[**第二章 身份验证——《跟我学Shiro》**](http://jinnianshilongnian.iteye.com/blog/2019547)

**博客分类：**

* [跟我学Shiro](http://jinnianshilongnian.iteye.com/category/305053)

[跟我学Shiro](http://www.iteye.com/blogs/tag/%E8%B7%9F%E6%88%91%E5%AD%A6Shiro)

**目录贴：**[**跟我学Shiro目录贴**](http://jinnianshilongnian.iteye.com/blog/2018398)

**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。

另外两个相关的概念是之前提到的**Subject**及**Realm**，分别是主体及验证主体的数据源。

**2.2  环境准备**

本文使用Maven构建，因此需要一点Maven知识。首先准备环境依赖：

**Java代码  [收藏代码](javascript:void())**

1. <dependencies>
2. <dependency>
3. <groupId>junit</groupId>
4. <artifactId>junit</artifactId>
5. <version>4.9</version>
6. </dependency>
7. <dependency>
8. <groupId>commons-logging</groupId>
9. <artifactId>commons-logging</artifactId>
10. <version>1.1.3</version>
11. </dependency>
12. <dependency>
13. <groupId>org.apache.shiro</groupId>
14. <artifactId>shiro-core</artifactId>
15. <version>1.2.2</version>
16. </dependency>
17. </dependencies>

添加junit、common-logging及shiro-core依赖即可。

**2.3  登录/退出**

1、首先准备一些用户身份/凭据（shiro.ini）

**Java代码  [收藏代码](javascript:void())**

1. [users]
2. zhang=123
3. wang=123

此处使用ini配置文件，通过[users]指定了两个主体：zhang/123、wang/123。

2、测试用例（com.github.zhangkaitao.shiro.chapter2.LoginLogoutTest）

**Java代码  [收藏代码](javascript:void())**

1. @Test
2. **public** **void** testHelloworld() {
3. //1、获取SecurityManager工厂，此处使用Ini配置文件初始化SecurityManager
4. Factory<org.apache.shiro.mgt.SecurityManager> factory =
5. **new** IniSecurityManagerFactory("classpath:shiro.ini");
6. //2、得到SecurityManager实例 并绑定给SecurityUtils
7. org.apache.shiro.mgt.SecurityManager securityManager = factory.getInstance();
8. SecurityUtils.setSecurityManager(securityManager);
9. //3、得到Subject及创建用户名/密码身份验证Token（即用户身份/凭证）
10. Subject subject = SecurityUtils.getSubject();
11. UsernamePasswordToken token = **new** UsernamePasswordToken("zhang", "123");
13. **try** {
14. //4、登录，即身份验证
15. subject.login(token);
16. } **catch** (AuthenticationException e) {
17. //5、身份验证失败
18. }
20. Assert.assertEquals(**true**, subject.isAuthenticated()); //断言用户已经登录
22. //6、退出
23. subject.logout();
24. }

2.1、首先通过new IniSecurityManagerFactory并指定一个ini配置文件来创建一个SecurityManager工厂；

2.2、接着获取SecurityManager并绑定到SecurityUtils，这是一个全局设置，设置一次即可；

2.3、通过SecurityUtils得到Subject，其会自动绑定到当前线程；如果在web环境在请求结束时需要解除绑定；然后获取身份验证的Token，如用户名/密码；

2.4、调用subject.login方法进行登录，其会自动委托给SecurityManager.login方法进行登录；

2.5、如果身份验证失败请捕获AuthenticationException或其子类，常见的如： DisabledAccountException（禁用的帐号）、LockedAccountException（锁定的帐号）、UnknownAccountException（错误的帐号）、ExcessiveAttemptsException（登录失败次数过多）、IncorrectCredentialsException （错误的凭证）、ExpiredCredentialsException（过期的凭证）等，具体请查看其继承关系；对于页面的错误消息展示，最好使用如“用户名/密码错误”而不是“用户名错误”/“密码错误”，防止一些恶意用户非法扫描帐号库；

2.6、最后可以调用subject.logout退出，其会自动委托给SecurityManager.logout方法退出。

**从如上代码可总结出身份验证的步骤：**

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；

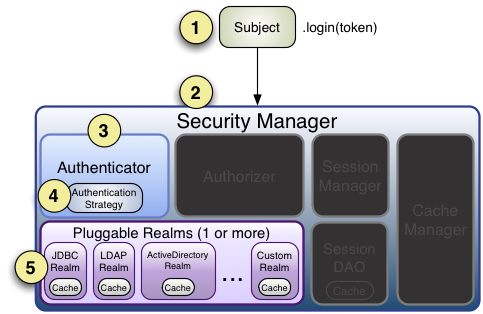
3、最后调用Subject.logout进行退出操作。

如上测试的几个问题：

1、用户名/密码硬编码在ini配置文件，以后需要改成如数据库存储，且密码需要加密存储；

2、用户身份Token可能不仅仅是用户名/密码，也可能还有其他的，如登录时允许用户名/邮箱/手机号同时登录。

**2.4  身份认证流程**



流程如下：

1、首先调用Subject.login(token)进行登录，其会自动委托给Security Manager，调用之前必须通过SecurityUtils. setSecurityManager()设置；

2、SecurityManager负责真正的身份验证逻辑；它会委托给Authenticator进行身份验证；

3、Authenticator才是真正的身份验证者，Shiro API中核心的身份认证入口点，此处可以自定义插入自己的实现；

4、Authenticator可能会委托给相应的AuthenticationStrategy进行多Realm身份验证，默认ModularRealmAuthenticator会调用AuthenticationStrategy进行多Realm身份验证；

5、Authenticator会把相应的token传入Realm，从Realm获取身份验证信息，如果没有返回/抛出异常表示身份验证失败了。此处可以配置多个Realm，将按照相应的顺序及策略进行访问。

**2.5  Realm**

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。如我们之前的ini配置方式将使用org.apache.shiro.realm.text.IniRealm。

org.apache.shiro.realm.Realm接口如下：

**Java代码  [收藏代码](javascript:void())**

1. String getName(); //返回一个唯一的Realm名字
2. **boolean** supports(AuthenticationToken token); //判断此Realm是否支持此Token
3. AuthenticationInfo getAuthenticationInfo(AuthenticationToken token)
4. **throws** AuthenticationException;  //根据Token获取认证信息

**单Realm配置**

1、自定义Realm实现（com.github.zhangkaitao.shiro.chapter2.realm.MyRealm1）：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** MyRealm1 **implements** Realm {
2. @Override
3. **public** String getName() {
4. **return** "myrealm1";
5. }
6. @Override
7. **public** **boolean** supports(AuthenticationToken token) {
8. //仅支持UsernamePasswordToken类型的Token
9. **return** token **instanceof** UsernamePasswordToken;
10. }
11. @Override
12. **public** AuthenticationInfo getAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {
13. String username = (String)token.getPrincipal();  //得到用户名
14. String password = **new** String((**char**[])token.getCredentials()); //得到密码
15. **if**(!"zhang".equals(username)) {
16. **throw** **new** UnknownAccountException(); //如果用户名错误
17. }
18. **if**(!"123".equals(password)) {
19. **throw** **new** IncorrectCredentialsException(); //如果密码错误
20. }
21. //如果身份认证验证成功，返回一个AuthenticationInfo实现；
22. **return** **new** SimpleAuthenticationInfo(username, password, getName());
23. }
24. }

2、ini配置文件指定自定义Realm实现(shiro-realm.ini)

**Java代码  [收藏代码](javascript:void())**

1. #声明一个realm
2. myRealm1=com.github.zhangkaitao.shiro.chapter2.realm.MyRealm1
3. #指定securityManager的realms实现
4. securityManager.realms=$myRealm1

通过$name来引入之前的realm定义

3、测试用例请参考com.github.zhangkaitao.shiro.chapter2.LoginLogoutTest的testCustomRealm测试方法，只需要把之前的shiro.ini配置文件改成shiro-realm.ini即可。

**多Realm配置**

1、ini配置文件（shiro-multi-realm.ini）

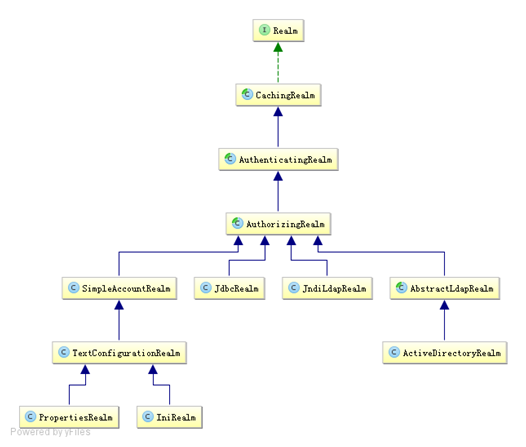
**Java代码  [收藏代码](javascript:void())**

1. #声明一个realm
2. myRealm1=com.github.zhangkaitao.shiro.chapter2.realm.MyRealm1
3. myRealm2=com.github.zhangkaitao.shiro.chapter2.realm.MyRealm2
4. #指定securityManager的realms实现
5. securityManager.realms=$myRealm1,$myRealm2

securityManager会按照realms指定的顺序进行身份认证。此处我们使用显示指定顺序的方式指定了Realm的顺序，如果删除“securityManager.realms=$myRealm1,$myRealm2”，那么securityManager会按照realm声明的顺序进行使用（即无需设置realms属性，其会自动发现），当我们显示指定realm后，其他没有指定realm将被忽略，如“securityManager.realms=$myRealm1”，那么myRealm2不会被自动设置进去。

2、测试用例请参考com.github.zhangkaitao.shiro.chapter2.LoginLogoutTest的testCustomMultiRealm测试方法。

**Shiro默认提供的Realm**



以后一般继承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；

**JDBC Realm使用**

1、数据库及依赖

**Java代码  [收藏代码](javascript:void())**

1. <dependency>
2. <groupId>mysql</groupId>
3. <artifactId>mysql-connector-java</artifactId>
4. <version>5.1.25</version>
5. </dependency>
6. <dependency>
7. <groupId>com.alibaba</groupId>
8. <artifactId>druid</artifactId>
9. <version>0.2.23</version>
10. </dependency>

本文将使用mysql数据库及druid连接池；

2、到数据库shiro下建三张表：users（用户名/密码）、user\_roles（用户/角色）、roles\_permissions（角色/权限），具体请参照shiro-example-chapter2/sql/shiro.sql；并添加一个用户记录，用户名/密码为zhang/123；

3、ini配置（shiro-jdbc-realm.ini）

**Java代码  [收藏代码](javascript:void())**

1. jdbcRealm=org.apache.shiro.realm.jdbc.JdbcRealm
2. dataSource=com.alibaba.druid.pool.DruidDataSource
3. dataSource.driverClassName=com.mysql.jdbc.Driver
4. dataSource.url=jdbc:mysql://localhost:3306/shiro
5. dataSource.username=root
6. #dataSource.password=
7. jdbcRealm.dataSource=$dataSource
8. securityManager.realms=$jdbcRealm

1、变量名=全限定类名会自动创建一个类实例

2、变量名.属性=值 自动调用相应的setter方法进行赋值

3、$变量名 引用之前的一个对象实例

4、测试代码请参照com.github.zhangkaitao.shiro.chapter2.LoginLogoutTest的testJDBCRealm方法，和之前的没什么区别。

**2.6  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：

**Java代码  [收藏代码](javascript:void())**

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。

假设我们有三个realm：

myRealm1： 用户名/密码为zhang/123时成功，且返回身份/凭据为zhang/123；

myRealm2： 用户名/密码为wang/123时成功，且返回身份/凭据为wang/123；

myRealm3： 用户名/密码为zhang/123时成功，且返回身份/凭据为zhang@163.com/123，和myRealm1不同的是返回时的身份变了；

1、ini配置文件(shiro-authenticator-all-success.ini)

**Java代码  [收藏代码](javascript:void())**

1. #指定securityManager的authenticator实现
2. authenticator=org.apache.shiro.authc.pam.ModularRealmAuthenticator
3. securityManager.authenticator=$authenticator
5. #指定securityManager.authenticator的authenticationStrategy
6. allSuccessfulStrategy=org.apache.shiro.authc.pam.AllSuccessfulStrategy
7. securityManager.authenticator.authenticationStrategy=$allSuccessfulStrategy

**Java代码  [收藏代码](javascript:void())**

1. myRealm1=com.github.zhangkaitao.shiro.chapter2.realm.MyRealm1
2. myRealm2=com.github.zhangkaitao.shiro.chapter2.realm.MyRealm2
3. myRealm3=com.github.zhangkaitao.shiro.chapter2.realm.MyRealm3
4. securityManager.realms=$myRealm1,$myRealm3

2、测试代码（com.github.zhangkaitao.shiro.chapter2.AuthenticatorTest）

2.1、首先通用化登录逻辑

**Java代码  [收藏代码](javascript:void())**

1. **private** **void** login(String configFile) {
2. //1、获取SecurityManager工厂，此处使用Ini配置文件初始化SecurityManager
3. Factory<org.apache.shiro.mgt.SecurityManager> factory =
4. **new** IniSecurityManagerFactory(configFile);
6. //2、得到SecurityManager实例 并绑定给SecurityUtils
7. org.apache.shiro.mgt.SecurityManager securityManager = factory.getInstance();
8. SecurityUtils.setSecurityManager(securityManager);
10. //3、得到Subject及创建用户名/密码身份验证Token（即用户身份/凭证）
11. Subject subject = SecurityUtils.getSubject();
12. UsernamePasswordToken token = **new** UsernamePasswordToken("zhang", "123");
14. subject.login(token);
15. }

2.2、测试AllSuccessfulStrategy成功：

**Java代码  [收藏代码](javascript:void())**

1. @Test
2. **public** **void** testAllSuccessfulStrategyWithSuccess() {
3. login("classpath:shiro-authenticator-all-success.ini");
4. Subject subject = SecurityUtils.getSubject();
6. //得到一个身份集合，其包含了Realm验证成功的身份信息
7. PrincipalCollection principalCollection = subject.getPrincipals();
8. Assert.assertEquals(2, principalCollection.asList().size());
9. }

即PrincipalCollection包含了zhang和zhang@163.com身份信息。

2.3、测试AllSuccessfulStrategy失败：

**Java代码  [收藏代码](javascript:void())**

1. @Test(expected = UnknownAccountException.**class**)
2. **public** **void** testAllSuccessfulStrategyWithFail() {
3. login("classpath:shiro-authenticator-all-fail.ini");
4. Subject subject = SecurityUtils.getSubject();
5. }

shiro-authenticator-all-fail.ini与shiro-authenticator-all-success.ini不同的配置是使用了securityManager.realms=$myRealm1,$myRealm2；即myRealm验证失败。

对于AtLeastOneSuccessfulStrategy和FirstSuccessfulStrategy的区别，请参照testAtLeastOneSuccessfulStrategyWithSuccess和testFirstOneSuccessfulStrategyWithSuccess测试方法。唯一不同点一个是返回所有验证成功的Realm的认证信息；另一个是只返回第一个验证成功的Realm的认证信息。

自定义AuthenticationStrategy实现，首先看其API：

**Java代码  [收藏代码](javascript:void())**

1. //在所有Realm验证之前调用
2. AuthenticationInfo beforeAllAttempts(
3. Collection<? **extends** Realm> realms, AuthenticationToken token)
4. **throws** AuthenticationException;
5. //在每个Realm之前调用
6. AuthenticationInfo beforeAttempt(
7. Realm realm, AuthenticationToken token, AuthenticationInfo aggregate)
8. **throws** AuthenticationException;
9. //在每个Realm之后调用
10. AuthenticationInfo afterAttempt(
11. Realm realm, AuthenticationToken token,
12. AuthenticationInfo singleRealmInfo, AuthenticationInfo aggregateInfo, Throwable t)
13. **throws** AuthenticationException;
14. //在所有Realm之后调用
15. AuthenticationInfo afterAllAttempts(
16. AuthenticationToken token, AuthenticationInfo aggregate)
17. **throws** AuthenticationException;

因为每个AuthenticationStrategy实例都是无状态的，所有每次都通过接口将相应的认证信息传入下一次流程；通过如上接口可以进行如合并/返回第一个验证成功的认证信息。

自定义实现时一般继承org.apache.shiro.authc.pam.AbstractAuthenticationStrategy即可，具体可以参考代码com.github.zhangkaitao.shiro.chapter2.authenticator.strategy包下OnlyOneAuthenticatorStrategy 和AtLeastTwoAuthenticatorStrategy。

到此基本的身份验证就搞定了，对于AuthenticationToken 、AuthenticationInfo和Realm的详细使用后续章节再陆续介绍。

示例源代码：<https://github.com/zhangkaitao/shiro-example>；可加群134755960探讨Spring/Shiro技术。

* [查看图片附件](http://jinnianshilongnian.iteye.com/blog/2019547)