# Shiro

······ *鑫合汇Java组整理*

## Shiro简介

一． Shiro是一个java安全框架，可以帮助我们完成:认证、授权、加密、会话管理等功能。

功能图：



1. **Authentication：**身份认证/登录，验证用户是不是拥有相应的身份；
2. **Authorization：**授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情，常见的如：验证某个用户是否拥有某个角色。或者细粒度的验证某个用户对某个资源是否具有某个权限；
3. **Session Manager：**会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通JavaSE环境的，也可以是如Web环境的；
4. **Cryptography：**加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；
5. **Web Support：**Web支持，可以非常容易的集成到Web环境；
6. Caching：缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；
7. **Concurrency：**shiro支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；
8. **Testing：**提供测试支持；
9. **Run As：**允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；
10. **Remember Me：**记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录了。

**Shiro不会去维护用户、维护权限；这些需要我们自己去设计/提供；然后通过相应的接口注入给Shiro即可。**

**=>从外部的角度看Shiro:**



可以看到：应用代码直接交互的对象是Subject，也就是说Shiro的对外API核心就是Subject；其每个API的含义

S**ubject：**主体，代表了当前“用户”，这个用户不一定是一个具体的人，与当前应用交互的任何东西都是Subject，如网络爬虫，机器人等；即一个抽象概念；所有Subject都绑定到SecurityManager，与Subject的所有交互都会委托给SecurityManager；可以把Subject认为是一个门面；SecurityManager才是实际的执行者；

**SecurityManager：**安全管理器；即所有与安全有关的操作都会与SecurityManager交互；且它管理着所有Subject；可以看出它是Shiro的核心，它负责与后边介绍的其他组件进行交互，如果学习过SpringMVC，你可以把它看成DispatcherServlet前端控制器；

**Realm：**域，Shiro从从Realm获取安全数据（如用户、角色、权限），就是说SecurityManager要验证用户身份，那么它需要从Realm获取相应的用户进行比较以确定用户身份是否合法；也需要从Realm得到用户相应的角色/权限进行验证用户是否能进行操作；可以把Realm看成DataSource，即安全数据源。

也就是说对于我们而言，最简单的一个Shiro应用：

1、应用代码通过Subject来进行认证和授权，而Subject又委托给SecurityManager；

2、我们需要给Shiro的SecurityManager注入Realm，从而让SecurityManager能得到合法的用户及其权限进行判断。

从以上也可以看出，Shiro不提供维护用户/权限，而是通过Realm让开发人员自己注入。

=》从内部架构图看：



## 二． Shiro身份认证

### 1. 简单认证

身份验证：即在应用中谁能证明他就是他本人。一般提供如他们的身份ID一些标识信 息来表明他就是他本人，如提供身份证，用户名/密码来证明。

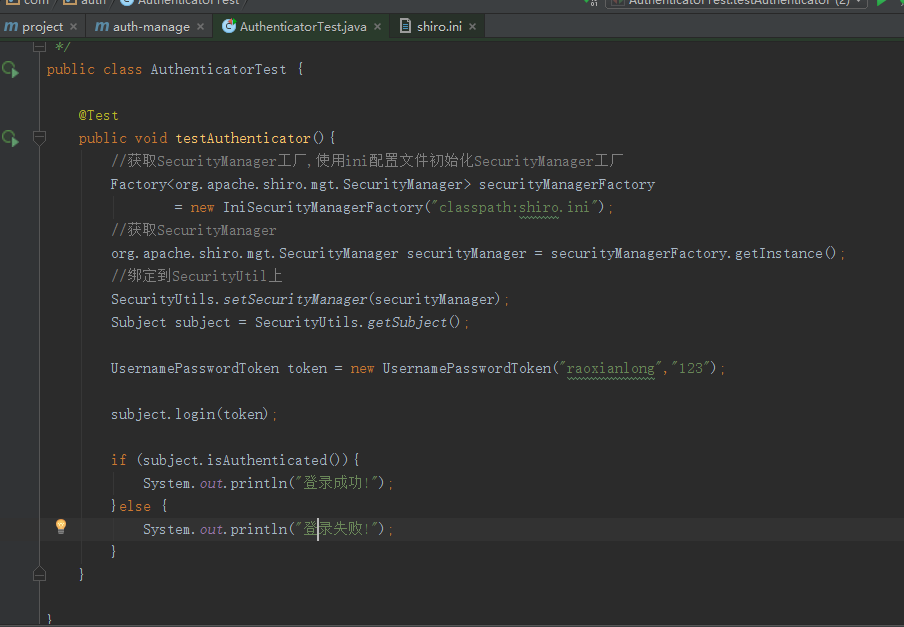
在shiro中，用户需要提供principals （身份）和credentials（证明）给shiro，从而应用能验证用户身份：

principals：身份，即主体的标识属性，可以是任何东西，如用户名、邮箱等，唯一即可。一个主体可以有多个principals，但只有一个Primary principals，一般是用户名/密码/手机号。

credentials：证明/凭证，即只有主体知道的安全值，如密码/数字证书等。

最常见的principals和credentials组合就是用户名/密码了。接下来先进行一个基本的身份认证。

=>简单登录认证代码：

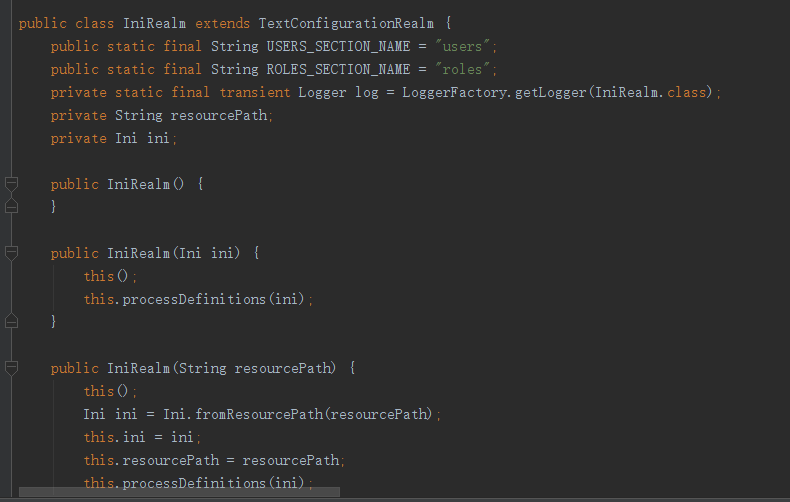


**总结出登录认证的步骤:**

1. 收集用户身份/凭证，即如用户名/密码；
2. 调用Subject.login进行登录，如果失败将得到相应的AuthenticationException异常，根据异常提示用户错误信息；否则登录成功；

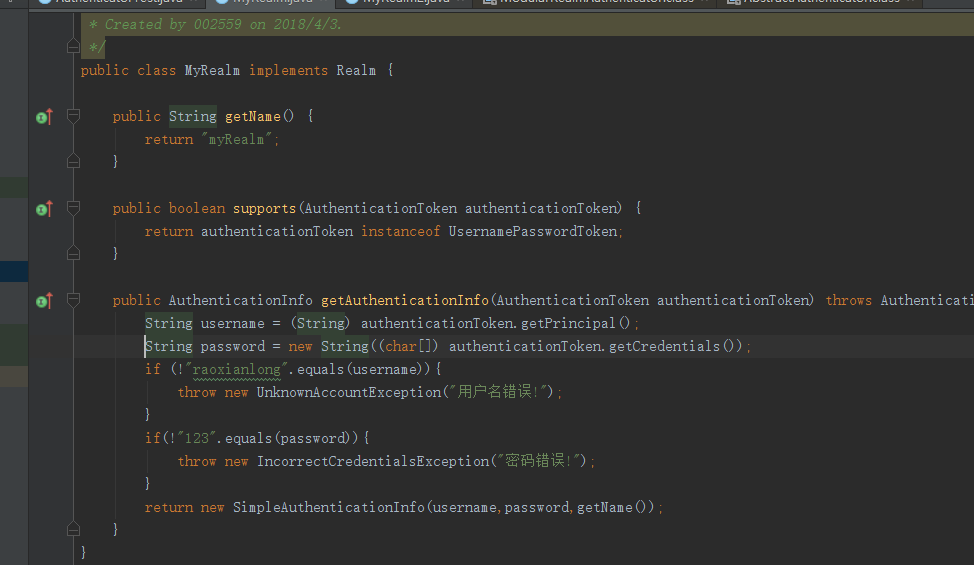
3、最后调用Subject.logout进行退出操作

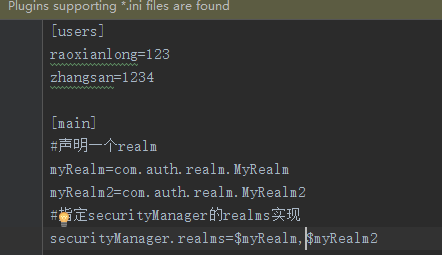
上面有说到realm，其实上面代码用到shiro自带的IniRealm:



它会读取shiro.ini文件，解析出用户、角色、权限等数据

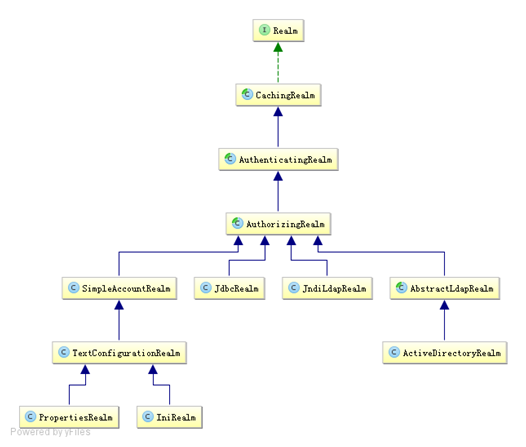
### 2. 自定义realm





配置realm

### 3. Shiro默认提供的realm



一般继承AuthorizingRealm（授权）即可；其继承了AuthenticatingRealm（即身份验证），而且也间接继承了CachingRealm（带有缓存实现）。其中主要默认实现如下：

**org.apache.shiro.realm.text.IniRealm**：[users]部分指定用户名/密码及其角色；[roles]部分指定角色即权限信息；

**org.apache.shiro.realm.text.PropertiesRealm**： user.username=password,role1,role2指定用户名/密码及其角色；role.role1=permission1,permission2指定角色及权限信息；

**org.apache.shiro.realm.jdbc.JdbcRealm**：通过sql查询相应的信息，如“select password from users where username = ?”获取用户密码，“select password, password\_salt from users where username = ?”获取用户密码及盐；“select role\_name from user\_roles where username = ?”获取用户角色；“select permission from roles\_permissions where role\_name = ?”获取角色对应的权限信息；也可以调用相应的api进行自定义sql；

### 4. 认证器和认证策越(Authenticator及AuthenticationStrategy)

Authenticator的职责是验证用户帐号，是Shiro API中身份验证核心的入口点：

1. **public** AuthenticationInfo authenticate(AuthenticationToken authenticationToken)
2. **throws** AuthenticationException;

如果验证成功，将返回AuthenticationInfo验证信息；此信息中包含了身份及凭证；如果验证失败将抛出相应的AuthenticationException实现。

SecurityManager接口继承了Authenticator，另外还有一个ModularRealmAuthenticator实现，其委托给多个Realm进行验证，验证规则通过AuthenticationStrategy接口指定，默认提供的实现：

FirstSuccessfulStrategy：只要有一个Realm验证成功即可，只返回第一个Realm身份验证成功的认证信息，其他的忽略；

AtLeastOneSuccessfulStrategy：只要有一个Realm验证成功即可，和FirstSuccessfulStrategy不同，返回所有Realm身份验证成功的认证信息；

AllSuccessfulStrategy：所有Realm验证成功才算成功，且返回所有Realm身份验证成功的认证信息，如果有一个失败就失败了。

ModularRealmAuthenticator默认使用AtLeastOneSuccessfulStrategy策略。

## 三． Shiro授权

### 1. 授权的几个概念

#### **1.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）。

#### 2.主体

主体，即访问应用的用户，在Shiro中使用Subject代表该用户。用户只有授权后才允许访问相应的资源。

#### 3.资源

在应用中用户可以访问的任何东西，比如访问JSP页面、查看/编辑某些数据、访问某个业务方法、打印文本等等都是资源。用户只要授权后才能访问。

权限

安全策略中的原子授权单位，通过权限我们可以表示在应用中用户有没有操作某个资源的权力。即权限表示在应用中用户能不能访问某个资源，如：

访问用户列表页面

查看/新增/修改/删除用户数据（即很多时候都是CRUD（增查改删）式权限控制）

打印文档等等。。。

如上可以看出，权限代表了用户有没有操作某个资源的权利，即反映在某个资源上的操作允不允许，不反映谁去执行这个操作。所以后续还需要把权限赋予给用户，即定义哪个用户允许在某个资源上做什么操作（权限），Shiro不会去做这件事情，而是由实现人员提供。

Shiro支持粗粒度权限（如用户模块的所有权限）和细粒度权限（操作某个用户的权限，即实例级别的），后续部分介绍。

#### 4.角色

角色代表了操作集合，可以理解为权限的集合，一般情况下我们会赋予用户角色而不是权限，即这样用户可以拥有一组权限，赋予权限时比较方便。典型的如：项目经理、技术总监、CTO、开发工程师等都是角色，不同的角色拥有一组不同的权限。

#### 5.隐式角色

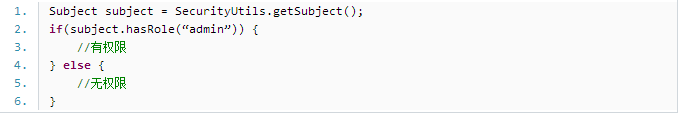
即直接通过角色来验证用户有没有操作权限，如在应用中CTO、技术总监、开发工程师可以使用打印机，假设某天不允许开发工程师使用打印机，此时需要从应用中删除相应代码；再如在应用中CTO、技术总监可以查看用户、查看权限；突然有一天不允许技术总监查看用户、查看权限了，需要在相关代码中把技术总监角色从判断逻辑中删除掉；即粒度是以角色为单位进行访问控制的，粒度较粗；如果进行修改可能造成多处代码修改。

#### 6.显示角色

在程序中通过权限控制谁能访问某个资源，角色聚合一组权限集合；这样假设哪个角色不能访问某个资源，只需要从角色代表的权限集合中移除即可；无须修改多处代码；即粒度是以资源/实例为单位的；粒度较细。

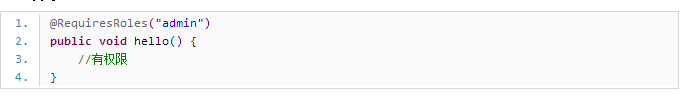
### Shiro的授权方式

#### 编程式



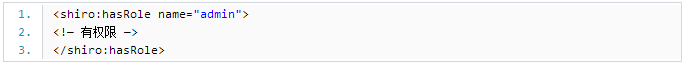
通过if/else判断有无权限。

#### 注解式



没有权限则抛出异常

#### JSP标签

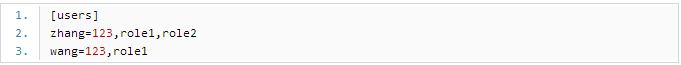


有权限则会显示标签里的内容

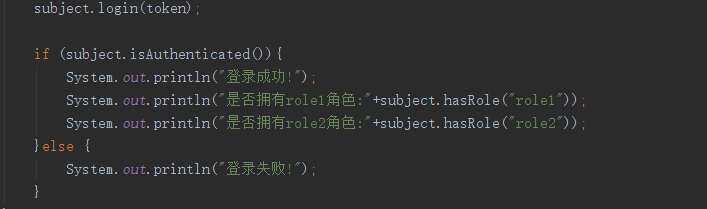
### 授权

#### 基于角色的访问控制(隐式角色)

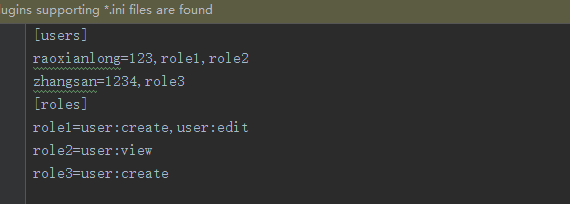
在ini配置文件配置用户拥有的角色（shiro-role.ini）



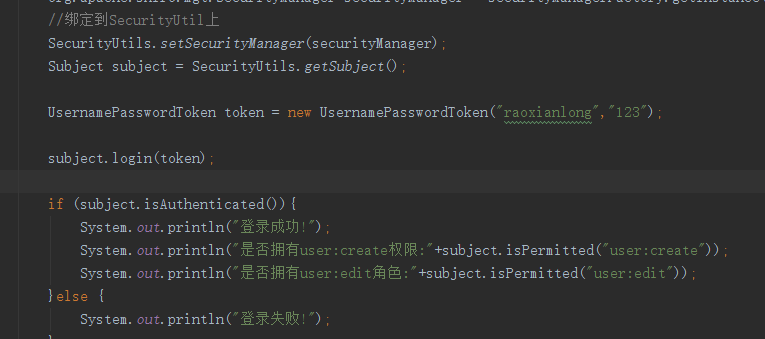
**规则即：“用户名=密码,角色1，角色2”，如果需要在应用中判断用户是否有相应角色，就需要在相应的Realm中返回角色信息，也就是说Shiro不负责维护用户-角色信息，需要应用提供，Shiro只是提供相应的接口方便验证**



#### 基于资源的访问控制（显示角色）



先定义角色下面的权限.然后给用户赋予角色



判断是否有某个权限

#### 字符串通配符权限

**规则：“资源标识符：操作：对象实例ID”  即对哪个资源的哪个实例可以进行什么操作。其默认支持通配符权限字符串，“:”表示资源/操作/实例的分割；“,”表示操作的分割；“\*”表示任意资源/操作/实例。**

##### 单个资源多个权限



判断代码：



##### 单个资源单个权限

判断代码:



##### ****3. 单个资源全部权限****



判断代码:



或



判断代码:



##### 4. Shiro对权限字符串缺失部分的处理

如“user:view”等价于“user:view:\*”；而“organization”等价于“organization:\*”或者“organization:\*:\*”。可以这么理解，这种方式实现了前缀匹配。

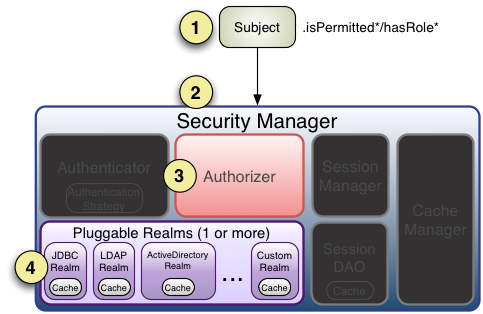
另外如“user:\*”可以匹配如“user:delete”、“user:delete”可以匹配如“user:delete:1”、“user:\*:1”可以匹配如“user:view:1”、“user”可以匹配“user:view”或“user:view:1”等。即\*可以匹配所有，不加\*可以进行前缀匹配；但是如“\*:view”不能匹配“system:user:view”，需要使用“\*:\*:view”，即后缀匹配必须指定前缀（多个冒号就需要多个\*来匹配）。

##### 5. 性能问题

通配符匹配方式比字符串相等匹配来说是更复杂的，因此需要花费更长时间，但是一般系统的权限不会太多，且可以配合缓存来提供其性能，如果这样性能还达不到要求我们可以实现位操作算法实现性能更好的权限匹配。另外实例级别的权限验证如果数据量太大也不建议使用，可能造成查询权限及匹配变慢。可以考虑比如在sql查询时加上权限字符串之类的方式在查询时就完成了权限匹配。

#### 授权流程

##### 普通流程



1、首先调用Subject.isPermitted\*/hasRole\*接口，其会委托给SecurityManager，而SecurityManager接着会委托给Authorizer；

2、Authorizer是真正的授权者，如果我们调用如isPermitted(“user:view”)，其首先会通过PermissionResolver把字符串转换成相应的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、Authorizer会判断Realm的角色/权限是否和传入的匹配，如果有多个Realm，会委托给ModularRealmAuthorizer进行循环判断，如果匹配如isPermitted\*/hasRole\*会返回true，否则返回false表示授权失败。

##### 2. ModularRealmAuthorizer进行多Realm匹配流程：

1、首先检查相应的Realm是否实现了实现了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。

##### 3. 如果Realm进行授权的话，应该继承AuthorizingRealm，其流程是：

1.、如果调用hasRole\*，则直接获取AuthorizationInfo.getRoles()与传入的角色比较即可；

2、首先如果调用如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；

3、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）；

4、接着调用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。

##### 4 .Authorizer、PermissionResolver及RolePermissionResolver

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。

**更改Authorizer实现:**



对于ModularRealmAuthorizer，相应的AuthorizingSecurityManager会在初始化完成后自动将相应的realm设置进去，我们也可以通过调用其setRealms()方法进行设置。

设置ModularRealmAuthorizer的permissionResolver，其会自动设置到相应的Realm上（其实现了PermissionResolverAware接口）



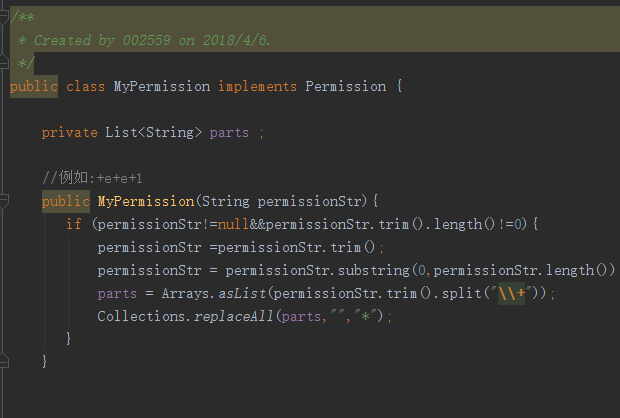
设置ModularRealmAuthorizer的rolePermissionResolver，其会自动设置到相应的Realm上（其实现了RolePermissionResolverAware接口）

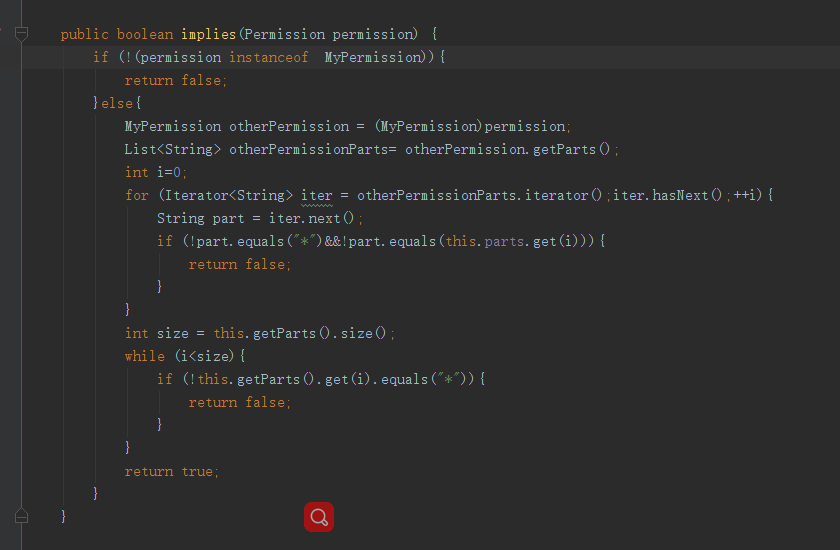


**实例:**

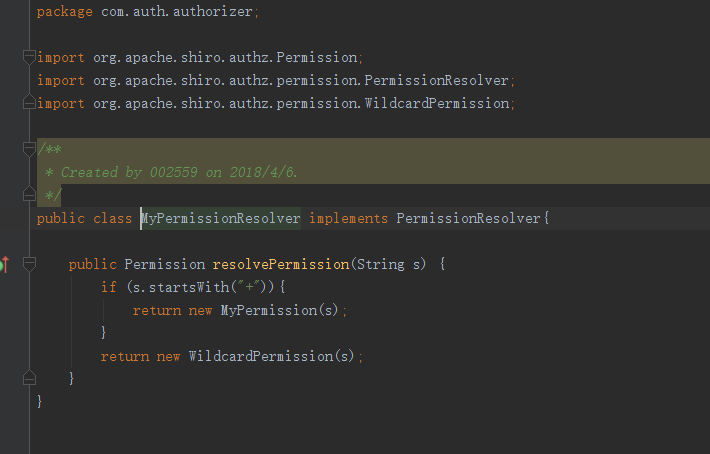
**需求：我们的权限字符串定义为:+资源字符串+权限位+实例ID**

1. **首先实现Permission**

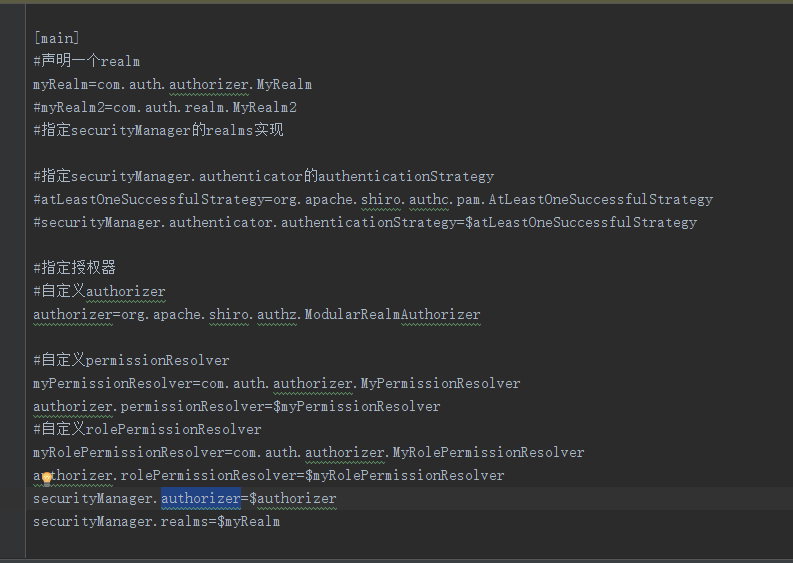




1. **实现permissionResolver解析器**



1. **配置ini**



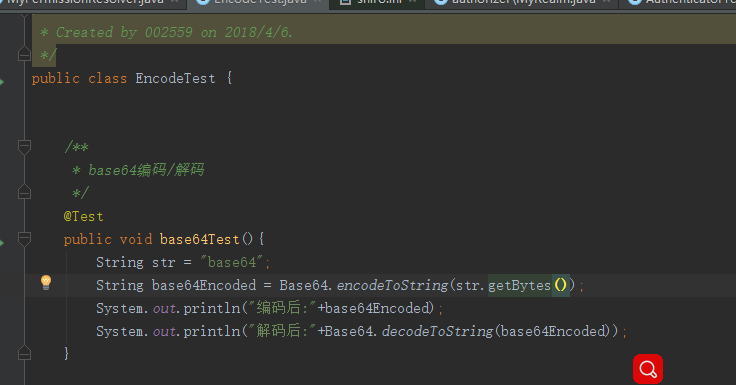
## 四 . Shiro编码/加密

在涉及到密码存储问题上，应该加密/生成密码摘要存储，而不是存储明文密码。比如之前的600w csdn账号泄露对用户可能造成很大损失，因此应加密/生成不可逆的摘要方式存储。

### 编码/解码

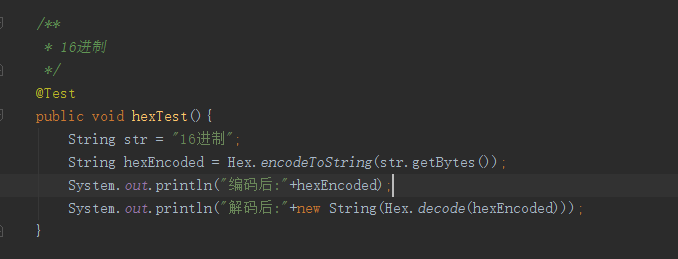
Shiro提供了base64和16进制字符串编码/解码的API支持，方便一些编码解码操作。Shiro内部的一些数据的存储/表示都使用了base64和16进制字符串。

#### Base64





#### 十六进制

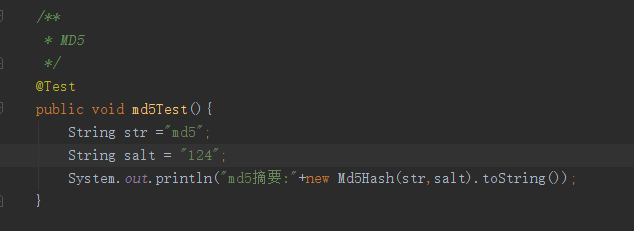




### 散列算法

散列算法一般用于生成数据的摘要信息，是一种不可逆的算法，一般适合存储密码之类的数据，常见的散列算法如MD5、SHA等。一般进行散列时最好提供一个salt（盐），比如加密密码“admin”，产生的散列值是“21232f297a57a5a743894a0e4a801fc3”，可以到一些md5解密网站很容易的通过散列值得到密码“admin”，即如果直接对密码进行散列相对来说破解更容易，此时我们可以加一些只有系统知道的干扰数据，如用户名和ID（即盐）；这样散列的对象是“密码+用户名+ID”，这样生成的散列值相对来说更难破解。

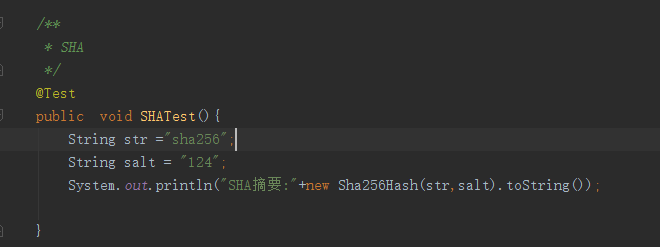
#### MD5





通过盐“123”MD5散列“md5”。另外散列时还可以指定散列次数，如2次表示：md5(md5(str))：“new Md5Hash(str, salt, 2).toString()”。

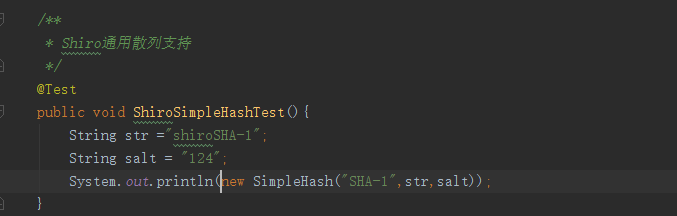
#### SHA





使用SHA256算法生成相应的散列数据，另外还有如SHA1、SHA512算法。

#### Shiro通用散列支持

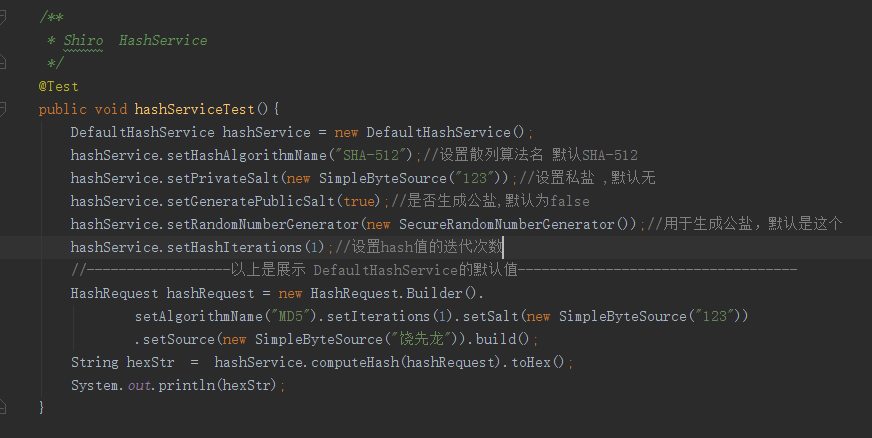




通过调用SimpleHash时指定散列算法，其内部使用了Java的MessageDigest实现。

#### Shiro提供的HashService

为了方便使用，Shiro提供了HashService，默认提供了DefaultHashService实现。





**步骤:**

1、首先创建一个DefaultHashService，默认使用SHA-512算法；

2、可以通过hashAlgorithmName属性修改算法；

3、可以通过privateSalt设置一个私盐，其在散列时自动与用户传入的公盐混合产生一个新盐；

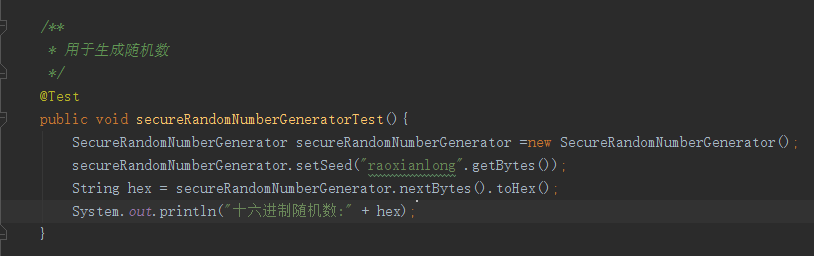
4、可以通过generatePublicSalt属性在用户没有传入公盐的情况下是否生成公盐；

5、可以设置randomNumberGenerator用于生成公盐；

6、可以设置hashIterations属性来修改默认加密迭代次数；

7、需要构建一个HashRequest，传入算法、数据、公盐、迭代次数。

SecureRandomNumberGenerator用于生成一个随机数：

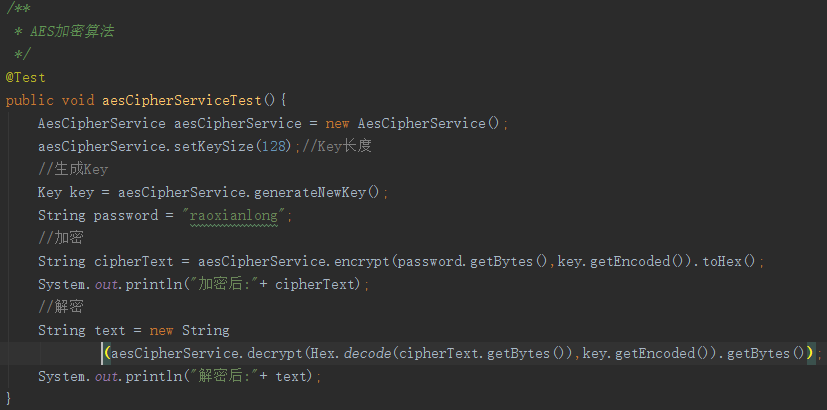




### 加密/解密

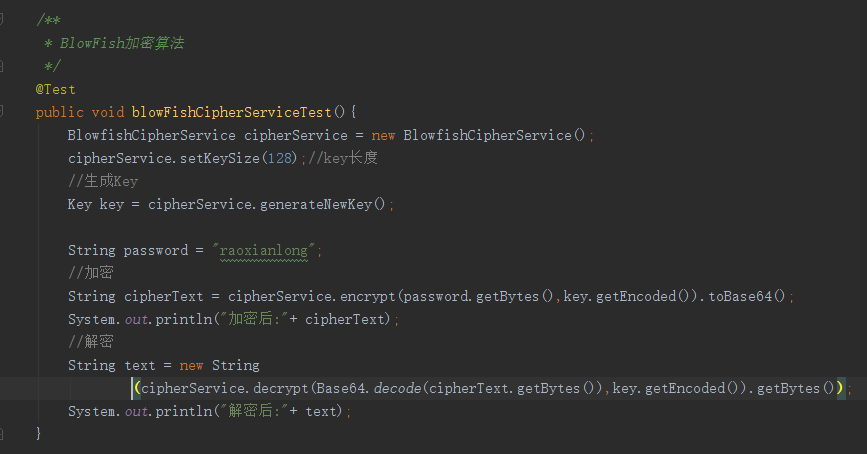
Shiro还提供对称式加密/解密算法的支持，如AES、Blowfish等；当前还没有提供对非对称加密/解密算法支持，未来版本可能提供。

#### AES算法





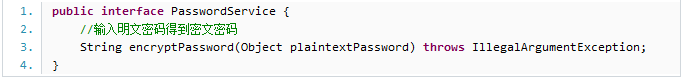
#### Blowfish算法

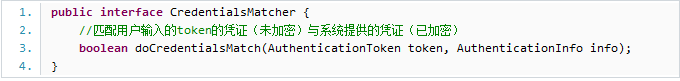




#### Shiro提供的PasswordService/CredentialsMatcher

Shiro提供了PasswordService及CredentialsMatcher用于提供加密密码及验证密码服务。

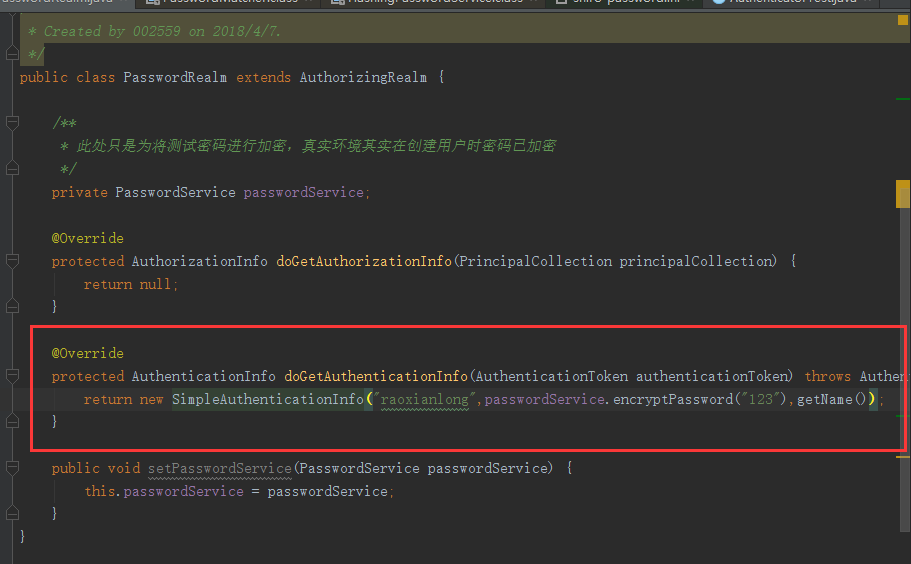




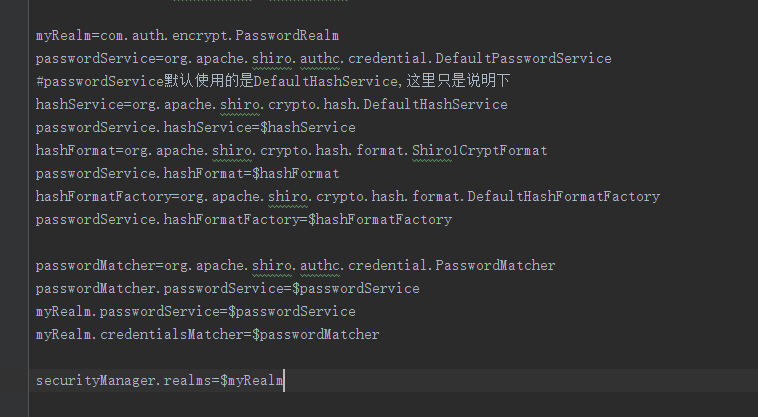
Shiro默认提供了PasswordService实现DefaultPasswordService；CredentialsMatcher实现PasswordMatcher及HashedCredentialsMatcher（更强大）。

##### DefaultPasswordService配合PasswordMatcher实现简单的密码加密与验证服务

Realm:

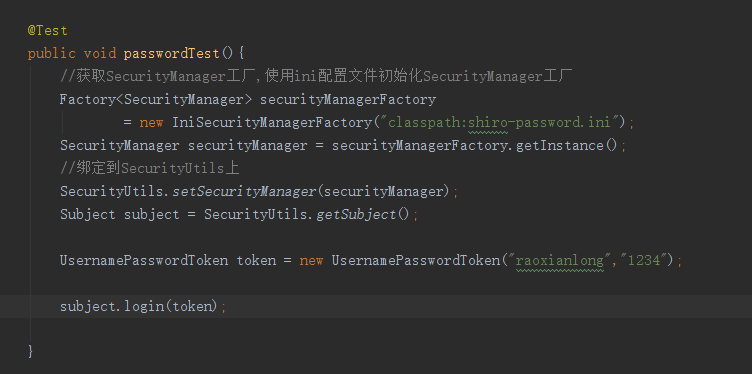


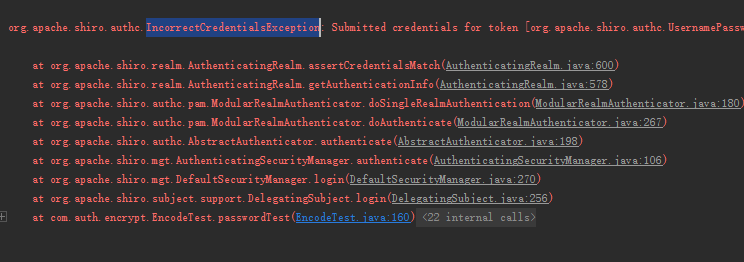
Ini配置:



结果:

如果用户名密码不匹配,则会抛出IncorrectCredentialsException异常





步骤:

1、passwordService使用DefaultPasswordService，如果有必要也可以自定义；

2、hashService定义散列密码使用的HashService，默认使用DefaultHashService（默认SHA-256算法）；

3、hashFormat用于对散列出的值进行格式化，默认使用Shiro1CryptFormat，另外提供了Base64Format和HexFormat，对于有salt的密码请自定义实现ParsableHashFormat然后把salt格式化到散列值中；

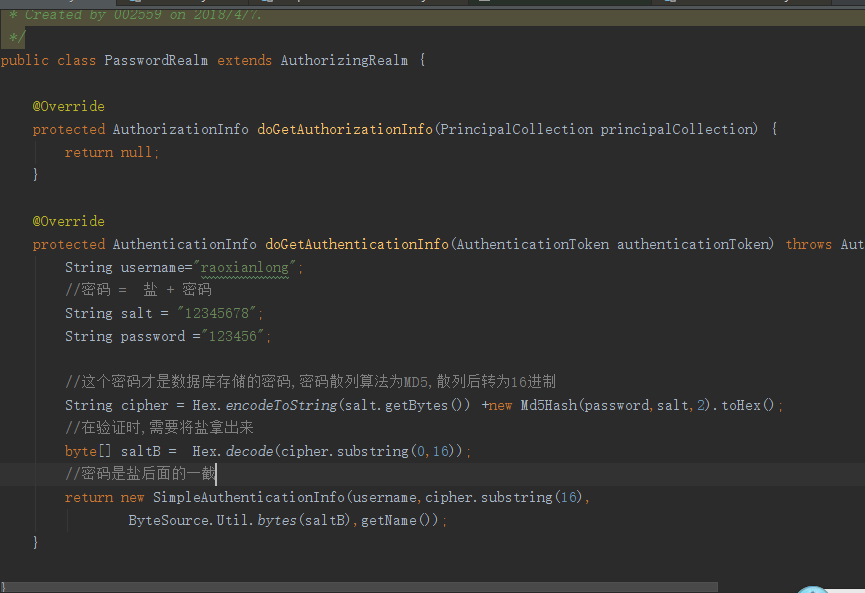
4、hashFormatFactory用于根据散列值得到散列的密码和salt；因为如果使用如SHA算法，那么会生成一个salt，此salt需要保存到散列后的值中以便之后与传入的密码比较时使用；默认使用DefaultHashFormatFactory；

5、passwordMatcher使用PasswordMatcher，其是一个CredentialsMatcher实现；

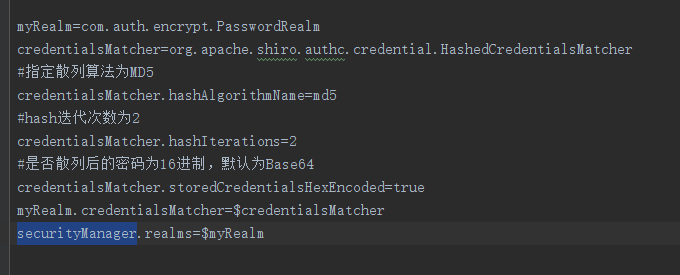
6、将credentialsMatcher赋值给myRealm，myRealm间接继承了AuthenticatingRealm，其在调用getAuthenticationInfo方法获取到AuthenticationInfo信息后，会使用credentialsMatcher来验证凭据是否匹配，如果不匹配将抛出IncorrectCredentialsException异常。

##### HashedCredentialsMatcher实现密码验证服务

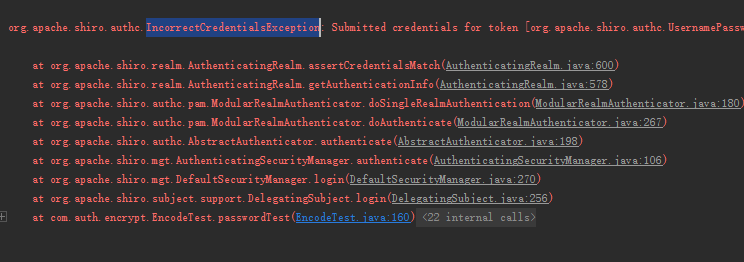
Realm：



Ini配置：



结果:如果密码不对也会报错



步骤:

1、通过credentialsMatcher.hashAlgorithmName=md5指定散列算法为md5，需要和生成密码时的一样；

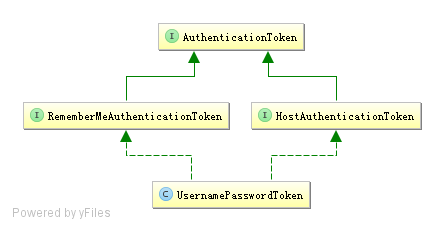
2、credentialsMatcher.hashIterations=2，散列迭代次数，需要和生成密码时的意义；

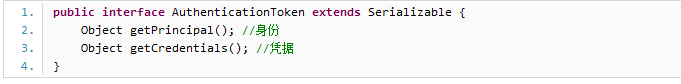
3、credentialsMatcher.storedCredentialsHexEncoded=true表示是否存储散列后的密码为16进制，需要和生成密码时的一样，默认是base64；

此处最需要注意的就是HashedCredentialsMatcher的算法需要和生成密码时的算法一样。另外HashedCredentialsMatcher会自动根据AuthenticationInfo的类型是否是SaltedAuthenticationInfo来获取credentialsSalt盐。

## 五 . Shiro相关对象

### AuthenticationToken



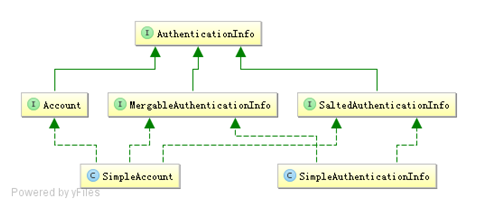
AuthenticationToken用于收集用户提交的身份（如用户名）及凭据（如密码）：

扩展接口RememberMeAuthenticationToken：提供了“boolean isRememberMe()”现“记住我”的功能；

扩展接口是HostAuthenticationToken：提供了“String getHost()”方法用于获取用户“主机”的功能。

Shiro提供了一个直接拿来用的UsernamePasswordToken，用于实现用户名/密码Token组，另外其实现了RememberMeAuthenticationToken和HostAuthenticationToken，可以实现记住我及主机验证的支持。

### AuthenticationInfo



AuthenticationInfo有两个作用：

1、如果Realm是AuthenticatingRealm子类，则提供给AuthenticatingRealm内部使用的CredentialsMatcher进行凭据验证；（如果没有继承它需要在自己的Realm中自己实现验证）；

2、提供给SecurityManager来创建Subject（提供身份信息）；

MergableAuthenticationInfo：

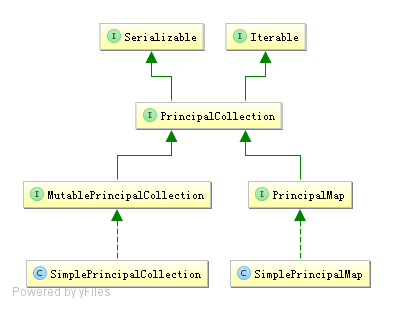
用于提供在多Realm时合并AuthenticationInfo的功能，主要合并Principal、如果是其他的如credentialsSalt，会用后边的信息覆盖前边的。

比如HashedCredentialsMatcher，在验证时会判断AuthenticationInfo是否是SaltedAuthenticationInfo子类，来获取盐信息。

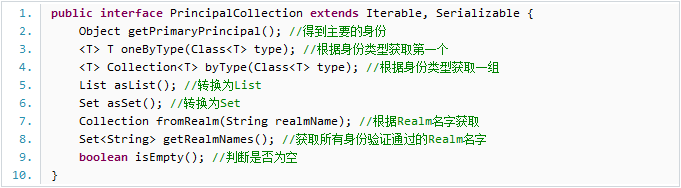
Account：

相当于我们之前的User，SimpleAccount是其一个实现；在IniRealm、PropertiesRealm这种静态创建帐号信息的场景中使用，这些Realm直接继承了SimpleAccountRealm，而SimpleAccountRealm提供了相关的API来动态维护SimpleAccount；即可以通过这些API来动态增删改查SimpleAccount；动态增删改查角色/权限信息。及如果您的帐号不是特别多，可以使用这种方式，

### PrincipalCollection



因为我们可以在Shiro中同时配置多个Realm，所以呢身份信息可能就有多个；因此其提供了PrincipalCollection用于聚合这些身份信息：



因为PrincipalCollection聚合了多个，此处最需要注意的是getPrimaryPrincipal，如果只有一个Principal那么直接返回即可，如果有多个Principal，则返回第一个（因为内部使用Map存储，所以可以认为是返回任意一个）；oneByType / byType根据凭据的类型返回相应的Principal；fromRealm根据Realm名字（每个Principal都与一个Realm关联）获取相应的Principal。

## 六． Shiro与web集成

Shiro提供了与Web集成的支持，其通过一个ShiroFilter入口来拦截需要安全控制的URL，然后进行相应的控制，ShiroFilter类似于如Strut2/SpringMVC这种web框架的前端控制器，其是安全控制的入口点，其负责读取配置（如ini配置文件），然后判断URL是否需要登录/权限等工作。