**四大基石**

Authentication：认证，Authorization：授权，Cryptography：加密，Session管理。

**三大核心概念**

**Subject(主体)**：当前操作用户。之所以没有用User这个单词，是因为User一般指人。而在安全界，Subject表示任何与软件交互的主体。使用下面语句来获取一个Subject：

*Subject currentUser = SecurityUtils.getSubject();*

**SecurityManager**：和Subject对应，管理所有安全操作(Security operations)，是Shiro的核心。在Web环境下，通过servlet filter来配置；在独立环境下，可以使用配置文件如ini和xml之类。SecurityManager是单例的，类似于Hibernate中的SessionFactory。

**Realms(领域)**：Shiro和安全数据(security data)之间的桥梁，相当于访问authentication和authorization数据(如username、password、roles、permissions)的DAO。

**基础讲解**

Authentication：

向系统提交用户识别信息(**principal**，如用户名)和识别验证(**credential**，如密码)，如果匹配则通过。

在Shiro中，把principal和credential组合为AuthenticationToken，最常用的就是username和password组合。因此，一个最简单的登录过程如下：

*AuthenticationToken token = new UsernamePasswordToken(username, password);*

*Subject currentUser = SecurityUtils.getSubject();*

*currentUser.login(token);*

Authorization：

本质是访问控制(Access Control)。一般系统使用角色(**role**)和许可(**permission**)来判断是否拥有某访问权限。

*if ( subject.hasRole(“administrator”) ) {*

*//show the ‘Create User’ button*

*} else {*

*//grey-out the button?*

*}*

这种硬编码方式显然有很大的不足。Shiro提供了自己的Permission：一个Permission描述一个功能点，如”user:create”。上面代码改为：

*if ( subject.isPermitted(“user:create”) ) {*

*//show the ‘Create User’ button*

*} else {*

*//grey-out the button?*

*}*

注意permission语句”user:create”，这是一个通配符许可(WildcardPermission)。

重点：Permission，通配符

Session Management：

不管是独立项目还是Web项目，不管部署在Tomcat还是Jetty，不管是单机还是集群，Shiro都提供了一致的Session API。另外，Shiro的Session支持终端共享，如同时使用client和browser登录系统。

*Session session = subject.getSession();* // 和HttpServletRequest中的Session API 一样

Cryptography：

加密。主要是两方面：信息摘要和加密。Shiro提供的API十分简洁：

*String hex = new Md5Hash(myFile).toHex();* // 获取myFile的MD5，结果为16进制字符串

*String encodedPassword =* // 对password进行sha512摘要并转成base64编码

*new Sha512Hash(password, salt, count).toBase64();*

*AesCipherService cipherService = new AesCipherService();*

*cipherService.setKeySize(256);*

//create a test key:

*byte[] testKey = cipherService.generateNewKey();*

//encrypt a file’s bytes:

*byte[] encrypted =*

*cipherService.encrypt(fileBytes, testKey);*

Web支持：

Shiro对Web项目提供了良好支持。

1、添加Shiro Servlet Filter

<filter>

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

<filter-class>

org.apache.shiro.web.servlet.IniShiroFilter

</filter-class>

<!-- no init-param means load the INI config from classpath:shiro.ini -->

</filter>

<filter-mapping>

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

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

</filter-mapping>

2、Taglib

**RBAC：Resource Based Access Control，基于资源的访问控制**

资源:动作:实例，如projectReport:view:123，意为可以view 标识为123的projectReport。在这里，projectReport即为resource，有的地方也叫domain；view是action；123是具体的实例，一般用标识符来区别。

**Meri的博客**

IniShiroFilter只是从ini文件中读取配置并初始化框架，本身并不做任何安全检查。它将这些代理给其他filter，它所做的仅仅是初始化。

ini配置文件包含4个部分。

[main]包含了shiro的初始化。Filter和用户对象也在这里配置。

[users]定义用户、密码和角色，格式：user=password,[role1][role2]...

[roles]将roles和permission关联起来。

[urls]指定url的访问权利，通过通过默认或者用户扩展的filter来实现。

**Realms**

realm承担authentication和authorization的职责。当用户想登录应用时，authentication信息会被收集并传给realm，realm验证提供的数据并决定是否允许用户登录、访问资源和拥有角色。

认证信息分两部分：

principal 账户唯一标识 如userName，accountID，PGP证书等

credential 用户标识证明 如password，指纹等

shiro提供了多种realm可以分别从ldap、ini、properties和数据库中读取authentication信息。如从数据库，需要在ini中配置realm如下：

*[main]*

*realmName = org.apache.shiro.realm.jdbc.JDBCRealm*

realm都实现了Realm接口。该接口有两个重要方法：**supports**和**getAuthenticationInfo**，都接收AuthenticationToken(内部包含principal和credential)作为参数。supports方法决定该Realm处理哪种AuthenticationToken；getAuthenticationInfo方法则对认证信息进行匹配，判断是否通过。

realm如果要实现authorization功能，则要实现Authorizer接口。该接口中的方法接收principal为参数并判断是否拥有role和permission。

Permission可以是字符串或者对象，如非必要，shiro提供的**WildcardPermissionResolver**已经足够。

**Shiro with ZK**

使用表单登录时，表单名称必须是loginform。当登录失败时，shiro会在Request中放置一个attribute来存储错误信息。

**Reference**

架构中的核心类：

**Authenticator**，当Subject登录时，Authenticator协调一到多个Realm确认Subject身份。

**AuthenticationStrategy**，当有多种登录形式时，需要配置该策略来决定，只要有一种形式成功就能登录还是必须全部通过才能登录等等。

**Authorizer**，和Authenticator一样，协调一到多个Realm判断Subject是否有权限做某事。

**SessionManager**，屏蔽底层差异，向独立环境和web环境下的应用提供统一的Session访问API。当系统中已存在Session时，会使用系统的Session，如Servlet中的Session。**SessionDAO**负责Session的持久化，同样支持多种数据源。

**CacheManager**，集成各种Cache中间件，提供cache功能。



**配置(ini文件形式)**

[main]：配置SecurityManager以及其依赖类。

属性设置，支持简单类型，对象类型($符号+名称)，二进制(默认使用Base64编码，也可使用16进制)。对于list、set类型，直接用”,”分开；对于map类型，用”:”和”,”分开。

注意：配置的顺序很重要，不要写错。

[users]：如果需要对密码加密，则需为realm配置一个CredentialsMatcher，如下

*sha256Matcher = org.apache.shiro.authc.credential.Sha256CredentialsMatcher*

*iniRealm.credentialsMatcher = $sha256Matcher*

remember me功能：只是说subject.getPrinciples()不为空，并不代表再次通过验证。主要应用场景是session失效但未退出系统。比如卓越网的相关功能。

logout后应该立即跳转到另一个页面。

**Authentication流程**

1、subject.login()，将principal和credential信息封装在AuthenticationToken中传入。

2、SecurityManager.login(token)，代理给相关对象执行。

3、代理给authenticator，调用其authenticate(token)方法，该方法协调一个或多个Realm来验证。

4、如果配有多个Realm，则需要根据AuthenticationStrategy来判断。

5、调用Realm的supports方法来判断该realm是否支持当前的AuthenticationToken验证，如支持，调用其getAuthenticationInfo()方法。

Authenticator一般使用默认的ModularRealmAuthenticator即可。

**Authorization**

1、基于role。主要用于系统菜单权限。

2、基于permission。主要用于页面按钮；对于数据实例级别的权限(比如id为1的book)，使用permission的”domain:action:instance”来标注。

使用注解(需开启AOP，可使用SpringAOP、AspectJ等)：

@RequiresAuthentication，没有通过认证则抛出异常。

@RequiresPermissions，没有通过授权则抛出异常。

其他，都以@Requires开头。

授权流程：

1、调用subject.isPermitted\*()，进入SecurityManager

2、SecurityManager将该方法代理给Authorizer。该类作用和Authenticator类似，协调多个realm，该类的实例一般是ModularRealmAuthorizer，和Authenticator那边类似。

3、进入Realm，进行授权验证的Realm都实现了Authorizer接口。没有实现该接口的Realm将被忽略。

Permission表达式默认为WildcardPermission格式，使用WildcardPermissionResolver来解析为Permission对象。在WildcardPermission表达式中，”\*”和”,”在每个部分都可以使用，如\*:print,query:lp7200；对于缺失的部分，只能向后匹配，比如printer=printer:\*，但query不等于\*:query:\*。

指定permission的时候越简略越方便，但检查permission的时候却是越具体越好。这是因为判断的时候使用的是匹配而非相等。另：使用Cache能大幅度提高判断速度。

**Realm**

Realm相当于DAO，同时获取authentication(principal、credential数据)和authorization(role、permission)数据，因此可以同时支持authentication和authorization操作，不过后者需要实现Authorizer接口。我们可以继承AuthorizingRealm来实现自己的realm，避免过多的编码。

CredentialsMatcher用于匹配Credential，比如一般使用密文来匹配用户密码。

**Session Management**

特性：对IOC友好，符合POJO规范；方便的自定义存储；不依赖于容器的集群；支持多种客户端访问；事件监听；Host IP记忆；对Web应用透明；可用于SSO。

**SessionManager**管理全局Session设置，自身是SecurityManager的管理对象。全局session超时设置：[main] securityManager.sessionManager.globalSessionTimeout = 3600000 默认30分钟。每一个session都可以自行管理自己的超时时间。

事件监听：可以实现SessionListener或者SessionListenerAdapter接口，针对所有的session。

存储：Session可以存放于内存、数据库、文本系统等等。内部的CRUD操作实际使用的是**SessionDAO**。在Web环境中，基于Servlet容器的应用使用的不是Shiro原生的Session，而是HttpSession。原生的Session默认使用的是内存模式，可能并不适合用于生产环境。此时可以替换SessionDAO，如改为Ehcache SessionDAO。

SessionID：默认使用的是UUID，足够了。

Session必须被校验才能将已失效的session从存储位置删除。对于不再使用的Session，如果长期不validate，则有可能挤满存储。Shirio提供了SessionValidationScheduler，定期检查session状态，如果已失效则将其删除。

**Web集成**

从1.2之后，shiro在Web环境中的初始化改为如下方式(在web.xml中)：

<listener>

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

</listener>

...

<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>

<dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher>

<dispatcher>INCLUDE</dispatcher>

<dispatcher>ERROR</dispatcher>

</filter-mapping>

shiro.ini文件依次从/WEB-INF/和classpath的根目录下加载。

EnvironmentLoaderListener初始化一个WebEnvironment，包含shiro所需的一切，包括SecurityManager。可以通过WebUtils.getRequiredWebEnvironment(servletContext) 来获得这个对象。ShiroFilter对所有请求拦截，进行可能的安全操作，它应该尽可能放在其他filter前面。

[urls]配置格式：ant-style url expression = filter1[optional\_config], filter2[optional\_config]...

如： /user/create/\*\* = authc,roles[administrator],perms[“user:create”]

匹配的时候，以第一个匹配上的url为准，后面如果还有能匹配的将被忽略。如果想实现自己的filter，可以继承PathMatchingFilter。多个filter，按顺序调用。

默认过滤器列表：

anon，匿名用户；authc，通过表单验证；authcBasic，通过Http验证；logout，退出；noSessionCreation，不创建session；perms，必须拥有某permission；port，端口；rest，必须经过某Http方法；roles，必须拥有某角色；ssl，通过sssl协议；user，必须是某用户。

在开发环境中，可能需要禁用某些过滤器以方便开发，比如SSL。可以全局禁用它，只需要[main]ssl.enabled = false

在web环境中，默认使用的SessionManager是ServletContainerSessionManager，这个类是与容器相关的，可能会造成开发环境、生产环境代码的不一致。可以改为shiro的native会话管理DefaultWebSessionManager，配置如下：

*[main]*

*sessionManager = org.apache.shiro.web.session.mgt.DefaultWebSessionManager*

*# configure properties (like session timeout) here if desired*

*# Use the configured native session manager:*

*securityManager.sessionManager = $sessionManager*

JSP Tag

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

<shiro:guest> 当前subject是guest，没有登录，没有被remember me

<shiro:user> 当前subject有identity标识，可以是remember me，也可以是登录用户

<shiro:authenticated> 已登录

<shiro: notAuthenticated > 未登录

<shiro:hasRole> <shiro:lacksRole> <shiro:hasAnyRole> 拥有[没有]某角色，拥有列表角色中任意一个。

<shiro:hasPermission> <shiro:lacksPermission> 拥有[没有]某权限。

**缓存**

三个接口：

CacheManager 和EhCache里的一样，管理cache；

Cache 和EhCache里的一样，管理key/value对；

CacheManagerAware 实现该接口可自动获取CacheManager，和Spring中xxAware一样

**集成**

集成CAS：Jasing CAS SSO

CAS登录步骤(假如用户未登录)

1、redirect到CAS服务器，url类似于

https://server.cas.com/login?service=http://application.examples.com/shiro-cas

<https://server.cas.com/login>是CAS服务登录地址，service是固定的参数

<http://application.examples.com/shiro-cas>是用户实际想登录的系统

2、CAS服务器对用户进行验证，如验证通过，返回

http://application.examples.com/shiro-cas?ticket=ST-4545454542121-cas

ticket是一个临时的令牌

3、shiro-cas系统向CAS服务器查询ticket是否可用，如可用，再redirect到

<http://application.examples.com/protected/index.jsp> (最初想访问的地址)

配置CasFilter，CasRealm

集成Spring

参见<http://shiro.apache.org/spring.html>