第三章 Shiro权限管理

泽林.王峰 (2019.4.15)

授课大纲

1、 Shiro简介

2、 Shiro的认证

3、 Shiro的授权

授课内容

1、什么是权限管理？

```
+ 权限管理是系统的安全范畴，要求必须是合法的用户才可以访问系统（用户认证），且必须具有该 资源的访问权限才可以访问该 资源（授权）。
1.认证：对用户合法身份的校验，要求必须是合法的用户才可以访问系统。
2.授权：访问控制，必须具有该 资源的访问权限才可以访问该 资源。
3.权限模型：标准权限数据模型包括 ：用户、角色、权限（包括资源和权限）、用户角色关系、角色权限关系。
4.权限分配：通过UI界面方便给用户分配权限，对上边权限模型进行增、删、改、查操作。
5.权限控制：
    5.1）基于角色的权限控制：根据角色判断是否有操作权限，因为角色的变化 性较高，如果角色修改需要修改控制代码，系统可扩展性不强。
    5.2）基于资源的权限控制：根据资源权限判断是否有操作权限，因为资源较为固定，如果角色修改或角色中权限修改不需要修改控制代码，使用此方法系统可维护性很强。建议使用。
```

2）权限管理的解决方案：

对于粗颗粒权限管理，建议在系统架构层面去解决，写系统架构级别统一代码（基础代码）。

> - 粗颗粒权限：比如对系统的url、菜单、jsp页面、页面上按钮、类方法进行权限管理，即对资源类型进行权限管理。

对于细颗粒权限管理：

> - 粗颗粒权限：比如用户id为001的用户信息（资源实例）、类型为t01的商品信息（资源实例），对资源实例进行权限管理，理解对数据级别的权限管理。
> - 细颗粒权限管理是系统的业务逻辑，业务逻辑代码不方便抽取统一代码，建议在系统业务层进行处理。

2.1）基于url的权限管理（掌握）：

> - 企业开发常用的方法，使用web应用中filter来实现，用户请求url，通过filter拦截，判断用户身份是否合法（用户认证），判断请求的地址是否是用户权限范围内的url(授权)。

2.2) 小结：

> 使用基于url拦截的权限管理方式，实现起来比较简单，不依赖框架，使用web提供filter就可以实现。

2.3）问题：

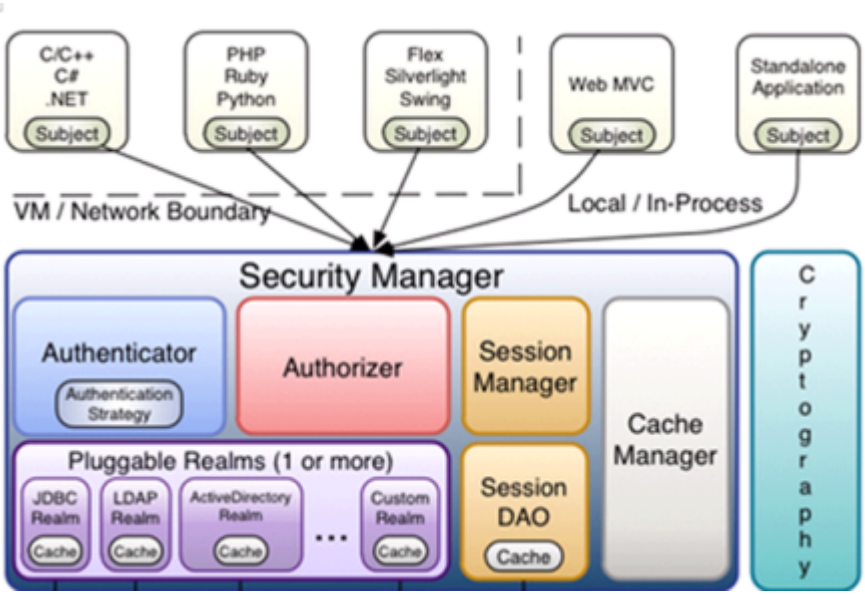
> 需要将所有的url全部配置起来，有些繁琐，不易维护，url(资源)和权限表示方式不规范。

1、 Shiro简介

1.1）什么是shiro？

- shiro是apache的一个开源框架，是一个权限管理的框架，实现 用户认证、用户授权。
- spring中有spring security (原名Acegi)，是一个权限框架，它和spring依赖过于紧密，没有shiro使用简单。
- shiro不依赖于spring，shiro不仅可以实现 web应用的权限管理，还可以实现c/s系统，分布式系统权限管理， shiro属于轻量框架，越来越多企业项目开始使用shiro。
- 使用shiro实现系统 的权限管理，有效提高开发效率，从而降低开发成本。

1.2）Shiro的架构？



> **各模块功能介绍：**
>
> 1. subject：主体，可以是用户也可以是程序，主体要访问系统，系统需要对主体进行认证、授权。
> 2. securityManager：安全管理器，主体进行认证和授权都 是通过securityManager进行。
> 3. authenticator：认证器，主体进行认证最终通过authenticator进行的。
> 4. authorizer：授权器，主体进行授权最终通过authorizer进行的。
> 5. sessionManager：web应用中一般是用web容器对session进行管理，shiro也提供一套session管理的方式。
> 6. SessionDao： 通过SessionDao管理session数据，针对个性化的session数据存储需要使用sessionDao。
> 7. cache Manager：缓存管理器，主要对session和授权数据进行缓存，比如将授权数据通过cacheManager进行缓存管理，和ehcache整合对缓存数据进行管理。
> 8. realm：域，领域，相当于数据源，通过realm存取认证、授权相关数据。 注意：在realm中存储授权和认证的逻辑。

> 9. cryptography：密码管理，提供了一套加密/解密的组件，方便开发。比如提供常用的散列、加/解密等功能。比如 md5散列算法。

1.3) 开发第一个shiro程序？

1.3.1) 定义maven工程在pom.xml文件中引入shiro的依赖？

```
<?xml version="1.0" encoding="UTF-8"?>
<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>com.zelin</groupId>
    <artifactId>Shiro_03</artifactId>
    <version>1.0-SNAPSHOT</version>
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.shiro/shiro-core -->
    <dependency>
        <groupId>org.apache.shiro</groupId>
        <artifactId>shiro-core</artifactId>
        <version>1.2.3</version>
    </dependency>
    <dependency>
        <groupId>org.apache.shiro</groupId>
        <artifactId>shiro-spring</artifactId>
        <version>1.2.3</version>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.9</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.6.4</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.3.2</version>
    </dependency>
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-io</artifactId>
```

```
        <version>1.3.2</version>
    </dependency>
</dependencies>
</project>
```

1.3.2) 在resources下定义shiroFirst.ini文件：

```
[users]
zhangsan=111111
```

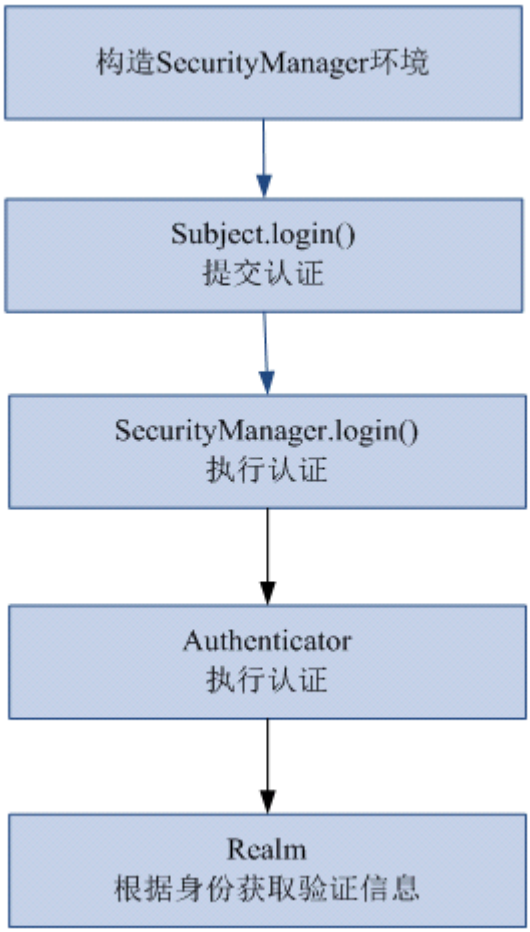
1.3.3) 进行单元测试：

```
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description:
 * @Date: Create in 2019/4/15 10:38
 */
public class TestShiroFirst {
    @Test
    public void testO1(){
        //1.根据ini文件得到SecurityManagerFactory对象
        IniSecurityManagerFactory iniSecurityManagerFactory = new
IniSecurityManagerFactory("classpath:shiroFirst.ini");
        //2.根据上面的工厂对象得到SecurityManager对象
        SecurityManager securityManager = iniSecurityManagerFactory.getInstance();
        //3.将当前的SecurityManager放到当前工作的上下文环境中
        SecurityUtils.setSecurityManager(securityManager);
        //4.得到主体对象
        Subject subject = SecurityUtils.getSubject();
        //5.构造用户的令牌对象（用户输入的信息）
        //5.1）如果用户名输入错误，出现:UnknownAccountException异常
        //5.2）如果密码不正确，出现：IncorrectCredentialsException异常
        AuthenticationToken token = new UsernamePasswordToken("zhangsan","11111");
        //6.通过suject进行认证
        subject.login(token);
        //7.判断认证是否通过
        boolean authenticated = subject.isAuthenticated();
        System.out.println("是否认证通过：" + authenticated);
    }
}
```

认证过程：

2）自定义Realm实现认证？

2.0）认证原理图



2.1) 自定义Realm类：

```
/**
 * @Author: Feng.Wang
 * @Date: 2019/4/15 12:36
 * @Company: Zelin.ShenZhen
 * @ClassName: CustomRealm
 * @Description:
*/
public class CustomRealm extends AuthorizingRealm {
    //授权的方法
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)
{
        return null;
    }
    //认证的方法
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
throws AuthenticationException {
       //1.得到身份信息
        Object principal = token.getPrincipal();
       //2.如果是null就返回null,其实就是此值返回给自定义realm的老大ModularRealmAuthticator这
个对象
       //此时ModularRealmAuthticator这个账户就会抛出：unknownAccoutException
```

```
        if(principal == null){
            return null;
        }
        //3.先模拟从数据库中取得密码
        String password = "111111";
        //4.返回AuthenticationInfo对象
        return new SimpleAuthenticationInfo(principal,password,"aaaa");
    }
}
```

2.2) shiroCustom.ini配置文件：

```
[main]
customRealm=com.zelin.realm.CustomRealm
securityManager.realms=$customRealm
```

2.3) 完成单元测试：

```
 @Test
    public void test01(){
        //1.根据ini文件得到SecurityManagerFactory对象
        IniSecurityManagerFactory managerFactory = new
IniSecurityManagerFactory("classpath:shiroCustom.ini");
        //2.根据上面的managerFactory得到SecurityManager对象
        SecurityManager securityManager = managerFactory.getInstance();
        //3.将此对象设置到上下文工作环境中
        SecurityUtils.setSecurityManager(securityManager);
        //4.得到主体对象
        Subject subject = SecurityUtils.getSubject();
        //5.构造一个令牌对象
        AuthenticationToken token = new UsernamePasswordToken("zhangsan","111111");
        //6.进行认证
        subject.login(token);
        //7.检查认证是否通过
        boolean authenticated = subject.isAuthenticated();
        System.out.println("是否认证通过：" + authenticated);

    }
```

2.4) 加盐单元测试：

```
  //加盐测试
    @Test
    public void testAddSalt(){
        String salt = "rbtwy";              //加的盐值
        String oldPassword = "111111";      //原始密码
        int hashIterations = 1;             //代表加密次数
        //加密算法一：
        Md5Hash md5Hash = new Md5Hash(oldPassword,salt,hashIterations);
        System.out.println(md5Hash);
        //加密算法二：
        SimpleHash simpleHash = new SimpleHash("md5",oldPassword,salt,hashIterations);
        System.out.println(simpleHash.toString());
    }
```

2.5 ）自定义Realm加盐

2.5.1 ）自定义加盐的realm类

```
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description: 自定义Realm加盐
 * @Date: Create in 2019/4/15 11:34
 */
public class Md5CustomRealm extends AuthorizingRealm {
    //授权
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)
{
        return null;
    }
    //认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
throws AuthenticationException {
        //1.得到身份信息
        Object principal = token.getPrincipal();
        //2.判断看是否为null
        if(null == principal){
            return null;
        }
        //3.模拟从数库中取得密码及加盐值
        String password = "ec1b86316c81b3f3440c07f65a74bf79";   //加密、加盐后的密码
        String salt = "rbtwy";                                  //盐值
        //4.返回SimpleAuthenticationInfo对象
        return new SimpleAuthenticationInfo(principal,password,
ByteSource.Util.bytes(salt),"aaa");
    }
}
```

2.5.2 ）加盐的自定义配置文件shiroCustomSaltRealm.ini

```
[main]
#定义凭证匹配器对象
customCredentialsMatcher=org.apache.shiro.authc.credential.Md5CredentialsMatcher
#指定凭证匹配器对象的加密次数为1
customCredentialsMatcher.hashIterations=1
#自定义加盐的Realm
customSaltCustomRealm=com.zelin.realm.Md5CustomRealm
#指定指定义realm的凭证匹配器对象
customSaltCustomRealm.credentialsMatcher=$customCredentialsMatcher
#将自定义realm加入到securityManager的realms集合中
securityManager.realms=$customSaltCustomRealm
```

认证小结：

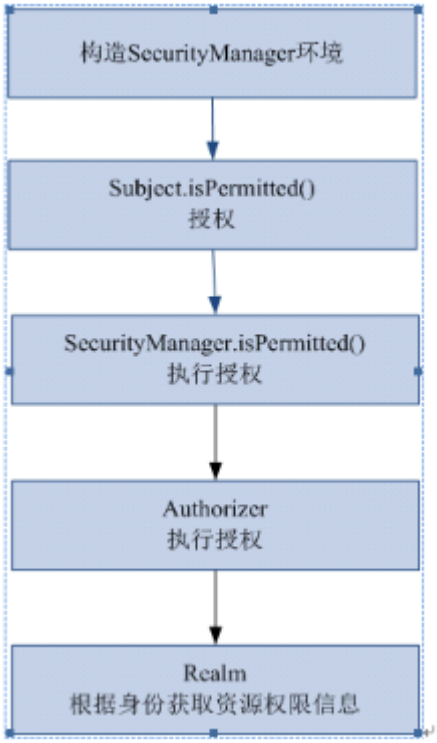
```
1. 认证步骤：
第一步：根据ini配置文件得到SecurityManager对象
第二步：由ModuarRealmAuthenticator调用iniRealm，将token中的主体信息传递到ini文件中，进行比对，如果内容一致，就将得到的认证主体给ModuarRealmAuthenticator,否则，抛出”unknown Account Exception”。
第三步：将返回得到的ini文件中的主体密码与token中输入的密码进行比对，如果不一致，就抛出“Incorrect credenticals Exception”（无效凭证异常）

2. 小结：
iniRealm如果在ini文件中查询到了主体信息，就将其返回给ModularRealmAuthenticator,如果没有找到主体，则将null返回给ModularRealmAuthenticator。
ModularRealmAuthenticator得到iniRealm返回给其的值，根据值再进行相应的处理，具体如下：
①如果iniRealm返回的是null，则抛出org.apache.shiro.authc.UnknownAccountException异常！
②如果iniRealm返回的主体对象不为null，则ModularRealmAuthenticator将此主体对象中的凭证信息（密码）与token中的密码进行比对，如果一致，则认证通过。否则，由ModularRealmAuthenticator抛出”org.apache.shiro.authc.IncorrectCredentialsException”
```

授权过程:

3）利用ini文件实现授权

3.0)授权原理图：



3.1) shiroPermission.ini文件定义：

```
[users]
zhangsan=111111,role1,role2
lisi=222222,role1
[roles]
role1=student:create,student:update,student:delete
role2=student:create
```

3.2) 单元测试

```
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description:
 * @Date: Create in 2019/4/15 11:58
 */
public class TestPermission {
    @Test
    public void test01(){
        //第一部分：进行认证操作
        //1.根据ini文件得到SecurityManagerFactory对象
        IniSecurityManagerFactory managerFactory = new
IniSecurityManagerFactory("classpath:shiroPermission.ini");
        //2.根据上面的managerFactory得到SecurityManager对象
        SecurityManager securityManager = managerFactory.getInstance();
        //3.将此对象设置到上下文工作环境中
        SecurityUtils.setSecurityManager(securityManager);
        //4.得到主体对象
        Subject subject = SecurityUtils.getSubject();
```

```
        //5.构造一个令牌对象
        AuthenticationToken token = new UsernamePasswordToken("zhangsan","111111");
        //6.进行认证
        subject.login(token);
        //7.检查认证是否通过
        boolean authenticated = subject.isAuthenticated();
        System.out.println("是否认证通过：" + authenticated);

        //第二部分：进行授权操作（下面第二部分必须在上面第一部分认证难过之后才能执行）
        //2.1)进行角色的授权操作
        //2.1.1)单个角色的授权操作
        boolean role1 = subject.hasRole("role1");
        System.out.println("是否有role1角色：" + role1);
        //2.1.2)多个角色的授权操作(没有角色，也不会抛出异常，只要有一个角色没有，就返回false)
        boolean hasAllRoles = subject.hasAllRoles(Arrays.asList("role1", "role3"));
        System.out.println("是否有role1,role3两个角色：" + hasAllRoles);

        //2.2)进行权限的授权操作
        //2.2.1)判断单个权限
        boolean permitted = subject.isPermitted("student:create");
        System.out.println("是否有单个权限：" + permitted);
        //2.2.2)判断多个权限(如果没有指定的权限，也不会抛出异常)
        boolean permittedAll = subject.isPermittedAll("student:create",
"student:update", "student:query");
        System.out.println("是否有多个权限：" + permittedAll);
    }
}
```

4) 利用ini文件+自定义realm实现授权

4.1) shiroPermissionRealm.ini文件定义

```
[main]
customRealm=com.zelin.realm.CustomPermissionRealm
securityManager.realms=$customRealm
```

4.2) CustomPermissionRealm自定义realm类

```
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description: 自定义授权realm
 * @Date: Create in 2019/4/15 12:08
 */
public class CustomPermissionRealm extends AuthorizingRealm {
    //授权
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)
{
        //1.得到身份信息
        Object primaryPrincipal = principals.getPrimaryPrincipal();
```

```
        //2.模拟从数据库中得到权限列表
        List<String> permissions = new ArrayList<>();
        permissions.add("student:create");
        permissions.add("student:delete");
        permissions.add("student:update");
        //3.构造AuthorizationInfo对象
        SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
        //4.将权限列表对象与SimpleAuthorizationInfo对象进行绑定
        authorizationInfo.addStringPermissions(permissions);
        return authorizationInfo;
    }
    //认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
throws AuthenticationException {
        //1.得到认证身份对象
        Object principal = token.getPrincipal();
        if(principal == null) return null;
        //2.从数据库中得到密码
        String password = "111111";
        return new SimpleAuthenticationInfo(principal,password,"aaa");
    }
}
```

4.3) 单元测试：

```
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description:
 * @Date: Create in 2019/4/15 11:58
 */
public class TestCustomRealmPermission {
    @Test
    public void test01(){
        //第一部分：进行认证操作
        //1.根据ini文件得到SecurityManagerFactory对象
        IniSecurityManagerFactory managerFactory = new
IniSecurityManagerFactory("classpath:shiroPermissionRealm.ini");
        //2.根据上面的managerFactory得到SecurityManager对象
        SecurityManager securityManager = managerFactory.getInstance();
        //3.将此对象设置到上下文工作环境中
        SecurityUtils.setSecurityManager(securityManager);
        //4.得到主体对象
        Subject subject = SecurityUtils.getSubject();
        //5.构造一个令牌对象
        AuthenticationToken token = new UsernamePasswordToken("zhangsan","111111");
        //6.进行认证
        subject.login(token);
        //7.检查认证是否通过
        boolean authenticated = subject.isAuthenticated();
```

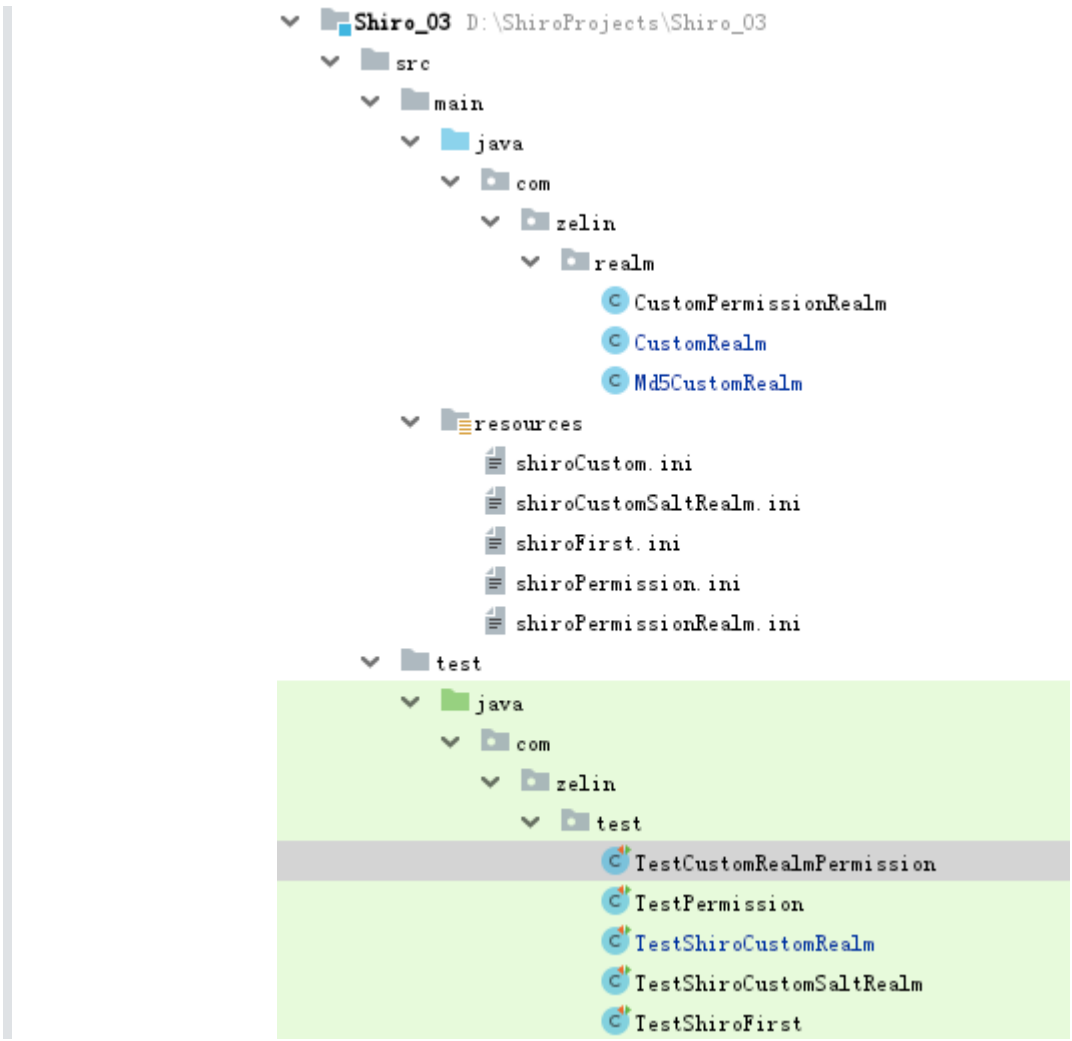
```
25          System.out.println("是否认证通过：" + authenticated);
26
27          //第二部分：进行授权操作（下面第二部分必须在上面第一部分认证难过之后才能执行）
28          //1.1)进行权限的授权操作
29          //1.1.1)判断单个权限
30          boolean permitted = subject.isPermitted("student:create");
31          System.out.println("是否有单个权限：" + permitted);
32          //1.1.2)(如果没有指定的权限，也不会抛出异常)
33          boolean permittedAll = subject.isPermittedAll("student:create",
    "student:update", "student:query");
34          System.out.println("是否有多个权限：" + permittedAll);
35
36          //1.2)使用checkPermission检查权限，找不到抛出异常
37          //1.2.1)检查单个权限(如果没有此权限，则抛出：
    org.apache.shiro.authz.UnauthorizedException: Subject does not have permission
    [student:query])
38          subject.checkPermissions("student:create");
39          //1.2.2)检查多个权限(只要有一个没有，则抛出：
    org.apache.shiro.authz.UnauthorizedException: Subject does not have permission
    [student:query])
40          subject.checkPermissions("student:update","student:delete","student:query");
41
42      }
43  }
44
```

运行效果如下：(因没有student:query权限，报错.)

```
org.apache.shiro.authc.IncorrectCredentialsException: Submitted credentials for token [org.apache.shiro.authc
.UsernamePasswordToken - zhangsan, rememberMe=false] did not match the expected credentials.

    at org.apache.shiro.realm.AuthenticatingRealm.assertCredentialsMatch(AuthenticatingRealm.java:600)
    at org.apache.shiro.realm.AuthenticatingRealm.getAuthenticationInfo(AuthenticatingRealm.java:578)
    at org.apache.shiro.authc.pam.ModularRealmAuthenticator.doSingleRealmAuthentication(ModularRealmAuthenticator
.java:180)
    at org.apache.shiro.authc.pam.ModularRealmAuthenticator.doAuthenticate(ModularRealmAuthenticator.java:267)
    at org.apache.shiro.authc.AbstractAuthenticator.authenticate(AbstractAuthenticator.java:198)
```

附项目结构图：


Shiro_03 D:\ShiroProjects\Shiro_03
src
main
java
com
zelin
realm
CustomPermissionRealm
CustomRealm
Md5CustomRealm
resources
shiroCustom.ini
shiroCustomSaltRealm.ini
shiroFirst.ini
shiroPermission.ini
shiroPermissionRealm.ini
test
java
com
zelin
test
TestCustomRealmPermission
TestPermission
TestShiroCustomRealm
TestShiroCustomSaltRealm
TestShiroFirst