# 一 Shiro简介

Apache Shiro是一个强大易用的**[Java](http://lib.csdn.net/base/java" \t "_blank" \o "Java 知识库)**安全框架，提供了认证、授权、加密和会话管理功能，可为任何应用提供安全保障。

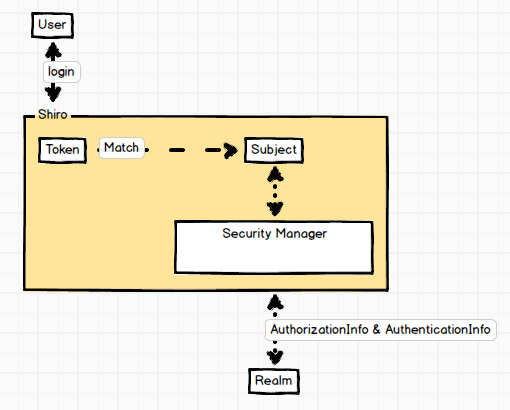


Shiro的具体功能点如下：

（1）身份认证/登录，验证用户是不是拥有相应的身份；   
（2）授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情，常见的如：验证某个用户是否拥有某个角色。或者细粒度的验证某个用户对某个资源是否具有某个权限；   
（3）会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通JavaSE环境的，也可以是如Web环境的；   
（4）加密，保护数据的安全性，如密码加密存储到[数据库](http://lib.csdn.net/base/mysql" \t "_blank" \o "MySQL知识库)，而不是明文存储；   
（5）Web支持，可以非常容易的集成到Web环境；   
Caching：缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；   
（6）shiro支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；   
（7）提供[测试](http://lib.csdn.net/base/softwaretest" \t "_blank" \o "软件测试知识库)支持；   
（8）允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；   
（9）记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录了。

# 二 身份认证Authentication

## 1 工作流程



## 2 认证主体 Subject

Subject 认证主体包含两个信息：

Principals：身份。可以是用户名，邮件，手机号码等等，用来标识一个登录主体身份；   
Credentials：凭证。常见有密码，数字证书等等

在Shiro中可以在.ini文件中指定一个认证主体，也可以从[数据库](http://lib.csdn.net/base/mysql" \t "_blank" \o "MySQL知识库)中取。

ini文件写法：

[users]

zhangsan=12345

lisi=12345

## 3 realm

realm就是一个域，Shiro就是从realm中获取验证数据的，比如，上一节中提到的.ini文件，realm有很多种，如text realm、jdbc realm、jndi realm、自定义realm等。

## 4 程序实现

1）在maven中引入使用的jar

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.2.5</version>

</dependency>

2）代码：

// 读取配置文件，初始化SecurityManager工厂

Factory<SecurityManager> factory = **new** IniSecurityManagerFactory("classpath:shiro.ini");

// 获取securityManager实例

SecurityManager securityManager = factory.getInstance();

// 把securityManager实例绑定到SecurityUtils

SecurityUtils.*setSecurityManager*(securityManager);

// 创建token令牌，用户名/密码

UsernamePasswordToken token = **new** UsernamePasswordToken("csdn", "123");

// 得到用户主体（[Users]）

Subject currentUser = SecurityUtils.*getSubject*();

**try**{

// 身份认证

currentUser.login(token);

System.***out***.println("身份认证成功！");

}**catch**(AuthenticationException e){

e.printStackTrace();

System.***out***.println("身份认证失败！");

}

// 退出

currentUser.logout();

## 5 jdbc realm

1）配置文件写法

[main]

#数据源选择的是c3p0

dataSource=com.mchange.v2.c3p0.ComboPooledDataSource

dataSource.driverClass=com.mysql.jdbc.Driver

dataSource.jdbcUrl=jdbc:mysql://localhost:3306/db\_shiro

dataSource.user=root

dataSource.password=root

#定义一个jdbc的realm，取名为jdbcRealm

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

#jdbcRealm中有个属性是dataSource，选择我们上边定义的dataSource

jdbcRealm.dataSource=$dataSource

#SecurityManager中的realm选择上面定义的jdbcRealm

securityManager.realms=$jdbcRealm

2）数据库中建立表users

users中字段为username、password

注意：表名必须为users、字段必须包括username和password，且字段名不能修改

# 三 自定义realm认证

如果针对相对简单的系统，使用.ini文件包括用户信息，角色信息，权限信息等是可行的。但是实际项目中，各种信息都是存放在**[数据库](http://lib.csdn.net/base/mysql" \t "_blank" \o "MySQL知识库)**中，此时需要使用自定义realm

## 1 配置文件

custome\_realm.ini

[main]

#自定义realm

customRealm=com.rr.realm.CustomRealm

#设置realm值，相当于注入

securityManager.realms=$customRealm

## 2 自定义realml类

自定义realm需要继承AuthorizingRealm类

重写如下两个方法：

doGetAuthenticationInfo()方法：用来验证当前登录的用户，获取认证信息

doGetAuthorizationInfo()方法：用来为当前登陆成功的用户授予权限和角色（已经登陆成功了）

//认证

@Override

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

// **TODO** Auto-generated method stub

// token表示用户输入的内容

// 1从token中取出身份信息

String userCode = (String) token.getPrincipal();

// 2根据用户输入的用户名从数据库查询对应数据，如果查询不到返回null，如果查询到，获取密码进行判断

// 本例模拟从数据库查询到密码

String password = "123";

// 3返回认证信息AuthenticationInfo

SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(

userCode, password, **this**.getName());

**return** simpleAuthenticationInfo;

}

## 3 测试

// 读取配置文件，初始化SecurityManager工厂

Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(classpath:customer\_realm.ini);

## 4 hash算法

hash（散列）算法一般用于生成一段文本的摘要信息，散列算法不可逆，不同的内容获取相同的hash值的概率非常小，通常用于对密码进行处理，提高安全性。常用的散列算法有MD5、SHA-1。

使用散列算法时，还可提供一个salt（盐）值，然后与原始内容生成摘要信息，这样做的目的是为了更安全

1）shiro中的hash处理

**public** **static** **void** main(String[] args) {

//原始 密码

String source = "123";

//盐值

String salt = **null**;

//散列次数

**int** hashIterations = 1;

//构造方法中：

//第一个参数：明文，原始密码

//第二个参数：盐值，可以使用随机数

//第三个参数：hash的次数，比如2次，相当 于md5(md5(''))

Md5Hash md5Hash = **new** Md5Hash(source, salt, hashIterations);

String password\_md5 = md5Hash.toString();

System.***out***.println(password\_md5);

//另外的方式，第一个参数：指定hash算法

SimpleHash simpleHash = **new** SimpleHash("md5", source, salt, hashIterations);

System.***out***.println(simpleHash.toString());

}

[main]

#定义凭证匹配器

credentialsMatcher=org.apache.shiro.authc.credential.HashedCredentialsMatcher

#hash算法

credentialsMatcher.hashAlgorithmName=md5

#次数

credentialsMatcher.hashIterations=1

customRealm=com.rr.realm.CustomRealmMd5

#设置realm的凭证匹配器

customRealm.credentialsMatcher=$credentialsMatcher

securityManager.realms=$customRealm

注意：

如果使用salt，可以传参指定salt值

SimpleAuthenticationInfo info = **new** SimpleAuthenticationInfo(name, pwd, ByteSource.Util.*bytes*("bbb"), **this**.getName());

# 四 授权Authorization

权限认证，也叫访问控制，即在应用中控制谁能访问哪些资源

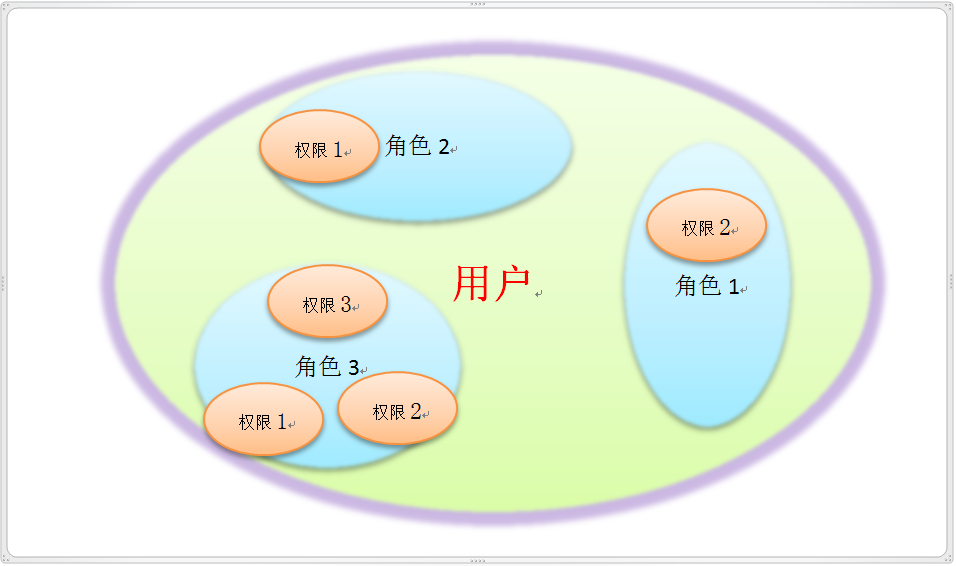
## 1 权限认证的要素

最核心的三个要素是：权限，角色和用户

权限(permission)：即操作资源的权利，比如访问某个页面，以及对某个模块的数据的添加，修改，删除，查看的权利；

角色(role)：指的是用户担任的的角色，一个角色可以有多个权限；

用户(user)：在Shiro 中，代表访问系统的用户。



## 2 访问控制

1）基于角色的访问控制

授权过程是通过判断角色来完成的，哪个角色可以做这件事

2）基于资源（权限）的访问控制

## 3 程序实现

1)realm文件

#用户，role表示各个角色

[users]

zhangsan=123,role1,role2,role3

lisi=123,role1,role2

#定义不同角色都拥有哪些权限

[roles]

role1=user:select

role2=user:add,user:update

role3=user.delete

2)测试代码

@Test

**public** **void** testHasRole() {

String configFile = "classpath:shiro\_role.ini";

String username = "zhangsan";

String password = "123";

Subject currentUser = ShiroUtil.*login*(configFile, username, password);

//测试hasRole

System.***out***.println(currentUser.hasRole("role2")? "有role2这个角色" : "没有role2这个角色");

//测试hasRoles

**boolean**[] results = currentUser.hasRoles(Arrays.*asList*("role1","role2","role3"));

System.***out***.println(results[0]? "有role1这个角色" : "没有role1这个角色");

System.***out***.println(results[1]? "有role2这个角色" : "没有role2这个角色");

System.***out***.println(results[2]? "有role3这个角色" : "没有role3这个角色");

//测试hasAllRoles

System.***out***.println(currentUser.hasAllRoles(Arrays.*asList*("role1","role2","role3")));

currentUser.logout();

}

@Test

**public** **void** testHasPermit() {

String configFile = "classpath:shiro\_role.ini";

String username = "zhangsan";

String password = "123";

Subject currentUser = ShiroUtil.*login*(configFile, username, password);

//测试是否有权限

System.***out***.println(currentUser.isPermitted("user:select")? "有select这个权限" : "没有select这个权限");

**boolean**[] results = currentUser.isPermitted(**new** String[]{"user:select","user:add","user:update"});

System.***out***.println(results[0]? "有user:select这个角色" : "没有user:select这个角色");

System.***out***.println(results[1]? "有user:add这个角色" : "没有user:add这个角色");

System.***out***.println(results[2]? "有user:update这个角色" : "没有user:update这个角色");

currentUser.logout();

}

# 五 与Web整合

## 1 引入jar

<!-- shiro核心包 -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.2.5</version>

</dependency>

<!-- 添加shiro web支持 -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>1.2.5</version>

</dependency>

## 2 web.xml配置

配置Shiro的监听器和过滤器

<listener>

<listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>

</listener>

<context-param>

<param-name>shiroEnvironmentClass</param-name>

<param-value>org.apache.shiro.web.env.IniWebEnvironment</param-value>

</context-param>

<context-param>

<param-name>shiroConfigLocations</param-name>

<param-value>classpath:shiro.ini</param-value>

</context-param>

<filter>

<filter-name>ShiroFilter</filter-name>

<filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>ShiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

## 3 shiro配置文件

[main]

#如果没有进行身份认证，访问需要认证后才能访问的资源时，请求的url，loginUrl是身份认证过滤器中的一个属性

authc.loginUrl=/login.jsp

#定义角色认证失败后的请求url映射，unauthorizedUrl是角色认证过滤器中的一个属性

roles.unauthorizedUrl=/unauthorized.jsp

#定义角色认证失败后请求url映射，unauthorizedUrl是角色认证过滤器中的一个属性

perms.unauthorizedUrl=/unauthorized.jsp

#定义几个用户和角色

[users]

zhang=123,admin,teacher

wang=123,teacher

li=123,student

Tom=123

#定义不同角色的权限

[roles]

admin=user:\*,student:\*

teacher=student:\*

#定义请求的地址需要做什么验证

[urls]

#请求login的时候不需要权限，游客身份即可(anon)

/login=anon

#请求/admin的时候，需要身份认证(authc)

/admin=authc

#请求/student的时候，需要角色认证，必须是拥有teacher角色的用户才行

/student=roles[teacher]

#请求/teacher的时候，需要权限认证，必须是拥有user:create权限的角色的用户才行

/teacher=perms["user:create"]

内置的过滤器

| **过滤器名** | **对应的类** |
| --- | --- |
| anon（匿名） | org.apache.shiro.web.filter.authc.AnonymousFilter |
| authc（身份验证） | org.apache.shiro.web.filter.authc.FormAuthenticationFilter |
| authcBasic（http基本验证） | org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter |
| logout（退出） | org.apache.shiro.web.filter.authc.LogoutFilter |
| noSessionCreation（不创建session） | org.apache.shiro.web.filter.session.NoSessionCreationFilter |
| perms(许可验证) | org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter |
| port（端口验证） | org.apache.shiro.web.filter.authz.PortFilter |
| rest(rest方面) | org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter |
| roles（权限验证） | org.apache.shiro.web.filter.authz.RolesAuthorizationFilter |
| ssl（ssl方面） | org.apache.shiro.web.filter.authz.SslFilter |
| user（用户方面） | org.apache.shiro.web.filter.authc.UserFilter |

# 六 自定义realm授权

如果针对相对简单的系统，使用.ini文件包括用户信息，角色信息，权限信息等是可行的。但是实际项目中，各种信息都是存放在[数据库](http://lib.csdn.net/base/mysql" \t "_blank" \o "MySQL知识库)中，此时需要使用自定义realm

## 1 建立表

用户表、角色表、资源表、用户角色对应表、角色资源对应表

## 2 自定义realm

自定义realm需要继承AuthorizingRealm类

重写如下两个方法：

doGetAuthenticationInfo()方法：用来验证当前登录的用户，获取认证信息

// 验证当前登录的用户，获取认证信息

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) **throws** AuthenticationException {

System.***out***.println("456");

String username = (String) token.getPrincipal(); // 获取用户名

Connection conn = **null**;

**try** {

conn = JdbcUtil.*getConnection*();

User user = userDao.getByUsername(conn, username); // 仅仅是根据用户名查出的用户信息，不涉及到密码

**if** (user != **null**) {

AuthenticationInfo authcInfo = **new** SimpleAuthenticationInfo(

user.getUsername(), user.getPassword(), "myrealm");

**return** authcInfo;

} **else** {

**return** **null**;

}

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **finally** {

**try** {

JdbcUtil.*close*(conn, **null**, **null**);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**return** **null**;

}

doGetAuthorizationInfo()方法：用来为当前登陆成功的用户授予权限和角色（已经登陆成功了）

// 为当前登陆成功的用户授予权限和角色，已经登陆成功了

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

System.***out***.println("123");

String username = (String) principals.getPrimaryPrincipal(); //获取用户名

SimpleAuthorizationInfo authorizationInfo = **new** SimpleAuthorizationInfo();

Connection conn = **null**;

**try** {

conn = JdbcUtil.*getConnection*();

authorizationInfo.setRoles(userDao.getRoles(conn, username)); //设置角色

authorizationInfo.setStringPermissions(userDao.getPerms(conn, username)); //设置权限

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **finally** {

**try** {

JdbcUtil.*close*(conn, **null**, **null**);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**return** authorizationInfo;

}

## 3 修改ini文件

[main]

#如果没有进行身份认证，访问需要认证后才能访问的资源时，loginUrl是身份认证过滤器中的一个属性

authc.loginUrl=/login

#定义角色认证失败后的请求url映射，unauthorizedUrl是角色认证过滤器中的一个属性

roles.unauthorizedUrl=/unauthorized.jsp

#定义角色认证失败后请求url映射，unauthorizedUrl是角色认证过滤器中的一个属性

perms.unauthorizedUrl=/unauthorized.jsp

#定义自己的realm

myRealm=com.rr.realm.MyRealm

securityManager.realms=$myRealm

#定义请求的地址需要做什么验证

[urls]

#请求login的时候不需要权限，游客身份即可(anon)

/login=anon

#请求/admin的时候，需要身份认证(authc)

/admin=authc

#请求/student的时候，需要角色认证，必须是拥有teacher角色的用户才行

/student=roles[teacher]

#请求/teacher的时候，需要权限认证，必须是拥有user:create权限的角色的用户才行

/teacher=perms["user:create"]

# 七 与SSM整合

## 1 引入jar

<!-- shiro核心包 -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.2.5</version>

</dependency>

<!-- 添加shiro web支持 -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>1.2.5</version>

</dependency>

<!-- 添加shiro spring支持 -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.2.5</version>

</dependency>

## 2 web.xml

添加shiro过滤器

<!-- 添加shiro过滤器 这里的filter-name 要和spring 的applicationContext.xml 里的

org.apache.shiro.spring.web.ShiroFilterFactoryBean 的bean name 相同-->

<filter>

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

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

<init-param>

<!-- 该值缺省为false，表示声明周期由SpringApplicationContext管理，设置为true表示ServletContainer管理 -->

<param-name>targetFilterLifecycle</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

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

<url-pattern>/\*</url-pattern>

</filter-mapping>

## 3 修改spring配置文件

<!-- 自定义Realm -->

<bean id=*"myRealm"* class=*"com.rr.realm.MyRealm"*/>

<!-- 安全管理器 -->

<bean id=*"securityManager"* class=*"org.apache.shiro.web.mgt.DefaultWebSecurityManager"*>

<property name=*"realm"* ref=*"myRealm"*/>

<property name=*"cacheManager"* ref=*"cacheManager"* />

</bean>

<!-- Shiro过滤器 -->

<bean id=*"shiroFilter"* class=*"org.apache.shiro.spring.web.ShiroFilterFactoryBean"*>

<!-- Shiro的核心安全接口,这个属性是必须的 -->

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

<!-- 未登陆时，访问需要认证的资源，则跳转到登录页面的配置 -->

<property name=*"loginUrl"* value=*"/login.jsp"*/>

<!-- 权限认证失败，则跳转到指定页面 -->

<property name=*"unauthorizedUrl"* value=*"/unauthorized.jsp"*/>

<!-- Shiro连接约束配置,即过滤链的定义 -->

<property name=*"filterChainDefinitions"*>

<value>

<!-- /login=anon

/user/admin\*=authc

/user/student\*/\*\*=roles[teacher]

/user/teacher\*/\*\*=perms["user:create"] -->

/user/login.do=anon

<!-- user下的所有内容都需要身份认证 -->

/user/\*=authc

</value>

</property>

</bean>

## 4 修改springmvc配置

如果使用shiro的注解，必须将该配置放在springmvc的配置文件中，否则注解不起作用

<!-- 保证实现了Shiro内部lifecycle函数的bean执行 -->

<bean id=*"lifecycleBeanPostProcessor"* class=*"org.apache.shiro.spring.LifecycleBeanPostProcessor"*/>

<!-- 生成代理，通过代理进行控制 -->

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

<!-- 开启Shiro注解,必须放在springmvc的配置文件中 -->

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

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

</bean>

<!-- 注解方式下的异常处理，身份认证异常处理、权限认证异常处理 -->

<bean class=*"org.springframework.web.servlet.handler.SimpleMappingExceptionResolver"*>

<property name=*"exceptionMappings"*>

<props>

<!--登录异常，跳转到login.jsp-->

<!-- <prop key="org.apache.shiro.authz.UnauthenticatedException">

/login

</prop> -->

<!-- 授权异常，跳转到unauthorized.jsp -->

<prop key=*"org.apache.shiro.authz.UnauthorizedException"*>/unauthorized</prop>

</props>

</property>

</bean>

## 5 shiro注解

@RequiresPermissions ：要求当前Subject在执行被注解的方法时具备一个或多个对应的权限。

@RequiresRoles ：要求当前Subject在执行被注解的方法时具备所有的角色，否则将抛出AuthorizationException异常。

@RequiresAuthentication：要求在访问或调用被注解的类/实例/方法时，Subject在当前的session中已经被验证。

## 6 Jsp中使用的shiro标签

需要引入标签

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

标签：

<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> 显示用户身份名称

<shiro:principal property="username"/> 显示用户身份中的属性值

## 7 缓存

本例使用ehcache缓存角色、权限等数据，如果不使用缓存，只要需要进行权限判断，都会进入自定的realm中获取权限相关数据

1）引入jar

<dependency>

<artifactId>ehcache-core</artifactId>

<groupId>net.sf.ehcache</groupId>

<version>2.5.0</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-ehcache</artifactId>

<version>1.2.5</version>

</dependency>

2）增加ehcache的配置文件

ehcache.xml配置文件说明：

name:缓存名称。  
 maxElementsInMemory：缓存最大个数。  
 eternal:对象是否永久有效，一但设置了，timeout将不起作用。  
 timeToIdleSeconds：设置对象在失效前的允许闲置时间（单位：秒）。仅当eternal=false对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间[无穷大](http://zhidao.baidu.com/search?word=%E6%97%A0%E7%A9%B7%E5%A4%A7&fr=qb_search_exp&ie=utf8" \t "_blank)。  
 timeToLiveSeconds：设置对象在失效前允许存活时间（单位：秒）。最大时间介于创建时间和失效时间之间。仅当eternal=false对象不是永久有效时使用，默认是0.，也就是对象存活时间[无穷大](http://zhidao.baidu.com/search?word=%E6%97%A0%E7%A9%B7%E5%A4%A7&fr=qb_search_exp&ie=utf8" \t "_blank)。  
 overflowToDisk：当内存中对象数量达到maxElementsInMemory时，Ehcache将会对象写到磁盘中。  
 diskSpoolBufferSizeMB：这个参数设置DiskStore（[磁盘缓存](http://zhidao.baidu.com/search?word=%E7%A3%81%E7%9B%98%E7%BC%93%E5%AD%98&fr=qb_search_exp&ie=utf8" \t "_blank)）的缓存区大小。默认是30MB。每个Cache都应该有自己的一个[缓冲区](http://zhidao.baidu.com/search?word=%E7%BC%93%E5%86%B2%E5%8C%BA&fr=qb_search_exp&ie=utf8" \t "_blank)。  
 maxElementsOnDisk：硬盘最大缓存个数。  
 diskPersistent：是否缓存[虚拟机](http://zhidao.baidu.com/search?word=%E8%99%9A%E6%8B%9F%E6%9C%BA&fr=qb_search_exp&ie=utf8" \t "_blank)重启期数据 Whether the disk store persists between restarts of the Virtual Machine. The default value is false.  
 diskExpiryThreadIntervalSeconds：磁盘失效线程运行[时间间隔](http://zhidao.baidu.com/search?word=%E6%97%B6%E9%97%B4%E9%97%B4%E9%9A%94&fr=qb_search_exp&ie=utf8" \t "_blank)，默认是120秒。  
 memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略是LRU（最近最少使用）。你可以设置为FIFO（[先进先出](http://zhidao.baidu.com/search?word=%E5%85%88%E8%BF%9B%E5%85%88%E5%87%BA&fr=qb_search_exp&ie=utf8" \t "_blank)）或是LFU（较少使用）。  
 clearOnFlush：内存数量最大时是否清除。

3)修改安全管理器配置

<!-- 安全管理器 -->

<bean id=*"securityManager"* class=*"org.apache.shiro.web.mgt.DefaultWebSecurityManager"*>

<property name=*"realm"* ref=*"myRealm"*/>

<property name=*"cacheManager"* ref=*"cacheManager"* />

</bean>

<!-- 缓存管理器 使用Ehcache实现 -->

<bean id=*"cacheManager"* class=*"org.apache.shiro.cache.ehcache.EhCacheManager"*>

<property name=*"cacheManagerConfigFile"* value=*"classpath:ehcache.xml"* />

</bean>

## 8 加密处理

<!-- 自定义Realm -->

<bean id="myRealm" class="com.rr.realm.MyRealm">

<!-- 配置密码匹配器 -->

<property name="credentialsMatcher">

<bean class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">

加密算法为MD5

<property name="hashAlgorithmName" value="MD5"></property>

加密次数

<property name="hashIterations" value="1"></property>

</bean>

</property>

</bean>

## 9 rememberMe功能

开启rememberMe功能，登录后，会将用户名和密码数据已密文的形式存储在cookie中，关闭程序后，可以不用登录，直接访问相关资源

1）页面

2）代码：

UsernamePasswordToken token = **new** UsernamePasswordToken(user.getUsername(), user.getPassword());

//记住我功能

token.setRememberMe(**true**);

3）配置：

<!-- 安全管理器 -->

<bean id=*"securityManager"* class=*"org.apache.shiro.web.mgt.DefaultWebSecurityManager"*>

<property name=*"realm"* ref=*"myRealm"*/>

<property name=*"cacheManager"* ref=*"cacheManager"* />

<property name=*"rememberMeManager"* ref=*"rememberMeManager"* />

</bean>

<!-- remenberMe配置 -->

<bean id=*"rememberMeCookie"* class=*"org.apache.shiro.web.servlet.SimpleCookie"*>

<constructor-arg value=*"rememberMe"* />

<property name=*"httpOnly"* value=*"true"* />

<!-- 记住7天（单位：秒） -->

<property name=*"maxAge"* value=*"604800"* />

</bean>

<!-- rememberMe管理器 -->

<bean id=*"rememberMeManager"* class=*"org.apache.shiro.web.mgt.CookieRememberMeManager"*>

<property name=*"cipherKey"* value=*"#{T(org.apache.shiro.codec.Base64).decode('4AvVhmFLUs0KTA3Kprsdag==')}"* />

<property name=*"cookie"* ref=*"rememberMeCookie"* />

</bean>

## 10 退出功能

*<!-- 手动创建了一个logout过滤器的对象，id和 -->*<**bean id="logout" class="org.apache.shiro.web.filter.authc.LogoutFilter"**>  
 <**property name="redirectUrl" value="/login.jsp"** />  
</**bean**>

Shiro的过滤器中配置

*<!-- 配置退出的操作 -->*/logout.do = logout

## 11 ajax请求时，权限验证不通过处理

自定义权限验证不通过的处理类

**public class** MyPermFilter **extends** PermissionsAuthorizationFilter{  
  
 @Override  
 **protected boolean** onAccessDenied(ServletRequest request, ServletResponse response) **throws** IOException {  
  
 HttpServletRequest httpRequest = (HttpServletRequest)request;  
 HttpServletResponse httpResponse = (HttpServletResponse)response;  
 *// 和ajax请求相关的请求头信息* String header = httpRequest.getHeader(**"X-Requested-With"**);  
 **if**(header != **null** && !header.equals(**""**) && header.equals(**"XMLHttpRequest"**)){  
 JsonBean bean = JsonUtils.*createJsonBean*(0, **"没有访问权限"**);  
 ObjectMapper objMapper = **new** ObjectMapper();  
 *// 通过响应返回json格式字符串* response.getWriter().write(objMapper.writeValueAsString(bean));  
 }**else**{  
 *// 非ajax方式，进行重定向* httpResponse.sendRedirect(((HttpServletRequest) request).getContextPath() + **"/unauthorized.jsp"**);  
 }  
 **return false**;  
 }  
}

修改配置

*<!-- 创建自定义的授权处理的对象 -->*<**bean id="myPermFilter" class="com.qianfeng.realm.MyPermFilter"**></**bean**>

并且在shiro的过滤器里设置：

*<!-- 权限认证失败，则跳转到指定页面 -->  
<!--<property name="unauthorizedUrl" value="/unauthorized.jsp"/>-->  
<!-- 设置自定义的filter -->*<**property name="filters"**>  
 <**map**>  
 *<!-- 注意key的值 -->* <**entry key="perms" value-ref="myPermFilter"** />  
 </**map**>  
</**property**>