1. Authentication身份认证

本章详细介绍Shiro安全框架的身份认证，包括身份认证的思想、API、流程以及Subject的记住状态和认证状态的区别和实际项目中的应用。关于本章，我们需要把握如下方面内容。

* Shiro对Subjects进行身份认证的步骤
* Shiro对Subjects进行身份认证的API
* Subject的Remembered状态同Authenticated的区别和应用
* Shiro对身份认证的内部流程
* Shiro Authenticator对象
* Shiro的AuthenticationStrategy的种类和特点

目录

[3 Authentication身份认证 1](#_Toc493755144)

[3.1 认证Subjects对象 2](#_Toc493755145)

[3.2 Remembered同Authenticated区别 4](#_Toc493755146)

[3.3 退出登录 5](#_Toc493755147)

[3.4 认证身份的流程 5](#_Toc493755148)

[3.5 Authenticator认证器 6](#_Toc493755149)

[3.6 AuthenticationStrategy 9](#_Toc493755150)

[3.7 总结 10](#_Toc493755151)

身份认证，就是认证Subject身份的过程，也就是证明用户是他自己的过程。Shiro作为Java开发中流行的安全框架，身份认证当然是Shiro安全框架的一个基本功能，而且正如图3-1 Shiro身份认证所示，Authentiction功能是Shiro安全框架的核心功能之一，而且是所有其他功能的入口功能。

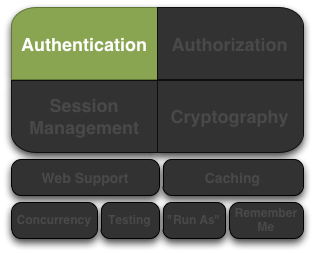


图3-1 Shiro身份认证

为了完成身份认证，需要认证的用户需要提供他的身份标识信息，和证明自己身份的凭证信息，而且这些信息需要在我们系统中有注册、能识别、且值得信赖。Shiro接收到用户提供的身份标识信息和身份凭证信息后，根据我们系统中的数据进行匹配验证，并且返回验证后的结果。为了理解Shiro的身份认证过程，我们需要理解两个概念：

* Principals(身份)

Principals(身份)是Subject的标识属性，它可以是任何标识数据，比如姓名、用户名、一串编号等等。当然，作为一个需要进行认证的Subject的标识信息，在整个系统中应该是唯一的，比如常用的用户名和邮件地址等。

* Credentials(身份凭证)

Credentials(身份凭证)是只有Subject才知道的机密信息，因为只有Subject自己才会知道，所有可以用来证明Subject的本身是否真实。通常情况下，我们会用密码做为Subject的Credentials(身份凭证)，当然也可以扫描用户的指纹和视网膜等信息做Credentials(身份凭证)

通常情况下，我们用用户名/密码表示认证功能里面的Principals/Credentials。用户名为唯一身份标识，密码是用来证明唯一身份标识真实性的凭证。如果Subject提供的密码同他的身份标识用户名匹配，我们的系统就会认为该Subject是真实合法的，因为再整个系统环境下，只有Subject自己才知道自己的密码。Shiro安全框架的身份认证功能就是基于这点实现的。

* 1. 认证Subjects对象

对Subject的身份认证，一般就按如下三个步骤进行

1. 收集提交过来的Subject的身份标识信息(Principals)和身份凭证信息(Credentials)
2. 提交身份标识信息(Principals)和身份凭证信息(Credentials)信息给安全服务进行验证
3. 根据验证结果，判断后面的走向。如果成功，继续操作，如果失败，转向到验证提示页面，中断前面的操作

Shiro安全框架的身份认证，也是按上面介绍的三个步骤进行的。接下来我们介绍Shiro安全框架的身份认证步骤和相关API，以及注意事项。

1. 收集Subject的身份标识信息(Principals)和身份凭证信息(Credentials)，样例代码块如下：

//使用简单的username/password对:

UsernamePasswordToken token = new UsernamePasswordToken(username, password);

//设置记住我属性

token.setRememberMe(true);

在上面代码块中，我们使用的是UsernamePasswordToken类的对象，封装了username/password对的验证信息，以便后面步骤里面，提交给Shiro安全框架进行身份识别和验证，使用UsernamePasswordToken进行Subject的Principals和Credentials进行封装后提交，也是我们在Shiro中常用的方式。

UsernamePasswordToken实现了Shiro中的org.apache.shiro.authc.  
AuthenticationToken接口。AuthenticationToken接口是Shiro安全框架中封装Subject的Principals和Credentials进行身份验证的标准接口，定义了两接口：Object getPrincipal()和Object getCridential()，通过调用这两个接口，可以获取要验证Subject对象的Principals和Cridentials。如果我们不想使用UsernamePasswordToken，可以自己实现AuthenticationToken接口，自己完成Subject的Principals和Credentials的封装。

这里有一点要注意，Shiro安全框架是不关心Subject的Principals和Credentials来源于哪里，怎么来的。比如可以来自HTTP的请求；来自另外一个服务发送过来的请求内容；来自另外一块代码的方法调用等等，对Shiro安全框架都无所谓，只要把Principals和Credentials封装到一个AuthenticationToken对象中，Shiro安全框架能通过Object getPrincipal()和Object getCridential()两个方法分别获取得到就行。

1. 提交身份信息(Principals/Credentials)基于Shiro安全框架进行认证，样例代码块如下：

// 获取当前的Subject对象

Subject currUser = SecurityUtils.getSubject();

// 基于当前的Subject对象，验证身份

currUser.login(token);

如上面代码所示，用AuthenticationToken封装好Subject的Principals和Credentials后，我们先需要通过SecurityUtils的getSubject()方法获取到当前的Subject对象，再调用Subject对象的login方法，传入AuthenticationToken对象，完成Subject对象的身份认证。

1. 处理认证后的成功和失败的结果

执行完currUser.login(token)代码后，如果完成认证后，证明当前Subject的身份是正确的，当前这个应用线程就会继续往后面执行。而且不管什么时候再调用SecurityUtils.getSubject()方法获取Subject对象后，这个Subject对象的Authenticated状态都是true。

如果currUser.login(token)里面身份认证失败了会怎么样呢？比如密码身份标识不存在；密码不匹配；账号被锁定了等等。

这时候我们需要在currUser.login(token)代码外面包含try catch语句块，当认证失败的时候，会抛出对应的异常，我们在对应的catch中就可以对不同的失败情况，做不同的处理。样例代码块如下：

try {

currUser.login(token);

} catch ( UnknownAccountException uae ) {

// 账号不存在异常

...

} catch ( IncorrectCredentialsException ice ) {

// 密码错误异常

...

} catch ( LockedAccountException lae ) {

// 账号被锁定异常

...

} catch ( ExcessiveAttemptsException eae ) {

// 错误次数过多一下

...

} ... catch自定义异常 ...

} catch ( AuthenticationException ae ) {

// 其他异常

}

研究过java中try catch语句块执行顺序的应该知道，如果前面没有被对应的异常catch到，就被最后的AuthenticationException catch到。

* 1. Remembered同Authenticated区别

通过前面的样例代码我们知道，在调用Subject的login方法验证身份之前，会给UsernamePasswordToken对象设置一个rememberMe属性值为true，如果验证成功了就有一个被记住的特性。那remembered属性同authenticated属性有什么区别，它们各自在实际项目中有哪些使用呢？接下来我们详细介绍一下这两个概念。

* Remembered

Remembered，被记住的。一个被记住的Subject对象，一定不是一个匿名的Subject对象，它一定有自己的身份标识(Principals)。也就是说，调用他的getPrincipal()方法一定不会是空的。需要注意的是，这个记住状态是前面的Subject，在它自己的Session中被记住的，当调用Subject的isRemembered方法的时候，返回true。当前Session不是被记住的Subject对象的Session，是另外一个后来的Session。

* Authenticated

Authenticated，被认证过的。表示Subject是一个被成功认证过的对象，也就是说，执行login方法的时候，没有抛出任何异常，而且当前Session也是当前Subject对象的Session。如果我们调用该Subject的isAuthenticated方法，返回是true。

Authenticated表示当前的Subject对象处在完全被证明的状态，当前的Subject对象肯定是它自己。Remembered只是表示当前的Subject对象以前有被证明过，但是中间有可能有什么原因，Subject对象的同系统中断了联系，再回来新Session中的时候，Subject对象有可能是自己本身，也有可能是另外一个Subject在冒充它的身份。对那些安全性要求不是很高的操作，只要是记住状态的Subject就可以访问，如果是安全性要求比较高的操作，比如在线支付功能，这时候就一定要对记住状态的Subject重新进行认证，确保是Subject对象本身后，才能继续支付操作。

* 1. 退出登录

当验证通过的Subject对象完成了所有的安全操作后，我们可以调用Subject.logout（）方法退出。如下面样例代码：

// 退出Subject

currUser.logout();

如果Subject对象调用了logout，意味着跟它关联的Session对象也过期了，不仅仅Session的所有数据无法使用，就是Subject对象本身的Principals信息也自动清除，当前Subject对象自动有回归成匿名Subject，特别要注意的是，以前的记住状态同样没有了。如果要继续进行其他安全操作的话，需要重新调用login方法，完成新的身份认证。

* 1. 认证身份的流程

到现在为止，我们对Subject对象的身份认证API，都只是围绕这Subject对象，操作的都是Subject类相关的方法完成的。那在Subject方法内部，具体涉及到哪些对象，发生了哪些事情，交互流程是怎么样的呢？图3-2身份认证很好的描述了Subject对象认证涉及的对象和处理认证的流程，里面的数字，代表流程的顺序。

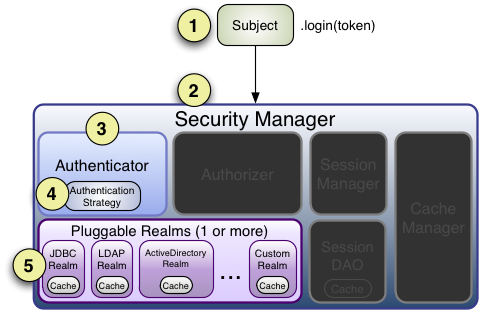


图3-2 身份认证

第一步：应用代码通过调用Subject的login方法，将要认证身份的Subject的principals和credentials信息，封装成AuthenticationToken对象传递给login方法。

第二步：Subject类是DelegationSubject的一个子类，当调用Subject对象的login方法的时候，自动会将它委托给对应的SecurityManager对象，调用securityManager对象的login(AuthenticationToken)方法。这是通过SecurityUtils.setSecurityManager()方法设置的委托关系（所以在调用Subject.login方法进行身份认证之前，一定要完成SecurityManager对象的委托设置）。

第三步：SecurityManager对象接收到委托的login验证请求后，会将认证动作再次委托给自己管理的Authenticator对象，调用Authenticator对象的authenticate(token)方法进行身份认证。一般情况下，SecurityManager会管理一个ModularRealmAuthenticator类型的对象，该类型的对象，支持基于一个或多个Realm对象的认证。

第四步：如果配置了多个Realm对象，Authenticator对象会把认证委托给AuthenticationStrategy进行多Realm身份验证。ModularRealmAuthenticator默认使用的AuthenticationStrategy是挨个使用Realm进行身份认证，只要有一个Realm认证通过，就表示身份认证通过。当然，我们也可自己设置不同的AuthenticationStrategy，后面紧接着会做相关介绍。

第五步：Shiro安全框架会判断注册的Realm对象是否支持AuthenticationToken认证，如果支持，token就会调用Realm对象的getAuthenticationInfo方法获取认证后的信息，如果没有信息返回或抛出异常，就表示认证失败。以同样的方式，调用每个Realm对象的getAuthenticationInfo方法，整个验证过程。

* 1. Authenticator认证器

通过前面的认证流程的介绍，当我们的应用程序调用Subject的login方法时，会自动委托给SecurityManager，由SecurityManager调用它管理的Authenticator对象的authenticate方法完成认证。Authenticator是一个接口，它的源代码如下：

package org.apache.shiro.authc;

public interface Authenticator {

public AuthenticationInfo authenticate(AuthenticationToken authenticationToken)

throws AuthenticationException;

}

如果认证通过，就会返回AuthenticationInfo结果信息，否则就返回null或抛出异常。当然，接口肯定是没有逻辑的，SecurityManager缺省是调用的ModularRealmAuthenticator实现类的认证方法，ModularRealmAuthenticator实现类的关键代码如下：

package org.apache.shiro.authc.pam;

public class ModularRealmAuthenticator extends AbstractAuthenticator {

... ...

protected AuthenticationInfo doSingleRealmAuthentication(Realm realm, AuthenticationToken token) {

… …

return info;

}

protected AuthenticationInfo doMultiRealmAuthentication(Collection<Realm> realms, AuthenticationToken token) {

… …

return aggregate;

}

**protected AuthenticationInfo doAuthenticate(AuthenticationToken authenticationToken) throws AuthenticationException** {

assertRealmsConfigured();

Collection<Realm> realms = getRealms();

**if (realms.size() == 1) {**

**return doSingleRealmAuthentication(realms.iterator().next(), authenticationToken);**

**} else {**

**return doMultiRealmAuthentication(realms, authenticationToken);**

**}**

}

... ...

}

在ModularRealmAuthenticator的doAuthenticate方法里面，会判断是否配置了多个Realm，如果只配置了一个（也知道配置一个Realm），就调用doSingleRealmAuthentication方法，基于该Realm进行验证，如果配置了多个Realm，就会调用doMultiReamAuthentication方法，基于多个Realm对象和配置的AuthenticationStrategy完成验证。

根据情况，我们也可以自己定义一个Authenticator实现了，接收SecurityManager的委托完成认证。如下面的DemoAuthenticator.java代码，就实现了一个简单的认证：只要用户名中包含了test，密码以123开头，就表示认证通过。详细代码可以查看随书代码[code/ShiroTP03/src/main/java/cn/ com/shiro/book/tp03/authz/DemoAutenti-  
cator.java。](../code/ShiroTP03/src/main/java/cn/com/shiro/book/tp03/authz/DemoAuthenticator.java)

package cn.com.shiro.book.tp03.authz;

... ...

public class DemoAuthenticator implements Authenticator {

private String name;

public AuthenticationInfo authenticate(

AuthenticationToken token)

throws AuthenticationException {

SimpleAuthenticationInfo authInfo = null;

if(token == null){

throw new AuthenticationException("验证token不能为null");

}

if(!(token instanceof UsernamePasswordToken)){

throw new AuthenticationException("暂时只能支持UsernamePasswordToken类型的凭证");

}

UsernamePasswordToken userPswToken = (UsernamePasswordToken)token;

String userName = userPswToken.getUsername();

String password = new String(userPswToken.getPassword());

if(!StringUtils.hasLength(userName) || !StringUtils.hasLength(password)){

throw new AuthenticationException("用户名或密码输入不合法");

}

if(userName.contains("test") && password.startsWith("123")){

authInfo = new SimpleAuthenticationInfo();

authInfo.setCredentials(password.toCharArray());

authInfo.setPrincipals(new SimplePrincipalCollection(userName, this.name));

}

return authInfo;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

接下来在shiro.ini中，添加在SecurityManager中注册DemoAuthenticator实现类的信息。样例代码块如下，详细信息请查看随书样例代码[code/ShiroTP03/src/main/resources/shiro.ini。](../code/ShiroTP03/src/main/resources/shiro.ini)

[main]

# 配置自定义验证器类，变量名是authenticator

authenticator=cn.com.shiro.book.tp03.authz.DemoAuthenticator

# 给authenticator对象的name属性设置值

authenticator.name=demoAuthenticator

# 将authenticator对象注册到securityManager对象中(代替默认的ModularRealmAuthenticator对象)

securityManager.authenticator=$authenticator

按第二章的方式，编写一个可以运行的测试代码，样例代码块如下，详细情况可以查看随书代码[code/ShiroTP03/src/test/java/cn/com/shiro/book/tp03/TestMain.java](../code/ShiroTP03/src/test/java/cn/com/shiro/book/tp03/TestMain.java)。

// 1.创建IniSecurityManagerFactory，基于shiro.ini进行初始化

Factory<org.apache.shiro.mgt.SecurityManager> factory = new IniSecurityManagerFactory(

"classpath:shiro.ini");

// 2.获取SecurityManager对象

org.apache.shiro.mgt.SecurityManager securityManager = factory

.getInstance();

// 3.注册全局的SecurityManager对象

SecurityUtils.setSecurityManager(securityManager);

Subject currUser = SecurityUtils.getSubject();

//UsernamePasswordToken token = new UsernamePasswordToken("root","123456");

UsernamePasswordToken token = new UsernamePasswordToken("1test2","123abc");

currUser.login(token);

System.out.println("登录成功");

currUser.logout();

运行测试代码，我们就会发现通过DemoAuthenticator.java的验证，只能验证用户名中包含有test，密码是123开头的Subject身份，其他的都验证不过，哪怕是我们在shiro.ini的[users]下配置好的users信息。

关于Authenticator对象，最后要强调的是，虽然我们可以自己实现Authenticator接口完成验证逻辑，但是实际应用中，用缺省的ModularRealmAuthenticator是一个更好的选择。

* 1. AuthenticationStrategy

前面的认证3.4节中的认证流程中有介绍，缺省的ModularRealmAuthenticator认证器会自己判断SecurityManager是否注册了一个Realm，还是注册了多个Realm，如果注册了两个以上的Realm，就会在每个Realm认证的基础上，结合AuthenticationStrategy完成最终的认证。比如通过AuthenticationStrategy确定是只要任何一个Realm认证通过就算认证成功了呢，还是一定要所有的Realm认证通过后才算成功。返回的AuthenticationInfo对象中，只要包含第一Realm认证成功后的信息就够了呢，还是要包含每个Realm认证成功的信息。

AuthenticationStrategy是一个接口，它里面有四个方法，样例代码块如下：

package org.apache.shiro.authc.pam;

public interface AuthenticationStrategy {

AuthenticationInfo beforeAllAttempts(Collection<? extends Realm> realms, AuthenticationToken token) throws AuthenticationException;

AuthenticationInfo beforeAttempt(Realm realm, AuthenticationToken token, AuthenticationInfo aggregate) throws AuthenticationException;

AuthenticationInfo afterAttempt(Realm realm, AuthenticationToken token, AuthenticationInfo singleRealmInfo, AuthenticationInfo aggregateInfo, Throwable t)

AuthenticationInfo afterAllAttempts(AuthenticationToken token, AuthenticationInfo aggregate) throws AuthenticationException;

}

* beforeAllAttempts

在执行所有Realms的认证方法前调用

* beforeAttempt

在执行每个Realm的getAuthenticationInfo方法前调用

* afterAttempt

在执行每个Realm的getAuthenticationInfo方法后调用

* afterAllAttempts

在执行完所有Realm的认证后调用

我们可以自己实现AuthenticationStrategy，在对应的方法里面封装合适的逻辑，再同在SecurityManager中注册自定义Authenticator对象一样，在shiro.ini配置文件中注册我们自定义的AuthenticationStrategy，样例代码如下。因为演示AuthenticationStrategy的效果，需要Realm组件的基础，我们这里就暂时没有在ShiroTP03工程中编写真正的AuthenticationStrategy实现类，下面的样例配置代码块也只是说明一下配置方式，不能真正运行出来。具体的AuthenticationStrategy效果体现，在后面的Realm章节详细介绍。

[main]

... ...

authcStrategy = org.apache.shiro.authc.pam.FirstSuccessfulStrategy

securityManager.authenticator.authenticationStrategy = $authcStrategy

当然，在Shiro安全框架中，对AuthenticationStrategy有做了三个具体的实现。

* 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，如果有一个失败就算认证失败。

ModularRealmAuthenticator认证器默认使用的是AtLeastOneSuccessfulStrategy

* 1. 总结

1. 基于Shiro安全框架进行身份认证的API是直接调用Subject对象的login(AuthenticationToken)方法，如果没有抛异常，就表示认证成功，如果认证失败的话，就会抛出对应的异常信息，我们的应用代码可以根据不同的异常信息做对应的认证失败后的逻辑处理。
2. 一个Subject对象，被唯一的一个Session对象管理，我们的应用程序，可以通过Session进行当前Subject对象独特的数据传递
3. 登录成功后的Subject对象，会Remembered状态和Authenticated两个状态。
4. Subject对象登录验证在Shiro安全框架中设计到SecurityManager、Authenticator、AuthenticationStrategy、Realm等对象，它们的执行顺序是Subject、SecurityManager、Authenticator、AuthenticationStrategy和Realm。正常情况下，最终是由Realm结合AuthenticationStrategy完成认证。
5. Shiro安全框架中对AuthenticationStrategy有AtLeastOneSuccessfulStrategy、FirstSuccessfulStrategy和AllSuccessfulStrategy等三个实现，ModularRealmAuthenticator验证器默认是用的是AtLeastOneSuccessfulStrategy