

SSM权限操作

1.数据库与表结构

1.1 用户表

1.1.1 用户表信息描述users

序号	字段名称	字段类型	字段描述
1	id	varchar2	无意义，主键uuid
2	email	varchar2	非空，唯一
3	username	varchar2	用户名
5	password	varchar2	密码（加密）
6	phoneNum	varchar2	电话
7	status	int	状态0 未开启 1 开启

1.1.2 sql语句

```
CREATE TABLE users(  
    id varchar2(32) default SYS_GUID() PRIMARY KEY,  
    email VARCHAR2(50) UNIQUE NOT NULL,  
    username VARCHAR2(50),  
    PASSWORD VARCHAR2(50),  
    phoneNum VARCHAR2(20),  
    STATUS INT  
)
```

1.1.3 实体类

```
public class UserInfo {  
  
    private String id;  
    private String username;  
    private String email;  
    private String password;  
    private String phoneNum;  
    private int status;  
    private String statusStr;  
    private List<Role> roles;  
  
}
```

1.2 角色表

1.2.1 角色表信息描述role

序号	字段名称	字段类型	字段描述
1	id	varchar2	无意义，主键uuid
2	roleName	varchar2	角色名
3	roleDesc	varchar2	角色描述

1.2.2 sql语句

```
CREATE TABLE role(  
    id varchar2(32) default SYS_GUID() PRIMARY KEY,  
    roleName VARCHAR2(50) ,  
    roleDesc VARCHAR2(50)  
)
```

1.2.3 实体类

```
public class Role {  
  
    private String id;  
    private String roleName;  
    private String roleDesc;  
    private List<Permission> permissions;  
    private List<User> users;  
  
}
```

1.2.4 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 users_role(  
    userId varchar2(32),  
    roleId varchar2(32),  
    PRIMARY KEY(userId,roleId),  
    FOREIGN KEY (userId) REFERENCES users(id),  
    FOREIGN KEY (roleId) REFERENCES role(id)  
)
```

1.3 资源权限表

1.3.1 权限资源表描述permission

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

1.3.2 sql语句

```
CREATE TABLE permission(  
    id varchar2(32) default SYS_GUID() PRIMARY KEY,  
    permissionName VARCHAR2(50) ,  
    url VARCHAR2(50)  
)
```

1.3.3 实体类

```
public class Permission {  
  
    private String id;  
    private String permissionName;  
    private String url;  
    private List<Role> roles;  
}
```

1.3.4.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 role_permission(  
    permissionId varchar2(32),  
    roleId varchar2(32),  
    PRIMARY KEY(permissionId,roleId),  
    FOREIGN KEY (permissionId) REFERENCES permission(id),  
    FOREIGN KEY (roleId) REFERENCES role(id)  
)
```

2.Spring Security概述

2.1 Spring Security介绍

Spring Security 的前身是 Acegi Security，是 Spring 项目组中用来提供安全认证服务的框架。

(<https://projects.spring.io/spring-security/>) Spring Security 为基于J2EE企业应用软件提供了全面安全服务。特别是使用领先的J2EE解决方案-Spring框架开发的企业软件项目。人们使用Spring Security有很多原因，不过通常吸引他们的是在J2EE Servlet规范或EJB规范中找不到典型企业应用场景的解决方案。特别要指出的是他们不能再 WAR 或 EAR 级别进行移植。这样，如果你更换服务器环境，就要，在新的目标环境进行大量的工作，对你的应用系统进行重新配置安全。使用Spring Security 解决了这些问题，也为你提供很多有用的，完全可以指定的其他安全特性。安全包括两个主要操作。

- “认证”，是为用户建立一个他所声明的主体。主题一般式指用户，设备或可以在你系统中执行动作的其他系统。
- “授权”指的是一个用户能否在你的应用中执行某个操作，在到达授权判断之前，身份的主题已经由身份验证过程建立了。

这些概念是通用的，不是Spring Security特有的。在身份验证层面，Spring Security广泛支持各种身份验证模式，这些验证模型绝大多数都由第三方提供，或则正在开发的有关标准机构提供的，例如 Internet Engineering Task Force.作为补充，Spring Security 也提供了自己的一套验证功能。

Spring Security 目前支持认证一体化如下认证技术： HTTP BASIC authentication headers (一个基于IETF RFC 的标准) HTTP Digest authentication headers (一个基于IETF RFC 的标准) HTTP X.509 client certificate exchange (一个基于IETF RFC 的标准) LDAP (一个非常常见的跨平台认证需要做法，特别是在大环境) Form-based authentication (提供简单用户接口的需求) OpenID authentication Computer Associates Siteminder JA-SIG Central Authentication Service (CAS，这是一个流行的开源单点登录系统) Transparent authentication context propagation for Remote Method Invocation and HttpInvoker (一个Spring远程调用协议)

Maven依赖

```
<dependencies>
  <dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>5.0.1.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>5.0.1.RELEASE</version>
  </dependency>
</dependencies>
```

2.2 Spring Security快速入门

2.2.1 pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
```



```

instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>spring_security_demo</groupId>
  <artifactId>SpringSecurity_quickStart</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <properties>
    <spring.version>5.0.2.RELEASE</spring.version>
    <spring.security.version>5.0.1.RELEASE</spring.security.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context-support</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-web</artifactId>
      <version>${spring.security.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-config</artifactId>
      <version>${spring.security.version}</version>
    </dependency>
  </dependencies>

```



```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
</dependency>
</dependencies>
<build>
  <plugins>
    <!-- java编译插件 -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.2</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.tomcat.maven</groupId>
      <artifactId>tomcat7-maven-plugin</artifactId>
      <configuration>
        <!-- 指定端口 -->
        <port>8080</port>
        <!-- 请求路径 -->
        <path>/</path>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

2.2.2 web.xml

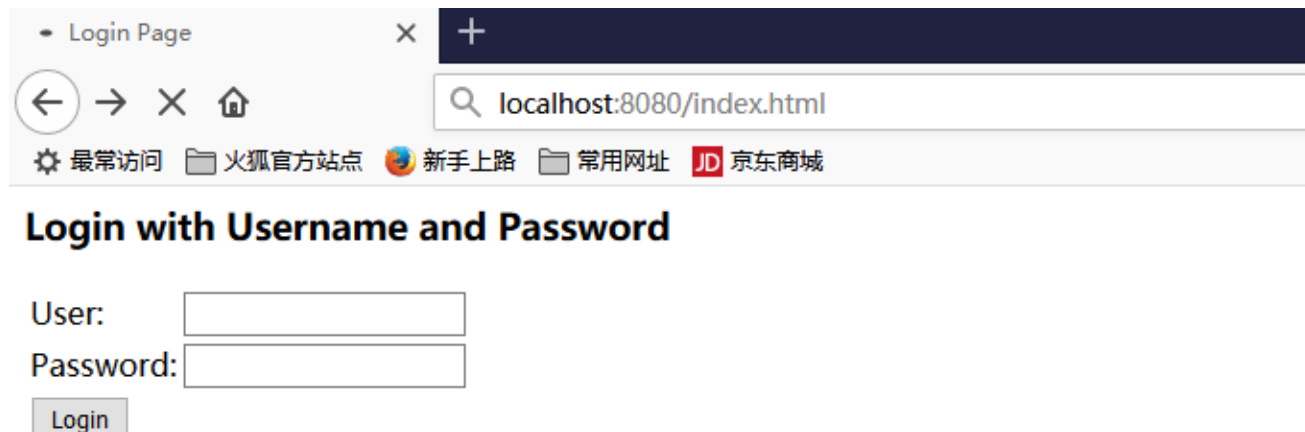
```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath:spring-security.xml</param-value>
</context-param>
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

2.2.3 spring security配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:security="http://www.springframework.org/schema/security"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd">
  <security:http auto-config="true" use-expressions="false">
    <!-- intercept-url定义一个过滤规则 pattern表示对哪些url进行权限控制，ccess属性表示在请求对应的URL时需要什么权限，
        默认配置时它应该是一个以逗号分隔的角色列表，请求的用户只需拥有其中的一个角色就能成功访问对应的URL -->
    <security:intercept-url pattern="/**" access="ROLE_USER" />
    <!-- auto-config配置后，不需要在配置下面信息 <security:form-login /> 定义登录表单信息
  <security:http-basic
    /> <security:logout /> -->
  </security:http>
  <security:authentication-manager>
    <security:authentication-provider>
      <security:user-service>
        <security:user name="user" password="{noop}user"
          authorities="ROLE_USER" />
        <security:user name="admin" password="{noop}admin"
          authorities="ROLE_ADMIN" />
      </security:user-service>
    </security:authentication-provider>
  </security:authentication-manager>
</beans>
```

2.2.4 测试

我们在webapp下创建一个index.html页面，在页面中任意写些内容。



Login Page

localhost:8080/index.html

最常访问 火狐官方网站 新手上路 常用网址 JD 京东商城

Login with Username and Password

User:

Password:

Login



当我们访问index.html页面时发现会弹出登录窗口，可能你会奇怪，我们没有建立下面的登录页面，为什么Spring Security会跳到上面的登录页面呢？这是我们设置http的auto-config="true"时Spring Security自动为我们生成的。

2.2.5 使用自定义页面

2.2.5.1 spring-security.xml配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:security="http://www.springframework.org/schema/security"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security.xsd">

    <!-- 配置不过滤的资源（静态资源及登录相关） -->
    <security:http security="none" pattern="/login.html" />
    <security:http security="none" pattern="/failer.html" />

    <security:http auto-config="true" use-expressions="false">
        <!-- 配置资源连接，表示任意路径都需要ROLE_USER权限 -->
        <security:intercept-url pattern="/**" access="ROLE_USER" />
        <!-- 自定义登陆页面，login-page 自定义登陆页面 authentication-failure-url 用户权限校验失败之后才会跳转到这个页面，如果数据库中没有一个用户则不会跳转到这个页面。
            default-target-url 登陆成功后跳转的页面。 注：登陆页面用户名固定 username，密码 password，action:login -->
        <security:form-login login-page="/login.html"
            login-processing-url="/login" username-parameter="username"
            password-parameter="password" authentication-failure-url="/failer.html"
            default-target-url="/success.html"
        />
        <!-- 登出， invalidate-session 是否删除session logout-url: 登出处理链接 logout-success-url: 登出成功页面
            注：登出操作 只需要链接到 logout即可登出当前用户 -->
        <security:logout invalidate-session="true" logout-url="/logout"
            logout-success-url="/login.jsp" />
        <!-- 关闭CSRF,默认是开启的 -->
        <security:csrf disabled="true" />
    </security:http>

    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service>
                <security:user name="user" password="{noop}user"
                    authorities="ROLE_USER" />
                <security:user name="admin" password="{noop}admin"
                    authorities="ROLE_ADMIN" />
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>
</beans>
```




2.2.5.2 login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="login" method="post">
  <table>
    <tr>
      <td>用户名: </td>
      <td><input type="text" name="username" /></td>
    </tr>
    <tr>
      <td>密码: </td>
      <td><input type="password" name="password" /></td>
    </tr>
    <tr>
      <td colspan="2" align="center"><input type="submit" value="登录" />
        <input type="reset" value="重置" /></td>
    </tr>
  </table>
</form>
</body>
</html>
```

2.2.5.3 success.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  success html<br>
  <a href="logout">退出</a>
</body>
</html>
```

2.2.5.4 failer.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>登录失败
</body>
</html>
```

2.3 Spring Security使用数据库认证

在Spring Security中如果想要使用数据库进行认证操作，有很多种操作方式，这里我们介绍使用UserDetails、UserDetailsService来完成操作。

- UserDetails

```
public interface UserDetails extends Serializable {
    Collection<? extends GrantedAuthority> getAuthorities();
    String getPassword();
    String getUsername();
    boolean isAccountNonExpired();
    boolean isAccountNonLocked();
    boolean isCredentialsNonExpired();
    boolean isEnabled();
}
```

UserDetails是一个接口，我们可以认为UserDetails作用是封装当前进行认证的用户信息，但由于其是一个接口，所以我们可以对其进行实现，也可以使用Spring Security提供的一个UserDetails的实现类User来完成操作

以下是User类的部分代码

```
public class User implements UserDetails, CredentialsContainer {

    private String password;
    private final String username;
    private final Set<GrantedAuthority> authorities;
    private final boolean accountNonExpired; //帐户是否过期
    private final boolean accountNonLocked; //帐户是否锁定
    private final boolean credentialsNonExpired; //认证是否过期
    private final boolean enabled; //帐户是否可用

}
```

- UserDetailsService

```
public interface UserDetailsService {  
  
    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;  
  
}
```

上面将UserDetails与UserDetailsService做了一个简单的介绍，那么我们具体如何完成Spring Security的数据库认证操作哪，我们通过用户管理中用户登录来完成Spring Security的认证操作。

3.用户管理

3.1 用户登录

spring security的配置

```
<security:authentication-manager>  
    <security:authentication-provider user-service-ref="userService">  
        <!-- 配置加密的方式 -->  
        <security:password-encoder ref="passwordEncoder"/>  
    </security:authentication-provider>  
</security:authentication-manager>
```

3.1.1.登录页面login.jsp

详细页面请查看资料

3.1.2.Service

```
public interface IUserService extends UserDetailsService{  
  
}
```

```
@Service("userService")  
@Transactional  
public class UserServiceImpl implements IUserService {  
  
    @Autowired  
    private IUserDao userDao;  
  
    @Override  
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
        UserInfo userInfo = userDao.findByUsername(username);  
        List<Role> roles = userInfo.getRoles();  
  
        List<SimpleGrantedAuthority> authoritys = getAuthority(roles);  
    }  
}
```



```
User user = new User(userInfo.getUsername(), "{noop}" + userInfo.getPassword(),
    userInfo.getStatus() == 0 ? false : true, true, true, true, authorities);

return user;
}

private List<SimpleGrantedAuthority> getAuthority(List<Role> roles) {
    List<SimpleGrantedAuthority> authoritys = new ArrayList();
    for (Role role : roles) {
        authoritys.add(new SimpleGrantedAuthority(role.getRoleName()));
    }
    return authoritys;
}
}
```

3.1.3.IUserDao

```
public interface IUserDao {

    @Select("select * from user where id=#{id}")
    public UserInfo findById(Long id) throws Exception;

    @Select("select * from user where username=#{username}")
    @Results({
        @Result(id = true, property = "id", column = "id"),
        @Result(column = "username", property = "username"),
        @Result(column = "email", property = "email"),
        @Result(column = "password", property = "password"),
        @Result(column = "phoneNum", property = "phoneNum"),
        @Result(column = "status", property = "status"),
        @Result(column = "id", property = "roles", javaType = List.class, many =
            @Many(select = "com.ithema.ssm.dao.IRoleDao.findRoleByUserId")) })
    public UserInfo findByUsername(String username);
}
```

3.2 用户退出

使用spring security完成用户退出，非常简单

- 配置

```
<security:logout invalidate-session="true" logout-url="/logout.do" logout-success-
url="/login.jsp" />
```

- 页面中

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

3.3 用户查询

3.3.1.用户查询页面 user-list.jsp

请在资料中查看具体代码

3.3.2.UserController

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

    @RequestMapping("/findAll.do")
    public ModelAndView findAll() throws Exception {
        List<UserInfo> users = userService.findAll();
        ModelAndView mv = new ModelAndView();
        mv.addObject("userlist", users);
        mv.setViewName("user-list");
        return mv;
    }
}
```

3.3.3.Dao

```
@Select("select * from user")
public List<UserInfo> findAll();
```

3.4 用户添加

3.4.1.用户添加页面 user-add.jsp

请在资料中查看具体页面代码

3.4.2.UserController



```
@Controller
@RequestMapping("/user")
public class UserController {
    @Autowired
    private IUserService userService;

    @RequestMapping("/save.do")
    public String save(UserInfo user) throws Exception {
        userService.save(user);
        return "redirect:findAll.do";
    }
}
```

3.4.3.Service

```
@Service("userService")
@Transactional
public class UserServiceImpl implements IUserService {

    @Autowired
    private IUserDao userDao;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public void save(UserInfo user) throws Exception {
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userDao.save(user);
    }
}
```

前期我们的用户密码没有加密，现在添加用户时，我们需要对用户密码进行加密

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

3.4.4.Dao

```
@Insert("insert into user(email,username,password,phoneNum,status) value(#{email},#{username},#{password},#{phoneNum},#{status})")
public void save(UserInfo user) throws Exception;
```

3.5 用户详情

3.5.1.用户详情页面user-show.jsp

请在资料中查看页面详细代码

注意：需要添加js

```
$("#collapse-table").treetable({ expandable : true    });
```

3.5.2.UserController

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

    @Autowired
    private IUserService userService;
    @RequestMapping("/findById.do")
    public ModelAndView findById(@RequestParam(name = "id", required = true) Long id) throws
Exception {
        UserInfo user = userService.findById(id);
        ModelAndView mv = new ModelAndView();
        mv.addObject("user", user);
        mv.setViewName("user-show");
        return mv;
    }
}
```

3.5.3.Dao

```
@Select("select * from user where id=#{id}")
@Results({ @Result(id = true, property = "id", column = "id"), @Result(column = "username",
property = "username"),
            @Result(column = "email", property = "email"), @Result(column =
"password", property = "password"),
            @Result(column = "phoneNum", property = "phoneNum"), @Result(column =
"status", property = "status"),
            @Result(column = "id", property = "roles", javaType = List.class, many =
@Many(select = "com.itheima.ssm.dao.IRoleDao.findRoleByUserId")) })
public UserInfo findById(Long id) throws Exception;
```

```
@Select("select * from role where id in( select roleId from user_role where userId=#{userId})")
@Results(
    {
        @Result(id=true,column="id",property="id"),
        @Result(column="roleName",property="roleName"),
        @Result(column="roleDesc",property="roleDesc"),

        @Result(column="id",property="permissions",javaType=List.class,many=@Many(select="com.itheima.ssm
        .dao.IPermissionDao.findById"))
    })
public List<Role> findRoleByUserId(Long userId);
```

我们需要将用户的所有角色及权限查询出来所以需要调用IRoleDao中的findRoleByUserId,而在IRoleDao中需要调用IPermissionDao的findById

```
@Select("select * from permission where id in (select permissionId from role_permission where
roleId=#{roleId})")
public List<Permission> findById(Long roleId);
```

4.角色管理

4.1 角色查询

4.1.1.角色页面role-list.jsp

请在资料中查看页面详细代码

4.1.2.RoleControlller

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

    @Autowired
    private IRoleService roleService;

    @RequestMapping("/findAll.do")
    public ModelAndView findAll() throws Exception {
        List<Role> roleList = roleService.findAll();
        ModelAndView mv = new ModelAndView();
        mv.addObject("roleList", roleList);
        mv.setViewName("role-list");
        return mv;
    }
}
```

4.1.3.Dao



```
@Select("select * from role")
public List<Role> findAll();
```

4.2 角色添加

4.2.1.角色添加页面role-add.jsp

请在页面中查看详细代码

4.2.2.RoleControlller

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

    @Autowired
    private IRoleService roleService;

    @RequestMapping("/save.do")
    public String save(Role role) {
        roleService.save(role);

        return "redirect:findAll.do";
    }
}
```

4.2.3.Dao

```
@Insert("insert into role(roleName,roleDesc) value(#{roleName},#{roleDesc})")
public void save(Role role);
```

5.资源权限管理

5.1 资源权限查询

5.1.1.权限资源页面permission-list.jsp

请在资料中查看详细代码

5.1.2.PermissionController

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

```
@Autowired
private IPermissionService permissionService;

@RequestMapping("/findAll.do")
public ModelAndView findAll() throws Exception {
    List<Permission> permissionList = permissionService.findAll();
    ModelAndView mv = new ModelAndView();
    mv.addObject("permissionList", permissionList);
    mv.setViewName("permission-list");
    return mv;
}
}
```

5.1.3.Dao

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

5.2 资源权限添加

5.2.1.权限资源添加页面permission-add.jsp

请在资料中查看页面详细代码

5.2.2.PermissionController

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

    @Autowired
    private IPermissionService permissionService;

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

5.2.3.Dao

```
@Insert("insert into permission(permissionName,url) value("#{permissionName},#{url})")
public void save(Permission p);
```

6. 权限关联与控制

6.1 用户角色关联

用户与角色之间是多对多关系，我们要建立它们之间的关系，只需要在中间表user_role插入数据即可。

6.1.1. 用户角色关联相关页面

- 在user-list.jsp页面上添加链接

```
<a href="${pageContext.request.contextPath}/user/findUserByIdAndAllRole.do?id=${user.id}"
class="btn bg-olive btn-xs">添加角色</a>
```

- 展示可以添加角色的页面user-roe-add.jsp

请在资料中查看页面详细代码

6.1.2. UserController

- findUserByIdAndAllRole(Long id)方法

此方法用于查找要操作的用户及可以添加的角色，参数是要操作的用户id

```
@RequestMapping("/findUserByIdAndAllRole.do")
public ModelAndView findUserByIdAndAllRole(Long id) throws Exception {
    UserInfo user = userService.findById(id);
    List<Role> roleList = roleService.findOtherRole(id);
    ModelAndView mv = new ModelAndView();
    mv.addObject("user", user);
    mv.addObject("roleList", roleList);
    mv.setViewName("user-role-add");
    return mv;
}
```

调用IUserService的findById方法获取要操作的User

调用IRoleService的findOtherRole方法用于获取可以添加的角色信息

- addRoleToUser(Long userId, Long[] ids)方法

些方法用于在用户与角色之间建立关系，参数userId代表要操作的用户id,参数ids代表的是角色id数组

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

6.1.3. Dao

- IRoleDao



```
@Select("select * from role where id not in( select roleId from user_role where userId=#{id})")  
public List<Role> findOtherRole(Long id);
```

用于查找可以添加的角色

- IUserDao

```
@Insert("insert into user_role(userId,roleId) value(#{userId},#{roleId})")  
public void addRoleToUser(@Param("userId") Long userId, @Param("roleId") Long roleId);
```

用于添加用户与角色关系

6.2 角色权限关联

角色与权限之间是多对多关系，我们要建立它们之间的关系，只需要在中间表role_permission插入数据即可。

6.2.1. 角色权限关联相关页面

- 在role-list.jsp页面上添加链接

```
<a href="${pageContext.request.contextPath}/role/findRoleByIdAndAllPermission.do?id=${role.id}" class="btn bg-olive btn-xs">添加权限</a>
```

- 展示可以添加权限的页面role-permission-add.jsp

请在资料中查看页面详细代码

6.2.2.RoleController

- findRoleByIdAndAllPermission(Long roleId)方法

此方法用于查找要操作的角色及可以添加的权限，参数是要操作的角色id

```
@RequestMapping("/findRoleByIdAndAllPermission.do")  
public ModelAndView findRoleByIdAndAllPermission(@RequestParam(name = "id", required = true) Long roleId)  
    throws Exception {  
    ModelAndView mv = new ModelAndView();  
    Role role = roleService.findById(roleid);  
    mv.addObject("role", role);  
    List<Permission> permissionList =  
    permissionService.findOtherPermission(roleid);  
    mv.addObject("permissionList", permissionList);  
    mv.setViewName("role-permission-add");  
    return mv;  
}
```

调用IRoleService的findById方法获取要操作的Role

调用IPermissionService的findOtherPermission方法用于获取可以添加的权限信息

- addPermissionToRole(Long roleId,Long[] ids)方法

些方法用于在角色与权限之间建立关系，参数roleId代表要操作的角色id,参数permissionIds代表的是权限id数组

```
@RequestMapping("/addPermissionToRole.do")
public String addPermissionToRole(@RequestParam(name = "roleId") Long roleId,
    @RequestParam(name = "ids") Long[] permissionIds) throws Exception {
    roleService.addPermissionToRole(roleId, permissionIds);
    return "redirect:findAll.do";
}
```

6.2.3.Dao

- IPermissionDao

```
@Select("select * from permission where id not in (select permissionId from role_permission
where roleId=#{roleId})")
public List<Permission> findOtherPermission(Long roleId);
```

用于查找可以添加的权限

- IRoleDao

```
@Insert("insert into role_permission (roleId,permissionId) value (#{roleId},#{
permissionId})")
public void addPermissionToRole(@Param("roleId") Long roleId, @Param("permissionId") Long
permissionId);
```

用于绑定角色与权限的关系

6.3 服务器端方法级权限控制

在服务器端我们可以通过Spring security提供的注解对方法来进行权限控制。Spring Security在方法的权限控制上支持三种类型的注解，JSR-250注解、@Secured注解和支持表达式的注解，这三种注解默认都是没有启用的，需要单独通过global-method-security元素的对应属性进行启用

6.3.1.开启注解使用

- 配置文件

```
<security:global-method-security jsr250-annotations="enabled"/>
<security:global-method-security secured-annotations="enabled"/>
<security:global-method-security pre-post-annotations="disabled"/>
```

- 注解开启

@EnableGlobalMethodSecurity：Spring Security默认是禁用注解的，要想开启注解，需要在继承WebSecurityConfigurerAdapter的类上加@EnableGlobalMethodSecurity注解，并在该类中将AuthenticationManager定义为Bean。

6.3.2.JSR-250注解

- @RolesAllowed表示访问对应方法时所应该具有的角色

示例:

```
@RolesAllowed({"USER", "ADMIN"}) 该方法只要具有"USER", "ADMIN"任意一种权限就可以访问。这里可以省略前缀ROLE_, 实际的权限可能是ROLE_ADMIN
```

- @PermitAll表示允许所有的角色进行访问, 也就是说不进行权限控制
- @DenyAll是和PermitAll相反的, 表示无论什么角色都不能访问

6.3.3.支持表达式的注解

- @PreAuthorize 在方法调用之前,基于表达式的计算结果来限制对方法的访问

示例:

```
@PreAuthorize("#userId == authentication.principal.userId or hasAuthority('ADMIN')")
void changePassword(@P("userId") long userId ){ }
```

这里表示在changePassword方法执行之前, 判断方法参数userId的值是否等于principal中保存的当前用户的userId, 或者当前用户是否具有ROLE_ADMIN权限, 两种符合其一, 就可以访问该方法。

- @PostAuthorize 允许方法调用,但是如果表达式计算结果为false,将抛出一个安全性异常

示例:

```
@PostAuthorize
User getUser("returnObject.userId == authentication.principal.userId or
hasPermission(returnObject, 'ADMIN')");
```

- @PostFilter 允许方法调用,但必须按照表达式来过滤方法的结果
- @PreFilter 允许方法调用,但必须在进入方法之前过滤输入值

6.3.4.@Secured注解

- @Secured注解标注的方法进行权限控制的支持, 其值默认为disabled。

示例:

```
@Secured("IS_AUTHENTICATED_ANONYMOUSLY")
public Account readAccount(Long id);
@Secured("ROLE_TELLER")
```

6.4 页面端标签控制权限

在jsp页面中我们可以使用spring security提供的权限标签来进行权限控制

6.4.1.导入

- maven导入

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
  <version>version</version>
</dependency>
```

- 页面导入

```
<%@taglib uri="http://www.springframework.org/security/tags" prefix="security"%>
```

6.4.2.常用标签

在jsp中我们可以使用以下三种标签，其中authentication代表的是当前认证对象，可以获取当前认证对象信息，例如用户名。其它两个标签我们可以用于权限控制

6.4.2.1 authentication

```
<security:authentication property="" htmlEscape="" scope="" var=""/>
```

- property: 只允许指定Authentication所拥有的属性，可以进行属性的级联获取，如“principal.username”，不允许直接通过方法进行调用
- htmlEscape: 表示是否需要将html进行转义。默认为true。
- scope: 与var属性一起使用，用于指定存放获取的结果的属性名的作用范围，默认我pageContext。Jsp中拥有的作用范围都进行指定
- var: 用于指定一个属性名，这样当获取到了authentication的相关信息后会将其以var指定的属性名进行存放，默认是存放在pageContext中

6.4.2.2 authorize

authorize是用来判断普通权限的，通过判断用户是否具有对应的权限而控制其所包含内容的显示

```
<security:authorize access="" method="" url="" var=""></security:authorize>
```

- access: 需要使用表达式来判断权限，当表达式的返回结果为true时表示拥有对应的权限
- method: method属性是配合url属性一起使用的，表示用户应当具有指定url指定method访问的权限，method的默认值为GET，可选值为http请求的7种方法
- url: url表示如果用户拥有访问指定url的权限即表示可以显示authorize标签包含的内容
- var: 用于指定将权限鉴定的结果存放在pageContext的哪个属性中

6.4.2.3 accesscontrollist

accesscontrollist标签是用于鉴定ACL权限的。其一共定义了三个属性：hasPermission、domainObject和var，其中前两个是必须指定的

```
<security:accesscontrollist hasPermission="" domainObject="" var=""></security:accesscontrollist>
```

- hasPermission: hasPermission属性用于指定以逗号分隔的权限列表
- domainObject: domainObject用于指定对应的域对象
- var: var则是用以将鉴定的结果以指定的属性名存入pageContext中，以供同一页面的其它地方使用