1. Realm身份认证

本章是在Authenticator验证器的基础上，介绍在Shiro安全框架中，Authenticator后面调用的Realm。主要介绍Shiro中Realm的体系和配置使用，以及我们怎么自定义Realm。对于读者，我们需要把握如下几个方面：

* Realm的接口定义
* Realm的体系
* Shiro自带的常用Realm实现类的作用和使用
* Realm的配置
* 自定义Realm和使用自定义Realm

目录

[4 Realm身份认证 1](#_Toc493755171)

[4.1 Realm简介 2](#_Toc493755172)

[4.2 Shiro自带的Realm 3](#_Toc493755173)

[4.3 使用单个Realm完成身份认证 4](#_Toc493755174)

[4.3.1 Realm的配置 4](#_Toc493755175)

[4.3.2 使用常用的Realm完成身份认证 5](#_Toc493755176)

[4.4 自定义Realm并认证身份 12](#_Toc493755177)

[4.5 使用多个Realm完成身份认证 17](#_Toc493755178)

[4.6 总结 21](#_Toc493755179)

通过前面章节的研究，我们知道Subject是应用代码同Shiro安全框架的接口，对Subject的身份认证，是委托给SecurityManager对象，调用SecurityManager管理的Authenticator对象实现的。虽然我们在3章 Authencation身份认证中能自定义一个Authencator类，完成SecurityManger的身份验证，当是实际情况下，SecurityManager是默认调用的ModularRealmAuthenticator验证器完成的验证。在ModularRealmAuthenticator内部，最终是委托给Realm真正完成的验证，而且在Shiro安全框架中，我们至少要配置一个Realm(当然也可以根据需要配置多个Realm，结合指定的AuthencationStrategy完成身份验证)。

可以把Realm理解成数据源，Shiro是通过Realm读取我们应用的业务数据，比如数据库中的用户名和密码，或ini配置文件中的用户配置等，完成Subject的身份认证。这一章我们将详细介绍Realm和Realm的基本使用。

* 1. Realm简介

Realm，域，是Shiro安全框架中的一个核心组件。这个组件能访问我们应用中的安全数据，比如用户、角色和权限等，并且能将这些安全数据，以Shiro安全框架需要并且理解的方式，传递给Shiro安全框架，一般Shiro安全框架能正确的完成安全操作。也就是说Realm是Shiro安全框架同我们应用程序的接口，由Realm从我们的应用从获取安全数据，组织成合适的数据返回给Shiro安全框架使用，它就是Shiro安全框架的数据源。

在实际应用中，一个Realm会专门针对一种数据的访问，如关系数据库、LDAP、文件系统获取其他情况，我们需要分别编写不同的Realm实现类，访问它们的安全数据（角色、权限等）。底层访问的技术也就是JDBC、文件流、Hibernate、MyBatis或JPA等其他数据持久化API。

因为大部分数据源都是同时包含身份认证数据(用户名、密码)和权限数据（角色、权限），所以Realm一般会同时支持身份认证操作和授权权限控制操作。在本章，我们主要研究Realm中的身份认证部分。如下是Shiro中定义的Realm接口：

package org.apache.shiro.realm;

public interface Realm {

String getName();

boolean supports(AuthenticationToken token);

AuthenticationInfo getAuthenticationInfo(AuthenticationToken token) throws AuthenticationException;

}

String getName()返回Realm的名称。每个Realm需要有名称，而且在SecurityManager中的名称必须唯一，不能重复。

boolean supports(AuthenticationToken token) 判断是否支持传入的认证令牌类型(AuthencationToken)，支持返回true，否则返回false

AuthenticationInfo getAuthenticationInfo(AuthenticationToken token) AuthenticationException 获取当前token对应的认证信息。如果认证成功，就返回该次认证的信息对象，否则返回null或者抛出AuthenticationException异常，获取根据实际认证失败的原因，抛出AccountException、CasAuthenticationException、CredentialsException、UnsupportedTokenException等异常。

* 1. Shiro自带的Realm

Shiro安全框架自己内部又对Realm域组件做了基本的实现，如图4-1Realm实现描述了Shiro的Realm实现类，和它们的层次关系。

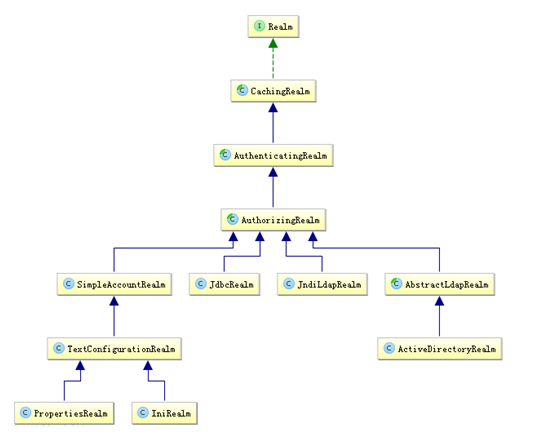


图4-1 Realm实现

* org.apache.shiro.realm. Realm

Realm，域，接口，定义了getName、supports和getAuthenticationInfo等三个基本方法，前面4.1小结中对这三个方法做了详细的阐述，请参考。

* org.apache.shiro.realm.CachingRealm

CachingRealm是一个抽象类，实现了Realm和CacheManagerAware接口，在基本的Realm的基础上，扩展了缓存管理功能。同时还提供了一个贯穿所有子类的，获取每个Realm特有的身份标识(principal/identity)的方法：Object getAvailablePrincipal(PrincipalCollection)。

* org.apache.shiro.realm.AuthentingRealm

AuthentingRealm是一个继承CachingRealm的抽象类，扩展了AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)方法，对登录身份认证功能的支持，但是没有提供对授权安全操作的支持（在后面的子类提供）

* org.apache.shiro.realm.AuthorizingRealm

AuthorizingRealm是一个继承了AuthentingRealm的一个抽象类，它处理继承了AuthentingRealm的所有功能外，还扩展了授权（访问控制）的功能，扩展方法是AuthorizationInfo getAuthorizationInfo(PrincipalCollection principals)。子类可以通过重新改方法，实行授权功能。

* org.apache.shiro.realm.SimpleAccountRealm

SimpleAccountRealm是一个实现类，继承了AuthorizingRealm抽象了，基于配置的用户和角色，以及角色包含的权限信息，对Realm的身份认证功能和授权控制功能做了初步的实现。因为用户信息和角色信息都是简单的通过Map保存在内存的，所以不支持大量的用户和角色的安全信息的认证和授权。

* org.apache.shiro.realm.text.TextConfigurationRealm

TextConfigurationRealm是一个实现类，继承了SimpleAccountRealm类，支持在启动的时候，读取文本配置文件，初始化安全操作需要的用户、角色和权限信息。

* org.apache.shiro.realm.text.PropertiesRealm

PropertiesRealm是一个实现类，继承了TextConfigurationRealm，支持通过读取.properties属性文件的方式，初始化安全操作的用户、角色和权限信息。

* org.apache.shiro.realm.text.IniRealm

IniRealm是一个实现类，继承了TextConfigurationRealm类，支持通过读取.ini文件，初始化安全操作的用户、角色和权限信息。

* org.apache.shiro.realm.jdbc.JdbcRealm

JdbcRealm是一个实现类，继承了AuthorizingRealm类，实现了基于JDBC访问数据库安全信息，完成安全操作的功能。它是通过相关的查询语句，从数据源中获取用户、角色和权限的安全操作所需要的信息的，我们可以通过设置对应的查询语句，让JdbcRealm从应用的数据库中获取对应的信息。

* org.apache.shiro.realm.ldap.JndiLdapRealm(DefaultLdapRealm)

JndiLapRealm类已经被DefaultLdapRealm类代替。

DefaultLdapRealm是一个类，继承了AuthorizingRealm，实现了基于Java的JNDI和LDAP获取安全操作需要的信息。

* org.apache.shiro.realm.ldap.AbstractLdapRealm

AbstractLdapRealm是一个抽象类，它实现了基于LDAP服务器获取安全操作信息的功能，需要注意的是，通过AbstractLdapRealm只支持用户和角色信息，不支持权限信息。它的子类ActivedirectoryReam实现了权限列表的支持。

* org.apache.shiro.realm.activedirectory.ActiveDirectoryRealm

ActivedirectoryRealm是一个实现类，它实现了基于动态目录LDAP服务进行认证的功能。

* 1. 使用单个Realm完成身份认证

在Shiro安全框架中，默认使用的是IniRealm，我们也可以通过配置或编码的方式，在SecurityManager中指定使用指定的Realm，接下来将介绍怎么样基于ini配置文件的方式，指定SecurityManager使用常用的Realm完成身份认证。

* + 1. Realm的配置
* 显示配置(Explicit Assignment)

显示配置是指在ini配置文件的[main]部分配置要使用的Realm对象和在SecurityManager的Authenticator调用这些Realm对象的顺序。样例配置代码块如下：

[main]

# 配置Realm对象

iniRealm=org.apache.shiro.realm.text.IniRealm

propRealm=org.apache.shiro.realm.text.PropertiesRealm

jdbcRealm=org.apache.shiro.realm.jdbc.JdbcRealm

# 按顺序在SecurityManager中配置要使用的Realm对象

securityManager.realms=$iniRealm, $jdbcRealm, $propRealm

如上面代码块，先是在[main]部分，使用name=Realm类名的方式，配置多个Ream对象，name需要唯一；再在最后一行，通过securityManager.realms=Realms的方式，将一个或多个Realm对象，设置到SecurityManager中。Realm对象是通过$name方式引用的前面配置的realm名称，每个reaml引用之间，使用逗号(,)隔开，引用的顺序，就是Authenticator执行认证的顺序。

* 隐式配置(Implicit Assignment)

隐式配置是指只要在ini配置文件的[main]部分配置要使用的Realm对象，不需要明确的使用配置将这些对象配置到SecurityManager对象的realms属性中去。SecurityManager在初始化的时候，自动将配置的Realm对象，按它们配置的顺序加载到realms属性中，Authenticator也是按配置的顺序基于Realm完成认证。样例配置代码块如下：

[main]

# 配置Realm对象

iniRealm=org.apache.shiro.realm.text.IniRealm

propRealm=org.apache.shiro.realm.text.PropertiesRealm

jdbcRealm=org.apache.shiro.realm.jdbc.JdbcRealm

* + 1. 使用常用的Realm完成身份认证

使用Realm完成身份认证，一般要基于如下五个步骤进行：

1. 配置Realm对象
2. 配置认证的用户安全信息
3. 配置AuthenticationStrategy策略(如果有多个Realm)
4. 编写测试代码
5. 运行测试

下面按这五个步骤，逐个介绍几个常用Realm身份认证的使用。

* 基于IniRealm完成身份认证

1. 配置IniRealm

在resources中创建shiro\_iniRealm.ini文件，在里面添加如下代码块，详细信息可以查看随书代码[code/ShiroTP04/src/main/resources/shiro\_iniRealm.ini](../code/ShiroTP04/src/main/resources/shiro_iniRealm.ini)中的[main]部分。

[main]

#配置iniRealm对象(realm名称=realm类名)

myIniRealm=org.apache.shiro.realm.text.IniRealm

#设置myIniRealm对象的资源路径

myIniRealm.resourcePath=classpath:shiro\_iniRealm.ini

#设置iniRealm对象

securityManager.realms=$myIniRealm

需要注意的是，Shiro安全框架中，默认会创建一个名称为iniRealm的IniRealm对象，基于IniSecurityManagerFactory工厂指定的资源文件进行初始化，因为在SecurityManager中的Realm名称必须唯一，所以我们要创建一个新的Realm对象的话，使用其他的名称。如上配置，我们使用的是myIniRealm做新配置的IniRealm对象的名称。

还有，因为myIniRealm是有SecurityManager按我们配置文件创建的，里面的resourcePath属性是没有被初始化的，我们需要基于myIniRealm.resourcePath=classpath:shiro\_iniRealm.ini指定资源路径，以便myIniRealm对象创建后，自动从shiro\_iniRealm.ini文件中读取[users]部分的账号信息。

1. 配置用户账号信息

在前面步骤中创建的shiro\_iniRealm.ini文件中，添加[users]部分的内容，如下面的样例代码块，详细信息查看随书代码[code/ShiroTP04/src/main/resources/shiro\_iniRealm.ini](../code/ShiroTP04/src/main/resources/shiro_iniRealm.ini)中的[users]部分。

[users]

#配置登录账号和对应的密码(账号=密码)

zhangsan=123456

lisi=abc123

wangwu=pswabc123

账号/密码对是以 账号=密码的格式，配置在[users]一下的，每一行描述一个账号/密码对。

1. 编写测试代码

编写测试代码TestIniRealm.java代码，内容如下样例代码块，详细内容请参考随书代码[code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/TestIniRealm  
.java](../code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/TestIniRealm.java)。

package cn.com.shiro.book.tp04;

… …

public class TestIniRealm {

public static void main(String[] args) {

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

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

"classpath:shiro\_iniRealm.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();

//创建用户名密码登录令牌

UsernamePasswordToken token = new UsernamePasswordToken("zhangsan","123456");

currUser.login(token);

System.out.println("登录成功");

currUser.logout();

}

}

1. 运行测试

为了基于exec-maven-plugin插件运行前面定义的TestIniRealm.java类，在pom.xml中按如下代码块配置exec-maven-plugin插件，详细内容请参考随书代码[code/ShiroTP04/pom.xml](../code/ShiroTP04/pom.xml)。

<!-- 该插件用来运行测试，在正式项目中不是必须的 -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<executions>

<execution>

<goals>

<goal>java</goal>

</goals>

</execution>

</executions>

<configuration>

<classpathScope>test</classpathScope>

<mainClass>**cn.com.shiro.book.tp04.TestIniRealm**</mainClass>

</configuration>

</plugin>

注意要准确配置如上粗体显示的mainClass元素内容，否则运行的时候会找到不启动类。

基于Maven运行exec:java，就可以查看运行结果。我们也可以试着将TestIniRealm.java代码中的账号/密码改成错误的，再运行检验是否能正常验证登录。

如上操作，只是让我们通过具体的配置，体现使用IniRealm完整身份认证的过程。在实际项目中，因为Shiro安全框架缺省就会根据ini配置文件初始化一个IniRealm对象，它的名称为iniRealm，并且自动设置到SecurityManager对象中去，所以我们可以使用#符合，注释掉shiro\_iniRealm.ini文件中的[main]部分的内容，一样可以达到同样的基于IniRealm对象的身份认证结果。

* 基于PropertiesRealm完成身份认证

1. 配置PropertiesRealm

在resources中创建shiro\_propRealm.ini文件，在里面添加如下代码块，详细信息可以查看随书代码[code/ShiroTP04/src/main/resources/shiro\_propRealm.ini](../code/ShiroTP04/src/main/resources/shiro_propRealm.ini)中的[main]部分。

[main]

#配置propRealm对象(realm名称=realm类名)

propRealm=org.apache.shiro.realm.text.PropertiesRealm

#设置propRealm对象的资源路径

propRealm.resourcePath=classpath:users\_prop.properties

#设置propRealm对象

securityManager.realms=$propRealm

如上代码，我们给propRealm的resourcePath属性设置了一个独立的配置账号信息的资源文件users\_prop.properties，这点同前面的IniRealm的不同，可以配置到同一个ini文件中的[users]部分。

1. 配置用户账号信息

在src/main/resources目录下创建一个名称为users\_prop.properties的属性文件，里面以user.用户名=密码的格式添加用户账号信息，如下面的样例代码块，详细信息查看随书代码[code/ShiroTP04/src/main/resources/users\_prop.ini](../code/ShiroTP04/src/main/resources/users_prop.properties)。

user.zhangsan=123456

user.lisi=abc123

user.wangwu=abctest

注意，所有的用户名都是以user.开头，后面跟用户名和匹配的密码，格式是：user.<账号名>=<密码>

1. 编写测试代码

编写测试代码TestPropertiesRealm.java代码，内容如下样例代码块，详细内容请参考随书代码[code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/  
TestPropertiesRealm.java](../code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/TestPropertiesRealm.java)。

package cn.com.shiro.book.tp04;

… …

public class TestPropertiesRealm {

public static void main(String[] args) {

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

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

"classpath:shiro\_propRealm.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();

//创建用户名密码登录令牌

UsernamePasswordToken token = new UsernamePasswordToken("lisi","abc123");

currUser.login(token);

System.out.println("登录成功");

currUser.logout();

}

}

1. 运行测试

为了基于exec-maven-plugin插件运行前面定义的TestPropertiesRealm.java类，在pom.xml中按如下代码块修改上面配置过的exec-maven-plugin插件，详细内容请参考随书代码[code/ShiroTP04/pom.xml](../code/ShiroTP04/pom.xml)。

<!-- 该插件用来运行测试，在正式项目中不是必须的 -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<executions>

<execution>

<goals>

<goal>java</goal>

</goals>

</execution>

</executions>

<configuration>

<classpathScope>test</classpathScope>

<mainClass>**cn.com.shiro.book.tp04.TestPropertiesRealm**</mainClass>

</configuration>

</plugin>

注意要准确配置如上粗体显示的mainClass元素内容，否则运行的时候会找到不启动类。

基于Maven运行exec:java，就可以查看运行结果。我们也可以试着将TestPropertiesRealm.java代码中的账号/密码改成错误的，再运行检验是否能正常验证登录。

* 基于JdbcRealm完成身份认证

1. 初始化数据库

在MySQL中创建一个数据库名称为demo\_shiro，在里面创建一张shr\_user表，包含ur\_id、ur\_user\_name、ur\_password和ur\_salt三个自动，分别是用户id、用户名、密码和盐(盐在当前例子中暂时不起作用)，并且插入三条用户数据，样例代码块如下，详细信息可以查看随书代码[code/ShiroTP04/shr\_user.sql](../code/ShiroTP04/shr_user.sql)。

-- ----------------------------

-- Table structure for shr\_user

-- ----------------------------

DROP TABLE IF EXISTS `shr\_user`;

CREATE TABLE `shr\_user` (

`ur\_id` int(11) NOT NULL AUTO\_INCREMENT,

`ur\_user\_name` varchar(20) NOT NULL,

`ur\_password` varchar(100) DEFAULT NULL,

`ur\_salt` varchar(50) DEFAULT NULL,

PRIMARY KEY (`ur\_id`),

UNIQUE KEY `ur\_user\_name` (`ur\_user\_name`)

) ENGINE=InnoDB AUTO\_INCREMENT=42 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records

-- ----------------------------

INSERT INTO `shr\_user` VALUES ('39', 'zhangsan', '123456', 'test');

INSERT INTO `shr\_user` VALUES ('40', 'lisi', 'abc123', 'test');

INSERT INTO `shr\_user` VALUES ('41', 'wangwu', 'abcefg', 'test');

1. 配置JdbcRealm

在resources中创建shiro\_jdbcRealm.ini文件，在里面添加如下代码块，详细信息可以查看随书代码[code/ShiroTP04/src/main/resources/shiro\_jdbcRealm.ini](../code/ShiroTP04/src/main/resources/shiro_jdbcRealm.ini)中的[main]部分。

[main]

#创建apache的dbcp连接池

dataSource=org.apache.commons.dbcp.BasicDataSource

#设置dataSource的连接数据库信息

dataSource.username=root

dataSource.password=root

dataSource.driverClassName=com.mysql.jdbc.Driver

dataSource.url=jdbc:mysql://localhost:3306/demo\_shiro

#配置jdbcRealm对象(realm名称=realm类名)

jdbcRealm=org.apache.shiro.realm.jdbc.JdbcRealm

#设置jdbcRealm的数据源

jdbcRealm.dataSource=$dataSource

#设置jdbcRealm查询用户和密码的查询语句

jdbcRealm.authenticationQuery=select ur\_password,ur\_salt from shr\_user where ur\_user\_name=?

#设置jdbcRealm对象

securityManager.realms=$jdbcRealm

如上代码，我们先要给JdbcRealm准备一个数据源，用来连接应用数据库，我们这里使用的是Apache的dbcp连接池，连接MySQL数据库。

为了让jdbcRealm从应用数据库中查询到账号信息，我们需要在配置文件中设置jdbcRealm.authenticationQuery=select ur\_password,ur\_salt from shr\_user where ur\_user\_name=?，表示从shr\_user表中，根据用户名查找对应的密码和盐值。

1. 编写测试代码

编写测试代码TestJdbcRealm.java代码，内容如下样例代码块，详细内容请参考随书代码[code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/  
TestJdbcRealm.java](../code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/TestPropertiesRealm.java)。

package cn.com.shiro.book.tp04;

… …

public class TestJdbcRealm {

public static void main(String[] args) {

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

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

"classpath:shiro\_propRealm.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();

//创建用户名密码登录令牌

UsernamePasswordToken token = new UsernamePasswordToken("wangwu","abcefg");

currUser.login(token);

System.out.println("登录成功");

currUser.logout();

}

}

因为运行过程中，需要用到Apache dbcp连接池和MySQL数据库的驱动，所有还需要在pom.xml中添加Apache dbcp连接池和MySQL数据库驱动的依赖，样例代码块如下，详细信息请查看随书代码[code/ShiroTP04/pom.xml](../code/ShiroTP04/pom.xml)的依赖配置部分。

<!-- https://mvnrepository.com/artifact/commons-dbcp/commons-dbcp -->

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.4</version>

</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.40</version>

</dependency>

1. 运行测试

为了基于exec-maven-plugin插件运行前面定义的TestJdbcRealm.java类，在pom.xml中按如下代码块修改上面配置过的exec-maven-plugin插件，详细内容请参考随书代码[code/ShiroTP04/pom.xml](../code/ShiroTP04/pom.xml)。

<!-- 该插件用来运行测试，在正式项目中不是必须的 -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<executions>

<execution>

<goals>

<goal>java</goal>

</goals>

</execution>

</executions>

<configuration>

<classpathScope>test</classpathScope>

<mainClass>**cn.com.shiro.book.tp04.TestJdbcRealm**</mainClass>

</configuration>

</plugin>

注意要准确配置如上粗体显示的mainClass元素内容，否则运行的时候会找到不启动类。

基于Maven运行exec:java，就可以查看运行结果。我们也可以试着将TestJdbcRealm.java代码中的账号/密码改成错误的，再运行检验是否能正常验证登录。

* 1. 自定义Realm并认证身份

前面4.3研究了对Shiro安全框架自带的常用Realm的使用，在实际应用中，我们可能需要结合实际情况，自定义Realm。接下来我们介绍怎么自定义Realm并使用自定义Realm。

自定义Realm，原则上可以实现最顶层的Realm接口，也可以根据实际需要，继承Realm的任何一个子类。比如我们只想实现身份的验证，不影响以前的授权实现，就可以继承AuthorizingRealm，只是要注意重写的方法，在整个流程中的角色和作用。接下来我们就从这两个方面阐述自定义Realm。

* 实现Realm接口

1. 编写MyRealm1.java类，实现Realm接口。

样例代码块如下，详细内容可以查看随书代码[code/ShiroTP04/src/main/java/cn/com/shiro/book/realm/MyRealm1.java](../code/ShiroTP04/src/main/java/cn/com/shiro/book/realm/MyRealm1.java)。

package cn.com.shiro.book.realm;

… …

public class MyRealm1 implements Realm {

public String getName() {

return name;

}

public boolean supports(AuthenticationToken token) {

// 只支持UsernamePasswordToken

return token instanceof UsernamePasswordToken;

}

/\*\*

\* 根据token获取应用中匹配上的AuthenticationInfo

\* \*/

public AuthenticationInfo getAuthenticationInfo(AuthenticationToken token)

throws AuthenticationException {

SimpleAuthenticationInfo info = null;

if (token != null) {

String userName = token.getPrincipal().toString();

String password = new String((char[]) token.getCredentials());

// 返回的Principals是输入的用户名+":"+realmName

if ("testUserName".equals(userName)) {

// testUserName用户可以输入任何密码

info = new SimpleAuthenticationInfo("testUserName:"

+ this.getName(), "anyPassword", this.getName());

} else if ((userName + "psw").equals(password)) {

// 其他用户的密码是用户名+psw

info = new SimpleAuthenticationInfo(userName + ":"

+ this.getName(), password, this.getName());

}

}

return info;

}

private String name = "myRealm1";

public void setName(String name) {

this.name = name;

}

}

上面代码的验证逻辑比较简单，只是起Demo作用：如果登录的Principals是testUserName，则任何密码都可以，返回的AuthenticationInfo中的Principals是testUserName+Realm的Name，Credentials是anyPassword；如果是其他Principals，则判断密码是不是Principals+pws，如果是，表示验证成功，返回Principals是传入的Principals+Realm的Name，Credentials是以前的Credentials，否则就返回null。

1. 配置MyRealm1

在resources中创建shiro\_myRealm1.ini文件，在里面添加如下代码块，详细信息可以查看随书代码[code/ShiroTP04/src/main/resources/shiro\_myRealm1.ini](../code/ShiroTP04/src/main/resources/shiro_myRealm1.ini)中的[main]部分。

[main]

#配置myRealm1对象

myRealm1=cn.com.shiro.book.realm.MyRealm1

#设置myReaml1的name

myRealm1.name=realm1

#设置propRealm对象

securityManager.realms=$myRealm1

1. 编写测试代码

编写测试代码TestMyRealm1.java代码，内容如下样例代码块，详细内容请参考随书代码[code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/  
TestMyRealm1.java](../code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/TestMyRealm1.java)。

package cn.com.shiro.book.tp04;

… …

public class TestMyRealm1 {

public static void main(String[] args) {

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

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

"classpath:shiro\_myRealm1.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();

//创建用户名密码登录令牌

UsernamePasswordToken token = new UsernamePasswordToken("lisi","lisipsw");

currUser.login(token);

System.out.println("登录成功");

currUser.logout();

}

}

1. 运行测试

为了基于exec-maven-plugin插件运行前面定义的TestMyRealm1.java类，在pom.xml中按如下代码块修改上面配置过的exec-maven-plugin插件，详细内容请参考随书代码[code/ShiroTP04/pom.xml](../code/ShiroTP04/pom.xml)。

<!-- 该插件用来运行测试，在正式项目中不是必须的 -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<executions>

<execution>

<goals>

<goal>java</goal>

</goals>

</execution>

</executions>

<configuration>

<classpathScope>test</classpathScope>

<mainClass>**cn.com.shiro.book.tp04.TestMyRealm1**</mainClass>

</configuration>

</plugin>

注意要准确配置如上粗体显示的mainClass元素内容，否则运行的时候会找到不启动类。

基于Maven运行exec:java，就可以查看运行结果。我们也可以试着将TestMyRealm1.java代码中的账号/密码改成错误的，再运行检验是否能正常验证登录。

* 继承AuthorizingRealm

同实现Realm接口的流程一样，先编写一个类MyRealm2.java，继承AuthorizingRealm，重写AuthorizingRealm中的doGetAuthenticationInfo方法，样例代码如下，详细内容请查看随书代码[code/ShiroTP04/src/main/java/cn/com/  
shiro/book/realm/MyRealm2.java](../code/ShiroTP04/src/main/java/cn/com/shiro/book/realm/MyRealm2.java)。

package cn.com.shiro.book.realm;

… …

public class MyRealm2 extends AuthorizingRealm {

@Override

protected AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

// 没有做授权控制实现

return null;

}

// 根据token的principals获取AuthenticationInfo，以便同token的Credentials匹配

@Override

protected AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) throws AuthenticationException {

SimpleAuthenticationInfo info = null;

if (token != null) {

String userName = token.getPrincipal().toString();

String password = new String((char[]) token.getCredentials());

if ("testUserName".equals(userName)) {

info = new SimpleAuthenticationInfo("testUserName:"

+ this.getName(), "anyPassword", this.getName());

**UsernamePasswordToken t = (UsernamePasswordToken) token;**

**// 修改token的密码，以便同返回的info密码一致，AuthorizingRealm里面才能成功匹配**

**t.setPassword("anyPassword".toCharArray());**

} else if ((userName + "psw").equals(password)) {

info = new SimpleAuthenticationInfo(userName + ":"

+ this.getName(), password, this.getName());

}

}

return info;

}

}

理解上面代码的doGetAuthenticationInfo方法的实现逻辑前，我们先看看AuthenticatingRealm里面实现身份验证方法getAuthenticationInfo，代码如下：

public final AuthenticationInfo getAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

AuthenticationInfo info = getCachedAuthenticationInfo(token);

if (info == null) {

//otherwise not cached, perform the lookup:

**info = doGetAuthenticationInfo(token); ①**

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) {

**assertCredentialsMatch(token, info); ②**

} else {

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

}

return info;

}

查看上面代码，getAuthenticationInfo方法一个final方法，也就是该方法是不能被重写的，它里面定义了Authenticate的流程，不能被改变。

**①**代码调用了抽象方法doGetAuthenticationInfo，根据token获取应用中，用来匹配验证token是否正确的AuthenticationInfo对象，这需要在子类中做具体实现（我们的MyRealm2中重写的就是这个方法），

**②**代码里面，完成通过doGetAuthenticationInfo方法获取的info同Subject传入的token的匹配验证。

所以MyRealm2.java代码中粗体部分，要修改token中以前的密码，这样才能在getAuthenticationInfo方法中实现正确的anyPassword的匹配。

如前面方式，自定义完MyRealm2类，后面的配置和测试，同前面使用实现Realm方式的自定义MyRealm1一样，我们这里就不重复介绍了，详细配置内容和测试代码，可以查看随书代码：

[code/ShiroTP04/src/main/resources/shiro\_myRealm2.ini](../code/ShiroTP04/src/main/resources/shiro_myRealm2.ini)

[code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/TestMyRealm2.java](../code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/TestMyRealm2.java)

[code/ShiroTP04/pom.xml](../code/ShiroTP04/pom.xml)

* 1. 使用多个Realm完成身份认证

前面4.3节，我们介绍了怎么使用单个的Realm对象，完成身份认证，接下来我们介绍在SecurityManager中设置多个Realm对象，结合不同的AuthenticationStrategy完成身份认证。

1. 配置shiro\_multiRealm.ini，样例代码块如下，详细信息请查看随书代码[code/ShiroTP04/src/main/resources/shiro\_multiRealm.ini](../code/ShiroTP04/src/main/resources/shiro_multiRealm.ini)。

[main]

#创建apache的dbcp连接池

dataSource=org.apache.commons.dbcp.BasicDataSource

#设置dataSource的连接数据库信息

dataSource.username=root

dataSource.password=root

dataSource.driverClassName=com.mysql.jdbc.Driver

dataSource.url=jdbc:mysql://localhost:3306/demo\_shiro

#配置jdbcRealm对象(realm名称=realm类名)

jdbcRealm=org.apache.shiro.realm.jdbc.JdbcRealm

#设置jdbcRealm的数据源

jdbcRealm.dataSource=$dataSource

#设置jdbcRealm查询用户和密码的查询语句

jdbcRealm.authenticationQuery=select ur\_password,ur\_salt from shr\_user where ur\_user\_name=?

#配置myRealm1对象

myRealm1=cn.com.shiro.book.realm.MyRealm1

myRealm1.name=realm1

#配置myRealm2对象

myRealm2=cn.com.shiro.book.realm.MyRealm2

myRealm2.name=realm2

**#指定securityManager的authenticator实现** ①

authenticator=org.apache.shiro.authc.pam.ModularRealmAuthenticator

securityManager.authenticator=$authenticator

**#配置系统默认的三个策略** ②

atLeastStrategy=org.apache.shiro.authc.pam.AtLeastOneSuccessfulStrategy

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

allStrategy=org.apache.shiro.authc.pam.AllSuccessfulStrategy

**#分别在验证器中设置不同的策略，体现不同的认证结果** ③

securityManager.authenticator.authenticationStrategy=$atLeastStrategy

**#设置Realms对象** ④

securityManager.realms= $myRealm2, $jdbcRealm, $myRealm1

查看前面的代码，实际上是前面案例中介绍的shiro\_myRealm1.ini，shir\_myRealm2.ini，shiro\_jdbcRealm.ini的一个组合，不同的是粗体标识部分。①是注册支持多个Realm的验证器，SecurityManager默认就是使用的ModularRealmAuthenticator，实际情况可以不配置这部分。这里配置是为了让我们理解更清晰。

②是Shiro安全框架默认实现的三种AuthenticationStrategy，前面章节有从理论上介绍过它们的区别，这里将在运行后的结果上面，体验它们的区别。

③是指定当前验证器使用哪种验证策略。

④设置多个realms对象。

1. 编写TestMultiRealm.java测试代码，样例代码块如下，详细内容请查看[code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/TestMultiRealm.java](../code/ShiroTP04/src/test/java/cn/com/shiro/book/tp04/TestMultiRealm.java)。

package cn.com.shiro.book.tp04;

… …

public class TestMultiRealm {

public static void main(String[] args) {

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

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

"classpath:shiro\_multiRealm.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();

//创建用户名密码登录令牌

UsernamePasswordToken token = new UsernamePasswordToken("test","testpsw");

currUser.login(token);

System.out.println("登录成功,返回的验证信息是：");

**PrincipalCollection principals = currUser.getPrincipals();**

**List list = principals.asList();**

**for(Object obj:list){**

**System.out.println(obj);**

**}**

currUser.logout();

}

}

上面代码的大部分同以前的测试代码一样，不同的是，最后添加了粗体标识部分内容。该内容的逻辑是输出认证后的所有身份信息。我们在SecurityManager中配置的三个Realm返回的Principals信息是不一样的：JdbcRealm返回的是输入的Principals；MyRealm1和MyRealm2返回的Principals是在输入的Principals后面加上了后缀“:”+realm名。根据ini配置信息知道，MyRealm1的后缀是:realm1，MyRealm2的后缀是:realm2。这样处理的目的是让我们方便测试识别每个Realm的验证。

1. 修改pom.xml的exec-maven-plugin中的mainClass元素内容为cn.com.shiro.book.tp04.TestMultiRealm（同前面测试案例代码一样），详细内容请好看随书代码[code/ShiroTP04/pom.xml](../code/ShiroTP04/pom.xml)。
2. 在shiro\_multiRealm.ini中，调整SecurityManager.authenticator.authentication  
   Strategy的不同策略对象，查看对比不同的结果。
3. atLeastStrategy

策略修改成：

securityManager.authenticator.authenticationStrategy=$atLeastStrategy

输出结果如下：

org.apache.shiro.authc.UnknownAccountException: No account found for user [test]

... ...

登录成功,返回的验证信息是：

test:realm2

test:realm1

因为JdbcRealm的数据源中，没有test这个用户，抛出一个账号不存在异常，同时显示登录成功，打印通过MyRealm2、MyRealm1成功验证后的身份信息。

说明基于AtLeastOneSuccessfulStrategy策略验证的规则是：使用所有Realm进行验证，只要有一个Realm验证成功，就整体验证通过，最后的AuthenticationInfo包含每个验证通过Realm返回的信息。

这也是ModularRealmAuthenticator默认使用的策略。

1. firstStrategy

策略修改成：

securityManager.authenticator.authenticationStrategy=$firstStrategy

输出结果如下:

org.apache.shiro.authc.UnknownAccountException: No account found for user [test]

... ...

登录成功,返回的验证信息是：

test:realm2

同前面的差不多，说明基于FirstSuccessfulStrategy策略验证规则是：使用所有的Realm进行验证，只要有一个验证成功，就整体通过，最后的AuthenticationInfo只包含第一个验证通过Realm返回的信息。

1. allStrategy

策略修改成：

securityManager.authenticator.authenticationStrategy=$allStrategy

输出结果：

org.apache.shiro.authc.UnknownAccountException: No account found for user [test]

... ...

[WARNING]

org.apache.shiro.authc.UnknownAccountException: No account found for user [test]

... ...

at cn.com.shiro.book.tp04.TestMultiRealm.main(TestMultiRealm.java:27)

输出两个No account found异常，第一个是JdbcRealm验证的时候，没有test账号抛出的异常。第二个是最后的登录结果异常。

如果我们接着在JdbcReam连接的shr\_user表中添加一条ur\_user\_name为test、ur\_password为testpsw的记录，再运行测试案例，输出结果是：

登录成功,返回的验证信息是：

test:realm2

test

test:realm1

说明AllSuccessfulStrategy的验证规则是：一定要所有Realm验证通过后，才能整体验证通过，如果都验证通过，返回的AuthenticationInfo中包含所有Ream返回的信息。

最后的正确输出信息，也验证了Realm的验证顺序，也同ini中各个Realm在SecurityManager的realms配置顺序一样($myRealm2, $jdbcRealm, $myRealm1)。

* 1. 总结

本章详细介绍Realm组件的如下内容(体系和身份验证部分内容)：

* Realm组件的体系

Realm组件以Realm接口为最上层接口，里面定义了三个方法：getName、supports和getAuthenticationInfo。getName返回当前Realm对象在SecurityManager中的唯一名称，一定要唯一；supports方法返回是否支持传入的AuthenticationToken的验证；getAuthenticationInfo返回验证通过后Subject的Principals信息，注意是验证通过后才有返回，否则返回null或抛对应的AuthenticationException异常。

* 自定义Realm组件

通过实行Realm接口，或继承Realm的任何一个子类，自定义一个Realm完成身份认证或配合完成身份认证。选择重写哪个方法的时候，一定要理解方法的作用和该方法在身份认证流程中的位置。

* 单个Realm组件的使用

以IniRealm、PropertiesRealm、JdbcRealm和自定义的MyRealm1、MyRealm2为案例说明了使用单个Realm组件的步骤和注意事项。

* 多个Realm组件的使用

以JdbcRealm和MyRealm1、MyRealm2为样例解释了怎样同时使用多个Realm的验证步骤和注意事项。

* 三个自带AuthenticationStrategy的区别

配个多个Realm组件的验证，体验了AtLeastOneSuccessfulStrategy、FirstSuccessfulStragegy和AllSuccessfulStrategy的区别。