

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？**

- 1 + 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）。
- 2 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。
- 3 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。
- 4 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。
- 5 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。
- 6 5.权限控制：
- 7 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。
- 8 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。

1.3)**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)。

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。

问题：

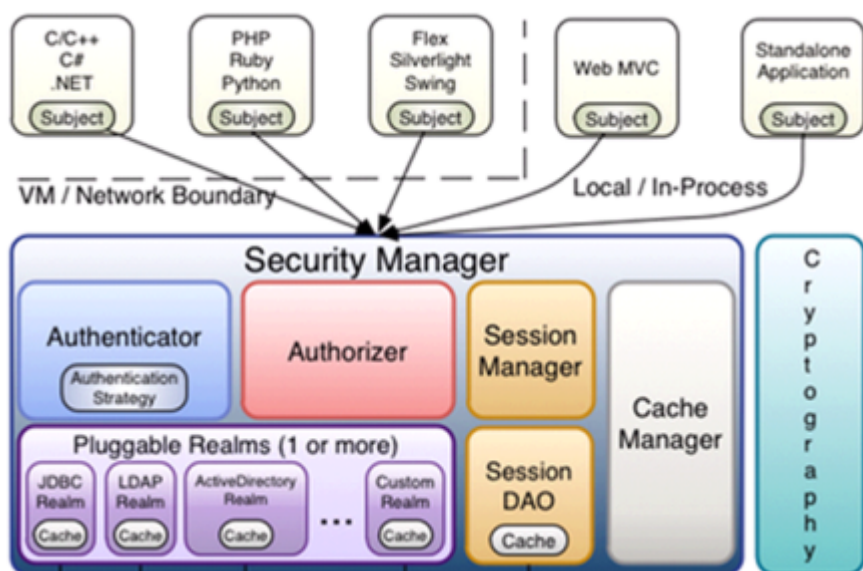
需要将所有的url全部配置起来，有些繁琐，不易维护，url(资源)和权限表示方式不规范。

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？

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5          http://maven.apache.org/xsd/maven-4.0.0.xsd">
6      <modelVersion>4.0.0</modelVersion>
7      <groupId>com.zelin</groupId>
8      <artifactId>Shiro_03</artifactId>
9      <version>1.0-SNAPSHOT</version>
10     <dependencies>
11         <!-- https://mvnrepository.com/artifact/org.apache.shiro/shiro-core -->
12         <dependency>
13             <groupId>org.apache.shiro</groupId>
14             <artifactId>shiro-core</artifactId>
15             <version>1.2.3</version>
16         </dependency>
17         <dependency>
18             <groupId>org.apache.shiro</groupId>
19             <artifactId>shiro-spring</artifactId>
20             <version>1.2.3</version>
21         </dependency>
22         <dependency>
23             <groupId>junit</groupId>
24             <artifactId>junit</artifactId>
25             <version>4.9</version>
26         </dependency>
27         <dependency>
28             <groupId>org.slf4j</groupId>
29             <artifactId>slf4j-log4j12</artifactId>
30             <version>1.6.4</version>
31         </dependency>
32         <dependency>
33             <groupId>org.apache.commons</groupId>
34             <artifactId>commons-lang3</artifactId>
35             <version>3.3.2</version>
36         </dependency>
37         <dependency>
38             <groupId>commons-logging</groupId>
39             <artifactId>commons-logging</artifactId>
40             <version>1.2</version>
41         </dependency>
42         <dependency>
43             <groupId>org.apache.commons</groupId>
44             <artifactId>commons-io</artifactId>
```

```
45     <version>1.3.2</version>
46   </dependency>
47 </dependencies>
48 </project>
```

1.3.2) 在resources下定义shiroFirst.ini文件：

```
1 [users]
2 zhangsan=111111
```

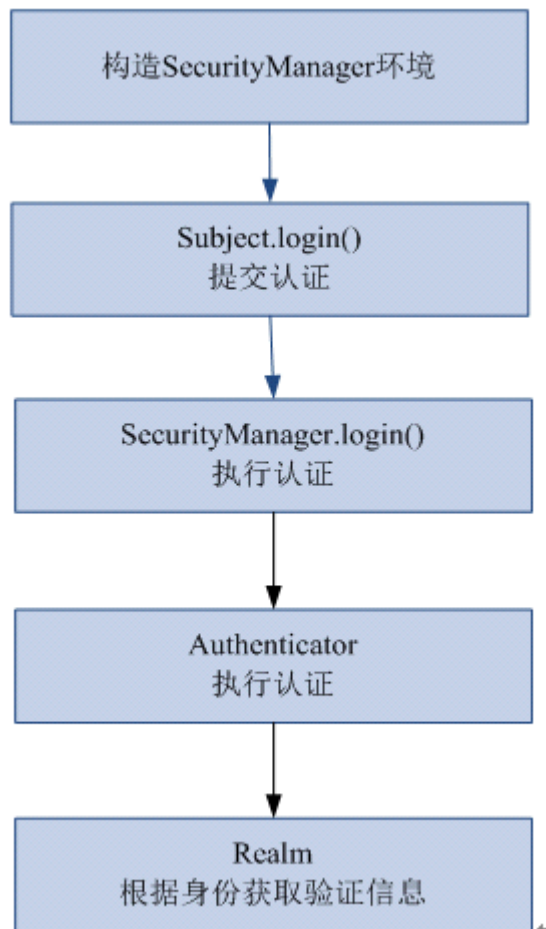
1.3.3) 进行单元测试：

```
1 /**
2  * @Author: Feng.Wang
3  * @Company: Zelin.ShenZhen
4  * @Description:
5  * @Date: Create in 2019/4/15 10:38
6  */
7 public class TestShiroFirst {
8     @Test
9     public void test01(){
10         //1.根据ini文件得到SecurityManagerFactory对象
11         IniSecurityManagerFactory iniSecurityManagerFactory = new
IniSecurityManagerFactory("classpath:shiroFirst.ini");
12         //2.根据上面的工厂对象得到SecurityManager对象
13         SecurityManager securityManager = iniSecurityManagerFactory.getInstance();
14         //3.将当前的SecurityManager放到当前工作的上下文环境中
15         SecurityUtils.setSecurityManager(securityManager);
16         //4.得到主体对象
17         Subject subject = SecurityUtils.getSubject();
18         //5.构造用户的令牌对象（用户输入的信息）
19         //5.1) 如果用户名输入错误，出现：UnknownAccountException异常
20         //5.2) 如果密码不正确，出现：IncorrectCredentialsException异常
21         AuthenticationToken token = new UsernamePasswordToken("zhangsan","11111");
22         //6.通过subject进行认证
23         subject.login(token);
24         //7.判断认证是否通过
25         boolean authenticated = subject.isAuthenticated();
26         System.out.println("是否认证通过：" + authenticated);
27     }
28 }
```

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：

```
1  /**
2   * @Author: Feng.Wang
3   * @Date: 2019/4/15 12:36
4   * @Company: Zelin.ShenZhen
5   * @ClassName: CustomRealm
6   * @Description:
7   */
8  public class CustomRealm extends AuthorizingRealm {
9      //授权的方法
10     @Override
11     protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)
12     {
13         return null;
14     }
15     //认证的方法
16     @Override
17     protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
18     throws AuthenticationException {
19         //1.得到身份信息
20         Object principal = token.getPrincipal();
21         //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ModularRealmAuthenticator这个对象
22         //此时ModularRealmAuthenticator这个账户就会抛出：unknownAccountException
```

```

21         if(principal == null){
22             return null;
23         }
24         //3.先模拟从数据库中取得密码
25         String password = "111111";
26         //4.返回AuthenticationInfo对象
27         return new SimpleAuthenticationInfo(principal,password,"aaa");
28     }
29 }

```

2.2) shiroCustom.ini配置文件：

```

1 [main]
2 customRealm=com.zelin.realm.CustomRealm
3 securityManager.realms=$customRealm

```

2.3) 完成单元测试：

```

1 @Test
2     public void test01(){
3         //1.根据ini文件得到SecurityManagerFactory对象
4         IniSecurityManagerFactory managerFactory = new
IniSecurityManagerFactory("classpath:shiroCustom.ini");
5         //2.根据上面的managerFactory得到SecurityManager对象
6         SecurityManager securityManager = managerFactory.getInstance();
7         //3.将此对象设置到上下文工作环境中
8         SecurityUtils.setSecurityManager(securityManager);
9         //4.得到主体对象
10        Subject subject = SecurityUtils.getSubject();
11        //5.构造一个令牌对象
12        AuthenticationToken token = new UsernamePasswordToken("zhangsan","111111");
13        //6.进行认证
14        subject.login(token);
15        //7.检查认证是否通过
16        boolean authenticated = subject.isAuthenticated();
17        System.out.println("是否认证通过：" + authenticated);
18
19    }

```

2.4) 加盐单元测试：

```

1 //加盐测试
2 @Test
3 public void testAddSalt(){
4     String salt = "rbtwy"; //加的盐值
5     String oldPassword = "111111"; //原始密码
6     int hashIterations = 1; //代表加密次数
7     //加密算法一：
8     Md5Hash md5Hash = new Md5Hash(oldPassword,salt,hashIterations);
9     System.out.println(md5Hash);
10    //加密算法二：
11    SimpleHash simpleHash = new SimpleHash("md5",oldPassword,salt,hashIterations);
12    System.out.println(simpleHash.toString());
13 }

```

2.5) 自定义Realm加盐

2.5.1) 自定义加盐的realm类

```

1 /**
2  * @Author: Feng.Wang
3  * @Company: Zelin.ShenZhen
4  * @Description: 自定义Realm加盐
5  * @Date: Create in 2019/4/15 11:34
6  */
7 public class Md5CustomRealm extends AuthorizingRealm {
8     //授权
9     @Override
10    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)
11    {
12        return null;
13    }
14    //认证
15    @Override
16    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
17    throws AuthenticationException {
18        //1.得到身份信息
19        Object principal = token.getPrincipal();
20        //2.判断看是否为null
21        if(null == principal){
22            return null;
23        }
24        //3.模拟从数据库中取得密码及加盐值
25        String password = "ec1b86316c81b3f3440c07f65a74bf79"; //加密、加盐后的密码
26        String salt = "rbtwy"; //盐值
27        //4.返回SimpleAuthenticationInfo对象
28        return new SimpleAuthenticationInfo(principal,password,
29        ByteSource.Util.bytes(salt),"aaa");
30    }
31 }

```

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

```

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

```

认证小结：

```

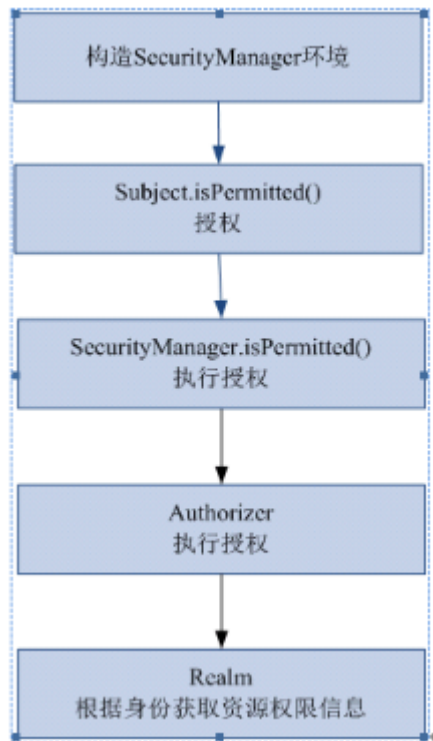
1  1. 认证步骤：
2  第一步：根据ini配置文件得到SecurityManager对象
3  第二步：由ModularRealmAuthenticator调用iniRealm，将token中的主体信息传递到ini文件中，进行比对，
   如果内容一致，就将得到的认证主体给ModularRealmAuthenticator，否则，抛出“unknown Account
   Exception”。
4  第三步：将返回得到的ini文件中的主体密码与token中输入的密码进行比对，如果不一致，就抛出“Incorrect
   credentials Exception”（无效凭证异常）
5
6  2. 小结：
7  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。
8  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：
9  ①如果iniRealm返回的是null，则抛出org.apache.shiro.authc.UnknownAccountException异常！
10 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：

```
1 [users]
2 zhangsan=111111,role1,role2
3 lisi=222222,role1
4 [roles]
5 role1=student:create,student:update,student:delete
6 role2=student:create
```

3.2) 单元测试

```
1 /**
2  * @Author: Feng.Wang
3  * @Company: Zelin.ShenZhen
4  * @Description:
5  * @Date: Create in 2019/4/15 11:58
6  */
7 public class TestPermission {
8     @Test
9     public void test01(){
10         //第一部分：进行认证操作
11         //1.根据ini文件得到SecurityManagerFactory对象
12         IniSecurityManagerFactory managerFactory = new
13         IniSecurityManagerFactory("classpath:shiroPermission.ini");
14         //2.根据上面的managerFactory得到SecurityManager对象
15         SecurityManager securityManager = managerFactory.getInstance();
16         //3.将此对象设置到上下文工作环境中
17         SecurityUtils.setSecurityManager(securityManager);
18         //4.得到主体对象
19         Subject subject = SecurityUtils.getSubject();
```

```

19 //5.构造一个令牌对象
20 AuthenticationToken token = new UsernamePasswordToken("zhangsan","111111");
21 //6.进行认证
22 subject.login(token);
23 //7.检查认证是否通过
24 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边第一部分认证通过之后才能执行）
28 //2.1)进行角色的授权操作
29 //2.1.1)单个角色的授权操作
30 boolean role1 = subject.hasRole("role1");
31 System.out.println("是否有role1角色：" + role1);
32 //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）
33 boolean hasAllRoles = subject.hasAllRoles(Arrays.asList("role1", "role3"));
34 System.out.println("是否有role1,role3两个角色：" + hasAllRoles);
35
36 //2.2)进行权限的授权操作
37 //2.2.1)判断单个权限
38 boolean permitted = subject.isPermitted("student:create");
39 System.out.println("是否有单个权限：" + permitted);
40 //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）
41 boolean permittedAll = subject.isPermittedAll("student:create",
42 "student:update", "student:query");
43 System.out.println("是否有多个权限：" + permittedAll);
44 }

```

4) 利用ini文件+自定义realm实现授权

4.1) shiroPermissionRealm.ini文件定义

```

1 [main]
2 customRealm=com.zelin.realm.CustomPermissionRealm
3 securityManager.realms=$customRealm

```

4.2) CustomPermissionRealm自定义realm类

```

1 /**
2  * @Author: Feng.Wang
3  * @Company: Zelin.Shenzhen
4  * @Description: 自定义授权realm
5  * @Date: Create in 2019/4/15 12:08
6  */
7 public class CustomPermissionRealm extends AuthorizingRealm {
8     //授权
9     @Override
10    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)
11    {
12        //1.得到身份信息
13        Object primaryPrincipal = principals.getPrimaryPrincipal();

```

```

13 //2.模拟从数据库中得到权限列表
14 List<String> permissions = new ArrayList<>();
15 permissions.add("student:create");
16 permissions.add("student:delete");
17 permissions.add("student:update");
18 //3.构造AuthorizationInfo对象
19 SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
20 //4.将权限列表对象与SimpleAuthorizationInfo对象进行绑定
21 authorizationInfo.addStringPermissions(permissions);
22 return authorizationInfo;
23 }
24 //认证
25 @Override
26 protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
    throws AuthenticationException {
27     //1.得到认证身份对象
28     Object principal = token.getPrincipal();
29     if(principal == null) return null;
30     //2.从数据库中得到密码
31     String password = "111111";
32     return new SimpleAuthenticationInfo(principal,password,"aaa");
33 }
34 }
35

```

4.3) 单元测试：

```

1 /**
2  * @Author: Feng.Wang
3  * @Company: Zelin.ShenZhen
4  * @Description:
5  * @Date: Create in 2019/4/15 11:58
6  */
7 public class TestCustomRealmPermission {
8     @Test
9     public void test01(){
10         //第一部分：进行认证操作
11         //1.根据ini文件得到SecurityManagerFactory对象
12         IniSecurityManagerFactory managerFactory = new
        IniSecurityManagerFactory("classpath:shiroPermissionRealm.ini");
13         //2.根据上面的managerFactory得到SecurityManager对象
14         SecurityManager securityManager = managerFactory.getInstance();
15         //3.将此对象设置到上下文工作环境中
16         SecurityUtils.setSecurityManager(securityManager);
17         //4.得到主体对象
18         Subject subject = SecurityUtils.getSubject();
19         //5.构造一个令牌对象
20         AuthenticationToken token = new UsernamePasswordToken("zhangsan","111111");
21         //6.进行认证
22         subject.login(token);
23         //7.检查认证是否通过
24         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边第一部分认证通过之后才能执行）
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",
34     "student:update", "student:query");
35     System.out.println("是否有多个权限：" + permittedAll);
36
37     //1.2)使用checkPermission检查权限，找不到抛出异常
38     //1.2.1)检查单个权限(如果没有此权限，则抛出：
39     org.apache.shiro.authz.UnauthorizedException: Subject does not have permission
40     [student:query])
41     subject.checkPermissions("student:create");
42     //1.2.2)检查多个权限(只要有一个没有，则抛出：
43     org.apache.shiro.authz.UnauthorizedException: Subject does not have permission
44     [student:query])
45     subject.checkPermissions("student:update", "student:delete", "student:query");
46
47 }
48 }

```

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