ssm练习第四天

第一章：用户模块

第一节：用户角色权限模块表结构

1. 角色表

1.角色表信息描述role

序号	字段名称	字段类型	字段描述
1	id	number	无意义，主键自动增长
2	roleName	varchar2	角色名
3	roleDesc	varchar2	角色描述

2.sql语句

```
CREATE TABLE sys_role(
    id NUMBER(9) PRIMARY KEY,
    roleName VARCHAR2(50) ,
    roleDesc VARCHAR2(50)
)
```

3.实体类

```
public class Role {

    private Long id;
    private String roleName;
    private String roleDesc;
    private List<Permission> permissions;

}
```

4.用户与角色关联关系

用户与角色之间是多对多关系，我们通过user_role表来描述其关联，在实体类中User中存在List，在Role中有List.而角色与权限之间也存在关系，我们会在后面介绍。

```
CREATE TABLE sys_user_role(
    userId NUMBER(9),
    roleId NUMBER(9),
    PRIMARY KEY(userId,roleId),
    FOREIGN KEY (userId) REFERENCES sys_USER(id),
    FOREIGN KEY (roleId) REFERENCES sys_role(id)
)
```

2. 权限资源表

1.权限资源表描述permission

序号	字段名称	字段类型	字段描述
1	id	number	无意义，主键自动增长
2	permissionName	varchar2	权限名
3	url	varchar2	资源路径

2.sql语句

```
CREATE TABLE sys_permission(
    id NUMBER(9) PRIMARY KEY ,
    permissionName VARCHAR2(50) ,
    url VARCHAR2(50)
)
```

3.实体类

```
public class Permission {

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

}
```

4.权限资源与角色关联关系

权限资源与角色是多对多关系，我们使用role_permission表来描述。在实体类Permission中存在List,在Role类中有List

```
CREATE TABLE sys_role_permission(
    permissionId NUMBER(9),
    roleId NUMBER(9),
    PRIMARY KEY(permissionId,roleId),
    FOREIGN KEY (permissionId) REFERENCES sys_permission(id),
    FOREIGN KEY (roleId) REFERENCES sys_role(id)
)
```

第二节：用户退出功能

1. 在header.jsp的页面中编写超链接

```
<a href="${pageContext.request.contextPath}/logout"
                                        class="btn btn-default btn-flat">注销</a>
```

第三节：用户的列表查询功能

1. 代码如下

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

    @Autowired
    private UserService userService;

    @RequestMapping("/findAll")
    public ModelAndView findAll() {
        List<SysUser> ulist = userService.findAll();
        ModelAndView mv = new ModelAndView();
        mv.addObject("ulist", ulist);
        mv.setViewName("user-list");
        return mv;
    }

}

    public List<SysUser> findAll() {
        return userDao.findAll();
    }

    @Select("select * from sys_user")
    List<SysUser> findAll();
```

第四节：用户添加功能

1. MD5加密介绍
    1. MD5是一个加密的算法，可以对铭文字符串进行加密，得到一个密文。
    2. MD5原文和密文是一对一的键值对。
        1. 例如：原文：123 密文：202cb962ac59075b964b07152d234b70
        2. 不管执行多少次，加密算法是固定的，得到的密文永远不变
        3. 相对来说就不安全，容易被破解
    3. 公司中使用MD5进行加密

1. 例如：原文密码是123，把首尾交换，再加上用户名进行加密
2. 例如：321meimei 进行加密e5212215b8cbf493e04f4290cf7cecc8
3. 这样做相对来说会安全

2. BCryptPasswordEncoder加密（加盐加密算法）
    1. shiro框架有md5hash算法加密，该加密是加盐的方式进行加密的。
    2. 在原有的密码中随机加入盐（就是字符串），再进行加密。

```
Md5Hash h = new Md5Hash("123","ha",2);
System.out.println(h.toString());
```

    3. 测试代码

```
    // $2a$10$.W5msBQws5yoJum7gEbMcOQ1J.p2UIjEH5l4lm1rLCers9QUhUVFS
    // $2a$10$UxZneVG78UkChbx4sZND7.mWWx6ulbnl38updtSMRK0KQw5QvEdUy
    public static void main(String[] args) {
        // System.out.println(md5("321meimei"));

        BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
        // String str = encoder.encode("123");
        // System.out.println(str);

        // 进行判断
        boolean b = encoder.matches("123",
"$2a$10$UxZneVG78UkChbx4sZND7.mWWx6ulbnl38updtSMRK0KQw5QvEdUy");
        System.out.println(b);
    }
```

3. 添加用户的代码
    1. 在security配置文件中配置加密类

```
<!-- 配置加密类 -->
    <bean id="passwordEncoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"></bean>
```

    2. 编写添加用户的方法

```
    @RequestMapping("/save")
    public String save(SysUser sysUser) {
        userService.save(sysUser);
        return "redirect:/user/findAll";
    }

    /**
     * 对密码加密保存
     */

    public void save(SysUser sysUser) {
```

```
        // 先对密码进行加密
        String newpwd = passwordEncoder.encode(sysUser.getPassword());
        sysUser.setPassword(newpwd);
        // 保存用户
        userDao.save(sysUser);
    }

    @Insert("insert into sys_user (username,email,password,phoneNum,status) values (#
{username},#{email},#{password},#{phoneNum},#{status})")
    void save(SysUser sysUser);
```

第五节：修改登录功能

1. 修改配置文件

```
    <!-- 在内存中临时提供用户名和密码的数据 -->
    <security:authentication-manager>
        <!-- 提供服务类，去数据库查询用户名和密码 -->
        <security:authentication-provider user-service-ref="userService">
            <!-- 提供加密方式 -->
            <security:password-encoder ref="passwordEncoder"/>
        </security:authentication-provider>
    </security:authentication-manager>
```

2. 修改登录的方法，把{noop}代码去掉，表示采用加密方式登录

```
    /**
     * 该方法是认证的方法
     * 先编写一个默认的认证代码
     * 参数就是表单提交的用户名
     */
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
{
        // 先设置假的权限
        List<GrantedAuthority> authorities = new ArrayList<>();
        // 传入角色
        authorities.add(new SimpleGrantedAuthority("ROLE_USER"));

        // 通过用户名查询密码
        SysUser sysUser = userDao.findByUsername(username);

        // 创建用户
        User user = new User(username, sysUser.getPassword(), authorities) ;
        return user;
    }
```

第六节：查看用户详情的功能

1. 手动向角色表和权限表添加数据，手动添加外键关系

2. 具体的代码如下

UserController的代码

```
/**
 * 查询用户的详细信息
 * @return
 */
@RequestMapping("/findOne")
public ModelAndView findOne(Long id) {
    SysUser user = userService.findOne(id);
    ModelAndView mv = new ModelAndView();
    mv.addObject("user", user);
    mv.setViewName("user-show");
    return mv;
}
```

UserServiceImpl的代码

```
/**
 * 查询一个用户
 */
public SysUser findOne(Long id) {
    return userDao.findOne(id);
}
```

UserDao接口的代码

```
/**
 * 查询详细信息
 * 一对多延迟加载
 * @param id
 * @return
 */
@Select("select * from sys_user where id = #{id}")
@Results({

@Result(property="roles",column="id",javaType=List.class,many=@Many(select="cn.itcast.dao.RoleDao.findByUserId",fetchType=FetchType.LAZY))
})
SysUser findOne(Long id);
```

RoleDao接口的代码

```
// 通过用户id查询该用户所拥有的角色
@Select("SELECT sr.* FROM sys_role sr,sys_user_role sur WHERE sr.id = sur.roleId AND sur.userId = #{id}")
@Results({

@Result(property="permissions",column="id",javaType=List.class,many=@Many(select="cn.itcast.dao.PermissionDao.findByRoleId",fetchType=FetchType.LAZY))
})
public List<Role> findByUserId(Long id);
```

PermissionDao接口的代码

```
@Select("SELECT sp.* FROM sys_permission sp,sys_role_permission srp WHERE sp.id =
```

```
srp.permissionId AND srp.roleId = #{id}")
    public List<Permission> findByRoleId(Long id);
```

第七节：给用户设置角色

1. 具体的代码如下

```
    UserServiceImpl的代码
    /**
     * 该方法是认证的方法
     * 先编写一个默认的认证代码
     * 参数就是表单提交的用户名
     */
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
{
        // 先设置假的权限
        List<GrantedAuthority> authorities = new ArrayList<>();

        // 通过用户名查询用户
        // 通过用户名查询密码
        SysUser sysUser = userDao.findByUsername(username);
        if(sysUser != null) {
            // 获取到角色的集合
            List<Role> roles = sysUser.getRoles();
            // 遍历
            for (Role role : roles) {
                // 传入角色
                authorities.add(new SimpleGrantedAuthority("ROLE_"+role.getRoleName()));
            }
            // 创建用户
            User user = new User(username, sysUser.getPassword(), authorities) ;
            return user;
        }
        return null;
    }

    UserDao接口的代码
    @Select("select * from sys_user where username = #{username}")
    @Results({
@Result(property="roles",column="id",javaType=List.class,many=@Many(select="cn.itcast.dao.R
oleDao.findByUserId",fetchType=FetchType.LAZY))
    })
    SysUser findByUsername(String username);
```

2. access属性的配置

    1. access的值是一个字符串，其可以直接是一个权限的定义，也可以是一个表达式。常用的类型有简单的角色名称定义，多个名称之间用逗号分隔。

```
<security:intercept-url pattern="/**" access="ROLE_USER,ROLE_ADMIN"/>
```

1. 在上述配置中就表示secure路径下的所有URL请求都应当具有ROLE_USER或ROLE_ADMIN权限。当access的值是以“ROLE_”开头的则将会交由RoleVoter进行处理。

2. 此外，其还可以是一个表达式，use-expressions="true"

```
<security:intercept-url pattern="/secure/**"
access="hasAnyRole('ROLE_USER','ROLE_ADMIN')"/>
```

3. 或者是使用hasRole()表达式，然后中间以or连接

```
<security:intercept-url pattern="/secure/**" access="hasRole('ROLE_USER') or
hasRole('ROLE_ADMIN')"/>
```

4. 其他的值 IS_AUTHENTICATED_ANONYMOUSLY表示用户不需要登录就可以访问；

   IS_AUTHENTICATED_REMEMBERED表示用户需要是通过Remember-Me功能进行自动登录的才能访问；IS_AUTHENTICATED_FULLY表示用户的认证类型应该是除前两者以外的，也就是用户需要是通过登录入口进行登录认证的才能访问

第二章：角色模块

第一节：查询所有数据和添加角色

1. 代码如下

```
@Controller
@RequestMapping("/role")
public class RoleController {

    @Autowired
    private RoleService roleService;

    @RequestMapping("/save")
    public String save(Role role) {
        roleService.save(role);
        return "redirect:/role/findAll";
    }

    @RequestMapping("/findAll")
    public String save(Model model) {
        List<Role> list = roleService.findAll();
        model.addAttribute("roleList", list);
        return "role-list";
    }

}

@Service
```

```
public class RoleServiceImpl implements RoleService {

    @Autowired
    private RoleDao roleDao;

    public List<Role> findAll() {
        return roleDao.findAll();
    }

    public void save(Role role) {
        roleDao.save(role);
    }

}

@Repository
public interface RoleDao {

    @Select("select * from sys_role")
    List<Role> findAll();

    @Insert("insert into sys_role (roleName, roleDesc) values (#{roleName}, #{roleDesc})")
    void save(Role role);

}
```

第三章：权限模块

第一节：查询所有权限和添加权限

1. 具体的代码如下

```
@Controller
@RequestMapping("/permission")
public class PermissionController {

    @Autowired
    private PermissionService permissionService;

    @RequestMapping("/findAll")
    public String findAll(Model model) {
        List<Permission> list = permissionService.findAll();
        model.addAttribute("list", list);
        return "permission-list";
    }

    @RequestMapping("/save")
    public String findAll(Permission peimission) {
        permissionService.save(peimission);
        return "redirect:findAll";
    }
}
```

```java
@Service
public class PermissionServiceImpl implements PermissionService {

    @Autowired
    private PermissionDao permissionDao;

    public List<Permission> findAll() {
        return permissionDao.findAll();
    }

    public void save(Permission peimission) {
        permissionDao.save(peimission);
    }

}

@Repository
public interface PermissionDao {

    @Insert("insert into sys_permission (permissionName,url,pid) values (#{permissionName},
#{url}, #{pid})")
    public void save(Permission peimission);

    @Select("select * from sys_permission")
    public List<Permission> findAll();

}
```