第8章 授权（阶段实战）

学习目标

- 能够描述或描绘RBAC模型的7张表及其关系
- 理解管理员角色设置的需求与实现思路，完成管理员角色设置功能
- 理解角色权限设置的需求与实现思路，完成角色权限设置的功能
- 掌握spring security授权控制的两种方式，实现用户权限控制的功能
- 理解用户菜单展示的需求与实现思路，完成用户菜单展示的功能

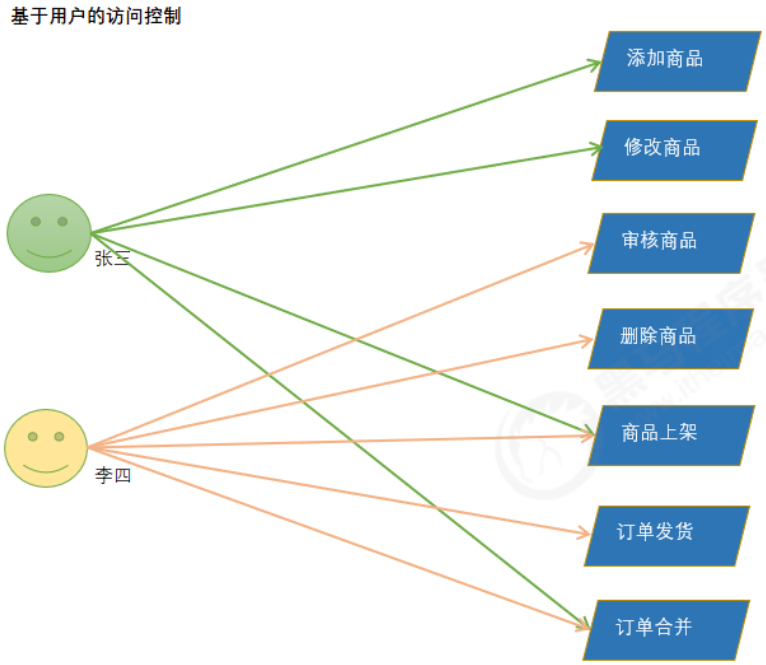
1. RBAC模型

1.1 什么是RBAC

权限系统提的最多的就是 RBAC（Role-Based Access Control 基于角色的访问控制）。 所谓角色，其实就是权限的集合，某个角色就是某几个权限的结合。其目的是为了简化授权和鉴权的过程。

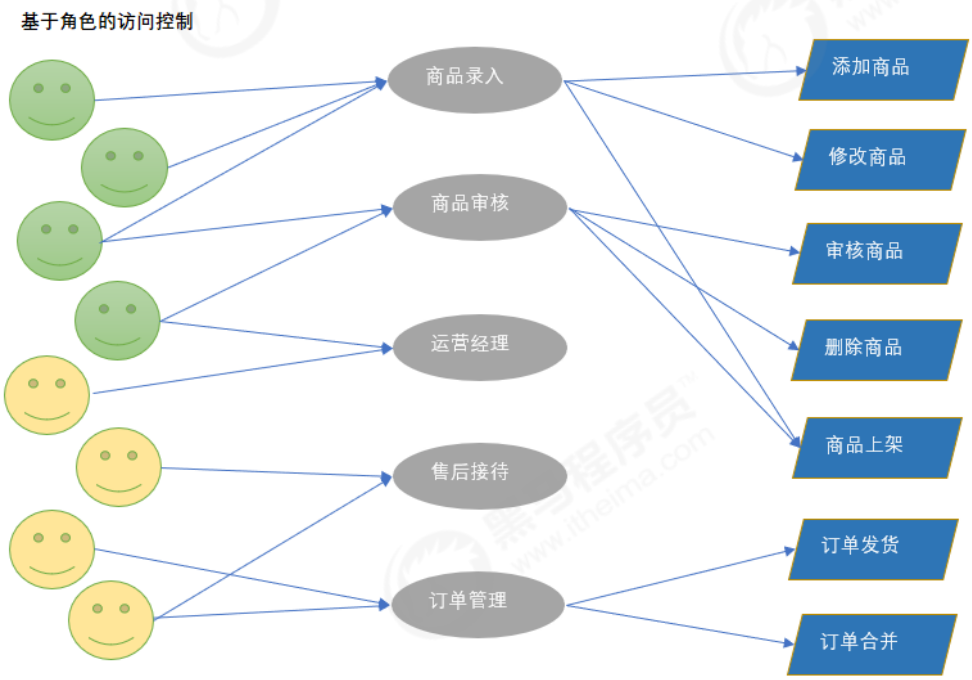
首选说一下为什么要进行访问控制。为了屏蔽无关人员操作系统的某个功能，防止误操作产生的垃圾数据或数据丢失。

小公司和比较简单的权限系统使用的基于用户的访问控制如下:



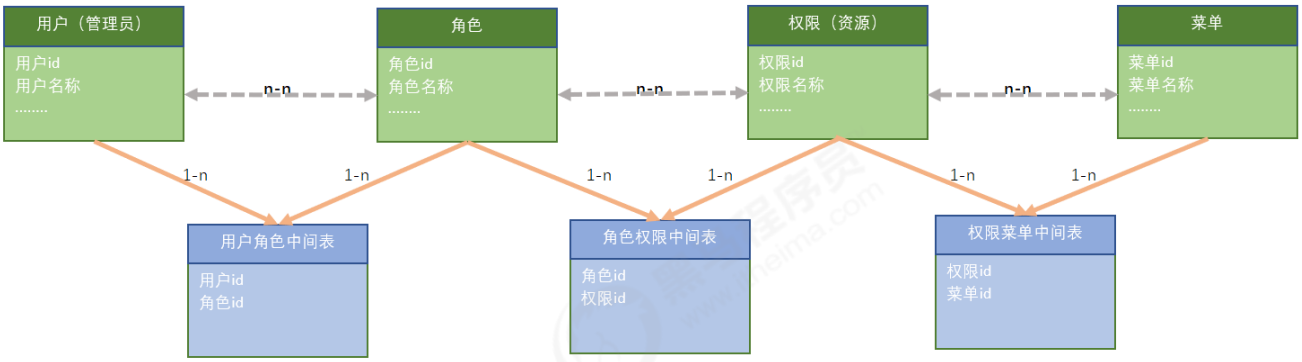
这种访问控制只适用于操作人员比较少的系统，如果操作人员较多，对每个操作人员都进行授权操作，无疑是非常繁琐的。

所以我们就需要基于角色的访问控制



1.2 表结构分析

企业开发中 RBAC模型设计为7张表，其中4张为基础表，3张为中间表。



1.2.1 用户与角色

用户和角色为多对多关系，通过用户角色中间表关联

tb_admin 管理员表：

字段名称	字段含义	字段类型	字段长度	备注
id	id	INT		主键
login_name	登录名称	VARCHAR		
password	密码			
status	状态			

tb_role 角色表：

字段名称	字段含义	字段类型	字段长度	备注
id	角色id	INT		主键
name	角色名称	VARCHAR		

tb_admin_role 管理员角色中间表：

字段名称	字段含义	字段类型	字段长度	备注
admin_id	管理员id	INT		主键
role_id	角色id	INT		主键

1.2.2 角色与权限

角色和权限为多对多关系，通过角色权限中间表关联

tb_role 角色表：

字段名称	字段含义	字段类型	字段长度	备注
id	角色id	INT		主键
name	角色名称	VARCHAR		

tb_resource 权限表（资源表）：

字段名称	字段含义	字段类型	字段长度	备注
id	资源id	INT		主键
res_key	资源key	VARCHAR		系统定义好的权限标识
res_name	资源名称	VARCHAR		
parent_id	上级资源id	INT		权限表的层次结构为2层

tb_role_resource 角色资源中间表：

字段名称	字段含义	字段类型	字段长度	备注
role_id	角色id	INT		主键
resource_id	资源id	INT		主键

1.2.3 权限与菜单

权限和菜单为多对多关系，通过权限菜单中间表关联

tb_resource 权限表（资源表）：

字段名称	字段含义	字段类型	字段长度	备注
id	资源id	INT		主键
res_key	资源key	VARCHAR		系统定义好的权限标识
res_name	资源名称	VARCHAR		
parent_id	上级资源id	INT		权限表的层次结构为2层

tb_menu 菜单表：

字段名称	字段含义	字段类型	字段长度	备注
id	id	VARCHAR		主键
name	菜单名称	VARCHAR		
……				
parent_id	上级菜单id	VARCHAR		菜单表的层次结构为3层

tb_resource_menu 资源菜单中间表：

字段名称	字段含义	字段类型	字段长度	备注
resource_id	资源id	INT		主键
menu_id	菜单id	VARCHAR		主键

2. 管理员角色设置

2.1 需求分析

管理员和角色为多对多关系，在保存管理员时实现对管理员角色中间表的添加。详见静态原型。

在修改某个管理员时，需要将该管理员的角色id列表读取出来。

2.2 思路分析

2.2.1 提交角色设置

（1）创建管理员角色中间表的实体类和数据访问接口

（2）创建组合实体类，包含管理员实体类和角色ID集合两个属性

（3）修改管理员实体类，为id添加以下注解可以标识该主键为自增

```
@GeneratedValue(strategy=GenerationType.IDENTITY)
```

添加此注解后，可以获取新增的id值。

（4）修改管理员add方法，取出管理员实体保存，取出角色ID集合，循环添加到管理员角色中间表。

（5）注意在保存管理员密码时需要对密码进行bcrypt加密。

（6）前端可以根据黑马架构师生成的代码修改即可。“所属角色”使用elementui的select选择器，为`el-select`设置`multiple`属性即可启用多选，详见官方文档。

2.2.2 读取角色设置

（1）修改findById方法的返回值为组合实体类，修改其中的逻辑，组合实体类的角色id集合需要查询管理员角色中间表。

（2）修改update方法，删除原来的相关的中间表数据，再循环添加中间表数据。

（3）读取后需要把密码属性设置为null，如果用户没有在界面输入密码则保持密码不变，如果填写了密码需要进行bcrypt加密。

3. 角色权限设置

3.1 需求分析

显示所有的权限列表，并自动勾选已经保存的权限。用户勾选权限后，点击提交，将勾选的权限id提交给后端保存

☐ 商品管理					
☐ 商品添加/编辑	☐ 商品删除/恢复	☐ 分类添加/编辑	☐ 分类转移/删除	☐ 商品属性管理	☐ 商品评价管理
☐ 商品自动上下架	☐ 图片批量处理	☐ 商品批量导出	☐ 图片库管理	☐ 商品添加/编辑	☐ 商品删除/恢复
☐ 分类转移/删除	☐ 商品属性管理	☐ 商品评价管理	☐ 商品自动上下架	☐ 图片批量处理	☐ 商品批量导出

☐ 订单管理					
☐ 商品添加/编辑	☐ 商品删除/恢复	☐ 分类添加/编辑	☐ 分类转移/删除	☐ 商品属性管理	☐ 商品评价管理
☐ 商品自动上下架	☐ 图片批量处理	☐ 商品批量导出	☐ 图片库管理	☐ 商品添加/编辑	☐ 商品删除/恢复
☐ 分类转移/删除	☐ 商品属性管理	☐ 商品评价管理	☐ 商品自动上下架	☐ 图片批量处理	☐ 商品批量导出

3.2 思路分析

3.2.1 提交权限设置

（1）创建角色权限中间表的实体类和数据访问接口

（2）前后端约定要提交的数据格式，包括“角色id”和“权限id列表”。根据约定的数据格式创建组合实体类。

（3）后端添加方法，接收组合实体类参数，提取“角色id”和“权限id列表”，循环读取权限id插入到角色权限中间表中。

3.2.2 读取权限设置

（1）后端查询权限表（资源表），以树状结构返回数据。前端使用两层v-for循环输出列表。

（2）后端添加方法，根据角色查询权限id列表，前端获取权限id列表后实现复选框的勾选。

4. 用户权限控制

4.1 需求分析

当用户执行一个不存在的权限的url，需要拦截请求。

4.2 spring security授权控制

4.2.1 基于URL访问控制

在UserDetailsServiceImpl的loadUserByUsername方法，实现对当前用户的授权

```
//构建权限列表
List<GrantedAuthority> grantedAuths = new ArrayList<GrantedAuthority>();
grantedAuths.add(new SimpleGrantedAuthority("goods_add"));
grantedAuths.add(new SimpleGrantedAuthority("goods_edit"));
.....
```

修改applicationContext_security.xml

```
<intercept-url pattern="/*/find*.do" access="hasAnyAuthority()" />
<intercept-url pattern="/brand/*.do" access="hasAuthority('brand')" />
<intercept-url pattern="/spu/save.do"
access="hasAnyAuthority('goods_add','goods_edit')" />
```

hasAnyAuthority()：拥有任意权限都可以访问

hasAuthority('brand')： 拥有brand的权限可以访问

hasAnyAuthority('goods_add','goods_edit')" ：拥有goods_add和goods_edit其中一个权限就可以访问

4.2.2 基于方法的访问控制

对当前用户授权，同上 ，对方法的访问控制如下：

（1）修改applicationContext_security.xml ， 增加配置 ， 启用注解

```
<global-method-security pre-post-annotations="enabled" />
```

（2）在进行权限控制的方法上添加注解

```
@PreAuthorize("hasAuthority('brand')")
```

brand为资源key

4.3 思路分析

（1）编写SQL语句，通过登录名查询资源KEY列表

```
SELECT res_key FROM tb_resource WHERE id IN (
  SELECT resource_id FROM tb_role_resource WHERE role_id IN (
     SELECT role_id FROM tb_admin_role WHERE admin_id IN  (
       SELECT id FROM tb_admin WHERE login_name='admin'
     )
  )
)
```

这里我们用到了4层嵌套循环

（2）数据访问接口新增方法，根据登录名查询资源KEY列表

（3）服务层实现根据登录名查询资源KEY列表

（4）UserDetailsServiceImpl的loadUserByUsername方法，调用根据登录名查询资源KEY列表的方法，将资源key列表添加到当前用户。

```
//构建权限列表
List<GrantedAuthority> grantedAuths = new ArrayList<GrantedAuthority>();
List<String> resKeyList = resourceService.findResKeyByLoginName(s);
for(String resKey :resKeyList ){
    grantedAuths.add(new SimpleGrantedAuthority(resKey));
}
```

（5）修改applicationContext_security.xml，添加对url的拦截，或在方法上添加注解实现对方法的拦截。

5. 用户菜单筛选

5.1 需求分析

用户登录后进入主界面，显示的菜单为用户所拥有的权限关联的菜单。不具有权限的菜单不显示。

5.2 思路分析

（1）编写SQL语句，根据当前登录名获取菜单列表的方法

```
SELECT * FROM tb_menu WHERE id IN(
    SELECT menu_id FROM tb_resource_menu WHERE resource_id IN (
      SELECT resource_id FROM tb_role_resource WHERE role_id IN (
         SELECT role_id FROM tb_admin_role WHERE admin_id IN  (
           SELECT id FROM tb_admin WHERE login_name='admin'
         )
      )
   )
)
```

注意通过上述语句，获取的菜单列表只包含三级菜单，而我们需要返回包括一级菜单、二级菜单和三级菜单的菜单列表。只要三级菜单存在，就要有它的二级菜单；只要有一个二级菜单就要有它的一级菜单。

查询二级菜单列表：

```
SELECT * FROM tb_menu WHERE id IN(
   SELECT parent_id FROM tb_menu WHERE id IN(
    SELECT menu_id FROM tb_resource_menu WHERE resource_id IN (
      SELECT resource_id FROM tb_role_resource WHERE role_id IN (
         SELECT role_id FROM tb_admin_role WHERE admin_id IN  (
           SELECT id FROM tb_admin WHERE login_name='admin'
         )
      )
    )
  )
)
```

查询一级菜单列表：

```
SELECT * FROM tb_menu WHERE id IN (
  SELECT parent_id FROM tb_menu WHERE id IN(
   SELECT parent_id FROM tb_menu WHERE id IN(
    SELECT menu_id FROM tb_resource_menu WHERE resource_id IN (
      SELECT resource_id FROM tb_role_resource WHERE role_id IN (
         SELECT role_id FROM tb_admin_role WHERE admin_id IN  (
           SELECT id FROM tb_admin WHERE login_name='admin'
         )
       )
      )
    )
  )
)
```

最后我们通过UNION运算符将三个语句连接成一条语句

```
SELECT * FROM tb_menu WHERE id IN(
    SELECT menu_id FROM tb_resource_menu WHERE resource_id IN (
      SELECT resource_id FROM tb_role_resource WHERE role_id IN (
         SELECT role_id FROM tb_admin_role WHERE admin_id IN  (
           SELECT id FROM tb_admin WHERE login_name='admin'
         )
      )
    )
)
UNION
SELECT * FROM tb_menu WHERE id IN(
   SELECT parent_id FROM tb_menu WHERE id IN(
    SELECT menu_id FROM tb_resource_menu WHERE resource_id IN (
      SELECT resource_id FROM tb_role_resource WHERE role_id IN (
         SELECT role_id FROM tb_admin_role WHERE admin_id IN  (
           SELECT id FROM tb_admin WHERE login_name='admin'
         )
       )
     )
   )
)
UNION
SELECT * FROM tb_menu WHERE id IN (
  SELECT parent_id FROM tb_menu WHERE id IN(
   SELECT parent_id FROM tb_menu WHERE id IN(
    SELECT menu_id FROM tb_resource_menu WHERE resource_id IN (
      SELECT resource_id FROM tb_role_resource WHERE role_id IN (
         SELECT role_id FROM tb_admin_role WHERE admin_id IN  (
           SELECT id FROM tb_admin WHERE login_name='admin'
         )
       )
      )
    )
  )
)
```

（2）将上述SQL封装为查询方法

（3）在controller获取当前用户名，调用上述查询方法。