代码示例https://github.com/lilewa/shiro-example

**Shiro和数据源的桥梁——Realm**

Realm 是一个可以访问特定于应用程序的安全相关数据（如用户、角色和权限）的组件。Realm 将此特定于应用程序的数据转换为 Shiro 所理解的格式, 因此 Shiro 可以提供一个易于理解的编程 API, 而不管存在多少数据源, 也无论您的数据可能有多特定于应用程序。

Realms通常与数据源 (如关系数据库、LDAP 目录、文件系统或其他类似资源) 是1对1的关系。因此, Realm 接口的实现使用特定于数据源的 Api 来使用安全相关数据 (角色、权限等), 数据源api可以是 JDBC、File IO、Hibernate或 JPA 或任何其他数据访问 API。

**Realm配置**

**INI文件中的显示定义**

[main]

fooRealm = com.company.foo.Realm

barRealm = com.company.another.Realm

bazRealm = com.company.baz.Realm

securityManager.realms = $fooRealm, $barRealm, $bazRealm

**隐式定义**

[main]

blahRealm = com.company.blah.Realm

fooRealm = com.company.foo.Realm

barRealm = com.company.another.Realm

在身份验证期间和realm交互的是Authenticator，身份验证流程在上一篇“3-Authentication”中有介绍。

1. 在Realm 执行身份验证尝试之前, 它的supports方法先被调用。此方法通常情况下, 将检查收集到的用户信息(用户信息是一个继承AuthenticationToken的类), 以确定它是否可以处理它
2. 如果Realm可以处理提交的AuthenticationToken，Authenticator会调用Realm的[getAuthenticationInfo(token)](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/realm/Realm.html" \l "getAuthenticationInfo-org.apache.shiro.authc.AuthenticationToken-) 方法
   1. 检查token中的提供的principal和credentials
   2. 根据principal, 查找数据源中的相应帐户数据
   3. 确保token提供的credentials与存储在数据存储区中的credentials相匹配
   4. 如果credentials匹配, 则返回一个AuthenticationInfo实例, 该实例以 Shiro 所理解的格式封装帐户数据

直接实现 "Realm" 接口可能会耗费大量时间, 而且容易出错。大多数人选择实现AuthorizingRealm抽象类, 而不是从头开始。

AuthorizingRealm抽象类中的[getAuthenticationInfo(token)](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/realm/Realm.html#getAuthenticationInfo-org.apache.shiro.authc.AuthenticationToken-) 方法是一个final方法，它为以上步骤提供了固定模板

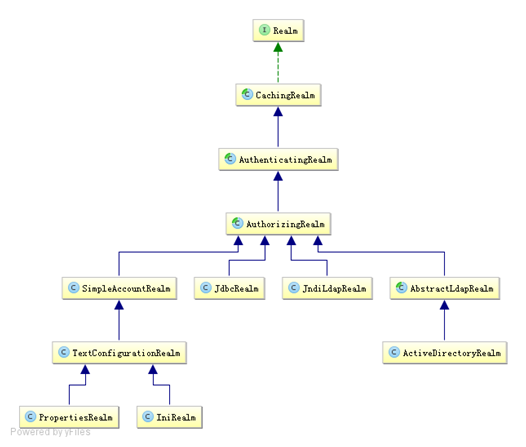
public final AuthenticationInfo getAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {  
  
 AuthenticationInfo info = getCachedAuthenticationInfo(token);//是否已经缓存  
 if (info == null) {  
 //根据用户信息在数据源中查找:  
 info = doGetAuthenticationInfo(token);  
 log.debug("Looked up AuthenticationInfo [{}] from doGetAuthenticationInfo", info);  
 if (token != null && info != null) {  
 cacheAuthenticationInfoIfPossible(token, info);  
 }  
 } else {  
 log.debug("Using cached authentication info [{}] to perform credentials matching.", info);  
 }  
  
 if (info != null) {

//匹配用户提供的credentials与存储在数据存储区中的credentials  
 assertCredentialsMatch(token, info);

} else {  
 log.debug("No AuthenticationInfo found for submitted AuthenticationToken [{}]. Returning null.", token);  
 }  
  
 return info;  
 }

开发者可以重写doGetAuthenticationInfo(token)从特定数据源得到相应帐户数据。提供自定义的**credentialsMatcher**覆盖默认的匹配方法

以下是shiro提供的Realm实现



org.apache.shiro.realm.text.IniRealm：IniRealm查找Ini 配置中的[user]section和[role]section。当在Ini 配置中定义非空 [user]section或[role]section 将自动触发 org.apache.shiro.realm.text.IniRealm 实例的创建,这个实例在[main]section下可通过"iniRealm" 名称引用和配置

org.apache.shiro.realm.text.PropertiesRealm：它的所有逻辑都与父类相关, 但除了父类的字符串配置外, 还启用了基于属性的配置。每行的 key/value对表示用户到角色的映射或角色到权限映射。

前缀必须是**user.**

**user.**username = password,role1,role2,...//格式

user.root = rootPassword,administrator  
user.jsmith = jsmithPassword,manager,engineer,employee  
user.abrown = abrownPassword,qa,employee  
user.djones = djonesPassword,qa,contractor

前缀必须是**role.**

**role.**rolename = permissionDefinition1, permissionDefinition2, ... //格式

org.apache.shiro.realm.jdbc.JdbcRealm：通过 JDBC 调用进行身份验证和授权的Realm。使用默认的schema和sql语句查询用于检索用户的密码进行身份验证, 并查询用户的角色和权限。通过设置realm的查询属性, 可以重写默认查询。

**默认的查询**

**dataSource.** authenticationQuery=select password from users where username = ?

//使用了盐的查询

**dataSource.** authenticationQuery= select password, password\_salt from users where username = ?

**dataSource** .userRolesQuery=select role\_name from user\_roles where username = ?

**dataSource** .authenticationQuery=select permission from roles\_permissions where role\_name = ?

确保token提供的credentials与存储在数据存储区中的凭据相匹配，这个功能是由realm负责。由于不同程序使用credentials的格式和数据类型不同，需要实现的匹配规则不同，realm使用[CredentialsMatcher](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authc/credential/CredentialsMatcher.html)实现匹配判断，

Shiro提供的[CredentialsMatcher](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authc/credential/CredentialsMatcher.html)实现有[SimpleCredentialsMatcher](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authc/credential/SimpleCredentialsMatcher.html) 和[HashedCredentialsMatcher](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authc/credential/HashedCredentialsMatcher.html)

Shiro默认使用[SimpleCredentialsMatcher](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authc/credential/SimpleCredentialsMatcher.html)

可以使用自己的[CredentialsMatcher](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authc/credential/CredentialsMatcher.html)实现来满足不同需求

代码实现

Realm myRealm = new com.company.shiro.realm.MyRealm();

CredentialsMatcher customMatcher = new com.company.shiro.realm.CustomCredentialsMatcher();

myRealm.setCredentialsMatcher(customMatcher);

ini配置

[main]

customMatcher = com.company.shiro.realm.CustomCredentialsMatcher

myRealm = com.company.shiro.realm.MyRealm

myRealm.credentialsMatcher = $customMatcher

SimpleCredentialsMatcher使用相等比较。它支持常见的字节源, 如字符串、字符数组、字节数组、文件和输入流

**Hashedcreditalmatcher**

shiro提供了多个 Hashedcreditalmatcher 子类实现，用于使用了hash算法的credentials的匹配。在realm上使用Hashedcreditalmatcher时, 必须将Hashedcreditalmatcher的配置设置成和哈希用户凭据的哈希算法一致，例如

创建用户的java代码

//使用随机数发生器来生成盐

RandomNumberGenerator rng = new SecureRandomNumberGenerator();

Object salt = rng.nextBytes();

//密码加盐后哈希1024次，转换成Base64格式

String hashedPasswordBase64 = new Sha256Hash(plainTextPassword, salt, 1024).toBase64();

User user = new User(username, hashedPasswordBase64);

//把盐一起保存到用户账户中

user.setPasswordSalt(salt);

userDAO.create(user);

由于使用的 SHA-256 哈希用户的密码, 因此需要告诉 Shiro 使用Hashedcredentialmatcher 来匹配程序使用的哈希算法。为了识别上面生成的用户信息，需要如下配置realm。

[main]

credentialsMatcher = org.apache.shiro.authc.credential.Sha256CredentialsMatcher

# 使用base64格式而不是hex

credentialsMatcher.storedCredentialsHexEncoded = false

credentialsMatcher.hashIterations = 1024

myRealm = com.company.....

myRealm.credentialsMatcher = $credentialsMatcher

此外Realm 的doGetAuthenticationInfo实现需要返回[SaltedAuthenticationInfo](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authc/SaltedAuthenticationInfo.html)类型的结果，而不是AuthenticationInfo。SaltedAuthenticationInfo接口可确保在创建用户帐户时使用的盐可以被 Hashedcredentialmatcher引用。

引用

<http://shiro.apache.org/tutorial.html>

<http://shiro.apache.org/authentication.html>

https://github.com/zhangkaitao/shiro-example