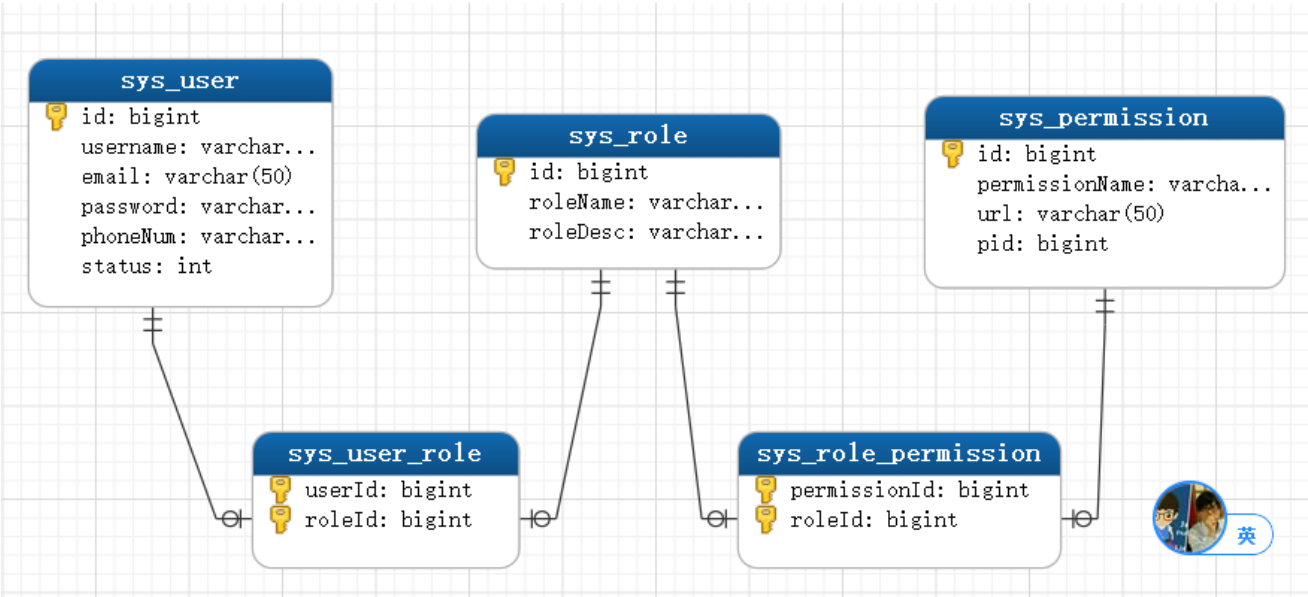
ssm练习第五天

第一章：授权操作

第一节：经典权限5张表的关系分析

1、数据库模型关系图



2、用户角色关系表

```
CREATE TABLE sys_user_role(
    userId BIGINT,
    roleId BIGINT,
    PRIMARY KEY(userId,roleId),
    FOREIGN KEY (userId) REFERENCES sys_USER(id),
    FOREIGN KEY (roleId) REFERENCES sys_role(id)
)
```

3、角色权限关系表

```
CREATE TABLE sys_role_permission(
    permissionId BIGINT,
    roleId BIGINT,
    PRIMARY KEY(permissionId,roleId),
    FOREIGN KEY (permissionId) REFERENCES sys_permission(id),
    FOREIGN KEY (roleId) REFERENCES sys_role(id)
)
```

3、修改用户SysUser实体

```java
public class SysUser {

    private Long id;
    private String username;
    private String email;
    private String password;
    private String phoneNum;
    private int status;

    //该用户包含哪些角色
    private List<Role> roleList;

    //此处省略getter和setter方法... ...

}
```

4、修改角色Role实体

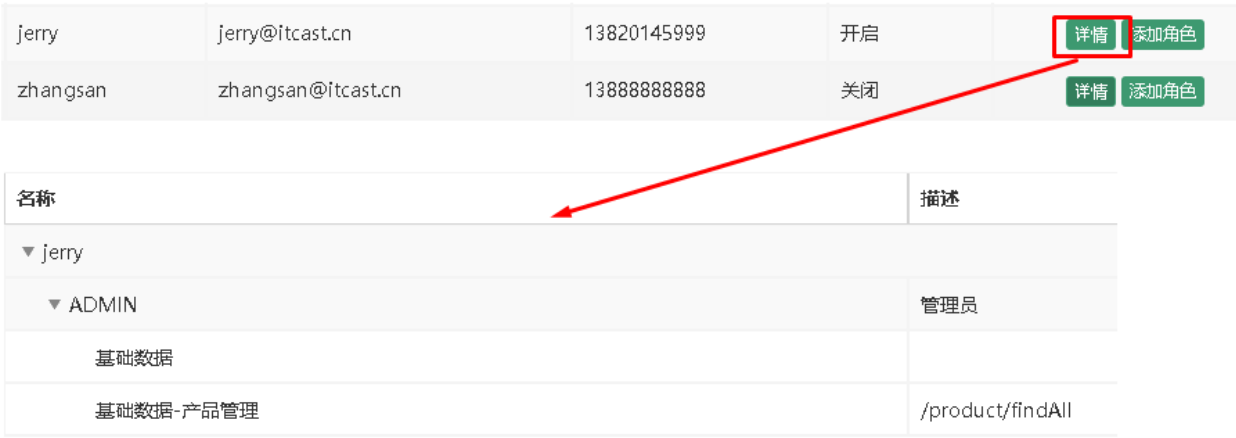
```java
public class Role {

    private Long id;
    private String roleName;
    private String roleDesc;

    //该角色具备哪些权限
    private List<Permission> permissionList;

    //该角色被哪些用户使用
    private List<SysUser> userList;

    //此处省略getter和setter方法... ...
}
```

5、修改权限Permission实体

```java
public class Permission {

    private Long id;
    private String permissionName;
    private String url;
    private Long pid;

    //该权限被哪些角色所拥有
    private List<Role> roleList;

    //此处省略getter和setter方法... ...
}
```

第二节：查看用户的角色详情功能

1、页面入口



2、编写Controller

```
@RequestMapping("/findById")
public ModelAndView findById(Long id){
    SysUser user = userService.findById(id);
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("user",user);
    modelAndView.setViewName("user-show");
    return modelAndView;
}
```

3、编写Service

接口

```
SysUser findById(Long id);
```

实现

```
@Override
public SysUser findById(Long id) {
    return userMapper.findById(id);
}
```

4、编写Mapper

UserMapper接口

```
@Select("select * from sys_user where id=#{id}")
@Results({
    @Result(id = true,property = "id",column = "id"),
    @Result(
        property = "roleList",
        column = "id",
        javaType = List.class,
        many = @Many(select = "com.itheima.dao.RoleMapper.findByUid",fetchType = FetchType.LAZY))
})
SysUser findById(Long id);
```

RoleMapper接口

```
@Select("select r.* from sys_role r,sys_user_role ur where r.id=ur.roleId and ur.userId=#{id}")
@Results({
    @Result(id = true,column = "id",property = "id"),
    @Result(
        property = "permissionList",
        column = "id",
        javaType = List.class,
        many=@Many(select = "com.itheima.dao.PermissionMapper.findByRId",fetchType = FetchType.LAZY)
    )
})
List<Role> findByUid(Long id);
```

PermissionMapper接口

```
@Select("select p.* from sys_permission p,sys_role_permission rp where p.id=rp.permissionId and rp.roleId=#{rid}")
List<Permission> findByRId(Long rid);
```

第三节：为用户分配角色-角色列表数据回显

1、页面入口

2、编写Controller

```
@RequestMapping("/addUserRoleUI")
public ModelAndView addUserRoleUI(Long id) {
    //查询所有的角色列表
    List<Role> roleList = roleService.findAll();
    //查询当前用户的角色列表
    SysUser user = userService.findById(id);
    //获得当前用户的角色
    List<Role> userRoleList = user.getRoleList();
    //拼接角色id的字符串
    // 1,2,3,
    StringBuffer stringBuffer = new StringBuffer();
    for (Role role : userRoleList) {
        stringBuffer.append(role.getId());
        stringBuffer.append(",");
    }

    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("roleList", roleList);
    modelAndView.addObject("userId", user.getId());
    modelAndView.addObject("userRoleStr", stringBuffer.toString());
    modelAndView.setViewName("user-role-add");
    return modelAndView;
}
```

3、页面回显数据

```
<input type="hidden" name="userId" value="${userId}">

<c:forEach items="${roleList}" var="role">
    <tr>
        <td>
            <input name="ids" type="checkbox" ${fn:contains(userRoleStr, role.id)?"checked":""}  value="${role.id}">
        </td>
        <td>${role.id}</td>
        <td>${role.roleName }</td>
        <td>${role.roleDesc}</td>
    </tr>
</c:forEach>
```

第四节：为用户分配角色-更新关系到用户角色中间表

1、页面入口

2、编写Controller

```
@RequestMapping("/addRoleToUser")
public String addRoleToUser(Long userId,Long[] ids) {
    userService.addRoleToUser(userId,ids);
    return "redirect:/user/findAll";
}
```

3、编写Service

接口

```
void addRoleToUser(Long userId, Long[] ids);
```

实现

```java
@Override
public void addRoleToUser(Long userId, Long[] ids) {
    //删除当前用户的所有关系，在添加新的关系
    userMapper.deleteUserRoleRelation(userId);

    if(ids!=null&&ids.length>0){
        //添加新关系
        for (Long roleId : ids) {
            userMapper.addRoleToUser(userId,roleId);
        }
    }

}
```

4、编写Mapper

```java
@Delete("delete from sys_user_role where userId=#{userId}")
void deleteUserRoleRelation(Long userId);

@Insert("insert into sys_user_role values(#{param1},#{param2})")
void addRoleToUser(Long userId,Long roleId);
```

第五节：为用户设置真正的角色

修改UserServiceImpl的loadUserByUsername方法，为用户设置真正的角色

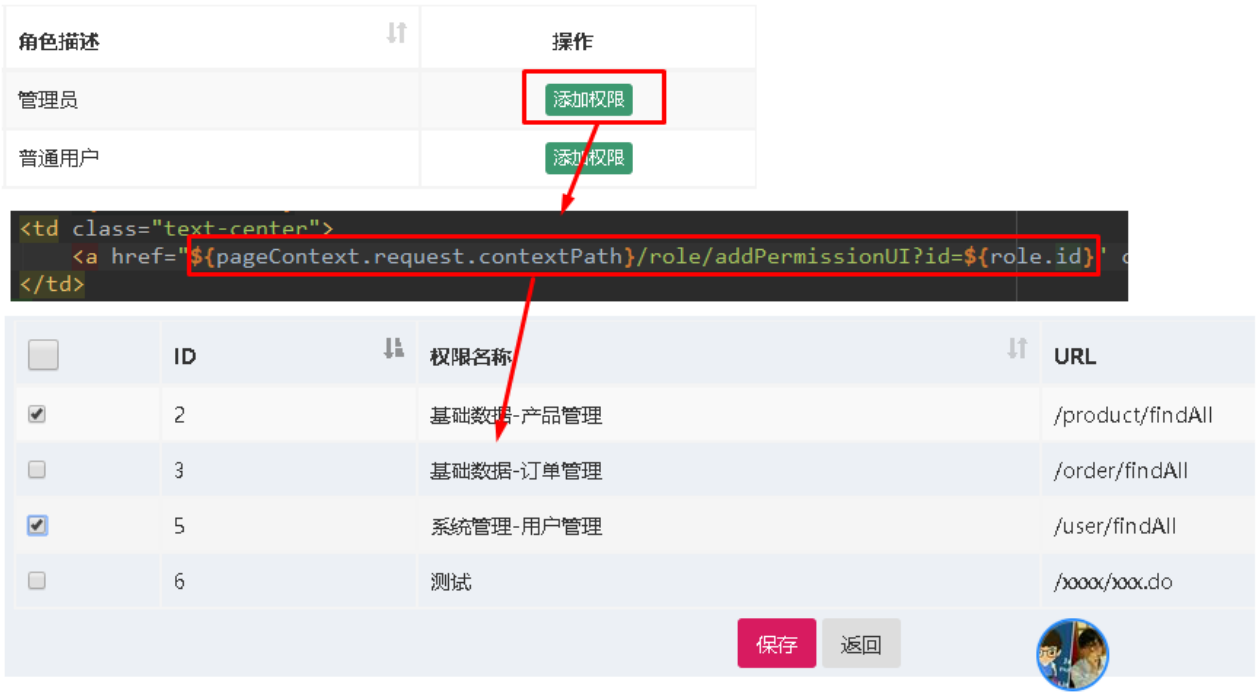
```java
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

    // 通过用户名查询密码
    SysUser sysUser = userMapper.findByUsername(username);
    //判断认证是否成功
    if(sysUser!=null){
        List<GrantedAuthority> authorities = new ArrayList<>();
        //获得用户的角色列表
        List<Role> roleList = sysUser.getRoleList();
        for (Role role : roleList) {
            authorities.add(new SimpleGrantedAuthority("ROLE_"+role.getRoleName()));
        }

        User user = new User(username,sysUser.getPassword(),authorities);
        return user;
    }

    return null;
}
```

修改UserMapper

```
@Select("select * from sys_user where username = #{username}")
@Results({
    @Result(id = true,property = "id",column = "id"),
    @Result(
        property = "roleList",
        column = "id",
        javaType = List.class,
        many = @Many(select = "com.itheima.dao.RoleMapper.findByUid",fetchType = FetchType.LAZY))
})
SysUser findByUsername(String username);
```

第六节：为角色添加权限-权限列表回显

1、页面入口



2、编写Controller

```
@RequestMapping("/addPermissionUI")
public ModelAndView addPermissionUI(Long id){
    //获得全部的非父的权限列表
    List<Permission> permissionList = permissionService.findNotParentList();

    //获得当前角色的权限列表
    Role role = roleService.findById(id);
    List<Permission> rolePermissionList = role.getPermissionList();
    StringBuffer stringBuffer = new StringBuffer();
    for (Permission permission : rolePermissionList) {
        stringBuffer.append(permission.getId());
```

```
        stringBuffer.append(",");
    }

    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("role",role);
    modelAndView.addObject("permissionList",permissionList);
    modelAndView.addObject("rolePermissionStr",stringBuffer.toString());
    modelAndView.setViewName("role-permission-add");
    return modelAndView;
}
```

3、编写Service

接口

```
List<Permission> findNotParentList();
```

实现

```
@Override
public List<Permission> findNotParentList() {
    return permissionMapper.findNotParentList();
}
```

4、编写Mapper

```
@Select("select * from sys_permission where pid!=0")
List<Permission> findNotParentList();
```

第七节：为角色添加权限-更新角色权限关系表

1、页面入口

2、编写Controller

```
@RequestMapping("/addPermissionToRole")
public String addPermissionToRole(Long roleId,Long[] ids){
    roleService.addPermissionToRole(roleId,ids);
    return  "redirect:/role/findAll";
}
```

3、编写Service

接口

```
void addPermissionToRole(Long roleId, Long[] ids);
```

实现

```
@Override
public void addPermissionToRole(Long roleId, Long[] ids) {
    //删除所有角色权限的关系
    roleMapper.deleteRolePermission(roleId);
    //为角色添加新的权限
    if(ids!=null&&ids.length>0){
        for (Long id : ids) {
            roleMapper.addRolePermission(roleId,id);
        }
    }
}
```

4、编写Mapper

```
@Delete("delete from sys_role_permission where roleId=#{roleId}")
void deleteRolePermission(Long roleId);

@Delete("insert into sys_role_permission(roleId,permissionId) values(#{param1},#{param2})")
void addRolePermission(Long roleId, Long id);
```

第二章：授权后的安全控制

第一节：在JSP页面控制菜单权限

在JSP页面中使用security:authorize标签，可以控制菜单是否显示。security:authorize标签的access="hasAnyRole('ROLE_USER','ROLE_ADMIN')"。因为标签的access使用的是表达式，所以需要将spring-security.xml配置文件的use-expressions设置为true。

spring-security.xml配置文件

```
<security:http auto-config="true" use-expressions="true">
    <!-- 配置拦截的请求地址，任何请求地址都必须有ROLE_USER的权限 -->
    <security:intercept-url pattern="/**" access="hasAnyRole('ROLE_ADMIN','ROLE_USER')"/>
    <!-- 配置具体的页面跳转 -->
    <security:form-login
                        login-page="/login.jsp"
                        login-processing-url="/login"
                        default-target-url="/index.jsp"
                        />

    <!-- 关闭跨越请求 -->
    <security:csrf disabled="true"/>

    <!-- 退出 -->
    <security:logout invalidate-session="true" logout-url="/logout" logout-success-url="/login.html"/>

</security:http>
```

在菜单页面aside.jsp中使用security标签进行菜单的选择性显示

```
<%--只有ROLE_ADMIN才能使用--%>
<security:authorize access="hasRole('ROLE_ADMIN')">
    <li class="treeview">
        <a href="#">
            <i class="fa fa-cogs"></i>
            <span>系统管理</span>
            <span class="pull-right-container">
                <i class="fa fa-angle-left pull-right"></i>
            </span>
```

```
        </a>
        <ul class="treeview-menu">
            <li id="system-setting">
                <a href="${pageContext.request.contextPath}/user/findAll">
                    <i class="fa fa-circle-o"></i> 用户管理
                </a>
            </li>
            <li id="system-setting">
                <a href="${pageContext.request.contextPath}/role/findAll">
                    <i class="fa fa-circle-o"></i> 角色管理
                </a>
            </li>
            <li id="system-setting">
                <a href="${pageContext.request.contextPath}/permission/findAll">
                    <i class="fa fa-circle-o"></i> 权限管理
                </a></li>
            <li id="system-setting">
                <a href="${pageContext.request.contextPath}/pages/syslog-list.jsp">
                    <i class="fa fa-circle-o"></i> 访问日志
                </a>
            </li>
        </ul>
    </li>
</security:authorize>

<%--ROLE_ADMIN和ROLE_USER都可以使用--%>
<security:authorize access="hasAnyRole('ROLE_ADMIN','ROLE_USER')">
    <li class="treeview">
        <a href="#">
            <i class="fa fa-cube"></i>
            <span>基础数据</span>
            <span class="pull-right-container">
                <i class="fa fa-angle-left pull-right"></i>
            </span>
        </a>
        <ul class="treeview-menu">
            <li id="system-setting">
                <a href="${pageContext.request.contextPath}/product/findAll">
                    <i class="fa fa-circle-o"></i> 产品管理
                </a>
            </li>
            <li id="system-setting">
                <a href="${pageContext.request.contextPath}/product/findByPage">
                    <i class="fa fa-circle-o"></i> 产品管理（手动分页）
                </a>
            </li>
            <li id="system-setting">
                <a
href="${pageContext.request.contextPath}/product/findByPageHelper">
                    <i class="fa fa-circle-o"></i> 产品管理（pageHelper分页）
                </a>
            </li>
            <li id="system-setting">
```

```
                <a href="${pageContext.request.contextPath}/order/findAll">
                    <i class="fa fa-circle-o"></i> 订单管理
                </a>
            </li>
        </ul>
    </li>
</security:authorize>
```

第二节：在服务器端控制权限

虽然页面菜单对不同角色显示类不同的菜单，但是如果直接访问服务器端的url还是可以访问到资源信息的，所以好需要对服务器端的资源进行安全控制。

控制方式就是借助于Spring的AOP，对Controller的的访问进行权限的功能增强。

修改spring-security.xml配置文件，添加aop的自定代理

```
<!--aop的自动代理
            其中proxy-target-class设置为true代表目标类的代理对象不需要借助于接口
            即使用cglib基于子类生成目标对象的代理对象
    -->
<aop:aspectj-autoproxy proxy-target-class="true"></aop:aspectj-autoproxy>
```

修改spring-security.xml中配置开启注解支持

```
<!--配置开启security的注解支持-->
<security:global-method-security secured-annotations="enabled"/>
```

在需要进行控制的Controller上添加@Secured注解

```
@Secured({"ROLE_ADMIN"})
public class UserController

@Secured({"ROLE_ADMIN","ROLE_USER"})
public class ProductController
```

第三章：系统日志功能

1、日志表sys_log和实体Log

sql语句

```sql
CREATE TABLE sys_log(
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    visitTime DATETIME,
    username VARCHAR(50),
    ip VARCHAR(30),
    method VARCHAR(200)
)
```

实体类

```java
public class SysLog {

    private Long id;
    private String visitTime;
    private String username;
    private String ip;
    private String method;

    //此处省略getter和setter方法... ...
}
```

2、springmvc.xml配置文件中开启aop的自动代理

```xml
<!--aop的自动代理
            其中proxy-target-class设置为true代表目标类的代理对象不需要借助于接口
            即使用cglib基于子类生成目标对象的代理对象
-->
<aop:aspectj-autoproxy proxy-target-class="true"></aop:aspectj-autoproxy>
```

3、在web.xml中配置监听request对象的监听器

```xml
<!-- 配置监听器，监听request域对象的创建和销毁的 -->
<listener>
    <listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>
</listener>
```

4、编写切面类（切面类内部有增强）

LogAop代码实现

```java
package com.itheima.aop;

import com.itheima.domain.SysLog;
import com.itheima.service.LogService;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.User;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import java.text.SimpleDateFormat;
import java.util.Date;

@Component
@Aspect
public class LogAop {

    @Autowired
    private HttpServletRequest request;

    @Autowired
    private LogService logService;

    @After("execution(* com.itheima.controller.*.*(..))")
    public void after(JoinPoint joinPoint){
        //封装Log实体，将实体传递给service 进行数据存储
        SysLog log = new SysLog();
        //1、private String visitTime;
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String visitTime = format.format(new Date());
        log.setVisitTime(visitTime);
        //2、private String username;
        User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String username = user.getUsername();
        log.setUsername(username);
        //3、private String ip;
        String remoteAddr = request.getRemoteAddr();
        log.setIp(remoteAddr);
        //4、private String method;
        Class<?> aClass = joinPoint.getTarget().getClass();//获得目标对象的字节码对象
        String className = aClass.getSimpleName(); //获得类名
        String methodName = joinPoint.getSignature().getName();//获得方法名称
        String method = className+"."+methodName;
        log.setMethod(method);

        //调用service方法
        logService.save(log);

    }

}

```

Service代码实现

接口

```java
package com.itheima.service;

import com.itheima.domain.SysLog;

public interface LogService {
    void save(SysLog log);
}

```

实现

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

import com.itheima.dao.LogMapper;
import com.itheima.domain.SysLog;
import com.itheima.service.LogService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service("logService")
public class LogServiceImpl implements LogService {

    @Autowired
    private LogMapper logMapper;

    @Override
    public void save(SysLog log) {
        logMapper.save(log);
    }
}

```

Mapper代码实现

```java
package com.itheima.dao;

import com.itheima.domain.SysLog;
import org.apache.ibatis.annotations.Insert;
import org.springframework.stereotype.Repository;

@Repository
public interface LogMapper {
    @Insert("insert into sys_log(visitTime,username,ip,method) values(#{visitTime},#{username},#{ip},#{method})")
    void save(SysLog log);
}

```