Shiro学习笔--快速上手

|  |  |
| --- | --- |
|  | 🢂 内容概览 |
|  | Why：此文档用来做什么？它存在的意义是什么？为解决什么问题？   |  | | --- | |  |   What：当前包含了那些内容？   |  | | --- | |  |   How：此文档应如何参考？   |  | | --- | |  |   Who：此文档适用于那些人员阅读参考？   |  | | --- | |  | |

目录

[1 快速上手 2](#_Toc445124669)

[1.1 示例 2](#_Toc445124670)

[1.1.1 maven配置 2](#_Toc445124671)

[1.1.2 java代码 4](#_Toc445124672)

[1.1.3 log4j.properties 5](#_Toc445124673)

[1.1.4 shiro.ini 6](#_Toc445124674)

[1.2 代码说明 6](#_Toc445124675)

[1.2.1 创建SecurityManager 6](#_Toc445124676)

[1.2.2 获取Subject 7](#_Toc445124677)

[1.2.3 获取用户session 8](#_Toc445124678)

[1.2.4 通过异常判断登录结果 9](#_Toc445124679)

[1.2.5 获取当前用户principal 10](#_Toc445124680)

[1.2.6 判断用户有无角色 10](#_Toc445124681)

[1.2.7 判断用户有无具体权限 10](#_Toc445124682)

[1.2.8 嵌套权限验证 11](#_Toc445124683)

[1.2.9 登出 11](#_Toc445124684)

[2 Shiro-core支持配置方式 11](#_Toc445124685)

[2.1 编程式配置 11](#_Toc445124686)

[2.2 基于INI的配置 11](#_Toc445124687)

[2.2.1 shiro.ini的文件结构说明 12](#_Toc445124688)

[3 如何使用加密后密码 14](#_Toc445124689)

# 快速上手

## 示例

### maven配置

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**groupId**>org.cime.shiro</**groupId**>  <**artifactId**>sample</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>   <**dependencies**>  *<!--shiro 基本模块-->* <**dependency**>  <**groupId**>org.apache.shiro</**groupId**>  <**artifactId**>shiro-core</**artifactId**>  <**version**>1.2.4</**version**>  </**dependency**>   *<!--shiro依赖slf4j-api-->* <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-api</**artifactId**>  <**version**>1.7.13</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-log4j12</**artifactId**>  <**version**>1.7.5</**version**>  </**dependency**>   *<!--shiro 使用ini方式配置,依赖commons-beanutils-->* <**dependency**>  <**groupId**>commons-beanutils</**groupId**>  <**artifactId**>commons-beanutils</**artifactId**>  <**version**>1.8.3</**version**>  </**dependency**>  </**dependencies**>   <**build**>  <**plugins**>  *<!--maven编译插件-->* <**plugin**>  <**artifactId**>maven-compiler-plugin</**artifactId**>  <**version**>3.3</**version**>  <**configuration**>  <**source**>1.8</**source**>  <**target**>1.8</**target**>  </**configuration**>  </**plugin**>   *<!--执行示例-->* <**plugin**>  <**groupId**>org.codehaus.mojo</**groupId**>  <**artifactId**>exec-maven-plugin</**artifactId**>  <**version**>1.1</**version**>  <**executions**>  <**execution**>  <**goals**>  <**goal**>java</**goal**>  </**goals**>  </**execution**>  </**executions**>  <**configuration**>  <**classpathScope**>test</**classpathScope**>  <**mainClass**>org.cime.quickstart.QuickStart</**mainClass**>  </**configuration**>  </**plugin**>  </**plugins**>  </**build**> </**project**> |

### java代码

|  |
| --- |
| **package** org.cime.quickstart;  **import** org.apache.shiro.SecurityUtils; **import** org.apache.shiro.authc.AuthenticationException; **import** org.apache.shiro.authc.IncorrectCredentialsException; **import** org.apache.shiro.authc.LockedAccountException; **import** org.apache.shiro.authc.UnknownAccountException; **import** org.apache.shiro.authc.UsernamePasswordToken; **import** org.apache.shiro.config.IniSecurityManagerFactory; **import** org.apache.shiro.mgt.SecurityManager; **import** org.apache.shiro.session.Session; **import** org.apache.shiro.subject.Subject; **import** org.apache.shiro.util.Factory; **import** org.slf4j.Logger; **import** org.slf4j.LoggerFactory;  */\*\*  \* shiro示例01--quickstart  \*/* **public class** QuickStart {   **private static final transient** Logger ***log*** = LoggerFactory.*getLogger*(QuickStart.**class**);   **public static void** main(String[] args) {   *//1.创建SecurityManager* Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(**"classpath:shiro.ini"**);  SecurityManager securMgr = factory.getInstance();   *//2.获取Subject* SecurityUtils.*setSecurityManager*(securMgr);  Subject currUser = SecurityUtils.*getSubject*();   *//3.获取session* Session session = currUser.getSession();   *//4.在session存放属性* session.setAttribute(**"name"**, **"张三丰"**);  ***log***.info(session.getAttribute(**"name"**).toString());   *//5.检查登录操作,如果未登录,则执行登录操作* **if** (!currUser.isAuthenticated()) {  **try** {  *//5.1 获取用户名和密码,创建token* UsernamePasswordToken token = **new** UsernamePasswordToken(**"zhangsan"**, **"passwd"**);  token.setRememberMe(**true**);   *//5.2 通过token执行登录操作* currUser.login(token);   *//通过异常划分认证失败的场景  //注:出于安全考虑,不应该给出认证失败的细节原因,只给出笼统提示即可* } **catch** (UnknownAccountException | IncorrectCredentialsException | LockedAccountException e) {  ***log***.info(**"用户帐号无效或者密码错误!"**);  *//认证类异常基类* } **catch** (AuthenticationException e) {  ***log***.info(**"因未知原因,登录失败!"**);  }  }   *//6.登录后,给出用户提示* ***log***.info(**"Hello "** + currUser.getPrincipal() + **" , welcome to our system!"**);   *//7.用户访问权限判断  //判断用户是否具有角色* **if** (currUser.hasRole(**"PM"**)) {  ***log***.info(**"You are PM!"**);  } **else** {  ***log***.info(**"You are not a PM!"**);  }   *//判断用户是否具有权限* **if** (currUser.isPermitted(**"coding"**)) {  ***log***.info(**"you can enjoy youer coding works!"**);  } **else** {  ***log***.info(**"Buddy,can you coding!"**);  }   *//判断是否具有嵌套权限* **if** (currUser.isPermitted(**"server:online:operation:view"**)) {  ***log***.info(**"you can play all the servers online!"**);  } **else** {  ***log***.info(**"Sorry,don't touch my precious!"**);  }   *//8.登出* currUser.logout();  System.*exit*(0);  } } |

### log4j.properties

|  |
| --- |
| **log4j.rootLogger**=**INFO, stdout  log4j.appender.stdout**=**org.apache.log4j.ConsoleAppender log4j.appender.stdout.layout**=**org.apache.log4j.PatternLayout log4j.appender.stdout.layout.ConversionPattern**=**%d %p [%c] - %m %n** *# General Apache libraries* **log4j.logger.org.apache**=**WARN** *# Default Shiro logging* **log4j.logger.org.apache.shiro**=**INFO** *# Disable verbose logging* **log4j.logger.org.apache.shiro.util.ThreadContext**=**WARN log4j.logger.org.apache.shiro.cache.ehcache.EhCache**=**WARN** |

### shiro.ini

|  |
| --- |
| *# 定义用户,及用户所属的角色 # 格式:用户名=密码,角色1[,角色2...角色n]* **[users] zhangsan** = **passwd,PM,RD lisi** = **passwd,RD,OP wangwu** = **passwd,QA** *# 定义角色,及角色所包含的权限 # 格式:角色名=权限1[,权限2...权限n] # 权限可以嵌套,多层嵌套以:分隔 # \*表示具有当前层次所有权限* **[roles]** *# 项目经理角色,具有所有权限* **PM** = **\*** *# RD,具有编码权限* **RD** = **coding** *# QA,具有测试,线上服务器查看权限* **QA** = **test,server:online:operation:view** *# OP,具有线上服务器操作权限* **OP** = **server:online:operation:\*** |

## 代码说明

### 创建SecurityManager

|  |
| --- |
| *//1.创建SecurityManager* Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(**"classpath:shiro.ini"**); SecurityManager securMgr = factory.getInstance(); |

示例中我们使用的是Ini配置方式创建SecurtiyManager实例，先创建工厂类，传入配置文件路径，然后通过工厂类”生产”相应的实例，创建SecurityManager用到了工厂模式。Factory作为工厂类的基类，SecurityManager作为工厂要生产的产品。shiro使用不同的工厂类创建不同的SecurityManager,以应对不同的使用场景；

Factory的接口定义如下，所有工厂的实现类通过getInstacne()方法获取SecurityManager实例；

|  |
| --- |
| **package** org.apache.shiro.util;  */\*\*  \* Generics-aware interface supporting the  \* <a href="http://en.wikipedia.org/wiki/Factory\_method\_pattern">Factory Method</a> design pattern.  \*  \** ***@param <T>*** *The type of the instance returned by the Factory implementation.  \** ***@since*** *1.0  \*/* **public interface** Factory<T> {   */\*\*  \* Returns an instance of the required type. The implementation determines whether or not a new or cached  \* instance is created every time this method is called.  \*  \** ***@return*** *an instance of the required type.  \*/* T getInstance(); } |

Shiro-core中Factory的实现类如下：

|  |
| --- |
|  |

复习：工厂模式类图（摘自网络）

|  |
| --- |
|  |

### 获取Subject

|  |
| --- |
| *//2.获取Subject* SecurityUtils.*setSecurityManager*(securMgr); Subject currUser = SecurityUtils.*getSubject*(); |

需要先在SecurityUtils中设置SecurityManager实例，然后通过SecurityUtils的getSubject（）获取Subject实例。SecurityUtils中的SecurityManager实例是作为备份使用的，通常会优先获取ThreadContext中的实例使用，只有当程序不依赖其它session运行时，才使用SecurityUtils中的实例。相关代码如下：

|  |
| --- |
| */\*\*  \* ONLY used as a 'backup' in VM Singleton environments (that is, standalone environments), since the  \* ThreadContext should always be the primary source for Subject instances when possible.  \*/* **private static** SecurityManager *securityManager*;  **public static void** setSecurityManager(SecurityManager securityManager) {  SecurityUtils.*securityManager* = securityManager; }  */\*\*  \* Returns the SecurityManager accessible to the calling code.  \* <p/>  \* This implementation favors acquiring a thread-bound {****@code*** *SecurityManager} if it can find one. If one is  \* not available to the executing thread, it will attempt to use the static singleton if available (see the  \* {****@link*** *#setSecurityManager setSecurityManager} method for more on the static singleton).  \* <p/>  \* If neither the thread-local or static singleton instances are available, this method throws an  \* {****@code*** *UnavailableSecurityManagerException} to indicate an error - a SecurityManager should always be accessible  \* to calling code in an application. If it is not, it is likely due to a Shiro configuration problem.  \*  \** ***@return*** *the SecurityManager accessible to the calling code.  \** ***@throws*** *UnavailableSecurityManagerException  \* if there is no {****@code*** *SecurityManager} instance available to the  \* calling code, which typically indicates an invalid application configuration.  \*/* **public static** SecurityManager getSecurityManager() **throws** UnavailableSecurityManagerException {  SecurityManager securityManager = ThreadContext.*getSecurityManager*();  **if** (securityManager == **null**) {  securityManager = SecurityUtils.*securityManager*;  }  **if** (securityManager == **null**) {  String msg = **"No SecurityManager accessible to the calling code, either bound to the "** +  ThreadContext.**class**.getName() + **" or as a vm static singleton. This is an invalid application "** +  **"configuration."**;  **throw new** UnavailableSecurityManagerException(msg);  }  **return** securityManager; } |

### 获取用户session

|  |
| --- |
| *//3.获取session* Session session = currUser.getSession();  *//4.在session存放属性* session.setAttribute(**"name"**, **"张三丰"**); ***log***.info(session.getAttribute(**"name"**).toString()); |

通过Subject的getSession()方法可以获取当前用户的session对象。此处的Session类似于HttpSession，但是shiro的session是可以独立创建和使用的，不依赖于Servlet,Ejb等具体模块；

### 通过异常判断登录结果

|  |
| --- |
| *//5.检查登录操作,如果未登录,则执行登录操作* **if** (!currUser.isAuthenticated()) {  **try** {  *//5.1 获取用户名和密码,创建token* UsernamePasswordToken token = **new** UsernamePasswordToken(**"zhangsan"**, **"passwd"**);  token.setRememberMe(**true**);   *//5.2 通过token执行登录操作* currUser.login(token);   *//通过异常划分认证失败的场景  //注:出于安全考虑,不应该给出认证失败的细节原因,只给出笼统提示即可* } **catch** (UnknownAccountException | IncorrectCredentialsException | LockedAccountException e) {  ***log***.info(**"用户帐号无效或者密码错误!"**);  *//认证类异常基类* } **catch** (AuthenticationException e) {  ***log***.info(**"因未知原因,登录失败!"**);  } } |

shiro的认证通过不同类别的异常进行判断，认证类异常的基类是AuthenticationException，继承关系如下：

|  |
| --- |
|  |

注：处于安全考虑，出于安全考虑,不应该给出认证失败的细节原因,只给出笼统提示即可。

### 获取当前用户principal

principal，相当于是用户的标识。

|  |
| --- |
| *//6.登录后,给出用户提示* ***log***.info(**"Hello "** + currUser.getPrincipal() + **" , welcome to our system!"**); |

### 判断用户有无角色

|  |
| --- |
| *//判断用户是否具有角色* **if** (currUser.hasRole(**"PM"**)) {  ***log***.info(**"You are PM!"**); } **else** {  ***log***.info(**"You are not a PM!"**); } |

### 判断用户有无具体权限

|  |
| --- |
| *//判断用户是否具有权限* **if** (currUser.isPermitted(**"coding"**)) {  ***log***.info(**"you can enjoy youer coding works!"**); } **else** {  ***log***.info(**"Buddy,can you coding!"**); } |

### 嵌套权限验证

|  |
| --- |
| *//判断是否具有嵌套权限* **if** (currUser.isPermitted(**"server:online:operation:view"**)) {  ***log***.info(**"you can play all the servers online!"**); } **else** {  ***log***.info(**"Sorry,don't touch my precious!"**); } |

### 登出

|  |
| --- |
| *//8.登出* currUser.logout(); |

# Shiro-core支持配置方式

为了保证shiro能够适应多样的环境，shiro提供了多种配置方式。其中shiro-core提供了编程式配置方式和基于INI文件的配置方式。

## 编程式配置

通过代码在程序中完成shiro的配置，并使用，这种方式是最简单的方式；

|  |
| --- |
| Realm realm = //instantiate or acquire a Realm instance. We'll discuss Realms later.  SecurityManager securityManager = new DefaultSecurityManager(realm);  //Make the SecurityManager instance available to the entire application via static memory:  SecurityUtils.setSecurityManager(securityManager); |

SecurityManager被设计成低耦合的类，起内部依赖的组件能够通过setter注入进行灵活替换，方便定制开发。

|  |
| --- |
| DefaultSecurityManager securityManager = new DefaultSecurityManager(realm);  SessionDAO sessionDAO = new CustomSessionDAO();  ((DefaultSessionManager)securityManager.getSessionManager()).setSessionDAO(sessionDAO); |

## 基于INI的配置

shiro提供了基于ini文件的配置方式，能够在ini文件中配置用户-角色-权限关系。通过IniSecurityManagerFactory指定配置文件路径，并创建基于INI配置的SecurityManager。配置文件可以是类路径、合法的URL等。

##### 通过配置文件创建securityManager；

|  |
| --- |
| *//1.创建SecurityManager* Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(**"classpath:shiro.ini"**); SecurityManager securMgr = factory.getInstance(); |

##### 通过INI对象创建SecurityManager;

Shiro提供了解析Ini文件的工具类，可以解析ini文件。

|  |
| --- |
| Ini ini = **new** Ini(); ini.loadFromPath(**"classpath:shiro.ini"**); Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(ini); SecurityManager securMgr = factory.getInstance(); |

### shiro.ini的文件结构说明

|  |
| --- |
| # =======================  # Shiro INI configuration  # =======================  [main]  # Objects and their properties are defined here,  # Such as the securityManager, Realms and anything  # else needed to build the SecurityManager  [users]  # The 'users' section is for simple deployments  # when you only need a small number of statically-defined  # set of User accounts.  [roles]  # The 'roles' section is for simple deployments  # when you only need a small number of statically-defined  # roles.  [urls]  # The 'urls' section is used for url-based security  # in web applications. We'll discuss this section in the  # Web documentation |

##### [main]

用来配置SecurityManager的属性,以及SecurityManager子模块的属性；

|  |
| --- |
| [main]  sha256Matcher = org.apache.shiro.authc.credential.Sha256CredentialsMatcher  myRealm = com.company.security.shiro.DatabaseRealm  myRealm.connectionTimeout = 30000  myRealm.username = jsmith  myRealm.password = secret  myRealm.credentialsMatcher = $sha256Matcher  securityManager.sessionManager.globalSessionTimeout = 1800000 |

Shiro实现了简单的注入框架，解析ini文件中的属性并注入到相应对象中，该功能依赖于commons-beanutils;

|  |
| --- |
|  |

注：使用ini方式注入属性，具有顺序性，调用对象的setter方法时，会按照ini文件中的配置顺序注入；

|  |
| --- |
|  |

##### [users]

用来配置系统包含的用户以及用户的密码，角色；适合于系统用户很少或者不需要动态创建用户的情况；

|  |
| --- |
| *# 定义用户,及用户所属的角色 # 格式:用户名=密码,角色1[,角色2...角色n]* **[users] zhangsan** = **passwd,PM,RD lisi** = **passwd,RD,OP wangwu** = **passwd,QA** |

定义[users]或者[roles]，将会触发shiro创建IniRealm，作为默认的Realm；

|  |
| --- |
|  |

##### [roles]

用来配置角色以及角色包含的权限；适合于系统角色很少或者不需要动态创建角色的情况；

|  |
| --- |
| *# 定义角色,及角色所包含的权限 # 格式:角色名=权限1[,权限2...权限n] # 权限可以嵌套,多层嵌套以:分隔 # \*表示具有当前层次所有权限* **[roles]** *# 项目经理角色,具有所有权限* **PM** = **\*** *# RD,具有编码权限* **RD** = **coding** *# QA,具有测试,线上服务器查看权限* **QA** = **test,server:online:operation:view** *# OP,具有线上服务器操作权限* **OP** = **server:online:operation:\*** |

注：

1. 为避免解析错误，当值中存在特殊字符时，使用双引号括住；
2. 当角色没有配置权限时，无须在[roles]中定义角色，直接在[user]中关联角色即可；

##### [url]

This section and its options is described in the [Web](http://shiro.apache.org/web.html) chapter.

# 如何使用加密后密码