# 1、简介

Apache Shiro是Java的一个安全框架。功能强大，使用简单的Java安全框架，它为开发人员提供一个直观而全面的认证，授权，加密及会话管理的解决方案。

实际上，Shiro的主要功能是管理应用程序中与安全相关的全部，同时尽可能支持多种实现方法。Shiro是建立在完善的接口驱动设计和面向对象原则之上的，支持各种自定义行为。Shiro提供的默认实现，使其能完成与其他安全框架同样的功能，这不也是我们一直努力想要得到的吗！

Apache Shiro相当简单，对比Spring Security，可能没有Spring Security做的功能强大，但是在实际工作时可能并不需要那么复杂的东西，所以使用小而简单的Shiro就足够了。对于它俩到底哪个好，这个不必纠结，能更简单的解决项目问题就好了。

Shiro可以非常容易的开发出足够好的应用，其不仅可以用在JavaSE环境，也可以用在JavaEE环境。Shiro可以帮助我们完成：认证、授权、加密、会话管理、与Web集成、缓存等。这不就是我们想要的嘛，而且Shiro的API也是非常简单；其基本功能点如下图所示：



**Authentication：**身份认证/登录，验证用户是不是拥有相应的身份；

**Authorization：**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；

**Session Manager：**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；

**Cryptography：**加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；

**Web Support：**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；

**Concurrency：**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；

**Testing：**提供测试支持；

**Run As：**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；

**Remember Me：**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。

**记住一点，Shiro不会去维护用户、维护权限；这些需要我们自己去设计/提供；然后通过相应的接口注入给Shiro即可。**

接下来我们分别从外部和内部来看看Shiro的架构，对于一个好的框架，从外部来看应该具有非常简单易于使用的API，且API契约明确；从内部来看的话，其应该有一个可扩展的架构，即非常容易插入用户自定义实现，因为任何框架都不能满足所有需求。

首先，我们从外部来看Shiro吧，即从应用程序角度的来观察如何使用Shiro完成工作。如下图：



可以看到：应用代码直接交互的对象是Subject，也就是说Shiro的对外API核心就是Subject；其每个API的含义：

**Subject：**主体，代表了当前“用户”，这个用户不一定是一个具体的人，与当前应用交互的任何东西都是Subject，如网络爬虫，机器人等；即一个抽象概念；所有Subject都绑定到SecurityManager，与Subject的所有交互都会委托给SecurityManager；可以把Subject认为是一个门面；SecurityManager才是实际的执行者；

**SecurityManager：**安全管理器；即所有与安全有关的操作都会与SecurityManager交互；且它管理着所有Subject；可以看出它是Shiro的核心，它负责与后边介绍的其他组件进行交互，如果学习过SpringMVC，你可以把它看成DispatcherServlet前端控制器；

**Realm：**域，Shiro从Realm获取安全数据（如用户、角色、权限），就是说SecurityManager要验证用户身份，那么它需要从Realm获取相应的用户进行比较以确定用户身份是否合法；也需要从Realm得到用户相应的角色/权限进行验证用户是否能进行操作；可以把Realm看成DataSource，即安全数据源。

也就是说对于我们而言，最简单的一个Shiro应用：

1、应用代码通过Subject来进行认证和授权，而Subject又委托给SecurityManager；

2、我们需要给Shiro的SecurityManager注入Realm，从而让SecurityManager能得到合法的用户及其权限进行判断。

**从以上也可以看出，Shiro不提供维护用户/权限，而是通过Realm让开发人员自己注入。**

接下来我们来从Shiro内部来看下Shiro的架构，如下图所示：



**Subject：**主体，可以看到主体可以是任何可以与应用交互的“用户”；

**SecurityManager：**相当于SpringMVC中的DispatcherServlet或者Struts2中的FilterDispatcher；是Shiro的心脏；所有具体的交互都通过SecurityManager进行控制；它管理着所有Subject、且负责进行认证和授权、及会话、缓存的管理。

**Authenticator：**认证器，负责主体认证的，这是一个扩展点，如果用户觉得Shiro默认的不好，可以自定义实现；其需要认证策略（Authentication Strategy），即什么情况下算用户认证通过了；

**Authrizer：**授权器，或者访问控制器，用来决定主体是否有权限进行相应的操作；即控制着用户能访问应用中的哪些功能；

**Realm：**可以有1个或多个Realm，可以认为是安全实体数据源，即用于获取安全实体的；可以是JDBC实现，也可以是LDAP实现，或者内存实现等等；由用户提供；注意：Shiro不知道你的用户/权限存储在哪及以何种格式存储；所以我们一般在应用中都需要实现自己的Realm；

**SessionManager：**如果写过Servlet就应该知道Session的概念，Session呢需要有人去管理它的生命周期，这个组件就是SessionManager；而Shiro并不仅仅可以用在Web环境，也可以用在如普通的JavaSE环境、EJB等环境；所有呢，Shiro就抽象了一个自己的Session来管理主体与应用之间交互的数据；这样的话，比如我们在Web环境用，刚开始是一台Web服务器；接着又上了台EJB服务器；这时想把两台服务器的会话数据放到一个地方，这个时候就可以实现自己的分布式会话（如把数据放到Memcached服务器）；

**SessionDAO：**DAO大家都用过，数据访问对象，用于会话的CRUD，比如我们想把Session保存到数据库，那么可以实现自己的SessionDAO，通过如JDBC写到数据库；比如想把Session放到Memcached中，可以实现自己的Memcached SessionDAO；另外SessionDAO中可以使用Cache进行缓存，以提高性能；

**CacheManager：**缓存控制器，来管理如用户、角色、权限等的缓存的；因为这些数据基本上很少去改变，放到缓存中后可以提高访问的性能

**Cryptography：**密码模块，Shiro提高了一些常见的加密组件用于如密码加密/解密的。

## 1.1 基本使用

### 基本概念

a.认证(authentication):

用户通过证明自己是谁，进而决定用户进入系统。

b.授权(authoriztion):

授权的前提，一定是通过认证。 主要控制用户在系统里，能对什么资源做什么操作！

c. shiro是一个权限管理框架，使用shiro管理权限，能够事半功倍！！

### 导包

shiro-all

|  |
| --- |
|  |
| **public** **void** test() **throws** Exception {  // 1. 创建shiro中的安全管理器  DefaultSecurityManager securityManager = **new** DefaultSecurityManager();    // 2. 使用IniRealm对象类加载配置文件  IniRealm realm = **new** IniRealm("classpath:com/cheney/hello/shiro.ini");    // 3. 把加载了配置信息的realm对象，设置给安全管理器  // 也就是说，安全管理器，已经"知道了"配置文件中的信息：  securityManager.setRealm(realm);    // 4. 把安全管理器，设置为全局的， 以后程序的任何地方都能畅通无阻地使用安全管理器，毕竟安全管理器被设置为全局的了！  // 也就可以认证，在程序的任何地方都可以读取到配置文件中的信息  // 在实际的web开发环境中，该行代码不用写，是由框架自动完成。这里是快速起步例子，所以要自己写出来！  SecurityUtils.*setSecurityManager*(securityManager);    // 5. 获取要登录当前系统的主体（用户)  // 要登录当前系统的主题，未必是人，也有可能是一个机器人程序，只要是与当前系统交换的主体就行。  Subject subject = SecurityUtils.*getSubject*();    // 6. 创建一个令牌，其中封装了用户提供账户和密码  UsernamePasswordToken token = **new** UsernamePasswordToken("zs","123");    // 7. 让主体拿着令牌，去进行认证！  // 注意：认证通过，则程序不会抛出任何异常， 一旦认证失败，就会抛出异常  // UnknownAccountException: 表示账户错误  // IncorrectCredentialsException: 密码错误  **try** {  subject.login(token);  System.***out***.println("认证通过");  } **catch** (Exception e) {  System.***out***.println("认证失败：" + e);  }  // 判断用户是否通过认证:  System.***out***.println("是否通过认证：" + subject.isAuthenticated());    // 8. 授权  // 在用户通过认证之后，就可以判断用户是否具有某个角色  System.***out***.println("admin:" + subject.hasRole("admin"));  System.***out***.println("guest:" + subject.hasRole("guest"));  // 在用户通过认证之后，就可以判断用户是否具有某个权限  System.***out***.println("user:save " + subject.isPermitted("user:save"));  System.***out***.println("user:delete " + subject.isPermitted("user:delete"));  System.***out***.println("user:update " + subject.isPermitted("user:update"));  System.***out***.println("user:find " + subject.isPermitted("user:find"));    // 9. 登出  subject.logout();  System.***out***.println("是否通过认证：" + subject.isAuthenticated());  } |

## 1.2 原理

|  |
| --- |
| http://dl2.iteye.com/upload/attachment/0093/9790/5e0e9b41-0cca-367f-8c87-a8398910e7a6.png |
| http://dl2.iteye.com/upload/attachment/0093/9792/9b959a65-799d-396e-b5f5-b4fcfe88f53c.png |

## 1.3 自定义shiro

必须继承AuthorizingRealm

|  |
| --- |
|  |
|  |

### 认证原理

|  |
| --- |
| 认证原理 |

## 1.4 多个realm

|  |
| --- |
|  |
|  |
|  |

## 1.5 jdbcRealm【】

### 表结构

|  |
| --- |
|  |

### 步骤

|  |
| --- |
| **public** **void** test() **throws** Exception {  // 1. 安全管理器  DefaultSecurityManager securityManager = **new** DefaultSecurityManager();  // 2. 定义数据源  DruidDataSource dataSource = **new** DruidDataSource();  dataSource.setDriverClassName("com.mysql.jdbc.Driver");  dataSource.setUrl("jdbc:mysql://localhost:3306/shiro");  dataSource.setUsername("root");  dataSource.setPassword("root");  // 3. JdbcRealm  JdbcRealm realm = **new** JdbcRealm();  realm.setDataSource(dataSource);  // 4. 重写JdbcRealm的sql语句  // 认证sql, 该sql执行时机是: subject.login(token);  realm.setAuthenticationQuery("select password from users where username = ?");  // 根据身份获取用户角色的sql,该sql执行时机是: subject.hasRole("admin");  realm.setUserRolesQuery("SELECT rname from users u JOIN users\_roles ur " +  "on u.uid = ur.uid " +  "join roles r " +  "on ur.rid = r.rid " +  "where username = ?");  // 根据角色获取用户权限的sql  realm.setPermissionsQuery("SELECT pname FROM roles\_permissions rp JOIN roles r " +  "ON rp.rid = r.rid " +  "JOIN permissions p " +  "ON rp.pid = p.pid " +  "WHERE rname = ?");  /\*注意，shiro默认情况下，只能根据身份来获取角色，  \* 当要判断用户是否拥有某个全显示，shiro不能那么“智能地”先根据身份获取角色，再根据角色获取权限。  \*为了让shiro能够根据身份 间接地获取到 权限，需要添加以下代码  \*/  realm.setPermissionsLookupEnabled(**true**);  securityManager.setRealm(realm);  SecurityUtils.*setSecurityManager*(securityManager);  Subject subject = SecurityUtils.*getSubject*();  // 认证  UsernamePasswordToken token = **new** UsernamePasswordToken("foo", "123");  **try** {  /\*\*  \* 此时，当我们认证时，shiro仍然会调用Realm的doGetAuthenticationInfo（这和以前的流程是一样的）  \* 值得一提的是，在JdbcRealm的doGetAuthenticationInfo方法中，是通过sql语句来完成认证的！  \* 现在的认证对应的sql句子是：select password from users where username = ?  \* 可以看出该sql中有一个占位符，shiro框架会自动把token中的账号，填入？中。 如果查出的结果集，一行都没有，证明账户错误！  \* 如果查出的结果集，有行，至少说明账户是对的，至于密码是否正确，还要再次比对。  \*/  subject.login(token);  System.***out***.println("认证成功");  } **catch** (Exception e) {  e.printStackTrace();  System.***out***.println("认证失败：" + e);  **return**;  }  //  System.***out***.println("身份：" + subject.getPrincipal());  // 授权  System.***out***.println("admin角色:" + subject.hasRole("admin"));  System.***out***.println("guest角色:" + subject.hasRole("guest"));    System.***out***.println("user:save " + subject.isPermitted("user:save"));  System.***out***.println("user:delete " + subject.isPermitted("user:delete"));  System.***out***.println("user:update " + subject.isPermitted("user:update"));  System.***out***.println("user:find " + subject.isPermitted("user:find"));  // 登出  subject.logout();  } |

## 1.6 授权表达式

\*：\*

\*：save

users：\*

users：update：1 --> 1指id

# 加密

什么是加密：

明文 --> 密钥 --> 密文

123 --> abc -> xyz

解密：

xyz --> abc --> 123

加密方式，总共分为3大类：

1. 对称加密：加密使用的密钥，和解密使用的密钥是同一个密钥

2. 非对称加密：加密使用的密钥，和解密使用的密钥不是同一个密钥

3. 不可逆加密：只能使用加密明文，得出的密文无法解密

## MD5

MD5加密属于不可逆加密。

MD5加密的原理，就是使用hash哈希算法

|  |
| --- |
|  |

## Shiro中使用MD5加密

|  |
| --- |
| **public** **static** **void** main(String[] args) {  DefaultSecurityManager dsm = **new** DefaultSecurityManager();    DruidDataSource ds = **new** DruidDataSource();  ds.setDriverClassName("com.mysql.jdbc.Driver");  ds.setUrl("jdbc:mysql://localhost:3306/shiro?characterEncoding=utf8");  ds.setUsername("root");  ds.setPassword("root");    JdbcRealm realm = **new** JdbcRealm();  realm.setDataSource(ds);    // 重写sql语句:  // 1. 认证sql, 该sql执行时机是: subject.login(token);  realm.setAuthenticationQuery("select password, salt from users where username = ?");  // 2. 根据身份获取用户角色的sql,该sql执行时机是: subject.hasRole("admin");  realm.setUserRolesQuery("SELECT rname FROM users\_roles ur JOIN users u ON ur.uid = u.uid JOIN roles r ON ur.rid = r.rid WHERE username = ?");  // 3. 根据角色获取用户权限的sql  realm.setPermissionsQuery("SELECT pname FROM roles\_permissions rp JOIN roles r ON rp.rid = r.rid JOIN permissions p ON rp.pid = p.pid WHERE rname = ?");  // 4. 为了能根据用户的身份来获取权限，需要添加以下配置： 1. 现根据身份获取角色， 2 再根据角色获取权限。  realm.setPermissionsLookupEnabled(**true**);    // 把加密需要的 算法、盐、迭代次数，告诉shiro，确切地说，就是告诉realm  // 1. 该对象只能设置2个信息：算法 迭代次数  HashedCredentialsMatcher hcm = **new** HashedCredentialsMatcher();  hcm.setHashAlgorithmName("md5");  hcm.setHashIterations(1024);  realm.setCredentialsMatcher(hcm);    // 2. 盐的设置  // 2.1. 给users表中，添加一个列，专门存放盐，也就说，盐是存放在数据库表中的！而不是这里配置的！  //ALTER TABLE users ADD COLUMN salt VARCHAR(20);  // 2.2 修改认证的sql语句  // select password, salt from users where username = ?  // 2.3. 告诉shiro，在认证sql语句中，查询出的结果中，包含盐。否则就算我们的认证sql语句确实把盐查出来了，shiro也不知道！  realm.setSaltStyle(SaltStyle.***COLUMN***);    dsm.setRealm(realm);    SecurityUtils.*setSecurityManager*(dsm);  Subject subject = SecurityUtils.*getSubject*();    UsernamePasswordToken token = **new** UsernamePasswordToken("foo", "123");  **try** {  // 认证  subject.login(token);  System.***out***.println("认证成功");  } **catch** (AuthenticationException e) {  e.printStackTrace();  System.***out***.println("认证失败:" + e);  **return**;  }  System.***out***.println("身份：" + subject.getPrincipal());  subject.logout();  } |

# 3、过滤器 权限拦截器

|  |  |
| --- | --- |
| 过滤器简称 | 对应的java类 |
| anon  （允许匿名访问） | org.apache.shiro.web.filter.authc.AnonymousFilter |
| authc  （需要认证） | org.apache.shiro.web.filter.authc.FormAuthenticationFilter |
| authcBasic | org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter |
| perms  （权限） | org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter |
| port | org.apache.shiro.web.filter.authz.PortFilter |
| rest | org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter |
| roles  （角色） | org.apache.shiro.web.filter.authz.RolesAuthorizationFilter |
| ssl | org.apache.shiro.web.filter.authz.SslFilter |
| user | org.apache.shiro.web.filter.authc.UserFilter |
| logout  （登出） | org.apache.shiro.web.filter.authc.LogoutFilter |

# 4、Jsp shiro标签

|  |  |
| --- | --- |
| **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 |

# 5、springmvc集成shiro

## 在web.xml中配置代理过滤器

|  |
| --- |
|  |

## Controller

|  |
| --- |
|  |

## spring-servlet.xml中配置shiro

|  |
| --- |
| <!-- 配置数据源 -->  <bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"* >  <property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*></property>  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/shiro"*></property>  <property name=*"username"* value=*"root"*></property>  <property name=*"password"* value=*"root"*></property>  </bean>  <!-- 配置shiro开始 -->  <!-- 0. 配置加密密钥 -->  <bean id=*"hash"* class=*"org.apache.shiro.authc.credential.HashedCredentialsMatcher"*>  <property name=*"hashAlgorithmName"* value=*"md5"*></property>  <property name=*"hashIterations"* value=*"1024"*></property>  </bean>  <!-- 1. 配置JdbcRealm -->  <bean id=*"realm"* class=*"org.apache.shiro.realm.jdbc.JdbcRealm"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  <property name=*"authenticationQuery"* value=*"select password, salt from users where username = ?"*></property>  <property name=*"userRolesQuery"* value=*"SELECT rname FROM users\_roles ur JOIN users u ON ur.uid = u.uid JOIN roles r ON ur.rid = r.rid WHERE username = ?"*></property>  <property name=*"permissionsQuery"* value=*"SELECT pname FROM roles\_permissions rp JOIN roles r ON rp.rid = r.rid JOIN permissions p ON rp.pid = p.pid WHERE rname = ?"*></property>  <property name=*"permissionsLookupEnabled"* value=*"true"*></property>  <property name=*"saltStyle"* value=*"COLUMN"*></property>  <property name=*"credentialsMatcher"* ref=*"hash"*></property>  </bean>  <!-- 2.配置安全管理器  我们自己测试时，环境是java环境，不是web环境，所有使用的是DefaultSecurityManager  而现在是web环境，所以类型也就替换为了： org.apache.shiro.web.mgt.DefaultWebSecurityManager  -->  <bean id=*"securityManager"* class=*"org.apache.shiro.web.mgt.DefaultWebSecurityManager"*>  <property name=*"realm"* ref=*"realm"*></property>  </bean>  <!-- 3.配置shiroFilter -->  <bean id=*"shiroFilter"* class=*"org.apache.shiro.spring.web.ShiroFilterFactoryBean"*>  <!-- 3.1 把安全管理器，设置为全局的，等价于以前的： SecurityUtils.setSecurityManager(securityManager) -->  <property name=*"securityManager"* ref=*"securityManager"*></property>  <!-- 3.2 用户没有通过认证的跳转界面， 默认为login.jsp -->  <property name=*"loginUrl"* value=*"index.jsp"*></property>  <!-- 3.3 用户经过认证，但是没有对应的角色或权限的跳转界面 -->  <property name=*"unauthorizedUrl"* value=*"/unauthorized.jsp"*></property>  <!-- 3.4 默认情况下，shiroFilter会放行所有请求 -->  <property name=*"filterChainDefinitions"*>  <value>  /index.jsp = anon <!-- anon允许匿名访问，也就是允许不认证就能访问 -->  /users/login = anon  /logout = logout <!-- 安全退出后，logout拦截器会自动跳转到应用默认路径 -->  /users.jsp = authc, roles[admin]  /cars.jsp = authc, roles[guest]  /home.jsp = authc <!-- 如果当前主体没有通过认证，就默认到login.jsp页面 -->  /\*\* = authc  </value>  </property>  </bean>  <!-- 配置shiro结束 --> |

## 配置权限及角色

路径会从上往下匹配

/\*\* 表示所有，放在最后面

logout拦截器会自动跳转到应用默认路径即根路径

|  |
| --- |
|  |

* Xml中多角色默认是and，如果想要or，则需要自定义拦截器

roles[“guest”,”admin”] and

# shiro缓存

shiro默认是没有缓存的，导致在一次会话中，会重复授权

## 缓存

1. 缓存是什么：

缓存是一个存储单元，这个存储单元中所存储的数据更接近于使用者!

1. 缓存是如何工作的:

先从缓存中读取数据，如果缓存中没有想要读取的数据，才会在慢速设备上读取实际数据！

比如cpu的缓存就是在cpu里，且cpu读取数据的顺序是：cpu-->L1/L2-->内存-->磁盘。

## shiro集成ehcache

### 导入依赖



### 添加ehcache.xml配置文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <ehcache>  <diskStore path=*"java.io.tmpdir"*/>  <defaultCache  maxElementsInMemory=*"10000"*  eternal=*"false"*  timeToIdleSeconds=*"5"*  timeToLiveSeconds=*"120"*  overflowToDisk=*"true"*  />  </ehcache> |

### 开启缓存功能

|  |
| --- |
|  |

# 7、Shiro注解

## 开启注解

|  |
| --- |
|  |

## 使用注解

|  |
| --- |
|  |

# 异常处理，页面优化

当认证授权时，出现异常，可以使用异常处理器，避免客户端出现500界面

## 方法一：自定义异常处理器

|  |
| --- |
|  |

## 方法二：SimpleMappingExceptionResolver

|  |
| --- |
|  |

# 9、Spring security 与apache shiro 差别：

shiro配置更加容易理解，容易上手；security配置相对比较难懂。

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。

shiro 不仅仅可以使用在web中，它可以工作在任何应用环境中。

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。

Shiro提供的密码加密使用起来非常方便。

# 10、控制精度：

Shiro也支持注解方式。

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。