**1、理解授权**

**授权，即访问控制**，在应用中，用来控制各种角色对资源的访问权限等，在授权中，有几个关键对象，主体（Subject）、资源（Resource）、权限（Permission）、角色（Role）。

* 主体：即访问者，应用的任何访问者，不只是人，也可能是个爬虫。
* 资源：在应用中，主体可以访问的任何东西，比如页面，某个业务方法，某些数据等。
* 权限：安全策略中的院子授权单位，通过权限我们可以表示在应用中，主体没有操作某个资源的权利，如：增删改查，Apache Shiro支持粗粒度权限（如用户某块的所有权限）和细粒度权限（操作某个用户的权限，即实例级别的）。
* 角色：**角色代表了操作结合**，可以理解为**权限的集合**，一般授予用户的是角色，即一组权限。

**2、授权方式**

（1）编程式方式

Subject subject = SecurityUtils.getSubject();

if(subject.hasRole(“admin”)) {

//有权限

} else {

//无权限

}

（2）注解式方式

@RequiresRoles("admin")

public void hello() {

//有权限

}

（3）标签方式

<shiro:hasRole name="admin">

<!— 有权限 —>

</shiro:hasRole>

**3、授权**

基于角色的访问控制(隐式角色)

[users]

chen=123,role1,role2

he=123,role3

[roles]

role1=user:create,user:update

role2=user:select

role3=user:delete

测试代码(测试用户是否拥有某角色)

@Test(expected = UnauthorizedException.class)

public void testHasRole() {

//用户登录

Subject subject = login("classpath:shiro-capter3.ini", "chen", "123");

//用户chen,拥有角色role1

Assert.assertTrue(subject.hasRole("role1"));

//用户chen,没有角色role2

Assert.assertFalse(subject.hasRole("noRole"));

//检查用户是否拥有列表里的权限,没有的话会抛出UnauthorizedException异常

subject.checkRoles("role1","role2","role3");

测试用户是否拥有某权限

@Test(expected = UnauthorizedException.class)

public void testHashRight(){

Subject subject = login("classpath:shiro-capter3.ini", "chen", "123");

//断言用户有user:create权限

Assert.assertTrue(subject.isPermitted("user:create"));

//断言用户没有user:delete

Assert.assertFalse(subject.isPermitted("user:delete"));

//如果不存在某个权限时,会抛出UnauthorizedException异常

subject.checkPermissions("user:create","user:select","user:delete");

}

ini配置文件通配符书写方式

单个资源,单个权限 role1=user:create,update

单个资源,所有权限 role=user:\*

所有资源,单个权限 role = \*:create

单个实例,单个权限 role=user:create:1 #对user的1实例,有create权限

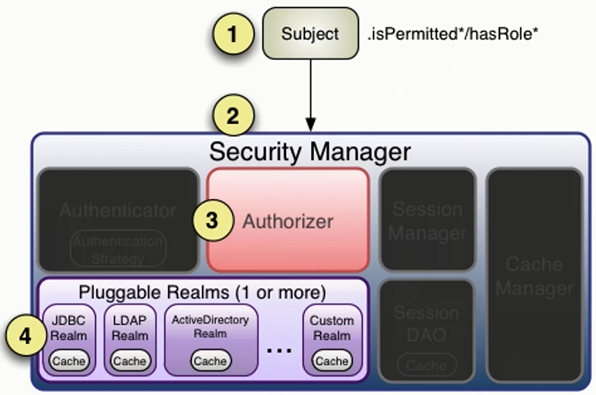
单个实例,多个权限 role=user:create,update:1

单个实例,所有权限 role=user:\*:1

所有实例,单个权限 role=use:create:\*

所有实例,所有权限 role=user:\*:\*

1. **授权流程**



（1）首先需要调用Subject.isPermitted("")接口,其会委托SecurityManager,而接着会委托给Authorizer。

（2）**Authorizer是真正的授权者**,如果我们调用isPermitted("user:create"),其首先会通过PermissionResolver把字符串转换成对应的Permission实例;

（3）在进行授权之前,其会调用相应的Realm获取Subject相应的角色/权限用于匹配传入的角色/权限;

（4）Authirizer会判断Realm的角色/权限和传入的是否匹配,匹配返回true,否则返回false.注意:如果存在多个Realm,会委托给ModularRealmAuthorizer进行循环匹配;

**ModularRealmAuthorizer进行多Realm匹配流程:**

（1）首先检查相应的Realm是否实现了Authorizer;

（2）如果实现了Authorizer,那么接着调用其相应的isPermitted/hasRole接口进行匹配

（3）如果有一个Realm匹配那么将返回true,否则返回false

如果进行Realm授权的话,应该继承AuthorizingRealm

Authorizer,PermissionResolver,RolePermissionResolver

Authorizer职责是进行授权,其提供了相应的角色权限判断接口.