1. Authorization授权

本章主要介绍基于Shiro安全框架的授权机制，也就是控制访问机制。关于本章需要把握如下几个方面：

* 授权设计到的主体（对象）以及各自的意义
* Shiro安全框架中实现授权的方式
* Shiro安全框架内部实现授权的流程
* 自定义授权控制器

目录

[6 Authorization授权 1](#_Toc494196712)

[6.1 授权简介 2](#_Toc494196713)

[6.1.1 权限 2](#_Toc494196714)

[6.1.2 角色 3](#_Toc494196715)

[6.1.3 用户 3](#_Toc494196716)

[6.2 授权方式 3](#_Toc494196717)

[6.2.1 编码式授权 5](#_Toc494196718)

[6.2.2 注解式授权 7](#_Toc494196719)

[6.2.3 Jsp标签式授权 8](#_Toc494196720)

[6.3 授权 8](#_Toc494196721)

[6.3.1 ModularRealmAuthorizer 8](#_Toc494196722)

[6.3.2 授权相关对象 9](#_Toc494196723)

[6.4 使用授权 9](#_Toc494196724)

[6.4.1 基于IniRealm实现授权 10](#_Toc494196725)

[6.4.2 基于PropertiesRealm实现授权 10](#_Toc494196726)

[6.4.3 基于JdbcRealm实现授权 10](#_Toc494196727)

[6.4.4 多个Realm实现授权 11](#_Toc494196728)

[6.4.5 自定义Authorizer 12](#_Toc494196729)

[6.5 总结 16](#_Toc494196730)

前面我们从设计的角度了解了权限安全模块的基本设计模型RBAC模型，理解了权限安全模块的需求、设计到的基本对象和对象之间的关系，接下来我们介绍在Shiro安全框架中多授权的实现和使用。如下图6-1shiro体系图描述了Authorization在Shiro安全框架中的位置。

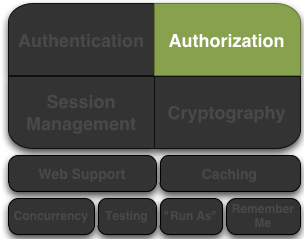


图6-1 Shiro体系

* 1. 授权简介

授权，也叫访问控制，即在应用中控制谁能访问哪些资源（如访问页面/编辑数据/页面操作等）。在授权中需了解的几个关键对象：主体（Subject）、资源（Resource）、权限（Permission）、角色（Role）。

比如验证某个用户是否能访问某个页面，修改某个数据，查看或点击某个按钮，甚至打印某些内容，这些都需要根据用户是否有对应的权限进行判断，然后才能够按判断结果执行。

实现授权功能，一般涉及到三个概念：权限、角色和用户。

* + 1. 权限

权限是Shiro安全框架中描述安全操作的最小单元（最小概念），它表示的是一个应用可以做的行为。比较准确的说法是权限是Subject可以操作的资源，它包含两层信息，一层是资源，一层是对资源的操作。比如打开文件、编辑文件、查看user/list页面、打印文档或删除用户等等。

一般来说，对资源的典型操作是CRUD(create、read、update和delete)，当然根据实际请求，还可以有其他操作。

需要注意的是，权限描述的是对什么资源的哪个操作，也就是操作和资源的对应关系，不包括谁可以执行哪些操作。

定义或描述谁可以对哪些资源做哪些操作，是权限分配的事情，应该是Shiro安全框架外，有各自应用自己实现的。

我们需要基于Realm组件，读取应用中权限配置相关的数据，以Shiro安全框架支持的格式提供给Authorizator进行认证授权。

Shiro安全框架中对权限的描述有两种：一种是Permission对象，比较常用的实现类是WildcardPermission；一种是以固定格式的字符串描述权限。

我们可以直接将固定格式的字符串权限，传给WildcardPermission类的构造器，就可以创建Permission对象，所以接下来介绍描述权限的字符串格式。

权限字符串的标准格式是 资源:操作:对象id ，整体意思是 怎么操作哪个资源的哪个对象，资源，操作和对象id之间用冒号(:)分开，同时还支持用统配符(\*)表示资源、操作和对象id的任何一个或多个。如下是权限字符串的常见几种形式和意义。

* 单个资源单个权限

user:update user:delete

* 单个资源多个权限

User:update,user:delete user:update,delete

* 单个资源全部权限

User:update,delete,add,query user:\*

* 所有资源单个权限

\*:view \*:add

* 单个对象单个权限

User:update:1 user:view:1

* 单个对象多个权限

User:update,view:1

* 所有对象单个权限

User:view:\*

* 所有对象全部权限

User:\*:\*

* + 1. 角色

角色是一个代号，它是一个包含多个权限的集合。在应用中，我们一般是把角色分配给用户，从而描述一个用户有哪些权限、可以怎么操作哪些资源。

一个系统中一般包含两种角色，显示角色和隐式角色。

* 显示角色是角色名称包含的权限，比如超级管理员、部门经理，根据名称我们就知道是拥有全系统的权限和拥有管理自己部门的权限，这些权限的控制逻辑，用if语句，固定写死在源代码里面。如果要修改权限，需要修改对应的代码。
* 隐式角色，开始只是一个名字，具体包含哪些权限，需要应用动态的赋予，也就是平常所说的权限分配。这种角色和权限的对应关系，是通过数据关系动态指定的，维护的时候，不需要修改代码，只要改变对应的持久数据就行。
  + 1. 用户

用户就是我们权限控制中（谁可以做什么事）的who(谁)， 它可以是某个人，也可以是某个第三方的应用，所以在Shiro安全框架中，我们称为Subject。不管代表人，还是第三方应用，用户在应用中都必须有自己的身份标识和只有它自己知道的，应用认可的，能证明自己身份的信息（一般是密码）

在实际应用中，可以将用户分组，每个组有不同的权限；也可以给用户分配不同角色，这样该用户就拥有相关角色对应的权限；当然，也可以将某些权限直接赋给用户。

* 1. 授权方式

在Shiro安全框架中，有三种方式可以实现授权

* 编码式

我们可以通过if else语句，判断是否有授权控制Subject的访问

* 注解式

在要执行的方法上面加上注解，Shiro框架根据对应的注解判断是否获得了授权，有授权才能执行该方法

* Jsp标签

通过jsp标签，控制页面的输出，从而控制用户的访问。

在本章，我们主要研究编码式的授权方法。注解式需要结合容器，jsp标签式的涉及到jsp页面，需要在web应用中才好使用，所以在后面的web集成的时候说明。为了方便后面的测试，我们先创建一个ShiroTP06 Maven工程，基于IniRealm，在ini配置文件中做好相关的配置。详情请查看随书代码[code/ShiroTP06/src/main/resources/shiro.ini](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\resources\shiro.ini)。

[main]

# -----------------------------------------------------------------------------

# 配置用户和每个用户的密码和角色信息

# username = password, role1, role2, ..., roleN

# -----------------------------------------------------------------------------

[users]

root = 123456, admin

guest = 123456, guest

zhangsan = 123456, role1,role2,role3

lisi = 123456, role1,role3

wangwu = 123456, role3

# -----------------------------------------------------------------------------

# 配置角色和对应的权限

# roleName = perm1, perm2, ..., permN

# -----------------------------------------------------------------------------

[roles]

admin = \*

guest = \*:view

role1 = user:add

role2 = user:update

role3 = user:delete

同时为了方便后面的测试，编写一个基于JUnit的测试案例类TestAuthorization，里面包含init方法和destory方法，分别实现Subject的登陆初始化和测试完后Subject登出。样例代码块如下，详细内容请查看随书代码[code/ShiroTP06/src/test/java/cn/com/shiro/book/tp06/TestAuthorization.java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\test\java\cn\com\shiro\book\tp06\TestAuthorization.java)。

@Before

public void init() {

// 1.创建IniSecurityManagerFactory，基于shiro.ini进行初始化

Factory<org.apache.shiro.mgt.SecurityManager> factory = new IniSecurityManagerFactory(

"classpath:shiro.ini");

// 2.获取SecurityManager对象

org.apache.shiro.mgt.SecurityManager securityManager = factory

.getInstance();

// 3.注册全局的SecurityManager对象

SecurityUtils.setSecurityManager(securityManager);

currUser = SecurityUtils.getSubject();

}

@After

public void destory(){

this.currUser.logout();

}

* + 1. 编码式授权

基于编码式判断Subject对象是否经过授权，从而控制操作，是一个用得比较普通而且容易理解的方法，一般分两种授权：一个是基于角色的编码授权；一个是基于权限的编码授权。

* 基于角色编码授权

在Shiro安全框架的Subject类里面，提供了boolean hasRole(String);boolean[] hasRoles(List<String>); boolean hasAllRoles(Collection<String>)方法，可以判断当前Subject对象是否是指定的角色，返回boolean数据表示判断结果。我们可以通过判断结果，结合if else 语句实现授权控制。

1. boolean hasRole(String roleName) 判断Subject是否有指定的角色，有返回true，否则返回false。
2. boolean[] hasRoles(List<String> roleNames) 判断Subject拥有指定角色名称集合的哪些角色，返回boolean数组。序号对应roleNames中的角色名称，如果有，结果为true，否则为false
3. boolean hasAllRoles(List<String> roleNames) 判断Subject是否拥有指定角色名称集合中的所有角色，如果所有角色都有，返回true，否则返回false

样例代码块如下，详细内容请查看随书代码[code/ShiroTP06/src/test/java/cn/  
com/shiro/book/tp06/TestAuthorization.java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\test\java\cn\com\shiro\book\tp06\TestAuthorization.java)代码中的testCheckRole方法。

@Test

public void testCheckRole(){

// 登录

UsernamePasswordToken token = new UsernamePasswordToken("zhangsan","123456");

this.currUser.login(token);

if(this.currUser.hasRole("admin")){

System.out.println(this.currUser.getPrincipal() + "是admin，拥有所有权限");

}else{

System.out.println(this.currUser.getPrincipal() + " 不是admin，可能拥有部分权限");

}

List<String> roleNames = new ArrayList<String>();

roleNames.add("role1");

roleNames.add("role2");

roleNames.add("admin");

boolean[] hasRoles = this.currUser.hasRoles(roleNames);

System.out.println(this.currUser.getPrincipal() + "拥有的角色请求:");

for(int i=0;i<roleNames.size();i++){

System.out.println(" "+roleNames.get(i) + ":" + hasRoles[i]);

}

System.out.println(this.currUser.hasAllRoles(roleNames));

}

我们可以通过编码的方式，通过调用Subject的hasRole\*方法判断是否有角色，根据结果控制授权动作。我们也可以在执行逻辑代码前，调用Subject的check\*方法什么当前Subject有角色，如果确实有，后面逻辑代码继续执行，否则就抛出Authorization异常。Subject中的角色声明方法如下：

1. void checkRole(String roleName)

如果实际情况同声明的一样，当前Subject拥有指定的角色，代码继续往后执行，否则抛出AuthorizationException异常。

1. void checkRoles(Collection<String> roleNames)

如果实际情况同声明的一样，当前Subject拥有集合中所有的角色，代码继续往后执行，否则抛出AuthorizationException

1. void checkRoles(String… roleNames)

作用同2）方法一样，不同的是改方法需要jdk5以上版本才能支持。

测试的样例代码块如下，详细请查看[code/ShiroTP06/src/test/java/cn/  
com/shiro/book/tp06/TestAuthorization.java](file:///C:\work\课件\shiro\shiro应用实战\code\ShiroTP06\src\test\java\cn\com\shiro\book\tp06\TestAuthorization.java)代码中的testAssertionRole方法。

@Test

public void testAssertionRole() {

// 登录

UsernamePasswordToken token = new UsernamePasswordToken("zhangsan",

"123456");

this.currUser.login(token);

this.currUser.checkRole("role1");

System.out.println(this.currUser.getPrincipal() + " 拥有role1角色，能执行role1相关的操作");

this.currUser.checkRoles("role1","role2");

System.out.println(this.currUser.getPrincipal() + " 拥有role1,role2角色，能执行role1,2相关的操作");

this.currUser.checkRole("admin");

System.out.println(this.currUser.getPrincipal() + " 拥有admin角色，能执行所有的操作");

}

* 基于权限编码授权

同样的，Subject类中同样存在isPermission\*() 和 checkPermission\*()方法，使用方式同前面的hasRole\*()和checkRole\*()方法一样，只是这里判断的目标是权限。详细代码我们这里就不累赘了，如下简单介绍一下相关的方法。

1. boolean isPermitted(Permission p)

判断Subject是否有权限，有返回true，否则返回false

1. boolean[] isPermitted(List<Permission> perms)

判断Subject是否有集合中的权限，对应的数组下标为true 或 false

1. boolean isPermittedAll(Collection<Permission> perms)

判断Subject是否具有集合中所有权限，有返回true，否则返回false

1. boolean isPermitted(String p)

同1）

1. boolean[] isPermitted(String … perms)

同2）

1. boolean isPermittedAll(String… perms)

同3）

1. void checkPermission(Permission perm)

声明subject具有指定权限，有就继续执行，否则抛AuthorizationException

1. void checkPermission(String perm)

同7）

1. void checkPermissions(Collection<Permission> perms)

声明subject具有指定集合所有权限，有就继续执行，否则抛AuthorizationException

1. void checkPermissions(String… perms)

同9）

* + 1. 注解式授权

Shiro安全框架中权限控制有五个注解，它们分别是：

* RequiresAuthentication

使用该注解标注的类，实例，方法在访问或调用时，当前Subject必须在当前session中已经过认证。

* RequiresGuest

使用该注解标注的类，实例，方法在访问或调用时，当前Subject可以是“gust”身份，不需要经过认证或者在原先的session中存在记录。

* RequiresPermissions

当前Subject需要拥有某些特定的权限时，才能执行被该注解标注的方法。如果当前Subject不具有这样的权限，则方法不会被执行。

* RequiresRoles

当前Subject必须拥有所有指定的角色时，才能访问被该注解标注的方法。如果当天Subject不同时拥有所有指定角色，则方法不会执行还会抛出AuthorizationException异常。

* RequiresUser

当前Subject必须是应用的用户，才能访问或调用被该注解标注的类，实例，方法。

多个注解可以同时加在一个方法上面，它们的固定执行顺序是RequiresRoles, RequiresPermissions, RequiresAuthentication, RequiresUser, RequiresGuest。因为在J2SE环境下不方便测试注解的效果，所以暂时不做代码演示。

* + 1. Jsp标签式授权

Shiro安全框架为jsp提供了一个控制标签集合，在web应用中再使用。

* 1. 授权

现在我们大致了解Shiro安全框架是怎么样基于Subject完成授权的了，接下来我们详细研究Shiro安全框架内部是怎么实现授权的。如下图6-2授权流程 描述了Shiro安全框架内部实现授权的流程。

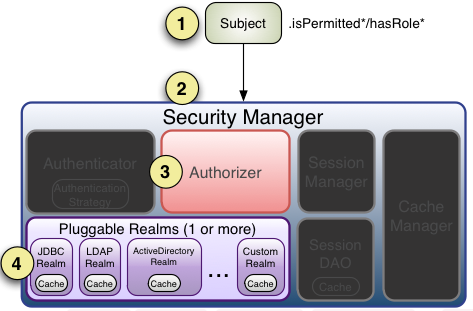


图6-2 授权流程

流程每步的说明如下：

1. 调用Subject的isPermitted\*/hasRole\*方法，Subject自己不执行，通过SecurityManager委托给Authorizer对象。
2. Authorizer是真正执行授权的组件，它先判断输入的是否是字符串格式的权限，如果是，调用PermissionResolver对象，将字符串权限转换成Permission对象。
3. 在正式完成授权前，需要调用Realm，从应用中读取当前Subject在应用中拥有的权限信息（角色、权限等）
4. 将Subject传入的权限同通过Realm获取的Subject在应用中的权限进行匹配，匹配上返回true，否则返回false。如果有多个Realm可以获取到Subject在应用的权限信息，就委托给ModularRealmAuthorizer进行循环匹配。
   * 1. ModularRealmAuthorizer

前面有介绍，SecurityManager默认使用的是ModularRealmAuthorizer对象完成授权的。ModularRealmAuthorizer通知直接基于一个Realm和多个Realm对象的授权，如下是ModularRealmAuthorizer的部分实现代码块。

public boolean isPermitted(PrincipalCollection principals, String permission) {

assertRealmsConfigured();

for (Realm realm : getRealms()) {

if (!(realm instanceof Authorizer)) continue;

if (((Authorizer) realm).isPermitted(principals, permission)) {

return true;

}

}

return false;

}

通过上面代码我们知道，ModularRealmAuthorizer是循环判断Realm是否实现了Authorizer接口，如果实现了Authorizer接口，就调用Realm的授权方法完成授权确定，否则就直接跳过；任何一个Realm授权成功后，就直接返回授权结果，后面的即使实现了Authorizer接口的Realm也不会执行。所有Realm执行完后，都没有授权成功的话，返回false。

需要说明的是，ModularRealmAuthorizer是按Realm在SecurityManager对象中注册的顺序进行调用的。

实现了Authorizer接口的Realm是Shiro安全框架中授权的最后环节，它的内部流程如下：

1. 初始化判断Subject调用的授权方法
2. 如果调用hasRole\*，则直接调用AuthorizationInfo.getRoles()获取角色集合，与传入的角色比较即可；
3. 如果调用如isPermitted(“user:view”)，首先通过PermissionResolver将权限字符串转换成相应的Permission实例，默认使用WildcardPermissionResolver，即转换为通配符的WildcardPermission；
4. 通过AuthorizationInfo.getObjectPermissions()得到Permission实例集合；通过AuthorizationInfo. getStringPermissions()得到字符串集合并通过PermissionResolver解析为Permission实例；然后获取用户的角色，并通过RolePermissionResolver解析角色对应的权限集合（默认没有实现，可以自己提供）；
5. 调用Permission. implies(Permission p)逐个与传入的权限比较，如果有匹配的则返回true，否则false。
   * 1. 授权相关对象

通过前面的介绍，Shiro安全框架中实现授权，主要关系到Authorizer、PermissionResolver、RolePermissionResolver等对象。

Authorizer的职责是进行授权（访问控制），是Shiro API中授权核心的入口点，其提供了相应的角色/权限判断接口。SecurityManager继承了Authorizer接口，且提供了ModularRealmAuthorizer用于多Realm时的授权匹配。PermissionResolver用于解析权限字符串到Permission实例，而RolePermissionResolver用于根据角色解析相应的权限集合。

这些对象的配置和使用，我们参考6.4小节的各种常用的使用授权。

* 1. 使用授权

前面介绍了Shiro安全框架授权的概念、流程和各个对象，接下来我们研究在实际项目中，基于常用的Realm实现授权。

* + 1. 基于IniRealm实现授权

编写shiro-iniRealm.ini，设置好SecurityManager的Authorizer和用户、角色、权限相关信息，如下样例代码块主要描述了SecurityManager中Authorizer的配置，详细内容请查看随书代码[code/ShiroTP06/src/main/resources/shiro\_iniRealm.ini](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\resources\shiro_iniRealm.ini)。

#自定义authorizer

authorizer=org.apache.shiro.authz.ModularRealmAuthorizer

#自定义permissionResolver

permissionResolver=org.apache.shiro.authz.permission.WildcardPermissionResolver

authorizer.permissionResolver=$permissionResolver

#设置SecurityManager授权器

securityManager.authorizer=$authorizer

… …

如上配置描述的是在SecurityManager中设置ModularRealmAuthorizer，使用的是WildcardPermission。这些都Shiro安全框架默认使用的。配置出来目的是明确说明它们的配置使用。

接下来编写测试授权代码，该代码同6.2小节中的测试代码类似，详细内容可以查看随书代码[code/ShiroTP06/src/test/java/cn/com/shiro/book/tp06/TestIniRealmAuthorizer.  
java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\test\java\cn\com\shiro\book\tp06\TestIniRealmAuthorizer.java)。

* + 1. 基于PropertiesRealm实现授权

基于PropertiesRealm实现授权的配置和代码，同6.4.1中基于IniRealm的配置和代码类似。不同的是PropertiesRealm的用户、角色和权限信息是单独用Properties文件配置的，样例代码块如下，详细内容请查看随书代码[code/ShiroTP06/src/main/resources/  
users\_prop.properties](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\resources\users_prop.properties)。

user.root=123456,admin

user.guest=123456,guest

user.zhangsan=123456,role1,role2

user.lisi=abc123,role1

user.wangwu=abctest,role3

role.admin=\*

role.guest=\*:view

role.role1=user:add, user:view

role.role2=user:add,update

role.role3=user:delete

ini配置文件请查看[code/ShiroTP06/src/main/resources/shiro\_propRealm.ini](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\resources\shiro_propRealm.ini)。测试代码请查看[code/ShiroTP06/src/test/java/cn/com/shiro/book/tp06/TestPropAuthorizer.java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\test\java\cn\com\shiro\book\tp06\TestPropRealmAuthorizer.java)

* + 1. 基于JdbcRealm实现授权

基于JdbcRealm实现授权的方式同前面大部分一样，包括测试代码。有点区别的是JdbcRealm在配置的时候，要根据自己的请情况设置根据用户名查询角色和根据角色名查询权限的查询语句，如下样例配置。详细情况请查看随书代码[code/ShiroTP06/src/main/resources/shiro\_jdbcRealm.ini](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\resources\shiro_jdbcRealm.ini)。

… …

#设置jdbcRealm查询用户和密码的查询语句

jdbcRealm.authenticationQuery=select ur\_password,ur\_salt from shr\_user where ur\_user\_name=?

#设置jdbRealm查询用户角色的语句

jdbcRealm.userRolesQuery=select rl\_name from shr\_view\_user\_role where ur\_user\_name=?

#设置jdbcRealm查询角色权限的语句

jdbcRealm.permissionsQuery=select pr\_desc from shr\_view\_role\_permission where rl\_name=?

#设置jdbcRealm对象

securityManager.realms=$jdbcRealm

测试代码请查看[code/ShiroTP06/src/test/java/cn/com/shiro/book/tp06/TestJdbcRealm  
Authorizer.java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\test\java\cn\com\shiro\book\tp06\TestJdbcRealmAuthorizer.java)。

另外，在我们第五章设计的表的基础上，添加shr\_demo\_permission, shr\_rl\_dmpr\_map表和shr\_view\_user\_role, shr\_view\_role\_permission视图。样例脚本如下，详细脚本请查看[code/ShiroTP06/demo\_jdbc.sql](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\demo_jdbc.sql)。

CREATE TABLE `shr\_demo\_permission` (

`pr\_id` int(11) NOT NULL AUTO\_INCREMENT,

`pr\_desc` varchar(50) DEFAULT NULL,

PRIMARY KEY (`pr\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8;

CREATE TABLE `shr\_rl\_dmpr\_map` (

`rl\_id` int(11) NOT NULL DEFAULT '0',

`pr\_id` int(11) NOT NULL DEFAULT '0',

PRIMARY KEY (`rl\_id`,`pr\_id`),

KEY `pr\_id` (`pr\_id`),

CONSTRAINT `shr\_rl\_dmpr\_map\_ibfk\_1` FOREIGN KEY (`rl\_id`) REFERENCES `shr\_role` (`rl\_id`),

CONSTRAINT `shr\_rl\_dmpr\_map\_ibfk\_2` FOREIGN KEY (`pr\_id`) REFERENCES `shr\_demo\_permission` (`pr\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `shr\_view\_role\_permission` AS select `rl`.`rl\_name` AS `rl\_name`,`pr`.`pr\_desc` AS `pr\_desc` from ((`shr\_role` `rl` join `shr\_demo\_permission` `pr`) join `shr\_rl\_dmpr\_map` `mp`) where ((`rl`.`rl\_id` = `mp`.`rl\_id`) and (`mp`.`pr\_id` = `pr`.`pr\_id`));

CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `shr\_view\_user\_role` AS select `rl`.`rl\_name` AS `rl\_name`,`ur`.`ur\_user\_name` AS `ur\_user\_name` from ((`shr\_role` `rl` join `shr\_user` `ur`) join `shr\_ur\_rl\_map` `mp`) where ((`rl`.`rl\_id` = `mp`.`rl\_id`) and (`mp`.`ur\_id` = `ur`.`ur\_id`));

* + 1. 多个Realm实现授权

基于多个Realm实现授权，就是将每个独立Realm的ini配置内容合并起来，定义多个Realm对象，最后在SecurityManager.realms中同时指定这些对象。测试代码同前面单个Realm实现授权的代码一样。在这里就不重复了。

* + 1. 自定义Authorizer

前面6.4.1到6.4.5，我们是使用已有的Authorizer和对应的实现类实现授权功能，接下来我们以自定义授权过程中相关的对象，体现实现授权。

1. 自定义BitPermission类。

BitPermission用于实现二进制位控制方式的权限，如规则是：

权限字符串格式：+资源字符串+权限位+实例ID；

以+开头中间通过+分割；

用权限位上的0和1表示是否有对应的权限，0表示无，1表示有；

1 新增（二进制：0001）、2 修改（二进制：0010）、4 删除（二进制：0100）、8 查看（二进制：1000）；如 +user+10 表示对资源user拥有修改/查看权限。

样例代码块如下，详细内容请查看随书代码[code/ShiroTP06/src/main/java/cn/com/  
shiro/book/tp06/authz/permission/BitPermission.java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\java\cn\com\shiro\book\tp06\authz\permission\BitPermission.java)。

… …

/\*\*

\* +资源id+操作码+对象id

\* \*/

public BitPermission(String perm){

String[] arrays = perm.split("\\+");

if(arrays.length>1){

this.resourceId = Integer.parseInt(arrays[1]);

}

if(arrays.length>2){

this.operateCode = Integer.parseInt(arrays[2]);

}

if(arrays.length>3){

this.instanceId = Integer.parseInt(arrays[3]);

}

}

@Override

public boolean implies(Permission p) {

boolean bool = false;

if (p != null && p instanceof BitPermission) {

BitPermission bitPrm = (BitPermission) p;

System.out.println("target->"+bitPrm.resourceId+":"+bitPrm.operateCode);

bool = (this.resourceId == bitPrm.resourceId)

&& ((this.operateCode & bitPrm.operateCode) != 0);

}

return bool;

}

如上代码，BitPermission构造器将以+号开通的字符串权限中的资源id、二进制权限标记和对象id解析出来。implies方法实现两个BitPermission对象比较，判断两个权限对象是否相互兼容，返回比较结果。

BitPermission类的主要作用是比较两个BitPermission是否兼容

1. 自定义BitAndWildcardPermissionResolver.java类

BitAndWildcardPermissionResolver类的作用是将字符串权限解析成对应的权限对象（WildcardPermission 或 BitPermission）。样例代码块如下，详细内容请查看随书代码[code/ShiroTP06/ src/main/java/cn/com/shiro/book/tp06/authz/permission/  
BitAndWildcardPermissionResolver.java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\java\cn\com\shiro\book\tp06\authz\permission\BitAndWildcardPermissionResolver.java)。

public class BitAndWildcardPermissionResolver implements PermissionResolver {

@Override

public Permission resolvePermission(String permissionString) {

if (permissionString.startsWith("+")) {

return new BitPermission(permissionString);

}

return new WildcardPermission(permissionString);

}

}

1. 自定义ShiroRolePermissionResolver.java类。

ShiroRolePermissionResolver类的作用是根据角色名，从应用数据找到对应的所有权限。样例代码如下，详细内容请查看随书代码[code/ShiroTP06/ src/main/java/cn/com/shiro/book/tp06/authz/permission/ShiroRolePermissionResolver.java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\java\cn\com\shiro\book\tp06\authz\permission\ShiroRolePermissionResovler.java)。

public class ShiroRolePermissionResovler implements RolePermissionResolver {

private Map<String,Collection<Permission>> rolePerms = new HashMap<String,Collection<Permission>>();

{

rolePerms.put("admin", Arrays.asList((Permission)new WildcardPermission("\*")));

rolePerms.put("role1", Arrays.asList((Permission)new BitPermission(1,1),

(Permission)new BitPermission(1,2)));

rolePerms.put("role2", Arrays.asList((Permission)new BitPermission(1,1),

(Permission)new BitPermission(1,3)));

rolePerms.put("role3", Arrays.asList((Permission)new BitPermission(1,1),

(Permission)new BitPermission(1,4)));

}

@Override

public Collection<Permission> resolvePermissionsInRole(String roleString) {

return this.rolePerms.get(roleString);

}

}

如上代码，使用代码块初始化了rolePerms Map变量，里面保存多个角色和对应的权限集合，模拟实现应用的数据源。

resolvePermissionsInRole方法，根据角色名找出数据源中该角色包含的所有权限集合。

1. 自定义ShiroRealm.java类

ShiroRealm类继承AuthorizingRealm父类，实现进行身份认证的基本方法(doGetAuthenticationInfo)和授权控制的基本方法(doGetAuthorizationInfo)。

* + 1. doGetAuthentictionInfo(AuthenticationToken)方法的作用是根据应用通过Subject的login方法传入的AuthenticationToken的身份标识(Principals)，从应用安全数据中找出对应的认证Subject身份合法性的凭证信息（Credentials），封装成AuthenticationInfo对象返回，以便后面流程的身份认证。
    2. doGetAuthorizationInfo(PrincipalCollection)反方法的作用是根据传入的Principals从应用安全数据中找出该Subject拥有的角色集合和权限集合信息，封装到AuthorizationInfo对象中返回，以便后面流程的授权判断。

ShiroRealm类的样例代码块如下，详细内容请查看随书代码[code/ShiroTP06/ src/main/java/cn/com/shiro/book/tp06/realm/ShiroRealm.java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\java\cn\com\shiro\book\tp06\realm\ShiroRealm.java)。

public class ShiroRealm extends AuthorizingRealm {

private Map<String, String> accountMap = new HashMap<String, String>();

private Map<String, List<String>> userRolesMap = new HashMap<String, List<String>>();

private Map<String, List<String>> userPermsMap = new HashMap<String, List<String>>();

// 模拟数据源初始化数据

{

// 初始化用户账号信息

accountMap.put("zhangsan", "abc123");

accountMap.put("lisi", "123456");

accountMap.put("wangwu", "123456");

// 初始化用户角色信息

userRolesMap.put("zhangsan", Arrays.asList("admin"));

userRolesMap.put("lisi", Arrays.asList("role1", "role2"));

userRolesMap.put("wangwu", Arrays.asList("role3"));

// 初始化用户权限信息

userPermsMap.put("wangwu",

Arrays.asList("order:create", "order:delete"));

}

/\*\*

\* 获取指定principals包含的角色和权限信息

\* \*/

@Override

protected AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

if (this.userRolesMap.get(principals.getPrimaryPrincipal()) != null) {

info.addRoles(this.userRolesMap.get(principals

.getPrimaryPrincipal()));

}

if (this.userPermsMap.get(principals.getPrimaryPrincipal()) != null) {

info.addStringPermissions(this.userPermsMap.get(principals

.getPrimaryPrincipal()));

}

return info;

}

/\*\*

\* 根据token的principals返回从数据源中获取的对应AuthenticationInfo

\* \*/

@Override

protected AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) throws AuthenticationException {

SimpleAuthenticationInfo info = null;

String password = this.accountMap.get(token.getPrincipal().toString());

if (password != null) {

info = new SimpleAuthenticationInfo(token.getPrincipal(), password,

this.getName());

}

return info;

}

}

如上代码中的三个Map成员变量，分别模拟的是应用安全数据中的用户密码集合、用户角色集合和用户权限集合。

1. 在resources目录下创建shiro\_shiroRealm.ini，在里面配置securityManager，使用我们自定义BitAndWildcardPermissionResolver、ShiroRolePermissionResolver和ShiroRealm对象，完成身份认证和权限授权。样例代码块如下，详细内容请查看随书代码[code/ShiroTP06/src/main/resources/shiro\_shiroRealm.ini](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\main\resources\shiro_shiroRealm.ini)。

[main]

#自定义authorizer

authorizer=org.apache.shiro.authz.ModularRealmAuthorizer

#自定义permissionResolver

permissionResolver=cn.com.shiro.book.tp06.authz.permission.BitAndWildcardPermissionResolver

authorizer.permissionResolver=$permissionResolver

#设置SecurityManager授权器

securityManager.authorizer=$authorizer

#配置shiroRealm对象(realm名称=realm类名)

shiroRealm=cn.com.shiro.book.tp06.realm.ShiroRealm

#设置shiroRealm对象

securityManager.realms=$shiroRealm

1. 编写TestShiroRealmAuthorizer代码进行测试。

因为这个测试代码同前面几个测试代码一样，这里就不重复累赘了，详细内容请查看随书代码[code/ShiroTP06/src/test/java/cn/com/shiro/book/tp06/TestShiroRealm  
Authorizer.java](file:///C:\Users\Noble\AppData\Roaming\Microsoft\code\ShiroTP06\src\test\java\cn\com\shiro\book\tp06\TestShiroRealmAuthorizer.java)。

* 1. 总结

本章从概念上介绍了授权相关的三个概念：用户、角色和权限，以及Shiro安全框架基于这三个概念的授权流程和涉及到的对象Subject、SecurityManager、Authorizer、Realm、Permission、PermissionResolver、RolePermissionResolver，以及它们的作用和相互关系。

同时从实际运用的角度，介绍了基于IniRealm、PropertiesRealm、JdbcRealm和自定义授权对象类的方法，实现基于Shiro安全框架的授权。