# Shiro 构建安全的企业级应用

Apache Shiro 是一个强大的，易用的Java 安全框架。它被用作于认证，授权，加密，session管理。依赖于Shiro简单易懂的API，你就可以快速的构建包括手机，大型web和商业应用。

## 权限基础

**两个基本的概念**

——安全实体：就是被权限系统保护的对象，比如工资数据。

——权限：就是需要被校验的行为，比如查看、修改等。

**分配权限**：

——把对某些安全实体的某些权限分配给某些人员。

——是向数据库里面添加数据、或是维护数据的过程

**权限验证**（权限匹配）：

——判断某个人员或程序对某个安全实体是否拥有某个或某些权限。

——从数据库中获取相应数据进行匹配的过程。

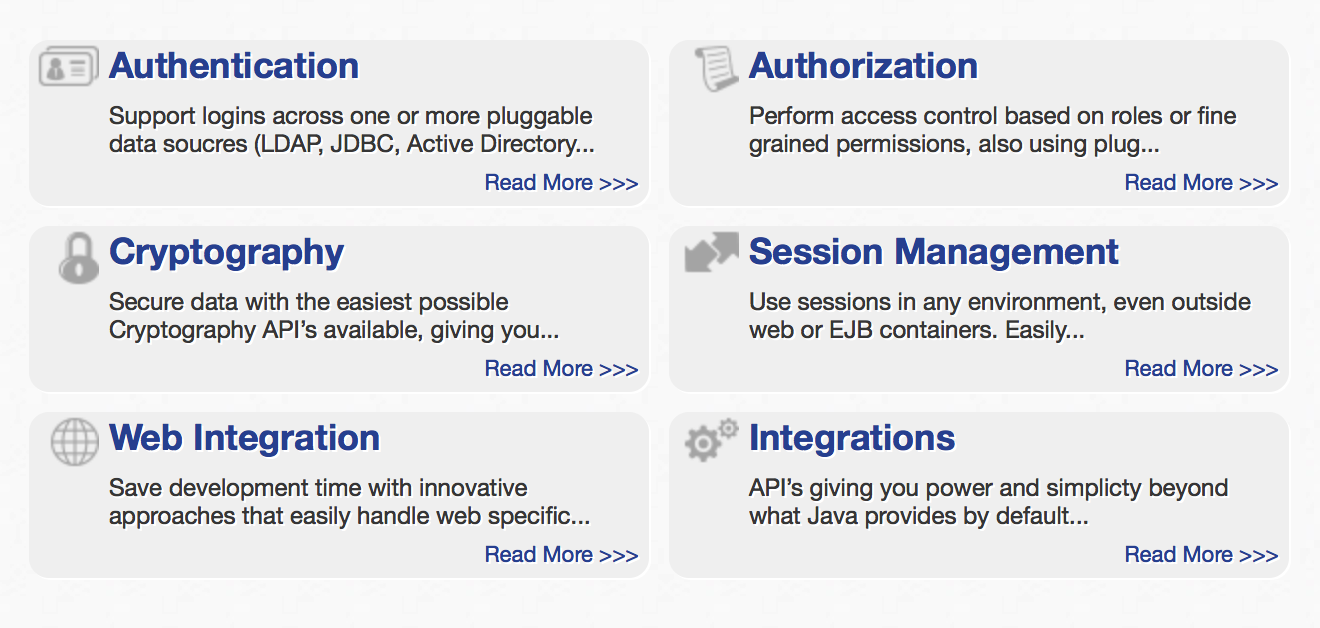
**权限的继承性**：

如果多个安全实体存在包含关系，而某个安全实体没有权限限制，则它会继承包含它的安全实体的相应权限。

**权限的最近匹配原则：**

如果多个安全实体存在包含关系，而某个安全实体没有权限限制，那么它会向上寻找并匹配相应权限限制，直到找到一个离这个安全实体最近的拥有相应权限限制的安全实体为止。如果把整个层次结构都寻找完了都没有匹配到相应权限限制的话，那就说明所有人对这个安全实体都拥有这个相应的权限限制。

## Shiro架构与功能介绍



Authentication : **认证。即验证是哪个用户登录。**

Authorization：也被称为访问控制，即决定当前登录用户是否有权限去访问受保护的资源。

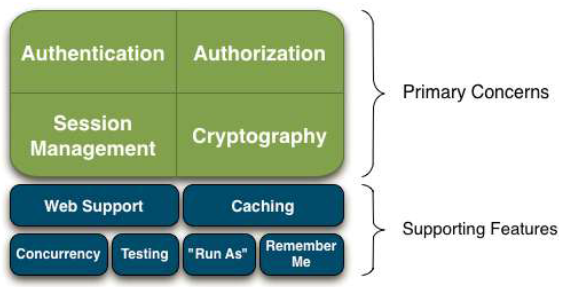
Cryptography：通过加密算法保护数据安全。

Session Management: 当用户使用你的应用是自身携带的数据。

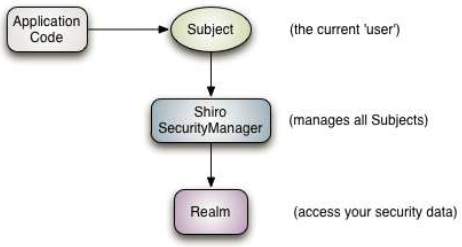
Web Integration: 虽然Shiro是设计用来确保任何基于JVM的应用，但是很多时候是用来保护Web应用。

Integrations:可以很好的集成其他的技术和框架。

下图展示了Shiro的的四大核心功能和所支持的其它特性。

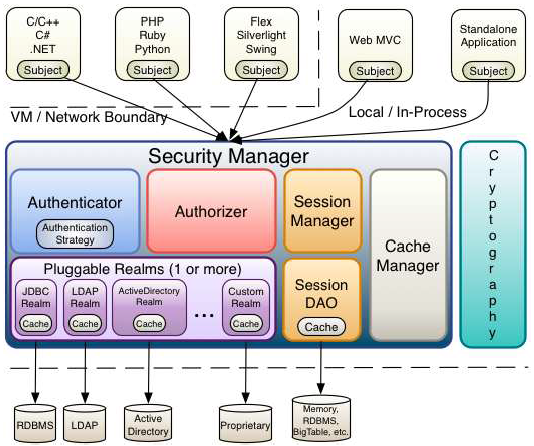


三大核心组件。

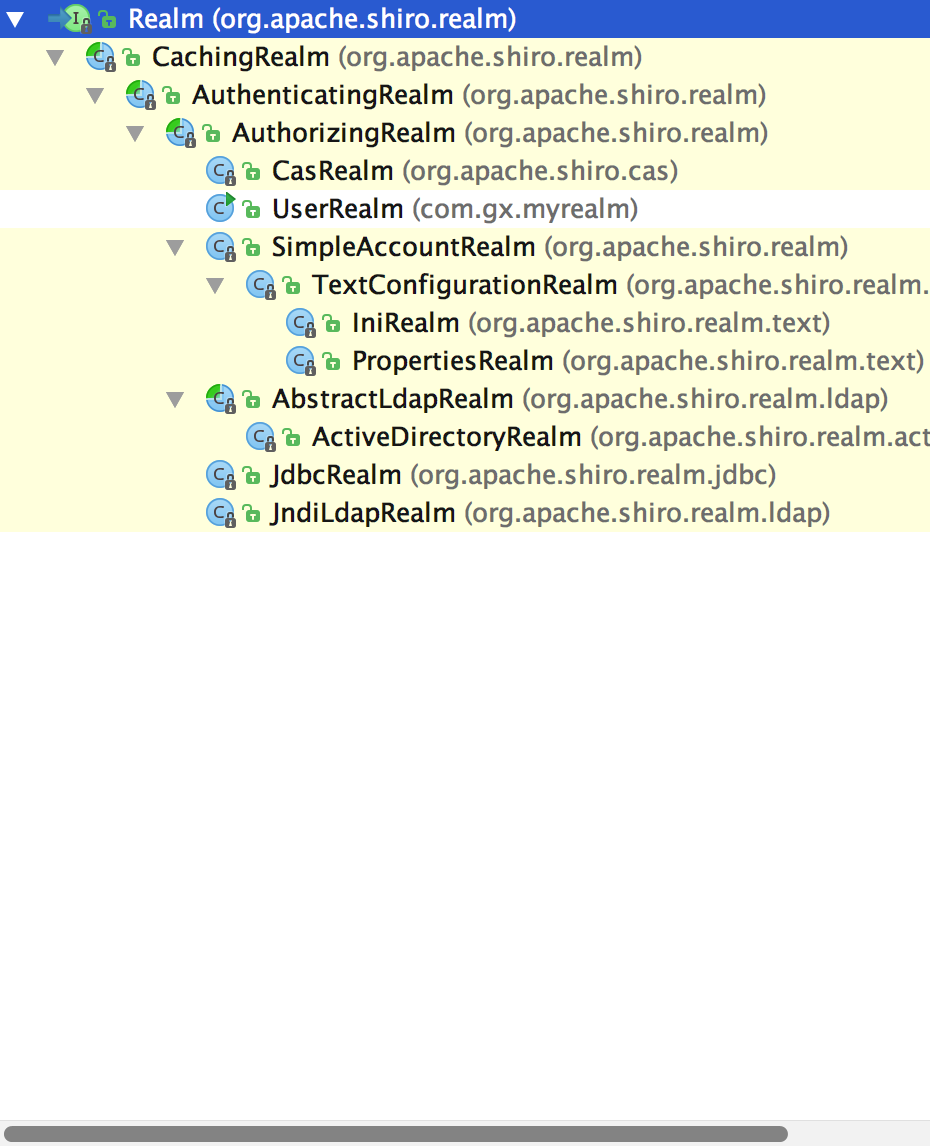


* **Subject** ：**正与系统进行交互的人，或某一个第三方服务**。所有 Subject 实例都被绑定到（且这是必须的）一个SecurityManager 上。
* **SecurityManager**：Shiro 架构的心脏，用来协调内部各安全组件，管理内部组件实例，并通过它来提供安全管理的各种服务。**当 Shiro 与一个 Subject 进行交互时，实质上是幕后的 SecurityManager 处理所有繁重的 Subject 安全操作**。
* **Realms** ：本质上是一个特定安全的 DAO。**当配置 Shiro 时，必须指定至少一个 Realm 用来进行身份验证和/或授权**。Shiro 提供了多种可用的 Realms 来获取安全相关的数据。如关系数据库(JDBC)，INI 及属性文件等。可以定义自己 Realm 实现来代表自定义的数据源。

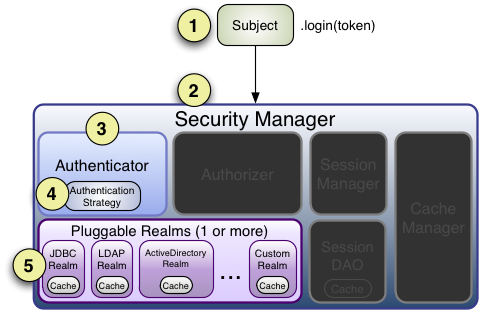
Shiro架构图

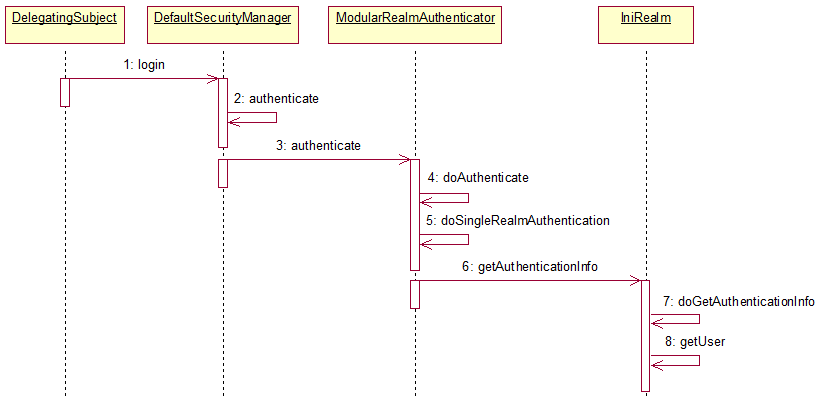


1. Authenticator ：执行对用户的身份验证（登录）的组件。Authenticator 从一个或多个 Realm 中获得数据以验证用户的身份。
2. 若存在多个realm，则接口 AuthenticationStrategy 会确定什么样算是验证成功（例如，如果一个 Realm 成功，而其他的均失败，是否登录成功）。
3. Authorizer ：验证用户能否访问应用中的受保护的资源
4. SessionManager ：可在任何应用或架构层一致地使用 Session API
5. SessionDAO：SessionManager 执行 Session 持久化（CRUD）操作。
6. CacheManager ：对 Shiro 组件提供缓存支持。
7. Cryptography：Shiro 的 API 大幅度简化 Java API 中繁琐的密码加密
8. Realms：Shiro 通过 Realms 来获取相应的安全数据



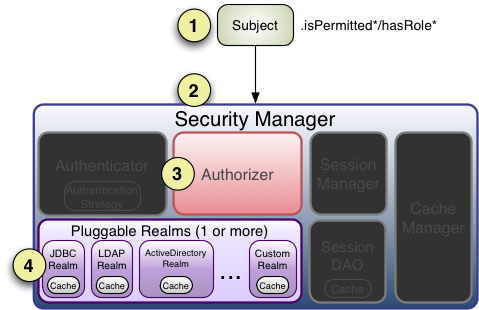
认证

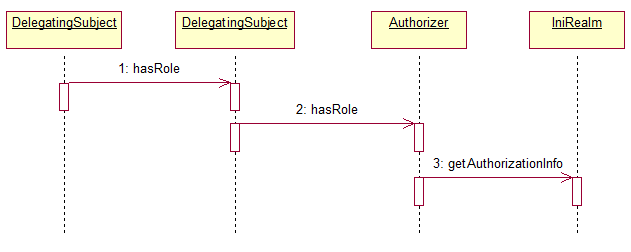




* Step 1：应用程序代码**调用 Subject.login 方法**，传递创建好的包含终端用户的 Principals(身份)和 Credentials(凭证)的 AuthenticationToken 实例
* Step 2：Subject 实例，通常为 DelegatingSubject（或子类）委托应用程序的 SecurityManager 通过调用 securityManager.login(token) 开始真正的验证。
* Step3：SubjectManager 接收 token，调用内部的 Authenticator 实例调用 authenticator.authenticate(token)。 Authenticator 通常是一个 ModularRealmAuthenticator 实例，支持在身份验证中协调一个或多个Realm 实例。
* Step 4：如果应用程序中配置了一个以上的 Realm，ModularRealmAuthenticator 实例将利用配置好的AuthenticationStrategy 来启动 Multi-Realm 认证尝试。在Realms 被身份验证调用之前，期间和以后，AuthenticationStrategy 被调用使其能够对每个Realm 的结果作出反应。
* Step 5：每个配置的 Realm 用来帮助看它是否支持提交的AuthenticationToken。如果支持，那么支持 Realm 的 getAuthenticationInfo 方法将会伴随着提交的 token 被调用。getAuthenticationInfo 方法有效地代表一个特定 Realm 的单一的身份验证尝试。

授权

[](http://shiro.apache.org/architecture.html)



* Step 1：应用程序或框架代码调用任何 Subject 的hasRole\*, checkRole\*, isPermitted\*,或者checkPermission\*方法的变体，传递任何所需的权限
* Step 2：Subject 的实例—通常是 DelegatingSubject(或子类)，调用securityManager 的对应的方法。
* Step 3：SecurityManager 调用 org.apache.shiro.authz.Authorizer 接口的对应方法。默认情况下，authorizer 实例是一个 ModularRealmAuthorizer 实例，它支持协调任何授权操作过程中的一个或多个Realm 实例。
* Step 4：每个配置好的 Realm 被检查是否实现了相同的 Authorizer 接口。如果是，Realm 各自的 hasRole\*, checkRole\*,isPermitted\*，或 checkPermission\* 方法将被调用。

注销

* **logout(注销)**：currentUser.logout();
* 调用 logout() 方法时，现有 Session 将失效，而且身份将失去关联（在Web 应用程序中，RememberMe cookie **将**被删除）。
* 在 Subject 注销后，该 Subject 的实例被再次认为是匿名的。
* **注意**：WEB 应用程序记住身份往往依靠 Cookie，然而Cookie 只能在 Response 被返回后被删除，所以**建议在调用subject.logout() 后立即向终端重定向一个新的视图或页面**。这样即能保证与安全相关的 Cookie 都能像预期的一样被删除。

<https://www.infoq.com/articles/apache-shiro>

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

Shiro中默认的过滤器



基于注解

* **Shiro提供的注解**
  + @RequiresAuthentication ：要求当前 Subject 已经在当前的session 中被验证通过才能被注解的类/实例/方法访问或调用。
  + @RequiresGuest ：要求当前的 Subject 是一个 “guest”，也就是他们必须是在之前的 session 中没有被验证或记住才能被注解的类/实例/方法访问或调用。
  + @RequiresPermissions：要求当前的 Subject 被允许一个或多个权限，以便执行注解的方法，比如：@RequiresPermissions("account:create")
  + @RequiresRoles：要求当前的 Subject 拥有所有指定的角色。如果他们没有，则该方法将不会被执行，而且 AuthorizationException 异常将会被抛出。比如：@RequiresRoles("administrator")
  + @RequiresUser：需要当前的Subject 是一个应用程序用户才能被注解的类/实例/方法访问或调用。要么是通过验证被确认，或者在之前session 中的 'RememberMe‘ 服务被记住。

<!-- Enable Shiro Annotations for Spring-configured beans. Only run after -->

<!-- the lifecycleBeanProcessor has run: -->

<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator" depends-on="lifecycleBeanPostProcessor"/>

<bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">

<property name="securityManager" ref="securityManager"/>

</bean>

基于标签

<%@ taglib prefix="shiro"uri="http://shiro.apache.org/tags" %>

guest 标签

guest 标签将显示它包含的内容，仅当当前的Subject 被认为是 guest 时。

guest 是指没有身份 ID 的任何 Subject：没有登录也没有在上一次的访问中被记住（RememberMe 服务）

guest 标签与user 标签逻辑相反。

<shiro:guest>

Hi there! Please <a href="login.jsp">Login</a> or

<a href="signup.jsp">Signup</a> today!

</shiro:guest>

user 标签

user 标签将显示它包含的内容，仅当当前的 Subject 被认为是 user 时。

user 在上下文中被定义为一个已知身份 ID 的 Subject，或是成功通过身份验证及通过 RememberMe 服务的。

该标签在语义上与authenticated 标签是不同的，authenticated 标签更为严格。

User 标签与guest 标签逻辑相反。

authenticated 标签

当当前用户在当前会话中成功地通过了身份验证 authenticated 标签才会显示包含的内容。比 user 标签更为严格。在逻辑上与 notAuthenticated 标签相反。

<shiro:authenticated>

<a href="updateAccount.jsp">Update your contact information</a>.

</shiro:authenticated>

notAuthenticated 标签

当前 Subject 还没有在其当前会话中成功地通过验证，将会显示它所包含的内容

<shiro:notAuthenticated>

Please <a href="login.jsp">login</a> in order to update your credit card information.

</shiro:notAuthenticated>

principal 标签

标签将会输出Subject 的主体（标识属性）或主要的属性

Hello, <shiro:principal/>, how are you today?

**Typed principal**

User ID: <principal type="java.lang.Integer"/>

**Principal property**

Hello, <shiro:principal property="firstName"/>, how are you today?

hasRole 标签

当当前Subject 被分配了具体的角色时显示它所包含的内容。hasRole 标签与lacksRole 标签逻辑相反。

<shiro:hasRole name="administrator">

<a href="admin.jsp">Administer the system</a>

</shiro:hasRole>

lacksRole 标签

<shiro:lacksRole name="administrator">

Sorry, you are not allowed to administer the system.

</shiro:lacksRole>

hasAnyRole 标签

当前的 Subject 被分配了任意一个来自于逗号分隔的角色名列表中的具体角色，将会显示它所包含的内容。

<shiro:hasAnyRoles name="developer, project manager, administrator">

You are either a developer, project manager, or administrator.

</shiro:lacksRole>

hasPermission 标签

当前 Subject 拥有特定的权限时，会显示它所包含的内容。hasPermission 标签与 lacksPermission 标签逻辑相反。

<shiro:hasPermission name="user:create">

<a href="createUser.jsp">Create a new User</a>

</shiro:hasPermission>

lacksPermission 标签

当前Subject 没有拥有（蕴含）特定的权限，将会显示它所包含的内容。也就是说，用户没有特定的能力。

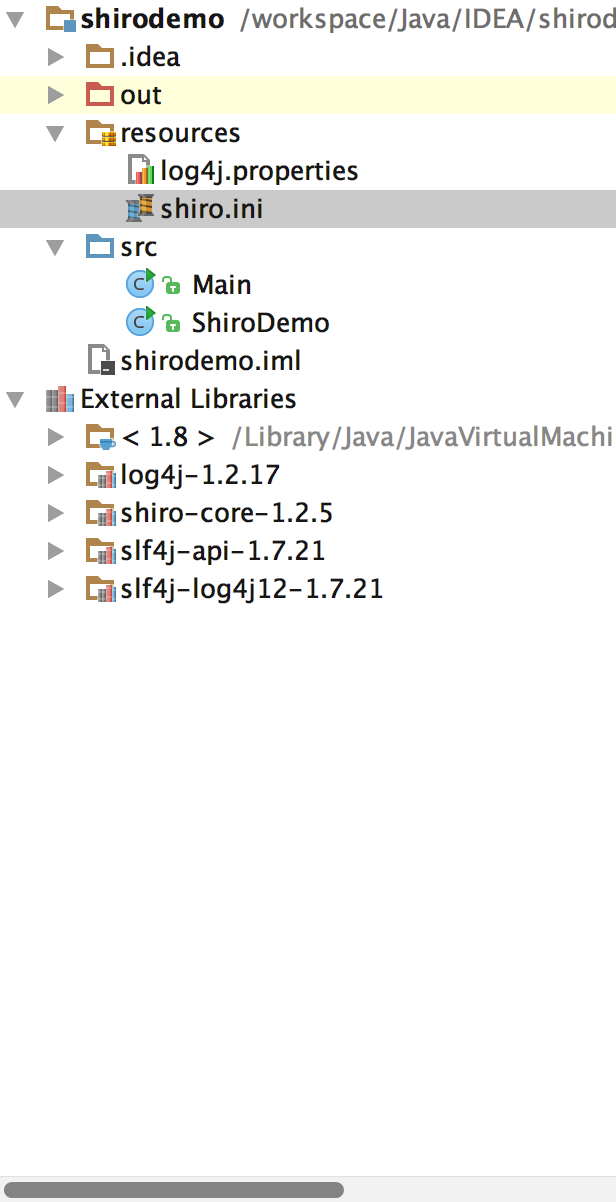
<shiro:lacksPermission name="user:delete">

Sorry, you are not allowed to delete user accounts.

</shiro:hasPermission>

## 简单例子

加入jar 包,log4j 和shiro的配置文件。



shiro.ini

|  |
| --- |
| *# ----------------------------------------------------------------------------- # 用户及其所分配的角色 # # Each line conforms to the format defined in the # org.apache.shiro.realm.text.TextConfigurationRealm#setUserDefinitions JavaDoc # -----------------------------------------------------------------------------* **[users]** *# user 'root' with password 'secret' and the 'admin' role* **root** = **secret, admin** *# user 'guest' with the password 'guest' and the 'guest' role* **guest** = **guest, guest** *# user 'presidentskroob' with password '12345' ("That's the same combination on # my luggage!!!" ;)), and role 'president'* **presidentskroob** = **12345, president** *# user 'darkhelmet' with password 'ludicrousspeed' and roles 'darklord' and 'schwartz'* **darkhelmet** = **ludicrousspeed, darklord, schwartz** *# user 'lonestarr' with password 'vespa' and roles 'goodguy' and 'schwartz'* **lonestarr** = **vespa, goodguy, schwartz** *# ----------------------------------------------------------------------------- # 角色及其分配的权限 #  # Each line conforms to the format defined in the # org.apache.shiro.realm.text.TextConfigurationRealm#setRoleDefinitions JavaDoc # -----------------------------------------------------------------------------* **[roles]** *# 'admin' role has all permissions, indicated by the wildcard '\*'* **admin** = **\*** *# The 'schwartz' role can do anything (\*) with any lightsaber:* **schwartz** = **lightsaber:\*** *# The 'goodguy' role is allowed to 'drive' (action) the winnebago (type) with # license plate 'eagle5' (instance specific id)* **goodguy** = **winnebago:drive:eagle5** |

[users]部分允许定义一组静态的用户帐户

* + 每行的格式：**username = password, roleName1, roleName2, …**
* **[roles] 部分允许把定义在 [users] 部分中的角色与权限关联起来**
  + 每行的格式：**rolename = permissionDefinition1, permissionDefinition2, …**
  + permissionDefinition 是一个任意的字符串，但大多数人将会使用符合 org.apache.shiro.authz.permission.WildcardPermission 格式的字符串。
  + 注意：
    - **如果一个独立的 permissionDefinition 需要被内部逗号分隔**（例如，printer:5thFloor:print,info），**则需要用户双引号环绕该定义，以避免错误解析**。
    - **如果角色不想关联权限，则不需要在 [roles] 部分把它们列出来。只需定义在 [user] 部分中定义角色名就足以创建尚不存在的角色**。
* 仅定义非空的 [users] 或 [roles] 部分就将自动地触发org.apache.shiro.realm.text.IniRealm 实例的创建

shiro 的permissions

* 基础语法之**简单的字符串**：即用简单的字符串来表示一个权限，如：user （相当于：user:\*）
* 基础语法之**多层次管理**：
  + 例如：user:query、user:edit
  + **冒号是一个特殊字符，它用来分隔权限字符串的下一部件**：第一部分是权限被操作的领域（打印机），第二部分是被执行的操作。
  + 多个值：**每个部件能够保护多个值**。因此，除了授予用户 user:query 和 user:edit 权限外，也**可以简单地授予他们一个：user:query, edit**
  + 还可以**用 \* 号代替所有的值**，如：user:\* ， 也可以写：\*:query，表示某个用户在所有的领域都有 query 的权限
* 基础语法之**实例级访问控制**
  + 这种情况通常会使用三个部件：**域、操作、被付诸实施的实例**。如：user:edit:manager
  + 也**可以使用通配符**来定义，如：user:edit:\*、user:\*:\*、user:\*:manager
  + **部分省略通配符**：缺少的部件意味着用户可以访问所有与之匹配的值，比如：user:edit  等价于 user:edit :\*、user  等价于 user:\*:\*
  + 注意：**通配符只能从字符串的结尾处省略部件**，也就是说 user:edit  并不等价于 user:\*:edit

|  |
| --- |
| * *// let's login the current user so we can check against roles and permissions:* **if** (!currentUser.isAuthenticated()) {  UsernamePasswordToken token = **new** UsernamePasswordToken(**"lonestarr"**, **"vespa"**);  token.setRememberMe(**true**);  **try** {  currentUser.login(token);  } **catch** (UnknownAccountException uae) {  log.info(**"There is no user with username of "** + token.getPrincipal());  } **catch** (IncorrectCredentialsException ice) {  log.info(**"Password for account "** + token.getPrincipal() + **" was incorrect!"**);  } **catch** (LockedAccountException lae) {  log.info(**"The account for username "** + token.getPrincipal() + **" is locked. "** +  **"Please contact your administrator to unlock it."**);  }  *// ... catch more exceptions here (maybe custom ones specific to your application?* **catch** (AuthenticationException ae) {  *//unexpected condition? error?* } ) |

|  |
| --- |
| *//say who they are: //print their identifying principal (in this case, a username):* ***log***.info(**"User ["** + currentUser.getPrincipal() + **"] logged in successfully."**); *//User [lonestarr] logged in successfully. //test a role:* **if** (currentUser.hasRole(**"schwartz"**)) {  ***log***.info(**"May the Schwartz be with you!"**);  } **else** {  ***log***.info(**"Hello, mere mortal."**); }  *//test a typed permission (not instance-level)* **if** (currentUser.isPermitted(**"lightsaber:weild"**)) {  ***log***.info(**"You may use a lightsaber ring. Use it wisely."**); } **else** {  ***log***.info(**"Sorry, lightsaber rings are for schwartz masters only."**); }  *//a (very powerful) Instance Level permission:* **if** (currentUser.isPermitted(**"winnebago:drive:eagle5"**)) {  ***log***.info(**"You are permitted to 'drive' the winnebago with license plate (id) 'eagle5'. "** +  **"Here are the keys - have fun!"**); } **else** {  ***log***.info(**"Sorry, you aren't allowed to drive the 'eagle5' winnebago!"**); } |

|  |
| --- |
| *//all done - log out!* currentUser.logout(); |

* **logout(注销)**：currentUser.logout();
* 调用 logout() 方法时，现有 Session 将失效，而且身份将失去关联（在Web 应用程序中，RememberMe cookie **将**被删除）。
* 在 Subject 注销后，该 Subject 的实例被再次认为是匿名的。
* **注意**：WEB 应用程序记住身份往往依靠 Cookie，然而Cookie 只能在 Response 被返回后被删除，所以**建议在调用subject.logout() 后立即向终端重定向一个新的视图或页面**。这样即能保证与安全相关的 Cookie 都能像预期的一样被删除。

## Spring集成Shiro

* 配置自定义 Realm：实现自定义认证和授权
* 配置 Shiro 实体类使用的缓存策略
* 配置 SecurityManager
* 配置保证 Shiro 内部 Bean 声明周期都得到执行的 Lifecycle Bean 后置处理器
* 配置AOP 式方法级权限检查
* 配置 Shiro Filter