

Shiro入门学习手册

简单的介绍，简单的配置，简单的
扩展

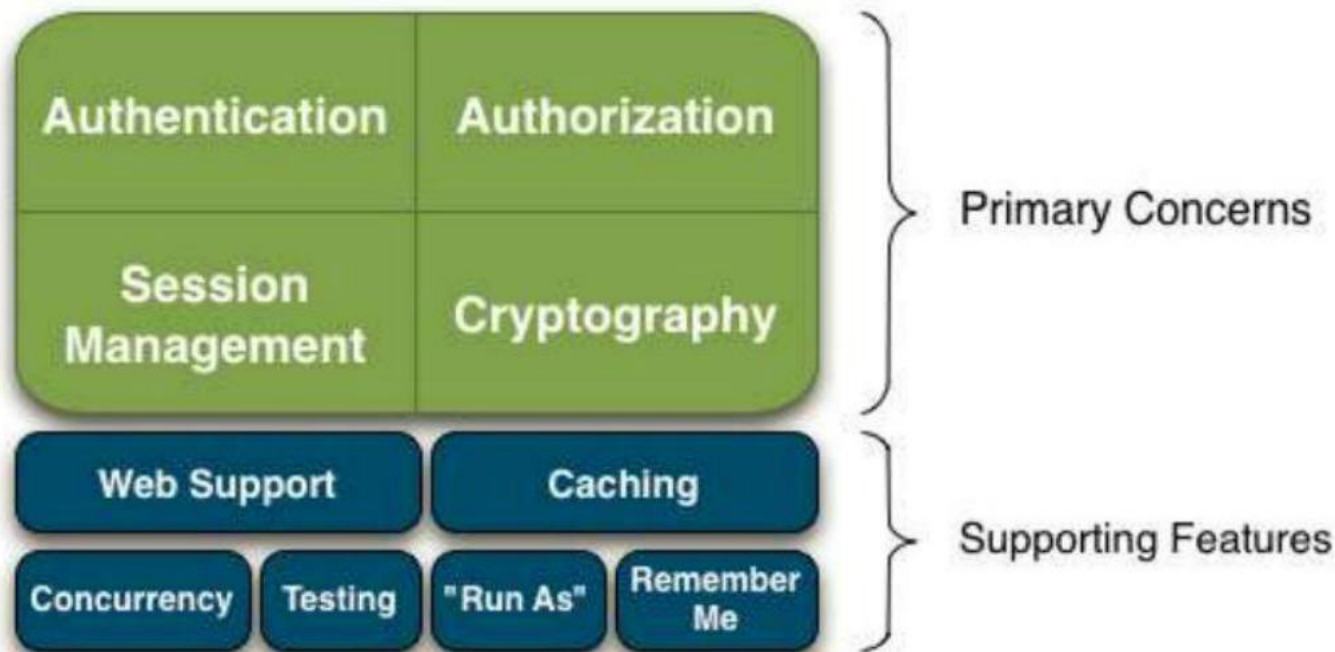
By jfm

一，shiro简介

Apache Shiro是一个强大而灵活的开源安全框架，它能够干净利落地处理身份认证，授权，企业会话管理和加密。

以下是你可以用 Apache Shiro所做的事情：







1. [?] 验证用户
2. [?] 对用户执行访问控制，如：
 - [?] 判断用户是否拥有角色admin。
 - 判断用户是否拥有访问的权限
3. [?] 在任何环境下使用 Session API。例如CS程序。
4. [?] 可以使用多个用户数据源。例如一个是oracle用户库，另外一个mysql用户库。
5. [?] 单点登录（SSO）功能。
6. [?] “Remember Me”服务，类似购物车的功能，shiro官方建议开启。

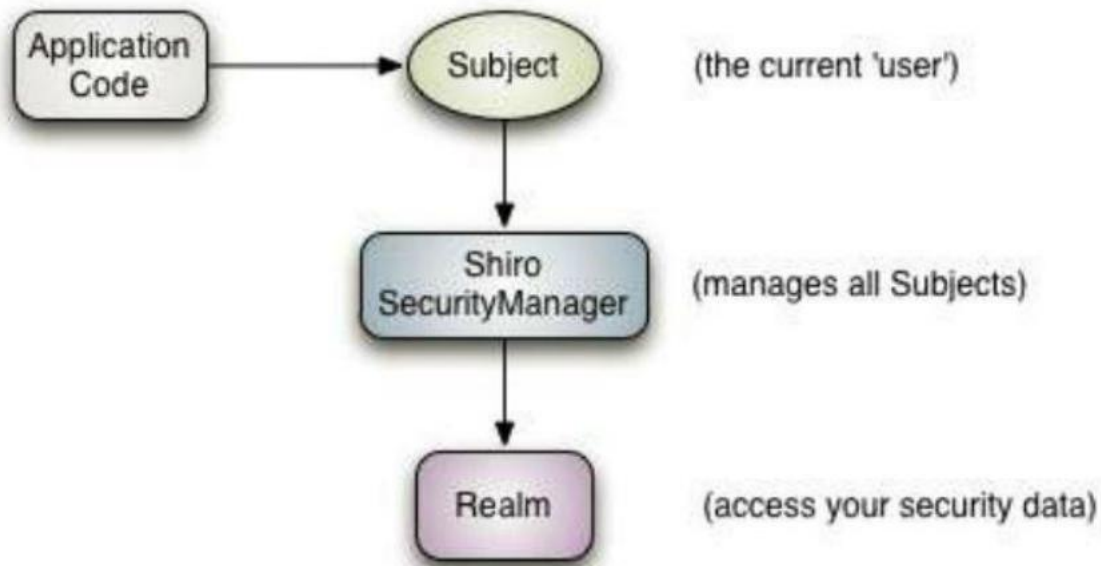


Shiro的4大部分——身份验证，授权，会话管理和加密

- **Authentication:** 身份验证，简称“登录”。
- **Authorization:** 授权，给用户分配角色或者权限资源
- **Session Management:** 用户session管理器，可以让CS程序也使用session来控制权限
- **Cryptography:** 把JDK中复杂的密码加密方式进行封装。

除了以上功能，shiro还提供很多扩展

-  **Web Support**: 主要针对web应用提供一些常用功能。
-  **Caching**: 缓存可以使应用程序运行更有效率。
-  **Concurrency**: 多线程相关功能。
-  **Testing**: 帮助我们进行测试相关功能
-  **"Run As"**: 一个允许用户假设为另一个用户身份（如果允许）的功能，有时候在管理脚本很有用。
-  **"Remember Me"**: 记住用户身份，提供类似购物车功能。



Subject:

Subject 是与程序进行交互的对象，可以是人也可以是服务或者其他，通常就理解为用户。

所有**Subject** 实例都必须绑定到一个**SecurityManager**上。我们与一个 **Subject** 交互，运行时shiro会自动转化为与 **SecurityManager**交互的特定 **subject**的交互。

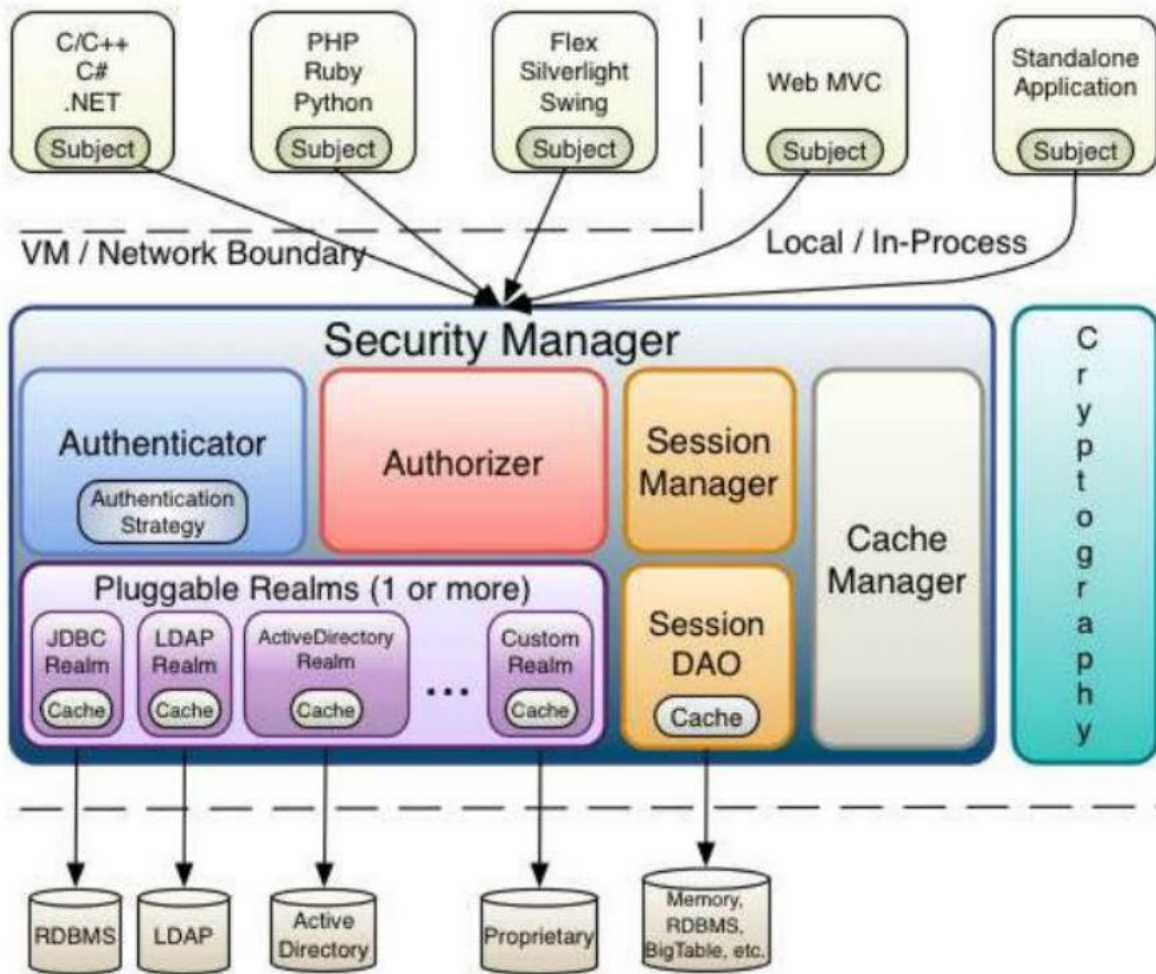
SecurityManager:

SecurityManager 是 Shiro 的核心，初始化时协调各个模块运行。然而，一旦 SecurityManager 协调完毕，SecurityManager 会被单独留下，且我们只需要去操作 Subject 即可，无需操作 SecurityManager。但是我们得知道，当我们正与一个 Subject 进行交互时，实质上是 SecurityManager 在处理 Subject 安全操作。

Realms:

Realms 在 Shiro 中作为应用程序和安全数据之间的“桥梁”或“连接器”。他获取安全数据来判断 subject 是否能够登录，subject 拥有什么权限。他有点类似 DAO。在配置 realms 时，需要至少一个 realm。而且 Shiro 提供了一些常用的 Realms 来连接数据源，如 LDAP 数据源的 JndiLdapRealm，JDBC 数据源的 JdbcRealm，ini 文件数据源的 IniRealm，properties 文件数据源的 PropertiesRealm，等等。我们也可以插入自己的 Realm 实现来代表自定义的数据源。像其他组件一样，Realms 也是由 SecurityManager 控制

小结:



1.Subject(org.apache.shiro.subject.Subject):

简称用户

2.SecurityManager(org.apache.shiro.mgt.SecurityManager)

如上所述，SecurityManager是shiro的核心，协调shiro的各个组件

3.Authenticator(org.apache.shiro.authc.Authenticator):

登录控制

注： Authentication Strategy

(org.apache.shiro.authc.pam.AuthenticationStrategy)

如果存在多个realm，则接口AuthenticationStrategy会确定什么样算是登录成功（例如，如果一个Realm成功，而其他的均失败，是否登录成功？）。

4.Authorizer(org.apache.shiro.authz.Authorizer) :
决定subject能拥有什么样角色或者权限。

5.SessionManager(org.apache.shiro.session.SessionManager) :
创建和管理用户session。通过设置这个管理器，shiro可以在任何环境下使用session。

6.CacheManager(org.apahce.shiro.cache.CacheManager) :
缓存管理器，可以减少不必要的后台访问。提高应用效率，增加用户体验。

7.Cryptography(org.apache.shiro.crypto.*) :
Shiro的api大幅度简化java api中繁琐的密码加密。

8.Realms(org.apache.shiro.realm.Realm) :
程序与安全数据的桥梁

二，简单配置

注：这里只介绍spring配置模式。

因为官方例子虽然中有更加简洁的ini配置形式，但是使用ini配置无法与spring整合。而且两种配置方法一样，只是格式不一样。

涉及的jar包

Jar包名称	版本
核心包shiro-core	1.2.0
Web相关包shiro-web	1.2.0
缓存包shiro-ehcache	1.2.0
与spring整合包shiro-spring	1.2.0
Ehcache缓存核心包ehcache-core	2.5.3
Shiro自身日志包slf4j-jdk14	1.6.4

使用maven时，在pom中添加依赖包

```
<dependency>
  <groupId>org.apache.shiro</groupId>
  <artifactId>shiro-core</artifactId>
  <version>1.2.0</version>
</dependency>
<dependency>
  <groupId>org.apache.shiro</groupId>
  <artifactId>shiro-web</artifactId>
  <version>1.2.0</version>
</dependency>
<dependency>
  <groupId>org.apache.shiro</groupId>
  <artifactId>shiro-ehcache</artifactId>
  <version>1.2.0</version>
</dependency>
```

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.2.0</version>
</dependency>
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache-core</artifactId>
    <version>2.5.3</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-jdk14</artifactId>
    <version>1.6.4</version>
</dependency>
```

Spring整合配置

1.在web.xml中配置shiro的过滤器

```
<!-- Shiro filter-->
```

```
<filter>
```

```
    <filter-name>shiroFilter</filter-name>
```

```
    <filter-class>
```

```
        org.springframework.web.filter.DelegatingFilterProxy
```

```
    </filter-class>
```

```
</filter>
```

```
<filter-mapping>
```

```
    <filter-name>shiroFilter</filter-name>
```

```
    <url-pattern>/*</url-pattern>
```

```
</filter-mapping>
```

2.在Spring的applicationContext.xml中添加shiro配置

```
<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
    <property name="securityManager" ref="securityManager" />
    <property name="loginUrl" value="/login" />
    <property name="successUrl" value="/login/loginSuccessFull" />
    <property name="unauthorizedUrl" value="/login/unauthorized" />
    <property name="filterChainDefinitions">
        <value>
            /home* = anon
            / = anon
            /logout = logout
            /role/** = roles[admin]
            /permission/** = perms[permssion:look]
            /** = authc
        </value>
    </property>
</bean>
```

securityManager: 这个属性是必须的。

loginUrl: 没有登录的用户请求需要登录的页面时自动跳转到登录页面，不是必须的属性，不输入地址的话会自动寻找项目web项目的根目录下的“/login.jsp”页面。

successUrl: 登录成功默认跳转页面，不配置则跳转至“/”。如果登陆前点击的一个需要登录的页面，则在登录自动跳转到那个需要登录的页面。不跳转到此。

unauthorizedUrl: 没有权限默认跳转的页面。

过滤器简称	对应的java类
anon	org.apache.shiro.web.filter.authc.AnonymousFilter
authc	org.apache.shiro.web.filter.authc.FormAuthenticationFilter
authcBasic	org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter
perms	org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter
port	org.apache.shiro.web.filter.authz.PortFilter
rest	org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter
roles	org.apache.shiro.web.filter.authz.RolesAuthorizationFilter
ssl	org.apache.shiro.web.filter.authz.SslFilter
user	org.apache.shiro.web.filter.authc.UserFilter
logout	org.apache.shiro.web.filter.authc.LogoutFilter

anon:例子/admins/**=anon 没有参数，表示可以匿名使用。

authc:例如/admins/user/**=authc表示需要认证(登录)才能使用，没有参数

roles: 例子/admins/user/**=roles[admin],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，当有多个参数时，例如admins/user/**=roles["admin,guest"],每个参数通过才算通过，相当于hasAllRoles()方法。

perms: 例子/admins/user/**=perms[user:add:*],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，例如/admins/user/**=perms["user:add:*,user:modify:*"],当有多个参数时必须每个参数都通过才通过，想当于isPermittedAll()方法。

rest: 例子/admins/user/**=rest[user],根据请求的方法,相当于/admins/user/**=perms[user:method],其中method为post, get, delete等。

port: 例子/admins/user/**=port[8081],当请求的url的端口不是8081是跳转到

schemal://serverName:8081?queryString,其中schmal是协议http或https等, serverName是你访问的host,8081是url配置里port的端口, queryString是你访问的url里的? 后面的参数。

authcBasic: 例如/admins/user/**=authcBasic没有参数表示httpBasic认证

ssl:例子/admins/user/**=ssl没有参数，表示安全的url请求，协议为https

user:例如/admins/user/**=user没有参数表示必须存在用户，当登入操作时不做检查

注：anon，authcBasic，auchc，user是认证过滤器，perms，roles，ssl，rest，port是授权过滤器

3. 在**applicationContext.xml**中添加 *securityManager*配置

```
<bean id="securityManager"  
    class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">  
    <!-- 单realm应用。如果有多个realm，使用 'realms'属性代替 -->  
    <property name="realm" ref="sampleRealm" />  
    <property name="cacheManager" ref="cacheManager" />  
</bean>  
  
<bean id="cacheManager"  
    class="org.apache.shiro.cache.ehcache.EhCacheManager" />
```

4. 配置jdbcRealm

```
<bean id="sampleRealm" class="org.apache.shiro.realm.jdbc.JdbcRealm">
  <property name="dataSource" ref="dataSource" />
  <property name="authenticationQuery"
    value="select t.password from my_user t where t.username = ?" />
  <property name="userRolesQuery"
    value="select a.rolename from my_user_role t left join my_role a on
t.roleid = a.id where t.username = ? " />
  <property name="permissionsQuery"
    value="SELECT B.PERMISSION FROM MY_ROLE T LEFT JOIN
MY_ROLE_PERMISSION A ON T.ID = A.ROLE_ID LEFT JOIN MY_PERMISSION
B ON A.PERMISSION = B.ID WHERE T.ROLENAME = ? " />
  <property name="permissionsLookupEnabled" value="true" />
  <property name="saltStyle" value="NO_SALT" />
  <property name="credentialsMatcher" ref="hashedCredentialsMatcher"
  />
</bean>
```

dataSource 数据源，配置不说了。

authenticationQuery 登录认证用户的查询SQL，需要用登录用户名作为条件，查询密码字段。

userRolesQuery 用户角色查询SQL，需要通过登录用户名去查询。查询角色字段

permissionsQuery 用户的权限资源查询SQL，需要用单一角色查询角色下的权限资源，如果存在多个角色，则是遍历每个角色，分别查询出权限资源并添加到集合中。

permissionsLookupEnabled 默认false。False时不会使用*permissionsQuery*的SQL去查询权限资源。设置为true才会去执行。

saltStyle 密码是否加盐，默认是NO_SALT不加盐。加盐有三种选择CRYPT,COLUMN,EXTERNAL。详细可以去看文档。这里按照不加盐处理。

credentialsMatcher 密码匹配规则。下面简单介绍。

```
<bean id="hashedCredentialsMatcher"  
    class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">  
    <property name="hashAlgorithmName" value="MD5" />  
    <property name="storedCredentialsHexEncoded" value="true" />  
    <property name="hashIterations" value="1" />  
</bean>
```

hashAlgorithmName 必须的，没有默认值。可以有MD5或者SHA-1，如果对密码安全有更高要求可以用SHA-256或者更高。这里使用MD5

storedCredentialsHexEncoded 默认是true，此时用的是密码加密用的是Hex编码；false时用Base64编码

hashIterations 迭代次数，默认值是1。

登录JSP页面

```
<form action="login" method="post">
```

```
<td>用户名: </td>
```

```
<td><input type="text" name="username"></input></td>
```

```
<td>密码: </td>
```

```
<td><input type="password" name="password"></input></td>
```

```
<td>记住我</td>
```

```
<td><input type="checkbox" name="rememberMe" /></td>
```

注：登录JSP，表单action与提交方式固定，用户名与密码的name也是固定。

5.配置shiro注解模式

<!-- 开启Shiro注解的Spring配置方式的beans。在
lifecycleBeanPostProcessor之后运行 -->

```
<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>
```

注意：在与springMVC整合时必须放在springMVC的配置文件中。
Shiro在注解模式下，登录失败，与没有权限均是通过抛出异常。并且默认并没有去处理或者捕获这些异常。在springMVC下需要配置捕获相应异常来通知用户信息，如果不配置异常会抛出到页面

```
<bean
  class="org.springframework.web.servlet.handler.SimpleMappingExceptionResol
  ver">
  <property name="exceptionMappings">
    <props>
      <prop key="org.apache.shiro.authz.UnauthorizedException">
        /unauthorized
      </prop>
      <prop key="org.apache.shiro.authz.UnauthenticatedException">
        /unauthenticated
      </prop>
    </props>
  </property>
</bean>
```

@RequiresAuthentication

验证用户是否登录，等同于方法`subject.isAuthenticated()` 结果为`true`时。

@ RequiresUser

验证用户是否被记忆，`user`有两种含义：

一种是成功登录的（`subject.isAuthenticated()` 结果为`true`）；

另外一种是被记忆的（`subject.isRemembered()`结果为`true`）。

@ RequiresGuest

验证是否是一个`guest`的请求，与`@ RequiresUser`完全相反。

换言之，`RequiresUser == ! RequiresGuest`。

此时`subject.getPrincipal()` 结果为`null`。

@ RequiresRoles

例如: `@RequiresRoles("aRoleName");`
`void someMethod();`

如果subject中有aRoleName角色才可以访问方法someMethod。如果没有这个权限则会抛出异常[AuthorizationException](#)。

@RequiresPermissions

例如: `@RequiresPermissions({"file:read", "write:aFile.txt"})`
`void someMethod();`

要求subject中必须同时含有file:read和write:aFile.txt的权限才能执行方法someMethod()。否则抛出异常[AuthorizationException](#)。

三.简单扩展

1. 自定义realm:

<!--自定义的myRealm 继承自 **AuthorizingRealm**，也可以选择 shiro提供的 -->
<bean id="myRealm" class="com.yada.shiro.MyReam"></bean>

//这是授权方法

```
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  
    String userName = (String) getAvailablePrincipal(principals);  
    //TODO 通过用户名获得用户的所有资源，并把资源存入info中  
    .....  
    SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();  
    info.setStringPermissions(set集合);  
    info.setRoles(set集合);  
    info.setObjectPermissions(set集合);  
    return info;  
}
```

//这是认证方法

```
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws  
AuthenticationException {
```

```
    //token中储存着输入的用户名和密码
```

```
    UsernamePasswordToken upToken = (UsernamePasswordToken)token;
```

```
    //获得用户名与密码
```

```
    String username = upToken.getUsername();
```

```
    String password = String.valueOf(upToken.getPassword());
```

```
    //TODO 与数据库中用户名和密码进行比对。比对成功则返回info，比对失败  
    则抛出对应信息的异常AuthenticationException
```

```
    .....
```

```
    SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(username,  
password.toCharArray(),getName());
```

```
    return info;
```

```
}
```

2. 自定义登录

//创建用户名和密码的令牌

UsernamePasswordToken token = new

UsernamePasswordToken(user.getUserName(),user.getPassWord());

//记录该令牌，如果不记录则类似购物车功能不能使用。

token.setRememberMe(true);

//**subject**理解成权限对象。类似**user**

Subject subject = SecurityUtils.getSubject();

try {

subject.login(token);

} catch (UnknownAccountException ex) {///用户名没有找到。

} catch (IncorrectCredentialsException ex) {///用户名密码不匹配。

}catch (AuthenticationException e) {///其他的登录错误

}

//验证是否成功登录的方法

if (subject.isAuthenticated()) {

}

3. 自定义登出

```
Subject subject = SecurityUtils.getSubject();  
subject.logout();
```

4. 基于编码的角色授权实现

```
Subject currentUser = SecurityUtils.getSubject();  
if (currentUser.hasRole("administrator")) {  
    //拥有角色administrator  
} else {  
    //没有角色处理  
}
```

断言方式控制

```
Subject currentUser = SecurityUtils.getSubject();  
//如果没有角色admin, 则会抛出异常, someMethod()也不会被执行  
currentUser.checkRole("admin");  
someMethod();
```


5. 基于编码的资源授权实现

```
Subject currentUser = SecurityUtils.getSubject();  
if (currentUser.isPermitted("permssion:look")) {  
    //有资源权限  
}else {  
    //没有权限  
}
```

断言方式控制

```
Subject currentUser = SecurityUtils.getSubject();  
//如果没有资源权限则会抛出异常。  
currentUser.checkPermission("permssion:look");  
someMethod();
```

6. 在JSP上的TAG实现

标签名称	标签条件（均是显示标签内容）
<shiro:authenticated>	登录之后
<shiro:notAuthenticated>	不在登录状态时
<shiro:guest>	用户在没有RememberMe时
<shiro:user>	用户在RememberMe时
<shiro:hasAnyRoles name="abc,123" >	在有abc或者123角色时
<shiro:hasRole name="abc">	拥有角色abc
<shiro:lacksRole name="abc">	没有角色abc
<shiro:hasPermission name="abc">	拥有权限资源abc
<shiro:lacksPermission name="abc">	没有abc权限资源
<shiro:principal>	默认显示用户名称

7.

默认，添加或删除用户的角色 或资源 ，系统不需要重启，但是需要用户重新登录。即用户的授权是首次登录后第一次访问需要权限页面时进行加载。

但是需要进行控制的权限资源，是在启动时就进行加载，如果要新增一个权限资源需要重启系统。

8.

Spring security 与apache shiro 差别：

- a) shiro配置更加容易理解，容易上手；security配置相对比较难懂。
- b) 在spring的环境下，security整合性更好。Shiro对很多其他的框架兼容性更好，号称是无缝集成。
- c) shiro 不仅仅可以使用在web中，它可以工作在任何应用环境中。
- d) 在集群会话时Shiro最重要的一个好处或许就是它的会话是独立于容器的。
- e) Shiro提供的密码加密使用起来非常方便。

9.

控制精度：

注解方式控制权限只能是在方法上控制，无法控制类级别访问。

过滤器方式控制是根据访问的URL进行控制。允许使用*匹配URL，所以可以进行粗粒度，也可以进行细粒度控制。