deptDialog.visible = false;
25         } else {
26             this.$message.error(res.message);
27         }
28     }
29 });
30 },

```

## 8.4 编辑部门

### 8.4.1 效果图

编辑部门

\* 所属部门

广州码农信息技术有限公司

\* 部门名称

软件技术部

部门电话

020-88881001

部门地址

广州市天河区

序号

1

取消

确定

#### 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。

```
1 //快速把obj1里面对应的数据复制到obj2
2 export default async function objCopy(obj1,obj2){
3   Object.keys(obj2).forEach(key =>{
4     obj2[key] = obj1[key]
5   })
6 }
```

(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。

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

#### 8.4.3 编辑按钮代码

```
1 <el-table-column label="操作" width="200" align="center">
2   <template slot-scope="scope">
3     <el-button
4       icon="el-icon-edit-outline"
5       type="primary"
6       size="small"
7       @click="handleEdit(scope.row)"
8     >编辑
9   </el-button>
10   <el-button
11     icon="el-icon-close"
12     type="danger"
13     size="small"
14     @click="handleDelete(scope.row)"
15   >删除
16   </el-button>
17 </template>
18 </el-table-column>
```

#### 8.4.4 编辑按钮点击事件

```
1  /**
2   * 编辑部门
3   * @param row
4   */
5  handleEdit(row) {
6      //数据回显
7      this.$objCopy(row, this.dept);
8      //设置窗口标题
9      this.deptDialog.title = "编辑部门";
10     //显示编辑部门窗口
11     this.deptDialog.visible = true;
12 },
```

#### 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：

```
1  /**
2   * 修改部门
3   * @returns
4   */
5  async updateDept(params) {
6      return await http.put("/api/department/update", params);
7  },
```

#### 8.4.6 确认按钮点击事件

```
1  /**
2   * 弹窗确认事件
3   */
4  onConfirm() {
5      this.$refs.deptForm.validate(async (valid) => {
6          if (valid) {
7              let res = null; //后端返回的数据
8              //判断部门ID是否有数据，如果部门ID为空，则表示新增，否则就是修改
9              if (this.dept.id === '') { //新增
10                  //发送新增请求
11                  res = await departmentApi.addDept(this.dept);
12              } else {
13                  //发送修改请求
14                  res = await departmentApi.updateDept(this.dept);
15              }
16              //判断是否成功
17              if (res.success) {
18                  this.$message.success(res.message);
19                  //刷新
20                  this.search();
21                  //关闭窗口
22                  this.deptDialog.visible = false;
23              } else {
24                  this.$message.error(res.message);
25              }
26          }
27      });
28  },
```

## 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：

```
1  /**
2   * 检查部门下是否存在子部门
3   */
4  async checkDepartment(params){
5    return await http.getRestApi("/api/department/check",params);
6  },
7  /**
8   * 删除部门
9   * @returns
10  */
11  async deleteById(params) {
12    return await http.delete("/api/department/delete", params);
13  }
```

### 8.5.3 删除按钮代码

```
1  <el-table-column label="操作" width="200" align="center">
2    <template slot-scope="scope">
3      <el-button
4        icon="el-icon-edit-outline"
5        type="primary"
6        size="small"
7        @click="handleEdit(scope.row)"
8        >编辑
9      </el-button>
10     <el-button
11       icon="el-icon-close"
12       type="danger"
13       size="small"
14       @click="handleDelete(scope.row)"
15       >删除
16     </el-button>
17   </template>
18 </el-table-column>
```

### 8.5.4 删除按钮点击事件

```
1  /**
2   * 删除部门
3   */
4  async handleDelete(row) {
5    //查询部门下是否存在子部门或用户
6    let result = await departmentApi.checkDepartment({ id: row.id });
7    //判断是否可以删除
```

```

8     if (!result.success) {
9         //提示不能删除
10        this.$message.warning(result.message);
11    } else {
12        //确认是否删除
13        let confirm = await this.$myconfirm("确定要删除该数据吗?");
14        if (confirm) {
15            //发送删除请求
16            let res = await departmentApi.deleteById({ id: row.id });
17            //判断是否成功
18            if (res.success) {
19                //成功提示
20                this.$message.success(res.message);
21                //刷新
22                this.search();
23            } else {
24                //失败提示
25                this.$message.error(res.message);
26            }
27        }
28    }
29 }

```

## 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。

```

1  package com.manong.vo.query;
2
3  import com.manong.entity.Permission;
4  import lombok.Data;
5
6  @Data
7  public class PermissionQueryVo extends Permission {
8  }

```

#### 9.1.2 PermissionService

```

1  /**
2   * 查询菜单列表
3   * @param permissionQueryVo
4   * @return
5   */
6  List<Permission> findPermissionList(PermissionQueryVo permissionQueryVo);
7
8  /**
9   * 查询上级菜单列表
10  * @return

```

```

11  */
12  List<Permission> findParentPermissionList();
13
14  /**
15   * 检查菜单是否有子菜单
16   * @param id
17   * @return
18   */
19  boolean hasChildrenOfPermission(Long id);

```

### 9.1.3 PermissionServiceImpl

```

1  /**
2   * 查询菜单列表
3   *
4   * @return
5   */
6  @Override
7  public List<Permission> findPermissionList(PermissionQueryVo permissionQueryVo) {
8      //创建条件构造器对象
9      QueryWrapper<Permission> queryWrapper = new QueryWrapper<Permission>();
10     //排序
11     queryWrapper.orderByAsc("order_num");
12     //调用查询菜单列表的方法
13     List<Permission> permissionList = baseMapper.selectList(queryWrapper);
14     //生成菜单树
15     List<Permission> menuTree = MenuTree.makeMenuTree(permissionList, 0L);
16     //返回数据
17     return menuTree;
18 }
19
20 /**
21  * 查询上级菜单列表
22  *
23  * @return
24  */
25 @Override
26 public List<Permission> findParentPermissionList() {
27     QueryWrapper<Permission> queryWrapper = new QueryWrapper<Permission>();
28     //只查询type为目录和菜单的数据(type=0或type=1)
29     queryWrapper.in("type", Arrays.asList(0,1));
30     //排序
31     queryWrapper.orderByAsc("order_num");
32     //查询菜单数据
33     List<Permission> permissionList = baseMapper.selectList(queryWrapper);
34     //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
35     Permission permission = new Permission();
36     permission.setId(0L);
37     permission.setParentId(-1L);
38     permission.setLabel("顶级菜单");
39     permissionList.add(permission); //将顶级菜单添加到集合
40     //生成菜单数据
41     List<Permission> menuTree = MenuTree.makeMenuTree(permissionList, -1L);
42     //返回数据
43     return menuTree;
44 }
45

```

```

46  /**
47   * 检查菜单是否有子菜单
48   *
49   * @param id
50   * @return
51   */
52  @Override
53  public boolean hasChildrenOfPermission(Long id) {
54      //创建条件构造器对象
55      QueryWrapper<Permission> queryWrapper = new QueryWrapper<Permission>();
56      //查询父级ID
57      queryWrapper.eq("parent_id", id);
58      //判断数量是否大于0，如果大于0则表示存在
59      if(baseMapper.selectCount(queryWrapper)>0){
60          return true;
61      }
62      return false;
63  }

```

#### 9.1.4 PermissionController

```

1   package com.manong.controller;
2
3
4   import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
5   import com.manong.entity.Permission;
6   import com.manong.service.PermissionService;
7   import com.manong.utils.Result;
8   import com.manong.vo.query.PermissionQueryVo;
9   import org.springframework.web.bind.annotation.*;
10
11  import javax.annotation.Resource;
12  import java.util.List;
13
14  @RestController
15  @RequestMapping("/api/permission")
16  public class PermissionController {
17      @Resource
18      private PermissionService permissionService;
19
20      /**
21       * 查询菜单列表
22       * @return
23       */
24      @GetMapping("/list")
25      public Result getMenuList(PermissionQueryVo permissionQueryVo){
26          //查询菜单列表
27          List<Permission> permissionList =
28              permissionService.findPermissionList(permissionQueryVo);
29          //返回数据
30          return Result.ok(permissionList);
31      }
32
33      /**
34       * 查询上级菜单列表
35       * @return
36       */

```



```
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

菜单名称	菜单类型	图标	路由名称	路由地址	操作
系统管理	目录	☰	system	/system/system	编辑 删除
部门管理	菜单	🗑️	department	/system/department/departme	编辑 删除
新增	按钮	+			编辑 删除
修改	按钮	✏️			编辑 删除
删除	按钮	🗑️			编辑 删除
用户管理	菜单	👤	userList	/system/user/userList	编辑 删除
新增	按钮	+			编辑 删除
修改	按钮	✏️			编辑 删除
删除	按钮	🗑️			编辑 删除

### 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-
5      button>
6      <!-- 数据表格 -->
7      <el-table
8        style="margin-top: 10px"
9        :height="tableHeight"
10       :data="menuList"
11       row-key="id"
12       default-expand-all
13       :tree-props="{ children: 'children' }"
14       :border="true"
15       stripe
16     >
17       <el-table-column prop="label" label="菜单名称"></el-table-column>
18       <el-table-column prop="type" label="菜单类型" align="center">

```

```

19       <el-tag v-if="scope.row.type == '0'" size="normal">目录</el-tag>
20       <el-tag type="success" v-else-if="scope.row.type == '1'" size="normal">
菜单</el-tag>
21       <el-tag type="warning" v-else-if="scope.row.type == '2'" size="normal">
按钮</el-tag>
22     </template>
23   </el-table-column>
24   <el-table-column prop="icon" label="图标" align="center">
25     <template slot-scope="scope">
26       <i :class="scope.row.icon" v-if="scope.row.icon.includes('el-icon')">
</i>
27       <svg-icon v-else :icon-class="scope.row.icon"></svg-icon>
28     </template>
29   </el-table-column>
30   <el-table-column prop="name" label="路由名称"></el-table-column>
31   <el-table-column prop="path" label="路由地址"></el-table-column>
32   <el-table-column prop="url" label="组件路径"></el-table-column>
33   <el-table-column align="center">
34     <template slot-scope="scope">
35       <el-button
36         type="primary"
37         icon="el-icon-edit"
38         size="small"
39         @click="editMenu(scope.row)"
40       >编辑
41     </el-button>
42     >
43     <el-button
44       type="danger"
45       size="small"
46       icon="el-icon-delete"
47       @click="deleteMenu(scope.row)"
48     >删除
49   </el-button>
50   </template>
51 </el-table-column>
52 </el-table-column>
53 </el-table>
54 </el-main>
55 </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。

```

1   import http from '@/utils/request';
2
3   export default {
4     /**
5      * 查询权限菜单列表
6      * @param params
7      */
8     async getMenuList(params){
9       return await http.get("/api/permission/list",params);
10    }
11  }

```

## 9.2.4 编写页面组件代码

```
1  //导入menu.js脚本代码
2  import menuApi from '@/api/menu';
3
4  export default {
5
6    name: 'menuList',
7
8    data() {
9      return {
10        menuList: [], //菜单列表
11        tableHeight: 0 //表格高度
12      }
13    },
14    methods: {
15      /**
16       * 查询菜单列表
17       */
18      async search() {
19        //发送查询请求
20        let res = await menuApi.getMenuList();
21        //判断是否成功
22        if(res.success){
23          //赋值
24          this.menuList = res.data;
25        }
26      },
27      /**
28       * 打开添加窗口
29       */
30      openAddWindow() {
31
32      },
33      /**
34       * 编辑菜单
35       * @param row
36       */
37      editMenu(row){
38
39      },
40      /**
41       * 删除菜单
42       * @param row
43       */
44      deleteMenu(row){
45
46      },
47    },
48    //初始化时调用
49    created(){
50      //调用查询菜单列表的方法
51      this.search();
52    },
53    mounted() {
54      this.$nextTick(() => {
55        this.tableHeight = window.innerHeight - 180
```

```
56     })
57   }
58 }
```

### 9.2.5 优化滚动条

在public目录下的index.html添加如下样式代码：

```
1  <style>
2    ::-webkit-scrollbar{
3      width: 5px;
4      height: 5px;
5      background-color: #F5F5F5;
6    }
7
8    ::-webkit-scrollbar-track {
9      box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.3);
10     -webkit-box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.3);
11     border-radius: 8px;
12     background-color: #F5F5F5;
13   }
14
15   ::-webkit-scrollbar-thumb{
16     border-radius: 8px;
17     box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.1);
18     -webkit-box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.1);
19     background-color: #c8c8c8;
20   }
21 </style>
```

## 9.3 新增权限菜单

### 9.3.1 效果图

(1) 新增目录

新增菜单

\* 菜单类型

☒ 目录 ☐ 菜单 ☐ 按钮

\* 所属菜单

\* 菜单名称

菜单图标

\* 路由地址

\* 权限字段

菜单序号

取消

确定

## (2) 新增菜单

新增菜单

\* 菜单类型

目录

●

菜单

按钮

\* 所属菜单

\* 菜单名称

菜单图标

\* 路由名称

\* 路由地址

\* 组件路径

\* 权限字段

菜单序号

取消

确定

## (3) 新增按钮

新增菜单

\* 菜单类型

目录

菜单

●

按钮

\* 所属菜单

菜单名称

菜单图标

\* 权限字段

菜单序号

取消

确定

### 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。

```

1  export default {
2    name: "menuList",
3    //注册组件
4    components:{
5      SystemDialog
6    },
7  }

```

### (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

```

1  <!-- 新增或编辑弹框 -->
2  <system-dialog
3    :title="menuDialog.title"
4    :width="menuDialog.width"
5    :height="menuDialog.height"
6    :visible="menuDialog.visible"
7    @onClose="onClose"
8    @onConfirm="onConfirm"
9  >
10   <div slot="content">
11     <el-form
12       :model="menu"
13       ref="menuForm"
14       :rules="rules"
15       label-width="80px"
16       :inline="true"
17       size="small"
18     >
19       <el-row>
20         <el-col :span="2">
21           <el-form-item prop="type" label="菜单类型">
22             <el-radio-group v-model="menu.type">
23               <el-radio :label="0">目录</el-radio>
24               <el-radio :label="1">菜单</el-radio>
25               <el-radio :label="2">按钮</el-radio>
26             </el-radio-group>
27           </el-form-item>
28         </el-col>
29       </el-row>
30       <el-form-item prop="parentName" size="small" label="所属菜单">
31         <el-input
32           @click.native="selectParentMenu"
33           v-model="menu.parentName"
34           :readonly="true"
35         ></el-input>
36       </el-form-item>
37       <el-form-item prop="label" size="small" label="菜单名称">
38         <el-input v-model="menu.label"></el-input>
39       </el-form-item>
40       <el-form-item size="small" label="菜单图标">
41         <el-input v-model="menu.icon"></el-input>
42       </el-form-item>
43       <el-form-item
44         prop="name"
45         v-if="menu.type == 1"
46         size="small"

```

```

47     label="路由名称"
48   >
49     <el-input v-model="menu.name"></el-input>
50   </el-form-item>
51   <el-form-item
52     prop="path"
53     v-if="menu.type !== 2"
54     size="small"
55     label="路由地址"
56   >
57     <el-input v-model="menu.path"></el-input>
58   </el-form-item>
59   <el-form-item
60     prop="url"
61     v-if="menu.type === 1"
62     size="small"
63     label="组件路径"
64   >
65     <el-input v-model="menu.url"></el-input>
66   </el-form-item>
67   <el-form-item prop="code" size="small" label="权限标识">
68     <el-input v-model="menu.code"></el-input>
69   </el-form-item>
70   <el-form-item size="small" label="菜单序号">
71     <el-input v-model="menu.orderNum"></el-input>
72   </el-form-item>
73 </el-form>
74 </div>
75 </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。

```

1  export default {
2    data() {
3      return {
4        //新增或编辑的框属性
5        menuDialog: {
6          title: "",
7          width: 630,
8          height: 270,
9          visible: false,
10       },
11       //菜单属性
12       menu: {
13         id: "",
14         type: "",
15         parentId: "",
16         parentName: "",
17         label: "",
18         icon: "",
19         name: "",
20         path: "",
21         url: "",
22         code: "",
23         orderNum: "",
24       },

```



```

25     rules: {
26         type: [{ required: true, trigger: "change", message: "请选择菜单类型" }],
27         parentName: [{ required: true, trigger: "change", message: "请选择所属菜单"
28     }],
29     label: [{ required: true, trigger: "blur", message: "请输入菜单名称" }],
30     name: [{ required: true, trigger: "blur", message: "请输入路由名称" }],
31     path: [{ required: true, trigger: "blur", message: "请输入路由路径" }],
32     url: [{ required: true, trigger: "blur", message: "请输入组件路径" }],
33     code: [{ required: true, trigger: "blur", message: "请输入权限字段" }],
34     },
35 };
36 methods: {
37     /**
38     * 打开添加窗口
39     */
40     openAddWindow() {
41         this.$resetForm("menuForm", this.menu); //清空表单数据
42         this.menuDialog.title = "新增菜单"; //设置窗口标题
43         this.menuDialog.visible = true; //显示窗口
44     },
45     /**
46     * 添加和修改菜单窗口关闭事件
47     */
48     onClose() {
49         this.menuDialog.visible = false; //关闭窗口
50     },
51     /**
52     * 添加和修改菜单窗口确认事件
53     */
54     onConfirm() {
55         //表单验证
56         this.$refs.menuForm.validate((valid) => {
57             if (valid) {
58                 //
59             }
60         });
61     },
62     /**
63     * 选择所属菜单
64     */
65     selectParentMenu() {
66     },
67 },
68 },
69 };

```

### 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

```

1  /**
2   * 获取上级菜单
3   * @returns
4   */
5  async getParentMenuList(params) {
6      return await http.get("/api/permission/parent/list", params)
7  }

```

## (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。

```

1  <!-- 选择所属菜单弹框 -->
2  <system-dialog
3      :title="parentDialog.title"
4      :width="parentDialog.width"
5      :height="parentDialog.height"
6      :visible="parentDialog.visible"
7      @onClose="onParentClose"
8      @onConfirm="onParentConfirm"
9  >
10     <div slot="content">
11         <el-tree
12             style="font-size: 14px"
13             ref="parentTree"
14             :data="parentMenuList"
15             node-key="id"
16             :props="defaultProps"
17             empty-text="暂无数据"
18             :show-checkbox="false"
19             default-expand-all
20             :highlight-current="true"
21             :expand-on-click-node="false"
22             @node-click="handleNodeClick"
23         >
24             <div class="customer-tree-node" slot-scope="{ node, data }">
25                 <!-- 当变为0说明没有下级 -->
26                 <span v-if="data.children.length == 0">
27                     <i class="el-icon-document" style="color: #8c8c8c; font-size: 18px"/>
28                     </span>
29                 <span v-else @click.stop="openBtn(data)">
30                     <svg-icon v-if="data.open" icon-class="add-s"/>
31                     <svg-icon v-else icon-class="sub-s"/>
32                 </span>
33                 <span style="margin-left: 3px">{{ node.label }}</span>
34             </div>
35         </el-tree>
36     </div>
37 </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。

```

1  export default {
2      name: 'menuList',
3      data() {
4          return {
5              //上级菜单弹框属性

```

```

6      parentDialog: {
7          title: '选择所属菜单',
8          width: 280,
9          height: 450,
10         visible: false
11     },
12     //树属性定义
13     defaultProps: {
14         children: 'children',
15         label: 'label'
16     },
17     parentMenuList: [] //所属菜单列表
18 }
19 },
20 methods: {
21     /**
22      * 选择所属菜单
23      */
24     async selectParentMenu() {
25         //显示窗口
26         this.parentDialog.visible = true;
27         //发送查询请求
28         let res = await menuApi.getParentMenuList();
29         //判断是否成功
30         if (res.success) {
31             //赋值
32             this.parentMenuList = res.data;
33         }
34     },
35     /**
36      * 选择所属菜单取消事件
37      */
38     onParentClose() {
39         this.parentDialog.visible = false //关闭窗口
40     },
41     /**
42      * 选择所属菜单确认事件
43      */
44     onParentConfirm() {
45         this.parentDialog.visible = false //关闭窗口
46     },
47     /**
48      * 切换图标
49      * @param data
50      */
51     openBtn(data) {
52         data.open = !data.open
53         this.$refs.parentTree.store.nodesMap[data.id].expanded = !data.open
54     },
55     /**
56      * 所属菜单节点点击事件
57      */
58     handleNodeClick(data) {
59         //所属父级菜单ID
60         this.menu.parentId = data.id;
61         //所属父级菜单名称
62         this.menu.parentName = data.label;
63     },

```

```
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脚本文件。

```
1  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", "s-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", "s-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。

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

(3) 定义图标选择器所需要的属性及事件。

```
1 export default {
2   name: 'menuList',
3   data() {
4     return {
5
6       iconList: [], //图标列表
7       userChooseIcon: "", //用户选中的图标
8     }
9   },
10  methods: {
11
12    /**
13     * 打开添加窗口
14     */
15    openAddWindow() {
16      this.$resetForm('menuForm', this.menu) //清空表单数据
17      this.menuDialog.title = '新增菜单' //设置窗口标题
18      this.menuDialog.visible = true //显示窗口
19      this.userChooseIcon = ""; //清空菜单图标
20    },
21    /**
22     * 获取图标列表
23     */
24    getIconList(){
25      this.iconList = elementIcons
26    },
27    //给icon绑定的点击事件
28    setIcon(icon) {
29      this.userChooseIcon = icon; //将i的样式设为选中的样式el-icon-xxx
30      this.menu.icon = icon;
31    },
32  },
33  //初始化时调用
34  created() {
35    //调用查询菜单列表的方法
36    this.search()
37    //调用获取图标列表
38    this.getIconList();
39  },
40
41 }
```

(4) 编写样式代码

```
1 .iconList {
2   width: 400px;
3   height: 300px;
4   overflow-y: scroll;
5   overflow-x: hidden;
6   display: flex;
7   justify-content: space-around;
```

```

8     flex-wrap: wrap;
9     i {
10        display: inline-block;
11        width: 60px;
12        height: 45px;
13        color: #000000;
14        font-size: 20px;
15        border: 1px solid #e6e6e6;
16        border-radius: 4px;
17        cursor: pointer;
18        text-align: center;
19        line-height: 45px;
20        margin: 5px;
21        &:hover {
22            color: orange;
23            border-color: orange;
24        }
25    }
26 }
27
28 .chooseIcons{
29     width: 175px;
30     background-color: #FFFFFF;
31     background-image: none;
32     border-radius: 4px;
33     border: 1px solid #DCDFE6;
34     box-sizing: border-box;
35     color: #606266;
36     display: inline-block;
37     font-size: inherit;
38     height: 33px;
39     line-height: 25px;
40     outline: none;
41     padding: 0 15px;
42     transition: border color 0.2s cubic-bezier(0.645, 0.045, 0.355, 1);
43 }

```

#### (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。

```

1  <div class="chooseIcons">
2    <el-popover placement="bottom" width="450" trigger="click">
3      <span
4        slot="reference"
5        style="
6          display: inline-block;
7          width: 200px;
8          height: 33px;
9          line-height: 33px;
10         "
11      >
12        <i :class="userChooseIcon"></i>
13        {{ userChooseIcon }}
14      </span>
15      <div class="iconList">
16        <i
17          v-for="item in iconList"

```

```

18         :key="item"
19         :class="item"
20         @click="setIcon(item)"
21         style="font-size: 20px"
22     ></i>
23 </div>
24 </el-popover>
25 </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。

```

1  <template>
2    <div class="chooseIcons">
3      <el-popover placement="bottom" width="450" trigger="click">
4        <span
5          slot="reference"
6          style="
7            display: inline-block;
8            width: 200px;
9            height: 33px;
10           line-height: 33px;
11         ">
12          >
13          <i :class="userChooseIcon"></i>
14          {{ userChooseIcon }}
15        </span>
16        <div class="iconList">
17          <i
18            v-for="item in iconList"
19            :key="item"
20            :class="item"
21            @click="setIcon(item)"
22            style="font-size: 20px"
23          ></i>
24        </div>
25      </el-popover>
26    </div>
27  </template>
28
29  <script>
30    //导入自定义的icon图标库
31    import { elementIcons } from "@utils/icons";
32    export default {
33      name: 'MyIcon',
34      data(){
35        return{
36          userChooseIcon:"", //用户选中的图标
37          iconList:[], //图标列表
38        }
39      },
40      methods:{
41        /**
42         * 获取图标列表
43         */

```

```

44     getIconList(){
45         this.iconList = elementIcons;
46     },
47     //给icon绑定的点击事件
48     setIcon(icon) {
49         //将i的样式设为选中的样式el-icon-xxx
50         this.userChooseIcon = icon;
51         //将选中的图标传递给父组件
52         this.$emit("selecticon", icon)
53     },
54 },
55 created() {
56     //获取图标列表
57     this.getIconList();
58 }
59 }
60 </script>
61
62 <style scoped lang="scss">
63 .iconList {
64     width: 400px;
65     height: 300px;
66     overflow-y: scroll;
67     overflow-x: hidden;
68     display: flex;
69     justify-content: space-around;
70     flex-wrap: wrap;
71     i {
72         display: inline-block;
73         width: 60px;
74         height: 45px;
75         color: #000000;
76         font-size: 20px;
77         border: 1px solid #e6f2eb;
78         border-radius: 4px;
79         cursor: pointer;
80         text-align: center;
81         line-height: 45px;
82         margin: 5px;
83         &:hover {
84             color: orange;
85             border-color: orange;
86         }
87     }
88 }
89
90 .chooseIcons{
91     width: 175px;
92     background-color: #FFFFFF;
93     background-image: none;
94     border-radius: 4px;
95     border: 1px solid #DCDFE6;
96     box-sizing: border-box;
97     color: #606266;
98     display: inline-block;
99     font-size: inherit;
100     height: 33px;
101     line-height: 25px;

```



```

102     outline: none;
103     padding: 0 15px;
104     transition: border-color 0.2s cubic-bezier(0.645, 0.045, 0.355, 1);
105 }
106 </style>

```

## (2) 编写menuList.vue脚本代码

```

1  //导入自定义图标组件
2  import MyIcon from '@components/system/MyIcon.vue'
3
4
5  export default {
6    name: 'menuList',
7    //注册组件
8    components: {
9      SystemDialog,
10     MyIcon
11   },
12   methods: {
13     /**
14      * 打开添加窗口
15      */
16     openAddWindow() {
17       this.$resetForm('menuForm', this.menu) //清空表单数据
18       this.menuDialog.title = '新增菜单' //设置窗口标题
19       this.menuDialog.visible = true //显示窗口
20       this.$nextTick(() => {
21         this.$refs["child"].userChooseIcon = ""; //清空菜单图标
22       })
23     },
24     /**
25      * 给icon绑定的点击事件
26      * @param icon
27      */
28     setIcon(icon) {
29       this.menu.icon = icon;
30     },
31   }
32 }

```

## (3) 编写menuList.vue页面代码

```

1  <el-form-item size="small" label="菜单图标">
2    <my-icon @selecticon="setIcon" ref="child"></my-icon>
3  </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：

```

1  /**
2   * 添加菜单
3   * @returns
4   */
5  async addMenu(params){
6      return await http.post("/api/permission/add",params)
7  }

```

## (2) 编写menuList.vue的窗口确认事件方法

```

1  /**
2   * 添加和修改菜单窗口确认事件
3   */
4  onConfirm() {
5      //表单验证
6      this.$refs.menuForm.validate(async (valid) => {
7          if (valid) {
8              let res = null;
9              //判断菜单ID是否为空
10             if (this.menu.id === "") {
11                 //发送添加请求
12                 res = await menuApi.addMenu(this.menu);
13             } else {
14                 //发送修改请求
15
16             }
17             //判断是否成功
18             if (res.success) {
19                 this.$message.success(res.message);
20                 //刷新
21                 this.search();
22                 //关闭窗口
23                 this.menuDialog.visible = false;
24             } else {
25                 this.$message.error(res.message);
26             }
27         }
28     })
29 }

```

## 9.4 编辑权限菜单

### 9.4.1 效果图

编辑菜单

\* 菜单类型

☒ 目录

☐ 菜单

☐ 按钮

\* 所属菜单

顶级菜单

\* 菜单名称

系统管理

菜单图标

el-icon-menu

\* 路由地址

/system

\* 权限字段

sys:manager

菜单序号

0

取消

确定

#### 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：

```
1  /**
2   * 修改菜单
3   * @returns
4   */
5  async updateMenu(params){
6    return await http.put( '/api/permission/update',params)
7  }
```

#### 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。

```
1  /**
2   * 编辑菜单
3   * @param row
4   */
5  editMenu(row) {
6    //把当前要编辑的数据复制到数据域，给表单回显
7    this.$objCopy(row, this.menu);
8    //设置弹框属性
9    this.menuDialog.title = "编辑菜单";
10   this.menuDialog.visible = true;
11   this.$nextTick(() => {
12     this.$refs["child"].userChooseIcon = row.icon;//菜单图标回显
13   })
14 },
15 /**
16  * 添加和修改菜单窗口确认事件
17  */
18  onConfirm() {
19    //表单验证
20    this.$refs.menuForm.validate(async (valid) => {
```

```

21     if (valid) {
22         let res = null;
23         //判断菜单ID是否为空
24         if (this.menu.id === "") {
25             //发送添加请求
26             res = await menuApi.addMenu(this.menu);
27         } else {
28             //发送修改请求
29             res = await menuApi.updateMenu(this.menu);
30         }
31         //判断是否成功
32         if (res.success) {
33             this.$message.success(res.message);
34             //刷新
35             //this.search();
36             window.location.reload();
37             //关闭窗口
38             this.menuDialog.visible = false;
39         } else {
40             this.$message.error(res.message);
41         }
42     }
43 })
44 },

```

## 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：

```

1  /**
2   * 检查菜单下是否存在子菜单
3   */
4  async checkPermission(param){
5      return await http.getRestApi("/api/permission/check",param);
6  },
7  /**
8   * 删除菜单
9   * @returns
10  */
11  async deleteById(params) {
12      return await http.delete("/api/permission/delete", params);
13  }

```

### 9.5.3 删除按钮代码

```

1  <el-table-column align="center">
2    <template slot-scope="scope">
3      <el-button
4        type="primary"
5        icon="el-icon-edit"

```

```

6         size="small"
7         @click="editMenu(scope.row)"
8     >编辑
9 </el-button>
10 <el-button
11     type="danger"
12     size="small"
13     icon="el-icon-delete"
14     @click="deleteMenu(scope.row)"
15 >删除
16 </el-button>
17 </template>
18 </el-table-column>

```

#### 9.5.4 删除按钮点击事件

```

1  /**
2   * 删除菜单
3   * @param row
4   */
5  async deleteMenu(row) {
6      //判断是否存在子菜单
7      let result = await menuApi.checkPermission({ id: row.id });
8      //判断是否可以删除
9      if (!result.success) {
10         //提示不能删除
11         this.$message.warning(result.message);
12     } else {
13         //确认是否删除
14         let confirm = await this.$myconfirm("确定要删除该数据吗?");
15         if (confirm) {
16             //发送删除请求
17             let res = await menuApi.deleteById({ id: row.id });
18             //判断是否成功
19             if (res.success) {
20                 //成功提示
21                 this.$message.success(res.message);
22                 //刷新
23                 this.search();
24             } else {
25                 //失败提示
26                 this.$message.error(res.message);
27             }
28         }
29     }
30 }

```

## 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。

```
1 package com.manong.vo.query;  
2  
3 import com.manong.entity.Role;  
4 import lombok.Data;  
5  
6 @Data  
7 public class RoleQueryVo extends Role {  
8     private Long pageNo = 1L; //当前页码  
9     private Long pageSize = 10L; //每页显示数量  
10    private Long userId; //用户ID  
11 }
```

### 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。

```
1 public interface RoleService extends IService<Role> {  
2     /**  
3      * 根据用户查询角色列表  
4      * @param page  
5      * @param roleQueryVo  
6      * @return  
7      */  
8     IPage<Role> findRoleListByUserId(IPage<Role> page, RoleQueryVo roleQueryVo);  
9 }
```

### 10.1.3 RoleServiceImpl

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

```
1 @Service  
2 @Transactional  
3 public class RoleServiceImpl extends ServiceImpl<RoleMapper, Role> implements  
4     RoleService {  
5     @Resource  
6     private UserMapper userMapper;  
7  
8     /**  
9      * 根据用户查询角色列表  
10     *  
11     * @param page  
12     * @param roleQueryVo  
13     * @return  
14     */  
15     @Override  
16     public IPage<Role> findRoleListByUserId(IPage<Role> page, RoleQueryVo  
17         roleQueryVo) {  
18         //创建条件构造器  
19         QueryWrapper<Role> queryWrapper = new QueryWrapper<Role>();
```

```

19         //角色名称
20         queryWrapper.like(!ObjectUtils.isEmpty(roleQueryVo.getRoleName()),
21             "role_name", roleQueryVo.getRoleName());
22         //排序
23         queryWrapper.orderByAsc("id");
24         //根据用户ID查询用户信息
25         User user = userMapper.selectById(roleQueryVo.getUserId());
26         //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
27         if(user!=null && !ObjectUtils.isEmpty(user.getIsAdmin()) &&
28             user.getIsAdmin() !=1){
29             queryWrapper.eq("create_user", roleQueryVo.getUserId());
30         }
31         return baseMapper.selectPage(page, queryWrapper);
32     }

```

### 10.1.4 RoleController

```

1  package com.manong.controller;
2
3
4  import com.baomidou.mybatisplus.core.metadata.IPage;
5  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
6  import com.manong.entity.Role;
7  import com.manong.service.RoleService;
8  import com.manong.utils.Result;
9  import com.manong.vo.query.RoleQueryVo;
10 import org.springframework.web.bind.annotation.*;
11
12 import javax.annotation.Resource;
13
14 @RestController
15 @RequestMapping("/api/role")
16 public class RoleController {
17     @Resource
18     private RoleService roleService;
19
20     /**
21      * 分页查询角色列表
22      * @param roleQueryVo
23      * @return
24      */
25     @GetMapping("/list")
26     public Result list(RoleQueryVo roleQueryVo){
27         //创建分页对象
28         IPage<Role> page = new Page<Role>
29             (roleQueryVo.getPageNo(), roleQueryVo.getPageSize());
30         //调用分页查询方法
31         roleService.findRoleListByUserId(page, roleQueryVo);
32         //返回数据
33         return Result.ok(page);
34     }
35
36     /**
37      * 添加角色
38      * @param role
39      * @return

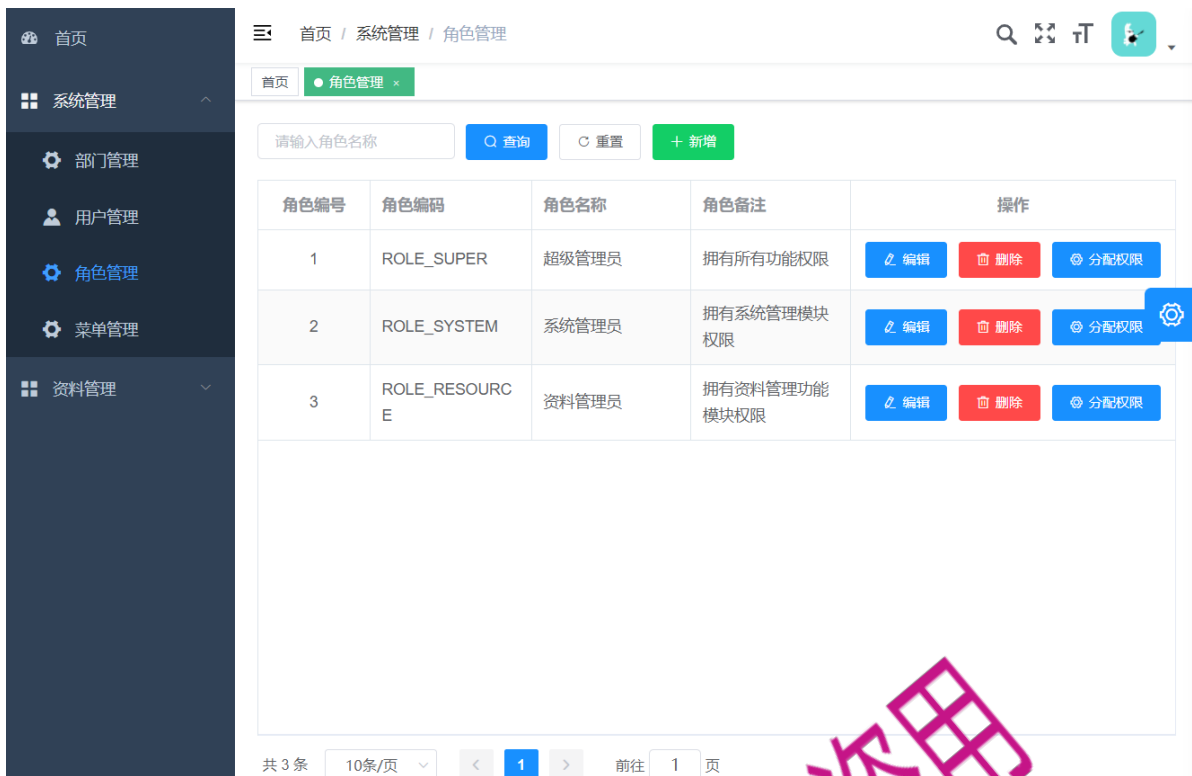
```

```
39     */
40     @PostMapping("/add")
41     public Result add(@RequestBody Role role){
42         if(roleService.save(role)){
43             return Result.ok().message("角色添加成功");
44         }
45         return Result.error().message("角色添加失败");
46     }
47
48     /**
49      * 修改角色
50      * @param role
51      * @return
52      */
53     @PutMapping("/update")
54     public Result update(@RequestBody Role role){
55         if(roleService.updateById(role)){
56             return Result.ok().message("角色修改成功");
57         }
58         return Result.error().message("角色修改失败");
59     }
60
61     /**
62      * 删除角色
63      * @param id
64      * @return
65      */
66     @DeleteMapping("/delete/{id}")
67     public Result delete(@PathVariable Long id){
68         if(roleService.removeById(id)){
69             return Result.ok().message("角色删除成功");
70         }
71         return Result.error().message("角色删除失败");
72     }
73
74 }
```

## 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。

```
1 <template>
2   <el-main>
3     <!-- 查询条件 -->
4     <el-form
5       :model="searchModel"
6       ref="searchForm"
7       label-width="80px"
8       :inline="true"
9       size="small"
10    >
11       <el-form-item>
12         <el-input v-model="searchModel.roleName" placeholder="请输入角色名称" />
13       </el-form-item>
14       <el-form-item>
15         <el-button type="primary" icon="el-icon-search">查询</el-button>
16         <el-button icon="el-icon-refresh-right">重置</el-button>
17         <el-button type="success" icon="el-icon-plus">新增</el-button>
18       </el-form-item>
19     </el-form>
20     <!-- 数据表格 -->
21     <el-table
22       :data="roleList"
23       :height="tableHeight"
24       border
25       stripe
26       style="width: 100%; margin-bottom: 10px"
27     >
28       <el-table-column
29         prop="id"
30         label="角色编号"
31         width="100"
32         align="center"
```

```

33     </el-table-column>
34     <el-table-column prop="name" label="角色名称"></el-table-column>
35     <el-table-column prop="remark" label="角色备注"></el-table-column>
36     <el-table-column label="操作" align="center" width="290">
37       <template slot-scope="scope">
38         <el-button
39           icon="el-icon-edit"
40           type="primary"
41           size="small"
42           @click="handleEdit(scope.row)"
43         >编辑
44       </el-button>
45       <el-button
46         icon="el-icon-delete"
47         type="danger"
48         size="small"
49         @click="handleDelete(scope.row)"
50       >删除
51     </el-button>
52     <el-button
53       icon="el-icon-setting"
54       type="primary"
55       size="small"
56       @click="assignRole(scope.row)"
57     >分配权限
58   </el-button>
59   </template>
60 </el-table-column>
61 </el-table>
62 <!-- 分页工具栏 -->
63 <el-pagination
64   background
65   @size-change="handleSizeChange"
66   @current-change="handleCurrentChange"
67   :current-page="pageNo"
68   :page-sizes="[10, 20, 30, 40, 50]"
69   :page-size="10"
70   layout="total, sizes, prev, pager, next, jumper"
71   :total="total"
72 >
73 </el-pagination>
74 </el-main>
75 </template>
76
77 <script>
78
79
80 export default {
81   name: 'roleList',
82   data() {
83     return {
84       // 查询条件
85       searchModel: {
86         roleName: ''
87       },
88       roleList: [], // 数据列表

```

```

91     tableHeight: 0, //表格高度
92     pageNo: 1, //当前页码
93     pageSize: 10, //每页显示数量
94     total: 0, //总数量
95   }
96 },
97 methods: {
98   /**
99    * 查询
100   */
101   search() {
102   },
103   /**
104    * 重置查询条件
105   */
106   resetValue() {
107   },
108   /**
109    * 当每页数量发生变化时触发该事件
110   */
111   handleSizeChange(size) {
112   },
113   /**
114    * 当页码发生变化时触发该事件
115   */
116   handleCurrentChange(page) {
117   },
118   /**
119    * 打开编辑窗口
120   */
121   handleEdit(row) {
122   },
123   /**
124    * 删除
125   */
126   handleDelete(row) {
127   },
128
129 },
130 //初始化时调用
131 created() {
132 },
133 //挂载后调用
134 mounted() {
135   this.$nextTick(() => {
136     this.tableHeight = window.innerHeight - 220
137   })
138 }
139 }
140 </script>

```

### 10.2.3 编写前端api脚本

在src/api目录下创建role.js。

```

1  import http from '@utils/request'
2
3  export default {
4    /**
5     * 查询角色列表
6     * @returns
7     */
8    async getRoleListApi(params) {
9      return await http.get("/api/role/list", params);
10   }
11 }

```

#### 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。

```

1  <template>
2    <el-main>
3      <!-- 查询条件 -->
4      <el-form
5        :model="searchModel"
6        ref="searchForm"
7        label-width="80px"
8        :inline="true"
9        size="small"
10     >
11        <el-form-item>
12          <el-input v-model="searchModel.roleName" placeholder="请输入角色名称" />
13        </el-form-item>
14        <el-form-item>
15          <el-button type="primary" icon="el-icon-search"
16            @click="search(pageNo, pageSize)">查询
17          </el-button>
18          <el-button icon="el-icon-refresh-right" @click="resetValue">重置</el-
19            button>
20          <el-button type="success" icon="el-icon-plus">新增</el-button>
21        </el-form-item>
22      </el-form>
23      <!-- 数据表格 -->
24      <el-table
25        :data="roleList"
26        :height="tableHeight"
27        border
28        stripe
29        style="width: 100%; margin-bottom: 10px"
30      >
31        <el-table-column
32          prop="id"
33          label="角色编号"
34          width="100"
35          align="center"
36        ></el-table-column>
37        <el-table-column prop="roleName" label="角色名称"></el-table-column>
38        <el-table-column prop="remark" label="角色备注"></el-table-column>
39        <el-table-column label="操作" align="center" width="290">
40          <template slot-scope="scope">
41            <el-button

```

```

40         icon="el-icon-edit"
41         type="primary"
42         size="small"
43         @click="handleEdit(scope.row)"
44     >编辑
45 </el-button>
46 <el-button
47     icon="el-icon-delete"
48     type="danger"
49     size="small"
50     @click="handleDelete(scope.row)"
51 >删除
52 </el-button>
53 <el-button
54     icon="el-icon-setting"
55     type="primary"
56     size="small"
57     @click="assignRole(scope.row)"
58 >分配权限
59 </el-button>
60 </template>
61 </el-table-column>
62 </el-table>
63 <!-- 分页工具栏 -->
64 <el-pagination
65     background
66     @size-change="handleSizeChange"
67     @current-change="handleCurrentChange"
68     :current-page="pageNo"
69     :page-sizes="[10, 20, 30, 40, 50]"
70     :page-size="10"
71     layout="total, sizes, prev, pager, next, jumper"
72     :total="total"
73 >
74 </el-pagination>
75 </el-main>
76 </template>
77
78 <script>
79 //导入role.js脚本
80 import roleApi from '@api/role';
81
82 export default {
83     name: 'roleList',
84     data() {
85         return {
86             //查询条件
87             searchModel: {
88                 roleName: '',
89                 pageNo: 1,
90                 pageSize: 10,
91                 userId: this.$store.getters.userId //用户ID
92             },
93             roleList: [], //数据列表
94             tableHeight: 0, //表格高度
95             pageNo: 1, //当前页码
96             pageSize: 10, //每页显示数量
97             total: 0, //总数量

```

```

98     }
99   },
100   methods: {
101     /**
102      * 查询
103      */
104     async search(pageNo=1, pageSize=10) {
105       this.searchModel.pageNo = pageNo; // 当前页码
106       this.searchModel.pageSize = pageSize; // 每页显示数量
107       // 发送查询请求
108       let res = await roleApi.getRoleListApi(this.searchModel);
109       // 执行成功
110       if (res.success) {
111         // 角色列表
112         this.roleList = res.data.records;
113         // 总数量
114         this.total = res.data.total;
115       }
116     },
117     /**
118      * 重置查询条件
119      */
120     resetValue() {
121       // 清空查询条件
122       this.searchModel.roleName = "";
123       // 重新查询
124       this.search();
125     },
126     /**
127      * 当每页数量发生变化时触发该事件
128      */
129     handleSizeChange(size) {
130       this.pageSize = size; // 将每页显示的数量交给成员变量
131       this.search(this.pageNo, size);
132     },
133     /**
134      * 当页码发生变化时触发该事件
135      */
136     handleCurrentChange(page) {
137       this.pageNo = page;
138       // 调用查询方法
139       this.search(page, this.pageSize);
140     },
141   },
142   created() {
143     // 调用查询角色列表的方法
144     this.search();
145   },
146   mounted() {
147     this.$nextTick(() => {
148       this.tableHeight = window.innerHeight - 220
149     })
150   }
151 }
152 </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_USERID', 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_USERID: (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

新增角色

\* 角色编码

\* 角色名称

角色描述

取消

确定

### 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。

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

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

```
1 export default {
2   name: "roleList",
3   //注册组件
4   components: {
5     SystemDialog
6   },
7 }
```

(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

```
1 <!-- 添加和修改角色窗口 -->
2 <system-dialog
3   :title="roleDialog.title"
4   :visible="roleDialog.visible"
5   :width="roleDialog.width"
6   :height="roleDialog.height"
7   @onClose="onClose"
8   @onConfirm="onConfirm"
9 >
10   <div slot="content">
11     <el-form
12       :model="role"
```



```

13     ref="roleForm"
14     :rules="rules"
15     label-width="80px"
16     :inline="false"
17     size="small"
18   >
19     <el-form-item label="角色编码" prop="roleCode">
20       <el-input v-model="role.roleCode"></el-input>
21     </el-form-item>
22     <el-form-item label="角色名称" prop="roleName">
23       <el-input v-model="role.roleName"></el-input>
24     </el-form-item>
25     <el-form-item label="角色描述">
26       <el-input type="textarea" v-model="role.remark" :rows="5"></el-input>
27     </el-form-item>
28   </el-form>
29 </div>
30 </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。

```

1  export default {
2    components: { SystemDialog },
3    name: 'roleList',
4    data() {
5      return {
6        rules: {
7          roleCode: [{ required: true, trigger: 'blur', message: '请输入角色编码' }],
8          roleName: [{ required: true, trigger: 'blur', message: '请输入角色名称' }],
9        },
10       //添加和修改角色窗口属性
11       roleDialog: {
12         title: '',
13         visible: false,
14         height: 230,
15         width: 500,
16       },
17       //角色对象
18       role: {
19         id: "",
20         roleCode: "",
21         roleName: "",
22         remark: "",
23         createUser: this.$store.getters.userId
24       }
25     },
26   },
27   methods: {
28     /**
29     * 打开添加窗口
30     */
31     openAddWindow() {
32       //清空表单数据
33       this.$resetForm("roleForm", this.role);
34       this.roleDialog.title = '新增角色' //设置窗口标题
35       this.roleDialog.visible = true //显示窗口

```

```

36     },
37     /**
38      * 窗口取消事件
39      */
40     onClose() {
41         this.roleDialog.visible = false
42     },
43     /**
44      * 窗口确认事件
45      */
46     onConfirm() {
47         //表单验证
48         this.$refs.roleForm.validate(async (valid) => {
49             if (valid) {
50
51             }
52         });
53     },
54 }
55 }

```

(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。

```

1  const mutations = {
2      //省略其他代码.....
3      //设置用户ID
4      SET_USERID: (state, userId) => {
5          state.userId = userId
6      },
7  }
8
9  const actions = {
10     // get user info
11     getInfo({ commit, state }) {
12         return new Promise((resolve, reject) => {
13             //调用api/user里面的getInfo方法获取用户信息和权限信息
14             getInfo(state.token).then(response => {
15                 const { data } = response
16                 if (!data) {
17                     reject('Verification failed, please Login again.')
18                 }
19                 //解构出用户id
20                 const { roles, name, avatar, introduction, id } = data
21
22                 if (!roles || roles.length <= 0) {
23                     reject('getInfo: roles must be a non-null array!')
24                 }
25                 //省略其他代码.....
26                 commit('SET_USERID', id); //用户ID
27                 resolve(data)
28             }).catch(error => {
29                 reject(error)
30             })
31         })
32     },
33 }

```

完成上述操作后，在src/store/getters.js中保存用户id。

```

1  const getters = {
2    //省略其它代码
3
4    //用户ID
5    userId: state => state.user.userId,
6
7  }
8  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。

```

1  /**
2   * 添加角色
3   * @returns
4   */
5  async addRoleApi(params) {
6    return await http.post("/api/role/add", params);
7  }

```

### 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。

```

1  /**
2   * 窗口确认事件
3   */
4  onConfirm() {
5    //表单验证
6    this.$refs.roleForm.validate(async (valid) => {
7      if (valid) {
8        let res = null;
9        //判断角色ID是否为空
10       if (this.role.id === "") {
11         //新增
12         //发送添加请求
13         res = await roleApi.addRoleApi(this.role);
14       } else {
15         //发送修改请求
16
17       }
18       //判断是否成功
19       if (res.success) {
20         this.$message.success(res.message);
21         //刷新
22         this.search(this.pageNo, this.pageSize);
23         //关闭窗口
24         this.roleDialog.visible = false;
25       } else {
26         this.$message.error(res.message);
27       }
28     }
29   });

```

## 10.4 编辑角色

### 10.4.1 效果图

**编辑角色**

\* 角色编码: ROLE\_SUPER

\* 角色名称: 超级管理员

角色描述: 拥有所有功能权限

取消 确定

### 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 === "") {
```

```

11         //新增
12         //发送添加请求
13         res = await roleApi.addRoleApi(this.role);
14     } else {
15         //发送修改请求
16         res = await roleApi.updateRoleApi(this.role);
17     }
18     //判断是否成功
19     if (res.success) {
20         this.$message.success(res.message);
21         //刷新
22         this.search(this.pageNo, this.pageSize);
23         //关闭窗口
24         this.roleDialog.visible = false;
25     } else {
26         this.$message.error(res.message);
27     }
28     }
29     });
30 },
31 /**
32  * 打开编辑窗口
33  */
34 handleEdit(row) {
35     //数据回显
36     this.$objCopy(row, this.role); //将当前编辑的数据复制到role对象中
37     //设置窗口标题
38     this.roleDialog.title = "编辑角色";
39     //显示编辑角色窗口
40     this.roleDialog.visible = true;
41 }

```

## 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

```
1  /**
2   * 根据角色ID查询权限列表
3   * @param roleId
4   * @return
5   */
6  List<Permission> findPermissionListByRoleId(Long roleId);
```

### 10.5.2.2 PermissionMapper.xml

```
1  <select id="findPermissionListByRoleId" resultType="com.manong.entity.Permission">
2      select t1.* from sys_permission t1 inner join sys_role_permission t2 on t1.id
3      = t2.permission_id
4      where t2.role_id = #{roleId}
5  </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。

```
1  package com.manong.vo;
2
3  import com.manong.entity.Permission;
4  import lombok.Data;
5
6  import java.util.ArrayList;
7  import java.util.List;
8
9  @Data
10 public class RolePermissionVo {
11     /**
12      * 菜单数据
13      */
14     private List<Permission> permissionList = new ArrayList<Permission>();
15
16     /**
17      * 该角色原有分配的菜单数据
18      */
19     private Object [] checkList;
20 }
```

### 10.5.2.4 PermissionService

```
1  /**
2   * 查询分配权限树列表
3   * @param userId
4   * @param roleId
5   * @return
6   */
7  RolePermissionVo findPermissionTree(Long userId, Long roleId);
```

### 10.5.2.5 PermissionServiceImpl

```
1  /**
2   * 查询分配权限树列表
3   *
4   * @param userId
5   * @param roleId
6   * @return
7   */
8  @Override
9  public RolePermissionVo findPermissionTree(Long userId, Long roleId) {
10     //1.查询当前用户信息
11     User user = userMapper.selectById(userId);
12     List<Permission> list = null;
13     //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
14     if(!ObjectUtils.isEmpty(user.getIsAdmin()) && user.getIsAdmin() == 1){
15         //查询所有权限
16         list = baseMapper.selectList(null);
17     }else{
18         //根据用户ID查询
19         list = baseMapper.findPermissionListByUserId(userId);
20     }
21     //3.组装成树数据
22     List<Permission> permissionList = MenuTree.makeMenuTree(list, 0L);
23     //4.查询要分配角色的原有权限
24     List<Permission> rolePermissions =
25     baseMapper.findPermissionListByRoleId(roleId);
26     //5.找出该角色存在的数据
27     List<Long> listIds = new ArrayList<Long>();
28     Optional.ofNullable(list).orElse(new ArrayList<>())
29         .stream()
30         .filter(Objects::nonNull) //等同于 obj -> obj!=null
31         .forEach(item -> {
32             Optional.ofNullable(rolePermissions).orElse(new ArrayList<>())
33                 .stream()
34                 .filter(Objects::nonNull)
35                 .forEach(obj ->{
36                     if(item.getId().equals(obj.getId())){
37                         listIds.add(obj.getId());
38                         return;
39                     }
40                 });
41         });
42     //创建
43     RolePermissionVo vo = new RolePermissionVo();
44     vo.setPermissionList(permissionList);
45     vo.setCheckList(listIds.toArray());
46     return vo;
47 }
```

### 10.5.2.6 RoleController

在RoleController编写查询权限树数据的方法。

```
1  /**
2   * 分配权限-查询权限树数据
3   *
```

```

4      * @param userId
5      * @param roleId
6      * @return
7      */
8      @GetMapping("/getAssignPermissionTree")
9      public Result getAssignPermissionTree(Long userId, Long roleId) {
10         //调用查询权限树数据的方法
11         RolePermissionVo permissionTree =
12         permissionService.findPermissionTree(userId, roleId);
13         //返回数据
14         return Result.ok(permissionTree);
15     }

```

### 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。

```

1      /**
2      * 查询分配权限树列表
3      * @returns
4      */
5      async getAssignTreeApi(params){
6          return await http.get("/api/role/getAssignPermissionTree", params);
7      }

```

#### 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。

```

1      <!-- 分配权限树窗口 -->
2      <system-dialog
3          :title="assignDialog.title"
4          :visible="assignDialog.visible"
5          :width="assignDialog.width"
6          :height="assignDialog.height"
7          @onClose="onAssignClose"
8          @onConfirm="onAssignConfirm"
9      >
10         <div slot="content">
11             <el-tree
12                 ref="assignTree"
13                 :data="assignTreeData"
14                 node-key="id"
15                 :props="defaultProps"
16                 empty-text="暂无数据"
17                 :show-checkbox="true"
18                 :highlight-current="true"
19                 default-expand-all
20             ></el-tree>
21         </div>
22     </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：

```
1 //判断菜单节点是否是末级节点
2 export default function leafUtils(){
3   const setLeaf =(arr) => {
4     if(arr && arr.length >0){
5       for(let i =0; i < arr.length;i++){
6         if(arr[i].children && arr[i].children.length > 0){
7           arr[i].open = false;
8           setLeaf(arr[i].children)
9         }else{
10           arr[i].open = true;
11         }
12       }
13     }
14     return arr
15   }
16   return{
17     setLeaf
18   }
19 }
```

### 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

```
1 //导入末级节点脚本
2 import leafUtils from '@/utils/leaf'
3
4 export default {
5   components: { SystemDialog },
6   name: 'roleList',
7   data() {
8     return {}
9     //分配权限窗口属性
10    assignDialog: {
11      title: '',
12      visible: false,
13      height: 450,
14      width: 300
15    },
16    roleId: '', //角色ID
17    assignTreeData: [], //树节点数据
18    //树节点属性
19    defaultProps: {
20      children: 'children',
21      label: 'label'
22    }
23  },
24 },
25 methods: {
26   /**
27    * 分配权限
28    */
29   async assignRole(row) {
30     //设置角色ID
```

```

31     this.roleId = row.id
32     //构建参数
33     let params = {
34         roleId: row.id,
35         userId: this.$store.getters.userId
36     }
37     //发送查询请求
38     let res = await roleApi.getAssignTreeApi(params)
39     //判断是否成功
40     if (res.success) {
41         //获取当前登录用户拥有的所有权限
42         let permissionList = res.data.permissionList
43         //获取当前被分配的角色已经拥有的权限信息
44         let checkList = res.data.checkList
45         //判断当前菜单是否是末级
46         let { setLeaf } = leafUtils()
47         //设置权限菜单列表
48         let newPermissionList = setLeaf(permissionList)
49         //设置树节点菜单数据
50         this.assignTreeData = newPermissionList
51         //将回调延迟到下次DOM更新循环之后执行,在修改数据之后立即使用它,然后等待DOM更新。
52         this.$nextTick(() => {
53             //获取树菜单的节点数据
54             let nodes = this.$refs.assignTree.children
55             //设置子节点
56             this.setChild(nodes, checkList)
57         })
58     }
59     //显示窗口
60     this.assignDialog.visible = true
61     //设置窗口标题
62     this.assignDialog.title = '给【${row.roleName}】分配权限`
63 },
64 /**
65  * 设置子节点
66  */
67 setChild(childNodes, checkList) {
68     //判断是否存在子节点
69     if (childNodes && childNodes.length > 0) {
70         //循环所有权限
71         for (let i = 0; i < childNodes.length; i++) {
72             //根据 data 或者 key 拿到 Tree 组件中的 node
73             let node = this.$refs.assignTree.getNode(childNodes[i])
74             //判断是否已经拥有对应的角色数据
75             if (checkList && checkList.length > 0) {
76                 //循环遍历已有的权限集合
77                 for (let j = 0; j < checkList.length; j++) {
78                     //找到已经存在的菜单权限节点
79                     if (checkList[j] == childNodes[i].id) {
80                         //如果节点是展开状态,则将树节点选中
81                         if (childNodes[i].open) {
82                             this.$refs.assignTree.setChecked(node, true)
83                             break
84                         }
85                     }
86                 }
87             }
88             //如果存在子节点,则递归选中

```

```

89         if (childNodes[i].children) {
90             this.setChild(childNodes[i].children, checkList)
91         }
92     }
93 }
94 },
95 /**
96  * 分配权限窗口取消事件
97  */
98 onAssignClose() {
99     this.assignDialog.visible = false
100 },
101 /**
102  * 分配权限窗口确认事件
103  */
104 async onAssignConfirm() {
105 }
106 },
107 },
108 }

```

## 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

```

1  package com.manong.dto;
2
3  import lombok.Data;
4
5  import java.util.List;
6
7  /**
8   * 用于给角色分配权限时保存 选中的权限数据
9   */
10 @Data
11 public class RolePermissionDTO {
12     private Long roleId;//角色编号
13     private List<Long> list;//权限菜单ID集合
14 }

```

### 10.5.4.2 RoleMapper

```

1  /**
2   * 删除角色权限关系
3   * @param roleId
4   */
5  @Delete("delete from sys_role_permission where role_id = #{roleId}")
6  void deleteRolePermission(Long roleId);
7
8  /**
9   * 保存角色权限关系
10  * @param roleId
11  * @param permissionIds
12  * @return

```

```

13     */
14     int saveRolePermission(Long roleId, List<Long> permissionIds);

```

#### 10.5.4.3 RoleMapper.xml

```

1     <insert id="saveRolePermission">
2         insert into sys_role_permission(role_id,permission_id) values
3         <foreach collection="permissionIds" item="item" index="index" separator=",">
4             ({roleId},{item})
5         </foreach>
6     </insert>

```

#### 10.5.4.4 RoleService

```

1     /**
2     * 保存角色权限关系
3     * @param roleId
4     * @param permissionIds
5     * @return
6     */
7     boolean saveRolePermission(Long roleId, List<Long> permissionIds);

```

#### 10.5.4.5 RoleServiceImpl

```

1     /**
2     * 保存角色权限关系
3     *
4     * @param roleId
5     * @param permissionIds
6     * @return
7     */
8     @Override
9     public boolean saveRolePermission(Long roleId, List<Long> permissionIds) {
10         //删除该角色对应的权限信息
11         baseMapper.deleteRolePermission(roleId);
12         //保存角色权限
13         return baseMapper.saveRolePermission(roleId,permissionIds)>0;
14     }

```

#### 10.5.4.6 RoleController

```

1     /**
2     * 分配权限-保存权限数据
3     *
4     * @param rolePermissionDTO
5     * @return
6     */
7     @PostMapping("/saveRoleAssign")
8     public Result saveRoleAssign(@RequestBody RolePermissionDTO rolePermissionDTO) {
9         if (roleService.saveRolePermission(rolePermissionDTO.getRoleId(),
10             rolePermissionDTO.getList())) {
11             return Result.ok().message("权限分配成功");
12         } else {
13             return Result.error().message("权限分配失败");
14         }
15     }

```

## 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。

```
1  /**
2   * 分配权限
3   * @returns
4   */
5  async assignSaveApi(params){
6    return await http.post("/api/role/saveRoleAssign",params);
7  }
```

### 10.5.5.2 编写分配权限窗口确认事件

```
1  /**
2   * 分配权限窗口确认事件
3   */
4  async onAssignConfirm() {
5    //获取选中的节点key
6    let ids = this.$refs.assignTree.getCheckedKeys()
7    //获取选中节点的父节点id
8    let pids = this.$refs.assignTree.getHalfCheckedKeys()
9    //组装选中的节点ID数据
10   let listId = ids.concat(pids)
11   //组装参数
12   let params = {
13     roleId: this.roleId,
14     list: listId
15   }
16   //发送请求
17   let res = await roleApi.assignSaveApi(params)
18   //判断是否成功
19   if (res.success) {
20     //关闭窗口
21     this.assignDialog.visible = false
22     //提示成功
23     this.$message.success(res.message)
24   } else {
25     //提示失败
26     this.$message.error(res.data)
27   }
28 }
```

## 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。

```
1 package com.manong.vo.query;  
2  
3 import com.manong.entity.User;  
4 import lombok.Data;  
5  
6 @Data  
7 public class UserQueryVo extends User {  
8     private Long pageNo = 1L; //当前页码  
9     private Long pageSize = 10L; //每页显示数量  
10 }
```

#### 11.1.1.2 UserService

在UserService接口中编写 分页查询用户信息 的方法。

```
1 /**  
2  * 分页查询用户信息  
3  * @param page  
4  * @param userQueryVo  
5  * @return  
6  */  
7 IPage<User> findUserListByPage(IPage<User> page, UserQueryVo userQueryVo);
```

#### 11.1.1.3 UserServiceImpl

```
1 /**  
2  * 分页查询用户信息  
3  *  
4  * @param page  
5  * @param userQueryVo  
6  * @return  
7  */  
8 @Override  
9 public IPage<User> findUserListByPage(IPage<User> page, UserQueryVo userQueryVo)  
10 {
```

```

10      //创建条件构造器对象
11      QueryWrapper<User> queryWrapper = new QueryWrapper<User>();
12      //部门编号
13      queryWrapper.eq(!ObjectUtils.isEmpty(userQueryVo.getDepartmentId()),
14          "department_id", userQueryVo.getDepartmentId());
15      //用户名
16      queryWrapper.like(!ObjectUtils.isEmpty(userQueryVo.getUsername()),
17          "username", userQueryVo.getUsername());
18      //真实姓名
19      queryWrapper.like(!ObjectUtils.isEmpty(userQueryVo.getRealName()),
20          "real_name", userQueryVo.getRealName());
21      //电话
22      queryWrapper.like(!ObjectUtils.isEmpty(userQueryVo.getPhone()),
23          "phone", userQueryVo.getPhone());
24      //查询并返回数据
25      return baseMapper.selectPage(page, queryWrapper);
26  }

```

#### 11.1.1.4 UserController

```

1  @Resource
2  private PasswordEncoder passwordEncoder;
3
4  /**
5   * 查询用户列表
6   * @param userQueryVo
7   * @return
8   */
9  @GetMapping("/list")
10 public Result list(UserQueryVo userQueryVo) {
11     //创建分页信息
12     IPage<User> page = new Page<User>(userQueryVo.getPageNo(),
13         userQueryVo.getPageSize());
14     //调用分页查询方法
15     userService.findUserListByPage(page, userQueryVo);
16     //返回数据
17     return Result.ok(page);
18 }

```

#### 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。

```

1  export default {
2      /**
3       * 查询用户列表
4       * @param params
5       * @returns
6       */
7      async getUserList(params){
8          return await http.get("/api/user/list", params);
9      }
10 }

```

### 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。

```
1   <template>
2     <el-container :style="{ height: containerHeight + 'px' }">
3       <!-- 左侧部门树形菜单列表 -->
4       <el-aside
5         style="
6           padding: 10px 0px 0px 0px;
7           background: #fff;
8           border-right: 1px solid #dfe6ec;
9         "
10        width="200px"
11      >
12        <el-tree
13          style="font-size: 14px"
14          ref="leftTree"
15          :data="deptList"
16          node-key="id"
17          :props="defaultProps"
18          default-expand-all
19          empty-text="暂无数据"
20          :show-checkbox="false"
21          :highlight-current="true"
22          :expand-on-click-node="false"
23          @node-click="handleNodeClick"
24        >
25          <div class="custom-tree-node" slot-scope="{ node, data }">
26            <div>
27              <span v-if="data.children.length == 0">
28                <i class="el-icon-document"></i>
29              </span>
30              <span v-else @click.stop="openBtn(data)">
31                <img v-if="data.open" icon-class="add-s"/>
32                <img v-else icon-class="sub-s"/>
33              </span>
34              <!-- 名称 -->
35              <span style="margin-left: 3px">{{ node.label }}</span>
36            </div>
37          </div>
38        </el-tree>
39      </el-aside>
40    </el-container>
41  </template>
42
43  <script>
44    //导入部门api脚本
45    import departmentApi from '@/api/department';
46
47    export default {
48      name: 'userList',
49      data() {
50        return {
51          containerHeight: 0, //容器高度
52          deptList: [], //左侧部门树形菜单列表
53          //树节点属性
54          defaultProps: {
```



```

55     children: 'children',
56     label: 'departmentName'
57   }
58 }
59 },
60 methods: {
61   /**
62    * 查询部门列表
63    */
64   async getDeptList() {
65     //发送查询请求
66     let res = await departmentApi.getDepartmentList(null);
67     //判断是否成功
68     if (res.success) {
69       this.deptList = res.data;
70     }
71   },
72   /**
73    * 左侧树节点点击事件
74    */
75   handleClick(data) {
76
77   },
78   //加减号图标点击事件
79   openBtn(data) {
80     //修改折叠展开状态
81     data.open = !data.open
82     this.$refs.leftTree.store.nodeMap[data.id].expanded = !data.open;
83   },
84
85 },
86 created() {
87   //查询部门列表
88   this.getDeptList();
89 },
90 mounted() {
91   this.$nextTick(() => {
92     //内容高度
93     this.containerHeight = window.innerHeight - 85;
94   })
95 }
96 }
97 </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。

```

1  <template>
2    <el-container :style="{ height: containerHeight + 'px' }">
3      <!-- 左侧部门树形菜单列表 -->
4
5      <!-- 右侧用户数据 -->
6      <!-- 表格数据 -->
7      <el-main>
8        <!-- 查询条件 -->
9        <el-form
10          :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"

```

```

63         :page-sizes="[10, 20, 30, 40, 50]"
64         :page-size="10"
65         layout="total, sizes, prev, pager, next, jumper"
66         :total="total"
67     >
68     </el-pagination>
69 </el-main>
70 </el-container>
71 </template>
72
73 <script>
74 //导入部门api脚本
75 import departmentApi from '@api/department';
76 //导入用户api脚本
77 import userApi from '@api/user';
78
79 export default {
80     name: 'UserList',
81     data() {
82         return {
83             //查询条件对象
84             searchModel: {
85                 username: "",
86                 realName: "",
87                 phone: "",
88                 departmentId: "",
89                 pageNo: 1,
90                 pageSize: 10,
91             },
92             userList: [], //用户列表
93             tableHeight: 0, //表格高度
94             pageNo: 1, //当前页码
95             pageSize: 10, //每页显示数量
96             total: 0, //总数量
97             departmentId: "", //部门编号
98         }
99     },
100     methods: {
101         /**
102          * 查询部门列表
103          */
104         async getDeptList() {
105             //发送查询请求
106             let res = await departmentApi.getDepartmentList(null)
107             //判断是否成功
108             if (res.success) {
109                 this.deptList = res.data;
110                 //树加载完成后，默认点击第一个节点
111                 this.$nextTick(() => {
112                     const firstNode = document.querySelector(".el-tree-node");
113                     firstNode.click();
114                 });
115             }
116         },
117         /**
118          * 左侧树节点点击事件
119          */
120         handleNodeClick(data) {

```

```

121         //给部门编号赋值
122         this.departmentId = data.id;
123         //查询数据
124         this.search(this.departmentId);
125     },
126     /**
127      * 查询用户列表
128      */
129     async search(departmentId, pageNo = 1, pageSize = 10) {
130         this.searchModel.pageNo = pageNo;
131         this.searchModel.pageSize = pageSize;
132         this.searchModel.departmentId = departmentId;
133         //发送查询请求
134         let res = await userApi.getUserList(this.searchModel);
135         if (res.success) {
136             this.userList = res.data.records;
137             this.total = res.data.total;
138         }
139     },
140     /**
141      * 当每页数量发生变化时触发该事件
142      */
143     handleSizeChange(size) {
144         this.pageSize = size; //将每页显示的数量交给成员变量
145         this.search(this.departmentId, this.pageNo, size);
146     },
147     /**
148      * 当页码发生变化时触发该事件
149      */
150     handleCurrentChange(page) {
151         this.pageNo = page;
152         //调用查询方法
153         this.search(this.departmentId, page, this.pageSize);
154     },
155     /**
156      * 重置查询条件
157      */
158     resetValue() {
159         //清空查询条件
160         this.searchModel = {};
161         //重新查询
162         this.search(this.departmentId);
163     },
164 },
165 created() {
166     //查询部门列表
167     this.getDeptList();
168     //调用查询用户列表
169     this.search(this.departmentId);
170 },
171 mounted() {
172     this.$nextTick(() => {
173         //内容高度
174         this.containerHeight = window.innerHeight - 85;
175         //表格高度
176         this.tableHeight = window.innerHeight - 220;
177     })
178 }

```

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

## 11.2 新增用户

### 11.2.1 效果图



新增用户

\* 用户名  \* 密码

\* 所属部门  \* 姓名

\* 电话  昵称

邮箱  \* 性别 ☐ 男 ☐ 女

头像

取消 确定

禁止盗用

### 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) {
```

```

16         return Result.error().message("该登录名称已被使用，请重新输入！");
17     }
18     //密码加密
19     user.setPassword(passwordEncoder.encode(user.getPassword()));
20     //调用保存用户信息的方法
21     if(userService.save(user)){
22         return Result.ok().message("用户添加成功");
23     }
24     return Result.error().message("用户添加失败");
25 }

```

## 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。

```

1  <!-- 添加和编辑用户窗口 -->
2  <system-dialog
3      :title="userDialog.title"
4      :height="userDialog.height"
5      :width="userDialog.width"
6      :visible="userDialog.visible"
7      @onClose="onClose"
8      @onConfirm="onConfirm"
9  >
10     <div slot="content">
11         <el-form
12             :model="user"
13             ref="userForm"
14             :rules="rules"
15             label-width="80px"
16             :inline="true"
17             size="small"
18         >
19             <el-form-item prop="username" label="用户名">
20                 <el-input v-model="user.username"></el-input>
21             </el-form-item>
22             <el-form-item prop="password" v-if="user.id === ''" label="密码">
23                 <el-input type="password" v-model="user.password"></el-input>
24             </el-form-item>
25             <el-form-item prop="departmentName" label="所属部门">
26                 <el-input
27                     v-model="user.departmentName"
28                     :readonly="true"
29                     @click.native="selectDepartment()"
30                 ></el-input>
31             </el-form-item>
32             <el-form-item prop="realName" label="姓名">
33                 <el-input v-model="user.realName"></el-input>
34             </el-form-item>
35             <el-form-item prop="phone" label="电话">
36                 <el-input v-model="user.phone"></el-input>
37             </el-form-item>
38             <el-form-item label="昵称">
39                 <el-input v-model="user.nickName"></el-input>

```

```

40     </el-form-item>
41     <el-form-item label="邮箱">
42       <el-input v-model="user.email"></el-input>
43     </el-form-item>
44     <el-form-item prop="gender" label="性别">
45       <el-radio-group v-model="user.gender">
46         <el-radio :label="0">男</el-radio>
47         <el-radio :label="1">女</el-radio>
48       </el-radio-group>
49     </el-form-item>
50     <br>
51     <!-- 用户头像: 待补充 -->
52   </el-form>
53 </div>
54 </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。

```

1  //导入对话框组件
2  import SystemDialog from '@/components/system/SystemDialog.vue'
3
4  export default {
5    name: 'UserList',
6    components: {
7      SystemDialog
8    },
9    data() {
10     //自定义验证规则
11     let validatePhone = (rule, value, callback) => {
12       if (!value) {
13         callback(new Error("请输入手机号码"));
14         //使用正则表达式进行验证手机号码
15       } else if (!( /^[13456789]\d{9}$/.test(value) )) {
16         callback(new Error("手机号格式不正确"));
17       } else {
18         callback();
19       }
20     };
21     return {
22       //添加和修改用户窗口属性
23       userDialog: {
24         title: '',
25         height: 410,
26         width: 610,
27         visible: false
28       },
29       //用户对象
30       user: {
31         id: '',
32         departmentId: '',
33         departmentName: '',
34         email: '',
35         realName: '',
36         phone: '',
37         nickName: '',
38         password: '',

```

```

39     username: '',
40     gender: '',
41     avatar: ''
42   },
43   rules: {
44     departmentName: [{ required: true, trigger: 'change', message: '请选择所属
部门' }],
45     realName: [{ required: true, trigger: 'blur', message: '请填写姓名' }],
46     phone: [{ trigger: 'blur', validator: validatePhone }],
47     username: [{ required: true, trigger: 'blur', message: '请填写登录名' }],
48     password: [{ required: true, trigger: 'blur', message: '请填写登录密码' }],
49     gender: [{ required: true, trigger: 'change', message: '请选择性别' }]
50   },
51   },
52 },
53 methods: {
54   /**
55    * 打开添加窗口
56    */
57   openAddWindow() {
58     this.$resetForm('userForm', this.user) //清空表单
59     this.userDialog.visible = true //显示窗口
60     this.userDialog.title = '新增用户' //设置标题
61   },
62   /**
63    * 新增或编辑取消事件
64    */
65   onClose() {
66     this.userDialog.visible = false //关闭窗口
67   },
68   /**
69    * 新增或编辑确认事件
70    */
71   onConfirm() {
72     //表单验证
73     this.$refs.userForm.validate((valid) => {
74       if (valid) {
75         //
76       }
77     })
78   },
79   /**
80    * 打开选择所属部门窗口
81    */
82   selectDepartment() {
83
84   },
85 }
86 }
87 </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。

```

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

```



```

4      :width="parentDialog.width"
5      :height="parentDialog.height"
6      :visible="parentDialog.visible"
7      @onClose="onParentClose"
8      @onConfirm="onParentConfirm"
9    >
10     <div slot="content">
11       <el-tree
12         ref="parentTree"
13         :data="parentList"
14         default-expand-all
15         node-key="id"
16         :props="parentProps"
17         :show-checkbox="false"
18         :highlight-current="true"
19         :expand-on-click-node="false"
20         @node-click="parentClick"
21       >
22         <div class="customer-tree-node" slot-scope="{ node, data }">
23           <span v-if="data.children.length == 0">
24             <i class="el-icon-document" />
25           </span>
26           <span v-else @click.stop="openParentBtn(data)">
27             <svg-icon v-if="data.open" icon-class="add-s" />
28             <svg-icon v-else icon-class="sub-s" />
29           </span>
30           <span style="margin-left: 3px">{{ node.label }}</span>
31         </div>
32       </el-tree>
33     </div>
34   </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。

```

1  export default {
2    name: 'UserList',
3    components: {
4      SystemDialog
5    },
6    data() {
7      return {
8        //选择所属部门窗口属性
9        parentDialog: {
10          title: '选择所属部门',
11          width: 300,
12          height: 450,
13          visible: false
14        },
15        //树节点属性
16        parentProps: {
17          children: 'children',
18          label: 'departmentName'
19        },
20        parentList: [], //所属部门节点数据

```

```

21     }
22 },
23 methods: {
24     /**
25      * 打开选择上级部门窗口
26      */
27     async selectDepartment() {
28         //查询上级部门数据
29         let res = await departmentApi.getDepartmentList(null)
30         //判断是否成功
31         if (res.success) {
32             this.parentList = res.data
33         }
34         //显示窗口
35         this.parentDialog.visible = true
36     },
37     /**
38      * 选择上级部门取消事件
39      */
40     onParentClose() {
41         this.parentDialog.visible = false //关闭窗口
42     },
43     /**
44      * 选择上级部门确认事件
45      */
46     onParentConfirm() {
47         this.parentDialog.visible = false
48     },
49     //上级部门树节点点击事件
50     parentClick(data) {
51         this.user.departmentId = data.id
52         this.user.departmentName = data.departmentName
53     },
54     //上级部门树加号、减号、图标点击事件
55     openParentBtn(data) {
56         data.open = !data.open
57         this.$refs.parentTree.store.nodesMap[data.id].expanded = !data.open
58     },
59 }
60 }
61 </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。

```

1     /**
2      * 添加用户
3      * @param params
4      * @returns
5      */
6     async addUser(params) {
7         return await http.post("/api/user/add", params);
8     }

```

### 11.2.3.6 编写窗口确认事件

```
1  /**
2   * 新增或编辑确认事件
3   */
4  onConfirm() {
5      this.$refs.userForm.validate(async(valid) => {
6          if (valid) {
7              let res = null
8              //判断用户ID是否为空
9              if (this.user.id === '') {
10                 //新增
11                 //发送添加请求
12                 res = await userApi.addUser(this.user)
13             } else {
14                 //发送修改请求
15
16             }
17             //判断是否成功
18             if (res.success) {
19                 this.$message.success(res.message)
20                 //刷新
21                 this.search(this.departmentId, this.pageNo, this.pageSize);
22                 //关闭窗口
23                 this.userDialog.visible = false
24             } else {
25                 this.$message.error(res.message)
26             }
27         }
28     })
29 }
```

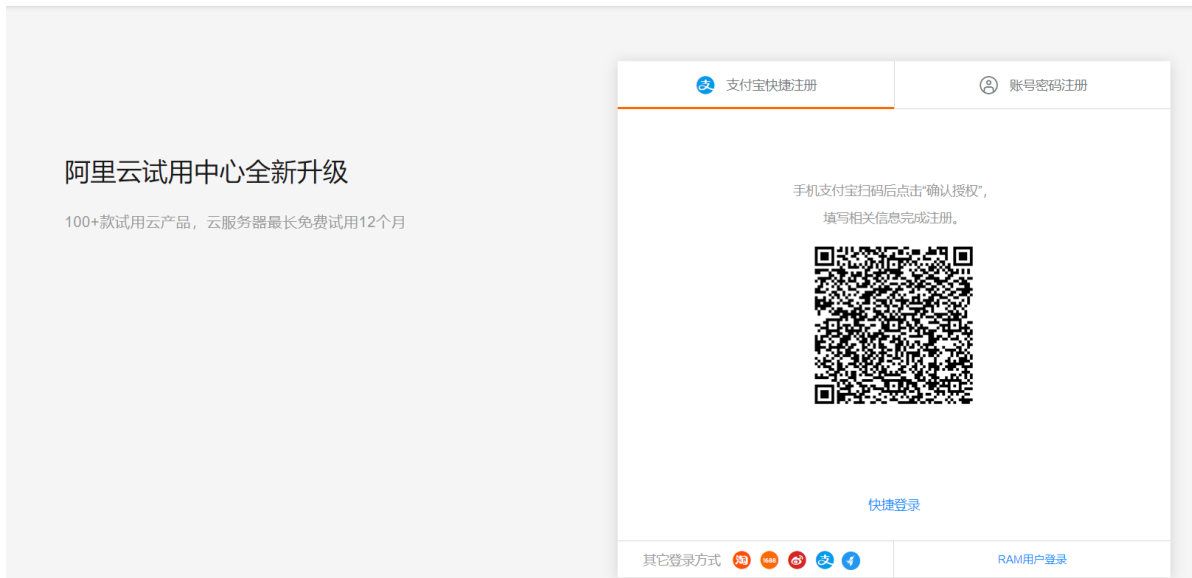
## 11.3 阿里云OSS对象存储服务

### 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

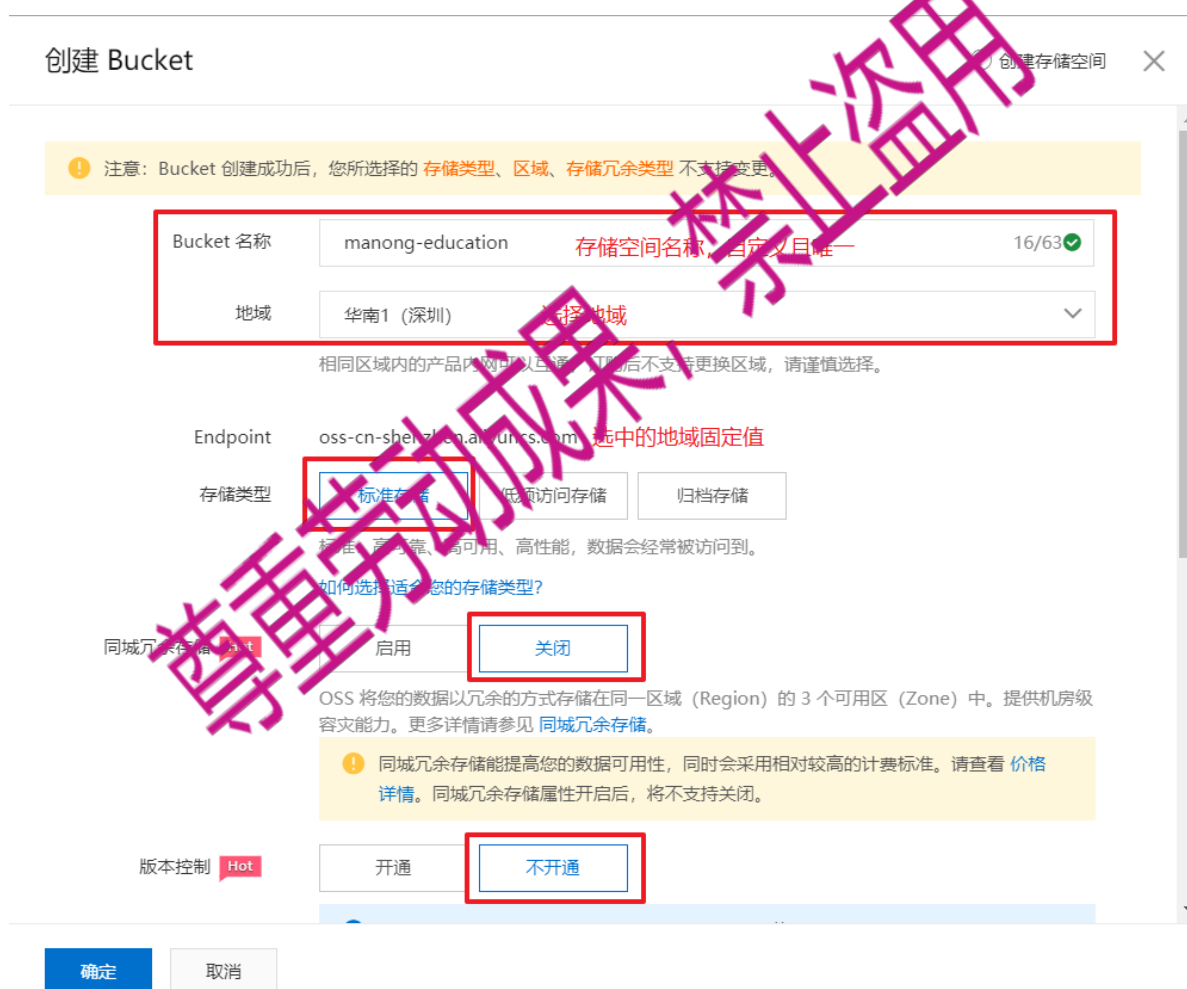
### 11.3.2.1 创建Bucket存储空间

(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。



## 创建 Bucket

创建存储空间

同城冗余存储能提高您的数据可用性，同时会采用相对较高的计费标准。请查看 [价格详情](#)。同城冗余存储属性开启后，将不支持关闭。

版本控制

开通

不开通

开启版本控制特性后，针对数据的覆盖和删除操作将会以历史版本的形式保存下来。了解 [版本控制](#)。当前未开启版本控制功能，数据删除或被覆盖后将无法找回。

读写权限

私有

公共读

公共读写

公共读：对文件与操作需要进行身份验证，可以对文件进行匿名读。

服务端加密方式

无

OSS 完全托管

KMS

实时日志查询

开通

不开通

OSS 与日志服务深度结合，免费提供最近 7 天内的 OSS 实时日志查询（限额 100 GB/天的日志写入额度，超出部分由日志服务单独收费。假设一条日志为 1 KB，约为 9 亿条）。开通该功能后，您可对 Bucket 的访问记录进行实时查询分析，[了解详情](#)。

定时备份

开通

不开通

混合云备份服务为 OSS 提供数据处理保护，防止误修改、误删除，可低成本地保存历史数据，服务首次开通后 2 个月内免费使用，[了解详情](#)。

确定

取消

阿里云控制台截图，显示 Bucket 创建成功后的概览页面。页面顶部有“基础数据”卡片，显示存储用量、流量、请求次数等统计信息。下方有“Endpoint (地域节点)”和“Bucket 域名”列表，支持 HTTPS 访问。页面右侧有“任务列表”。

Endpoint (地域节点)	Bucket 域名	HTTPS
oss-cn-shenzhen.aliyuncs.com	manong-education.oss-cn-shenzhen.aliyuncs.com	支持
oss-cn-shenzhen-internal.aliyuncs.com	manong-education.oss-cn-shenzhen-internal.aliyuncs.com	支持
oss-cn-shenzhen-internal.aliyuncs.com	manong-education.oss-cn-shenzhen-internal.aliyuncs.com	支持

### 11.3.2.2 文件管理

阿里云控制台截图，显示 Bucket 文件管理页面。页面顶部有“文件管理”卡片，显示文件数量、文件大小、存储类型等信息。下方有“文件列表”表格，显示文件名称、大小、类型、更新时间等。页面右侧有“任务列表”。

文件名	文件大小	存储类型	更新时间	操作
暂无数据				

### 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

The screenshot shows the 'manong-education' bucket configuration in the OSS console. The 'Endpoint (地域节点)' field is highlighted with a red box, showing 'oss-cn-shenzhen.aliyuncs.com'. Other fields like 'Bucket 域名' and 'HTTPS' are also visible.

- AccessKeyId：账号ID
- AccessKeySecret：账号密码

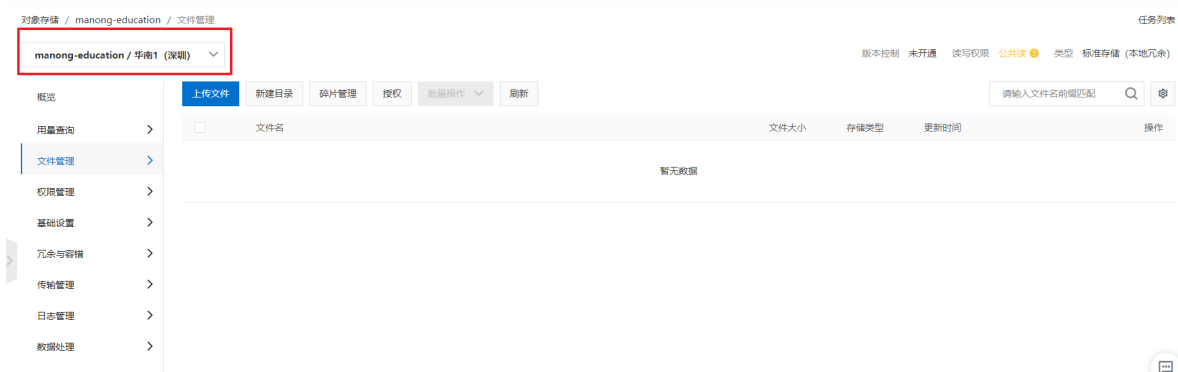
注意：AccessKeyId和AccessKeySecret 不是阿里云的账号密码，是由下图创建出来的ID和密码。

The screenshot shows the 'Create AccessKey' dialog box with a red box around the '继续使用该AccessKey' button. Below the dialog, the 'AccessKey' list is visible, showing the 'AccessKey ID' and 'AccessKey Secret' for the 'manong-education' bucket. The 'AccessKey Secret' is highlighted in blue.

点击 创建AccessKey 按钮后，会下载一份AccessKeyId和AccessKeySecret密码的文件，该密钥不能轻易暴露，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。

```
1  <!-- 阿里云OSS文件上传开始 -->
2  <!-- 阿里云 OSS -->
3  <dependency>
4      <groupId>com.aliyun.oss</groupId>
5      <artifactId>aliyun-sdk-oss</artifactId>
6      <version>3.14.1</version>
7  </dependency>
8
9  <!-- 日期时间工具 -->
10 <dependency>
11     <groupId>joda-time</groupId>
12     <artifactId>joda-time</artifactId>
13     <version>2.10.14</version>
14 </dependency>
15 <dependency>
16     <groupId>commons-io</groupId>
17     <artifactId>commons-io</artifactId>
18     <version>2.4</version>
19 </dependency>
20 <!-- 阿里云OSS文件上传结束 -->
```

### 11.4.2 编写application.properties配置文件

```
1  #阿里云OSS配置
2  aliyun.oss.file.endpoint=填写自己的地域节点
3  aliyun.oss.file.keyid=填写自己的keyId
4  aliyun.oss.file.keysecret=填写自己的AccessKeySecret
5  #bucket名称
6  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。

```
1  package com.manong.config.oss;
2
3  import lombok.Data;
4  import org.springframework.boot.context.properties.ConfigurationProperties;
```



```

5  import org.springframework.stereotype.Component;
6
7  @Data
8  @Component
9  @ConfigurationProperties(prefix = "aliyun.oss.file")
10 public class OssProperties {
11     private String endPoint;
12     private String keyId;
13     private String keySecret;
14     private String bucketName;
15 }

```

#### 11.4.4 编写文件上传业务代码

##### 11.4.4.1 编写FileService接口

```

1  package com.manong.service;
2
3  import org.springframework.web.multipart.MultipartFile;
4
5  public interface FileService {
6
7      /**
8       * 文件上传
9       * @param file      文件上传对象
10      * @param module     文件夹名称
11      * @return
12      */
13      String upload(MultipartFile file, String module);
14
15      /**
16       * 删除文件
17       * @param url
18       */
19      void deleteFile(String url);
20
21 }

```

##### 11.4.4.2 编写FileServiceImpl实现类

```

1  package com.manong.service.impl;
2
3  import com.aliyun.oss.OSS;
4  import com.aliyun.oss.OSSClientBuilder;
5  import com.manong.config.oss.OssProperties;
6  import com.manong.service.FileService;
7  import org.joda.time.DateTime;
8  import org.springframework.stereotype.Service;
9  import org.springframework.transaction.annotation.Transactional;
10 import org.springframework.web.multipart.MultipartFile;
11
12 import javax.annotation.Resource;
13 import java.io.IOException;
14 import java.io.InputStream;
15 import java.util.UUID;
16
17 @Service
18 @Transactional

```

```

19 public class FileServiceImpl implements FileService {
20     /**
21      * 文件上传
22      *
23      * @param file 文件上传对象
24      * @param module 文件夹名称
25      * @return
26      */
27     @Resource
28     private OssProperties ossProperties;
29
30     /**
31      * 文件上传
32      *
33      * @param file 文件上传对象
34      * @param module 文件夹名称
35      * @return
36      */
37     @Override
38     public String upload(MultipartFile file, String module) {
39         //获取地域节点
40         String endPoint = ossProperties.getEndPoint();
41         //获取AccessKeyId
42         String keyId = ossProperties.getKeyId();
43         //获取AccessKeySecret
44         String keySecret = ossProperties.getKeySecret();
45         //获取BucketName
46         String bucketName = ossProperties.getBucketName();
47
48         try {
49             //创建OSSClient实例
50             OSS ossClient = new OSSClientBuilder().build(endPoint, keyId,
keySecret);
51             //上传文件流
52             InputStream inputStream = file.getInputStream();
53             //获取旧名称
54             String originalFilename = file.getOriginalFilename();
55             //获取文件后缀名
56             String extension = FilenameUtils.getExtension(originalFilename);
57             //将文件名重命名
58             String newFileName = UUID.randomUUID().toString()
.replace("-", "") + "." + extension;
59             //使用当前日期进行分类管理
60             String datePath = new DateTime().toString("yyyy/MM/dd");
61             //构建文件名
62             newFileName = module + "/" + datePath + "/" + newFileName;
63             //调用OSS文件上传的方法
64             ossClient.putObject(bucketName, newFileName, inputStream);
65             //关闭OSSClient
66             ossClient.shutdown();
67             //返回文件地址
68             return "https://" + bucketName + "." + endPoint + "/" + newFileName;
69         } catch (IOException e) {
70             e.printStackTrace();
71         }
72     }
73     return null;
74 }
75

```

```

76     @Override
77     public void deleteFile(String url) {
78         //获取地域节点
79         String endPoint = ossProperties.getEndPoint();
80         //获取AccessKeyId
81         String keyId = ossProperties.getKeyId();
82         //获取AccessKeySecret
83         String keySecret = ossProperties.getKeySecret();
84         //获取BucketName
85         String bucketName = ossProperties.getBucketName();
86         try {
87             //创建OSSClient实例
88             OSS ossClient = new OSSClientBuilder().build(endPoint, keyId,
keySecret);
89             //组装文件地址
90             String host = "https://" + bucketName + "." + endPoint + "/";
91             //获取文件名称
92             String objectName = url.substring(host.length());
93             //删除文件
94             ossClient.deleteObject(bucketName, objectName);
95         } catch (Exception e) {
96             e.printStackTrace();
97         }
98     }
99 }

```

#### 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。

```

1  package com.manong.controller;
2
3  import com.manong.service.FileService;
4  import com.manong.utils.Result;
5  import org.springframework.web.bind.annotation.PostMapping;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RestController;
8  import org.springframework.web.multipart.MultipartFile;
9
10 import javax.annotation.Resource;
11
12 @RestController
13 @RequestMapping("/api/oss/file")
14 public class OSSController {
15
16     @Resource
17     private FileService fileService;
18
19     /**
20      * 文件上传
21      * @param file
22      * @param module
23      * @return
24      */
25     @PostMapping("/upload")
26     public Result upload(MultipartFile file, String module){
27         //返回上传到oss的路径
28         String url = fileService.upload(file,module);

```

```
29         return Result.ok(url).message("文件上传成功");
30     }
31
32 }
```

## 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

```
1  <!-- 用户头像 -->
2  <el-form-item label="头像">
3      <el-upload
4          :show-file-list="false"
5          :on-success="handleAvatarSuccess"
6          :before-upload="beforeAvatarUpload"
7          class="avatar-uploader"
8          :data="uploadHeader"
9          action="http://localhost:9999/api/oss/file/upload?module=avatar"
10     >
11         
12         <i v-else class="el-icon-plus avatar-uploader-icon"/>
13     </el-upload>
14 </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。

```
1  <style lang="scss">
2  //用户头像
3  .avatar-uploader .el-upload {
4      border: 1px dashed #d9d9d9 !important;
5      border-radius: 6px;
6      cursor: pointer;
7      position: relative;
8      overflow: hidden;
9  }
10
11 .avatar-uploader .el-upload:hover {
12     border-color: #409EFF;
13 }
14
15 .avatar-uploader .avatar-uploader-icon {
16     font-size: 28px;
17     color: #8c939d;
18     width: 178px;
19     height: 178px;
20     line-height: 178px;
21     text-align: center;
22 }
23
24 .avatar-uploader img {
25     width: 178px;
26     height: 178px;
27     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.$forceUpdate()
10 },
11 /**
12  * 上传校验
13  * @param file
14  * @returns {boolean}
15  */
16 beforeAvatarUpload(file) {
17   const isJPG = file.type === 'image/jpeg'
18   const isLt2M = file.size / 1024 / 1024 < 2
19   if (!isJPG) {
20     this.$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。

```
1  /**
2   * 修改用户
3   *
4   * @param user
5   * @return
6   */
7  @PutMapping("/update")
8  public Result update(@RequestBody User user) {
9      //查询用户
10     User item = userService.findUserByUserName(user.getUsername());
11     //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
12     if (item != null && item.getId() != user.getId()) {
13         return Result.error().message("登录名称已被占用!");
14     }
15     //调用修改用户信息的方法
16     if(userService.updateById(user)){
17         return Result.ok().message("用户修改成功");
18     }
19     return Result.error().message("用户修改失败");
20 }
```

### 11.5.2 编辑用户前端实现

#### 11.5.2.1 效果图

编辑用户

\* 用户名

admin

\* 所属部门

广州码农信息技术有限公司

\* 姓名

李明

\* 电话

13242587415

昵称

超级管理员

邮箱

111

\* 性别

☒ 男 ☐ 女

头像



取消

确定

#### 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。

```
1  /**
2   * 编辑用户
3   * @param params
4   * @returns
5   */
6  async updateUser(params){
7    return await http.put("/api/user/update", params);
8  }
```

#### 11.5.2.3 编写修改用户脚本代码

```
1  /**
2   * 编辑用户
3   */
4  handleEdit(row) {
5    //设置弹框属性
6    this.userDialog.title = '编辑用户'
7    this.userDialog.visible = true
8    //把当前编辑的数据复制到表单数据域，供回显使用
9    this.$objCopy(row, this.user)
10 },
11 /**
12  * 新增或编辑确认事件
13  */
```

```

14     onConfirm() {
15         this.$refs.userForm.validate(async(valid) => {
16             if (valid) {
17                 let res = null
18                 //判断用户ID是否为空
19                 if (this.user.id === '') {
20                     //新增
21                     //发送添加请求
22                     res = await userApi.addUser(this.user)
23                 } else {
24                     //发送修改请求
25                     res = await userApi.updateUser(this.user)
26                 }
27                 //判断是否成功
28                 if (res.success) {
29                     this.$message.success(res.message)
30                     //刷新
31                     this.search(this.departmentId, this.pageNo, this.pageSize);
32                     //关闭窗口
33                     this.userDialog.visible = false
34                 } else {
35                     this.$message.error(res.message)
36                 }
37             }
38         })
39     }

```

## 11.6 删除用户

### 11.6.1 删除用户后端接口实现

#### 11.6.1.1 UserMapper

```

1     public interface UserMapper extends BaseMapper<User> {
2
3         /**
4          * 删除用户角色关系
5          * @param userId
6          * @return
7          */
8         @Delete("delete from sys_user_role where user_id=#{userId}")
9         int deleteUserRole(Long userId);
10    }

```

#### 11.6.1.2 SystemConstants

```

1     package com.manong.utils;
2
3     public class SystemConstants {
4
5         /**
6          * 默认头像
7          */
8         public static final String DEFAULT_AVATAR =
9             "https://manong-authority.oss-cn-guangzhou.aliyuncs.com/avatar/male.jpg";
10
11    }

```



### 11.6.1.3 UserService

```
1  /**
2   * 删除用户信息
3   * @param id
4   * @return
5   */
6  boolean deleteById(Long id);
```

### 11.6.1.4 UserServiceImpl

```
1  @Override
2  @Transactional(rollbackFor = RuntimeException.class)
3  public boolean deleteById(Long id) {
4      //查询
5      User user = baseMapper.selectById(id);
6      //删除用户角色关系
7      baseMapper.deleteUserRole(id);
8      //删除用户
9      if (baseMapper.deleteById(id) > 0) {
10         //判断用户是否存在
11         if (user != null && !ObjectUtils.isEmpty(user.getAvatar())
12             && !user.getAvatar().equals(SystemConstants.DEFAULT_AVATAR)) {
13             //删除文件
14             fileService.deleteFile(user.getAvatar());
15         }
16         return true;
17     }
18     return false;
19 }
```

### 11.6.1.5 UserController

```
1  /**
2   * 删除用户
3   * @param id
4   * @return
5   */
6  @DeleteMapping("/delete/{id}")
7  public Result delete(@PathVariable Long id) {
8      //调用删除用户信息的方法
9      if(userService.deleteById(id)){
10         return Result.ok().message("用户删除成功");
11     }
12     return Result.error().message("用户删除失败");
13 }
```

## 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。

```

1  /**
2   * 删除用户
3   * @param params
4   * @returns
5   */
6  async deleteUser(params){
7      return await http.delete("/api/user/delete",params);
8  }

```

### 11.6.2.2 编写删除确认事件

```

1  /**
2   * 删除
3   */
4  async handleDelete(row) {
5      let confirm = await this.$myconfirm('确定要删除该数据吗?')
6      if (confirm) {
7          //封装条件
8          let params = { id: row.id }
9          //发送删除请求
10         let res = await userApi.deleteUser(params)
11         //判断是否成功
12         if (res.success) {
13             this.$message.success(res.message)
14             //刷新
15             this.search(this.departmentId, this.pageNo, this.pageSize);
16         } else {
17             this.$message.error(res.message)
18         }
19     }
20 }

```

## 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

```

1  /**
2   * 根据用户ID查询该用户拥有的角色ID
3   * @param userId
4   * @return
5   */
6  @Select("select role_id from `sys_user_role` where user_id = #{userId}")
7  List<Long> findRoleIdByUserId(Long userId);

```

### 11.7.2.2 RoleService

```
1  /**
2   * 根据用户ID查询该用户拥有的角色ID
3   * @param userId
4   * @return
5   */
6  List<Long> findRoleIdByUserId(Long userId);
```

### 11.7.2.3 RoleServiceImpl

```
1  /**
2   * 根据用户ID查询该用户拥有的角色ID
3   *
4   * @param userId
5   * @return
6   */
7  @Override
8  public List<Long> findRoleIdByUserId(Long userId) {
9      return baseMapper.findRoleIdByUserId(userId);
10 }
```

### 11.7.2.4 UserController

```
1  /**
2   * 获取分配角色列表
3   * @param roleQueryVo
4   * @return
5   */
6  @GetMapping("/getRoleListForAssign")
7  public Result getRoleListForAssign(RoleQueryVo roleQueryVo){
8      //创建分页对象
9      IPage<Role> page = new Page<Role>(roleQueryVo.getPageNo(),
10     roleQueryVo.getPageSize());
11     //调用查询方法
12     roleService.findRoleListByUserId(page, roleQueryVo);
13     //返回数据
14     return Result.ok(page);
15 }
16 /**
17 * 根据用户ID查询该用户拥有的角色列表
18 * @param userId
19 * @return
20 */
21 @GetMapping("/getRoleByUserId/{userId}")
22 public Result getRoleByUserId(@PathVariable Long userId){
23     //调用根据用户ID查询该用户拥有的角色ID的方法
24     List<Long> roleIds = roleService.findRoleIdByUserId(userId);
25     return Result.ok(roleIds);
26 }
```

### 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。

```
1  /**
2   * 查询用户角色列表
3   * @param params
4   * @returns
5   */
6  async getAssignRoleList(params){
7    return await http.get("/api/user/getRoleListForAssign",params);
8  },
9  /**
10   * 获取分配角色列表数据
11   * @param params
12   * @returns
13   */
14  async getRoleIdByUserId(params){
15    return await http.getRestApi("/api/user/getRoleByUserId",params);
16  }
```

#### 11.7.3.2 编写分配角色窗口代码

```
1  <!-- 分配角色窗口 -->
2  <system-dialog
3    :title="assignDialog.title"
4    :height="assignDialog.height"
5    :width="assignDialog.width"
6    :visible="assignDialog.visible"
7    @onClose="onAssignClose"
8    @onConfirm="onAssignConfirm"
9  >
10   <div slot="content">
11     <!-- 分配角色数据列表 -->
12     <el-table
13       ref="assignRoleTable"
14       :data="assignRoleList"
15       border
16       stripe
17       :height="assignHeight"
18       style="width: 100%; margin-bottom: 10px"
19       @selection-change="handleSelectionChange"
20     >
21       <el-table-column
22         type="selection"
23         width="55"
24         align="center"
25       ></el-table-column>
26       <el-table-column prop="roleCode" label="角色编码"/>
27       <el-table-column prop="roleName" label="角色名称"/>
28       <el-table-column prop="remark" label="角色备注"/>
29     </el-table>
30     <!-- 分页工具栏 -->
31     <el-pagination
32       @size-change="assignSizeChange"
33       @current-change="assignCurrentChange"
```

```

34     :current-page.sync="roleVo.pageNo"
35     :page-sizes="[10, 20, 30, 40, 50]"
36     :page-size="roleVo.pageSize"
37     layout="total, sizes, prev, pager, next, jumper"
38     :total="roleVo.total"
39     background
40   >
41   </el-pagination>
42 </div>
43 </system-dialog>

```

### 11.7.3.3 编写分配角色回显脚本代码

```

1  export default {
2    name: 'UserList',
3    components: {
4      SystemDialog
5    },
6    data() {
7      return {
8        //分配角色窗口属性
9        assignDialog: {
10          title: "",
11          visible: false,
12          width: 800,
13          height: 410,
14        },
15        //角色对象
16        roleVo: {
17          pageNo: 1,
18          pageSize: 10,
19          userId: "",
20          total: 0,
21        },
22        assignRoleList: [], //角色列表
23        assignHeight: 0, //分配角色表格高度
24        selectedIds: [], //被选中的角色id
25        selectedUserId: "", //被分配角色的用户ID
26      }
27    },
28    methods: {
29      /**
30       * 打开分配角色
31       */
32      async assignRole(row){
33        //防止回显出现问题
34        this.selectedIds = [];
35        this.selectedUserId = "";
36        //被分配用户的id
37        this.selectedUserId = row.id;
38        //显示窗口
39        this.assignDialog.visible = true;
40        //设置标题
41        this.assignDialog.title = `给【${row.realName}】分配角色`;
42        //查询当前登录用户的所有角色信息
43        await this.getAssignRoleList();
44        //获取当前被分配用户的ID
45        let params = {

```

```

46         userId: row.id,
47     };
48     //发送根据用户ID查询角色列表的请求
49     let res = await userApi.getRoleIdByUserId(params);
50     //如果存在数据
51     if (res.success && res.data){
52         //将查询到的角色ID列表交给选中角色数组
53         this.selectedIds = res.data;
54         //循环遍历
55         this.selectedIds.forEach((key) => {
56             this.assignRoleList.forEach((item) => {
57                 if (item.id === key) {
58                     this.$refs.assignRoleTable.toggleRowSelection(item, true);
59                 }
60             });
61         });
62     }
63 },
64 /**
65  * 查询当前登录用户的所有角色信息
66  */
67 async getAssignRoleList(pageNo = 1, pageSize = 10) {
68     //给用户ID赋值
69     this.roleVo.userId = this.$store.getters.userId;
70     this.roleVo.pageNo = pageNo;
71     this.roleVo.pageSize = pageSize;
72     //发送查询请求
73     let res = await userApi.getAssignRoleList(this.roleVo);
74     //判断是否成功
75     if (res.success) {
76         //角色列表赋值
77         this.assignRoleList = res.data.records;
78         //角色总数赋值
79         this.roleVo.total = res.data.total;
80     }
81 },
82 /**
83  * 分配角色取消事件
84  */
85 onAssignClose(){
86     //隐藏窗口
87     this.assignDialog.visible = false;
88 },
89 /**
90  * 分配角色确认事件
91  */
92 onAssignConfirm(){
93     //隐藏窗口
94     this.assignDialog.visible = false;
95 },
96 /**
97  * 当每页数量发生变化时触发该事件
98  */
99 assignSizeChange(size){
100     this.roleVo.pageSize = size; //将每页显示的数量交给成员变量
101     this.getAssignRoleList(this.roleVo.pageNo, size);
102 },
103 /**

```

```

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

```
1  /**
2   * 保存用户角色关系
3   * @param userId
4   * @param roleIds
5   * @return
6   */
7  int saveUserRole(Long userId, List<Long> roleIds);
```

#### 11.7.4.3 UserMapper.xml

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

#### 11.7.4.4 UserService

```
1  /**
2   * 分配角色
3   * @param userId
4   * @param roleIds
5   * @return
6   */
7  boolean saveUserRole(Long userId, List<Long> roleIds);
```

#### 11.7.4.5 UserServiceImpl

```
1  @Override
2  @Transactional(rollbackFor = RuntimeException.class)
3  public boolean saveUserRole(Long userId, List<Long> roleIds) {
4      //删除该用户对应的角色信息
5      baseMapper.deleteUserRole(userId);
6      //新增用户角色信息
7      return baseMapper.saveUserRole(userId, roleIds)>0;
8  }
```

#### 11.7.4.6 UserController

```
1  /**
2   * 分配角色
3   * @param userRoleDTO
4   * @return
5   */
6  @PostMapping("/saveUserRole")
7  public Result saveUserRole(@RequestBody UserRoleDTO userRoleDTO){
8      if (userService.saveUserRole(userRoleDTO.getUserId(),
9          userRoleDTO.getRoleIds())) {
10         return Result.ok().message("角色分配成功");
11     }
12     return Result.error().message("角色分配失败");
13 }
```



## 11.7.5 分配角色前端实现

### 11.7.5.1 编写前端api脚本

```
1  /**
2   * 分配角色
3   */
4  async assignRoleSave(params){
5      return await http.post("/api/user/saveUserRole",params)
6  }
```

### 11.7.5.2 编写分配角色窗口确认事件

```
1  /**
2   * 分配角色确认事件
3   */
4  async onAssignConfirm(){
5      //判断用户是否有选中角色
6      if (this.selectedIds.length === 0) {
7          this.$message.warning("请选择要分配的角色!");
8          return;
9      }
10     let params = {
11         userId: this.selectedUserId,
12         roleIds: this.selectedIds,
13     };
14     //发送请求
15     let res = await userApi.assignRoleSave(params);
16     //判断是否成功
17     if (res.success) {
18         this.$message.success(res.message);
19         //关闭窗口
20         this.assignDialog.visible = false;
21     } else {
22         this.$message.error(res.message);
23     }
24 }
```

## 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。

```
1  @Configuration
2  @EnableWebSecurity
3  //开启权限注解控制
4  @EnableGlobalMethodSecurity(prePostEnabled = true)
5  public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
6  }
```

### 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。

```
1  @RestController
2  @RequestMapping("/api/user")
3  public class UserController {
4
5      @Resource
6      private UserService userService;
7
8      @Resource
9      private RoleService roleService;
10
11     @Resource
12     private PasswordEncoder passwordEncoder;
13
14     /**
15      * 查询所有用户列表
16      * @return
17      */
18     @GetMapping("/listAll")
19     public Result listAll(){
20         return Result.ok(userService.list());
21     }
22
23     /**
24      * 查询用户列表
25      *
26      * @param userQueryVo
27      * @return
28      */
29     @GetMapping("/list")
30     public Result list(UserQueryVo userQueryVo) {
31         //创建分页信息
32         Page<User> page = new Page<User>(userQueryVo.getPageNo(),
33 userQueryVo.getPageSize());
34         //调用分页查询方法
35         userService.findUserListByPage(page, userQueryVo);
36         //返回数据
37         return Result.ok(page);
38     }
39
40     /**
41      * 添加用户
42      *
43      * @param user
44      * @return
45      */
46     @PostMapping("/add")
47     @PreAuthorize("hasAuthority('sys:user:add')")
48     public Result add(@RequestBody User user) {
49         //查询用户
50         User item = userService.findUserByUserName(user.getUsername());
51         //判断对象是否为空
52         if (item != null) {
53             return Result.error().message("该登录名称已被使用，请重新输入！");
54         }
55     }
56 }
```

```

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(),
110             roleQueryVo.getPageSize());
111         //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(),
138             userRoleDTO.getRoleIds())) {
139             return Result.ok().message("角色分配成功");
140         }
141         return Result.error().message("角色分配失败");
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;

```

```
11     }
12   }
13   return tag;
14 }
```

### 12.2.3 引入按钮权限判断脚本

在main.js中添加如下代码。

```
1 //导入按钮权限判断
2 import hasPermission from '@/permission/index'
3 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。

```
1 <el-button
2   icon="el-icon-close"
3   type="danger"
4   size="small"
5   @click="handleDelete(scope.row)"
6   v-if="hasPermission('sys:user:delete')"
7   >删除
8 </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

```
1 import { login, logout, getInfo } from '@api/user'
2 // 导入auth脚本
3 import { getToken, setToken, removeToken, setTokenTime } from '@utils/auth'
4 import router, { resetRouter } from '@router'
5
6 const state = {
7   token: getToken(), // 获取token信息
8   name: '',
9   avatar: '',
10  introduction: '',
11  roles: []
12 }
13
```

```

14  const mutations = {
15    //设置token信息
16    SET_TOKEN: (state, token) => {
17      state.token = token
18    },
19    //设置个人介绍
20    SET_INTRODUCTION: (state, introduction) => {
21      state.introduction = introduction
22    },
23    //设置用户姓名
24    SET_NAME: (state, name) => {
25      state.name = name
26    },
27    //设置用户头像
28    SET_AVATAR: (state, avatar) => {
29      state.avatar = avatar
30    },
31    //设置用户对应的角色
32    SET_ROLES: (state, roles) => {
33      state.roles = roles
34    }
35  }
36
37  const actions = {
38    // 用户登录
39    login({ commit }, userInfo) {
40      //从用户信息userInfo中解构出用户名和密码
41      const { username, password } = userInfo
42      return new Promise((resolve, reject) => {
43        //调用src/api/user.js文件中的login()方法
44        login({ username: username.trim(), password: password }).then(response => {
45          //从response中解构出返回的token数据
46          const { token, expireTime } = response
47          //将返回的token数据保存到store中，作为全局变量使用
48          commit('SET_TOKEN', token)
49          //将token信息保存到cookie中
50          setToken(token)
51          //设置token过期时间
52          setTokenTime(expireTime);
53          resolve()
54        }).catch(error => {
55          reject(error)
56        })
57      })
58    },
59  }

```

## 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。

```
1  /**
2   * 刷新token
3   * @returns
4   */
5  export async function refreshTokenApi(params){
6    return await http.post("/api/sysUser/refreshToken",params);
7  }
```

### 13.3 编写token过期方法

修改src/utlis下的auth.js代码。

```
1  import Cookies from 'js-cookie'
2
3  const TokenKey = 'Admin-Token'
4  //定义token过期时间的key
5  const timeKey = "expireTime";
6
7  /**
8   * 获取token
9   * @returns
10  */
11  export function getToken() {
12    return Cookies.get(TokenKey)
13  }
14
15  /**
16   * 设置token
17   * @returns
18   */
19  export function setToken(token) {
20    return Cookies.set(TokenKey, token)
21  }
22  /**
23   * 删除token
24   * @returns
25   */
26  export function removeToken() {
27    return Cookies.remove(TokenKey)
28  }
29
30  /**
31   * 清空sessionStorage
32   */
33  export function clearStorage(){
34    return sessionStorage.clear();
35  }
36
37  /**
38   * 设置token过期时间
39   * @returns
40   */
41  export function setTokenTime(time){
42    return sessionStorage.setItem(timeKey,time);
43  }
44
45  /**
```

```

46  * 获取token过期时间
47  * @returns
48  */
49  export function getTokenTime(){
50    return sessionStorage.getItem(timeKey);
51  }
52
53  /**
54   * 清空token过期时间
55   * @returns
56   */
57  export function removeTokenTime(){
58    return sessionStorage.setItem(timeKey,0);
59  }

```

## 13.4 编写请求拦截

修改src/utils下的request.js代码。

```

1  import axios from 'axios'
2  import { MessageBox, Message } from 'element-ui'
3  import store from '@/store'
4  //导入auth脚本
5  import {
6    getToken, setToken, clearStorage, getTokenTime, setTokenTime, removeTokenTime } from
7    '@/utils/auth'
8  import qs from 'qs'
9  //导入刷新token的api脚本
10 import { refreshTokenApi } from '@/api/user'
11
12 // create an axios instance
13 const service = axios.create({
14   baseURL: process.env.VUE_APP_BASE_API, // url = base url + request url
15   // withCredentials: true, // send cookies when cross-domain requests
16   timeout: 10000 // 请求超时时间
17 })
18
19 /**
20  * 刷新token
21  */
22 function refreshTokenInfo(){
23   //设置请求参数
24   let param = {
25     token:getToken()
26   }
27   return refreshTokenApi(param).then(res=>res);
28 }
29
30 //定义变量，标识是否刷新token
31 let isRefresh = false;
32 // 发送请求之前进行拦截
33 service.interceptors.request.use(
34   config => {
35     //获取当前系统时间
36     let currentTime = new Date().getTime();

```



```

37 //获取token过期时间
38 let expireTime = getTokenTime();
39 //判断token是否过期
40 if(expireTime>0){
41 //计算时间
42 let min = (expireTime - currentTime) / 1000 / 60;
43 //如果token离过期时间相差10分钟，则刷新token
44 if(min<10){
45 //判断是否刷新
46 if(!isRefresh){
47 //标识刷新
48 isRefresh = true;
49 //调用刷新token的方法
50 return refreshTokenInfo().then(res=>{
51 //判断是否成功
52 if(res.success){
53 //设置新的token和过期时间
54 setToken(res.data.token);
55 setTokenTime(res.data.expireTime);
56 //将新的token添加到header头部
57 config.headers.token = getToken();
58 }
59 return config;
60 }).catch(error=>{
61 }).finally(()=>{
62 //修改是否刷新token的状态
63 isRefresh = false;
64 });
65 }
66 }
67 }
68 // 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
69 if (store.getters.token) {
70 // 将token添加到请求的头部
71 config.headers['token'] = getToken()
72 }
73 return config
74 },
75 error => {
76 //清空sessionStorage
77 clearStorage();
78 //清空token过期时间
79 removeTokenTime();
80 // do something with request error
81 return Promise.reject(error)
82 }
83 )
84
85 // response interceptor
86 service.interceptors.response.use(
87 response => {
88 const res = response.data
89 // if the custom code is not 20000, it is judged as an error.
90 if (res.code !== 200) {
91 Message({
92 message: res.message || 'Error',
93 type: 'error',
94 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 !== '') {
172           _params += `${params[key]}/`
173         }
174         //去掉参数最后一位?
175         _params = _params.substr(0, _params.length - 1)
176       }
177       if (_params) {
178         return service.get(`${url}${_params}`)
179       } else {
180         return service.get(url)
181       }
182     },
183   delete(url, params) {
184     let _params
185     if (Object.is(params, undefined || null)) {
186       _params = ''
187     } else {
188       _params = '/'
189       for (const key in params) {
190         // eslint-disable-next-line no-prototype-builtins
191         if (params.hasOwnProperty(key) && params[key] !== null && params[key]
192 !== '') {
193           _params += `${params[key]}/`
194         }
195       }
196       //去掉参数最后一位?
197       _params = _params.substr(0, _params.length - 1)
198     }
199     if (_params) {
200       return service.delete(`${url}${_params}`).catch(err => {
201         message.error(err.msg)
202         return Promise.reject(err)
203       })
204     } else {
205       return service.delete(url).catch(err => {
206         message.error(err.msg)
207         return Promise.reject(err)
208       })
209     }
210   }
211 }

```

```
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
```

尊重劳动成果，禁止盗用