

shiro 权限框架介绍

李泽昊

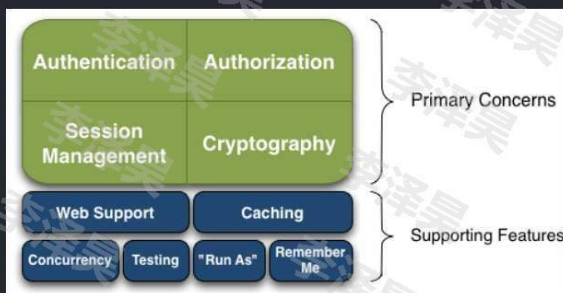
2016-12-20

<http://blog.csdn.net/Q0494406030>

Just for Smart Life

1.简介

- Shiro是一个安全框架，是Apache 的一个子项目。Shiro提供了：认证、授权、加密、会话管理、与Web集成、缓存等模块。



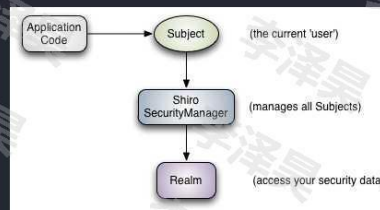
智享未来

1.1 模块介绍

- **Authentication** : 用户身份识别, 可以认为是登录;
- **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;
- **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不会去维护用户和权限之间的关系; 需要我们去设计/提供; 然后通过相应的接口注入给Shiro即可。

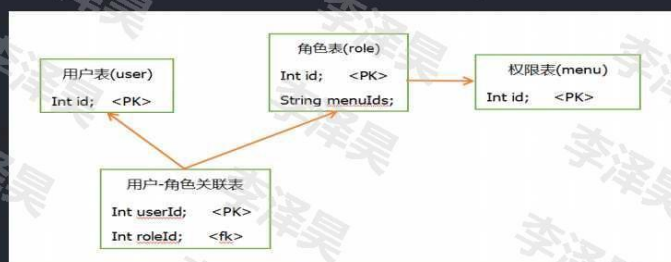
2. 核心概念:

- **Subject** : 主体, 代表了当前操作“用户”, 这个用户不一定是具体的人, 与当前应用交互的任何东西都是Subject, 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;
- **SecurityManager** : 安全管理器; 即所有与subject安全有关的操作都会与SecurityManager交互; 且它管理着所有Subject; 它负责与里面的各个组件进行交互, 可以把它理解看成SpringMVC DispatcherServlet前端控制器;
- **Realm** : 域, 安全数据源。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。



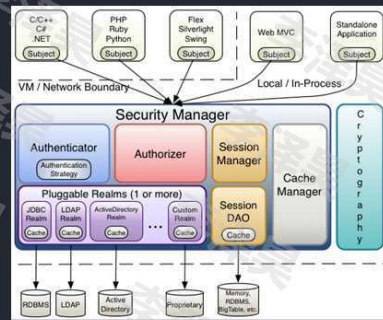
2. 核心概念:

- 从上图可以看出:
- 1、应用代码通过Subject来进行认证和授权, 而Subject又委托给SecurityManager;
- 2、SecurityManager要验证用户身份, 那么它需要从Realm获取相应的用户/角色/权限进行比较以确定用户身份是否合法;
- 总结: Shiro不提供维护用户/权限, 而是通过Realm让开发人员自己注入。



3.shiro内部架构介绍

- **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, 即登录的认证, 校验用户身份的合法性;
- **Authorizer** : 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;



智享未来

云和声 Unisound

3.shiro内部架构介绍

- **Realm** : 可以认为是安全实体数据源, 可以是JDBC实现, 或者内存实现等等, 由用户提供; 执行认证(登录)和授权(访问控制)时, Shiro会从应用配置的Realm中查找很多内容。注意: Shiro不知道你的用户/权限存储在哪以及以何种格式存储; 所以我们一般在应用中都需要实现自己的Realm;
- **SessionManager** : shiro有自己的session管理机制, 她位于web容器, 由SessionManager这个组件管理他的生命周期。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);

智享未来

云和声 Unisound

3.shiro内部架构介绍

- **SessionDAO** : DAO大家都用过, 数据访问对象, 用于会话的CRUD, 实现在线会话管理, 比如我们可以把Session保存到数据库, 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。

智享未来

云和声 Unisound

4. 身份认证

概念:

- **身份验证**，当前用户身份的合法性。一般提供如他们的身份ID一些标识信息来表明他，如用户名/密码来证明。
- **principals**：身份，即主体的标识属性，如用户名、邮箱等，唯一即可。一般是用户名/手机号。
- **credentials**：证明/凭证，即只有主体知道的安全值，如密码/数字证书等。
最常见的principals和credentials组合就是用户名/密码了。
- subject**：主体，外界通过Subject接口来和SecurityManager进行交互，**subject**接口含有登录、退出、权限判断、获取session,其中的Session可不是平常我们所使用的HttpSession等，而是shiro自定义的是一个数据上下文，与一个Subject相关联的。
- realm**：验证主体的数据源。

智享未来

云和声
Unisaund

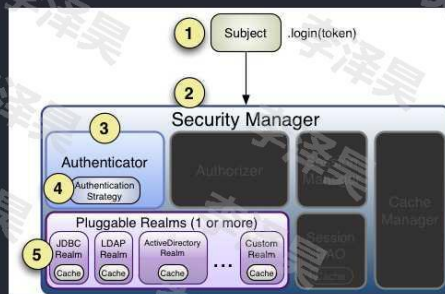
4.1 认证流程

1、应用程序构建了一个终端用户认证信息的AuthenticationToken实例后，调用Subject.login方法。

• 2、Sbjeect会委托应用程序设置的securityManager实例调用securityManager.login(token)方法。

• 3、SecurityManager接受到token(令牌)信息后会委托内置的Authenticator的实例（通常都是ModularRealmAuthenticator类的实例）调用authenticator.authenticate(token)。

ModularRealmAuthenticator在认证过程中会对设置的一个或多个Realm实例进行适配，它实际上为Shiro提供了一个可拔插的认证机制。



智享未来

云和声
Unisaund

4.1 认证流程

4、如果在应用程序中配置了多个Realm，ModularRealmAuthenticator会根据配置的AuthenticationStrategy(认证策略)来进行多Realm的认证过程。在Realm被调用后，AuthenticationStrategy将对每一个Realm的结果作出响应。

• 注：如果应用程序中仅配置了一个Realm，Realm将被直接调用而无需再配置认证策略。

5、Realm将调用getAuthenticationInfo(token);getAuthenticationInfo方法就是实际认证处理，我们通过覆盖Realm的doGetAuthenticationInfo方法来编写我们自定义的认证处理。

认证策略

FirstSuccessfulStrategy：只要有一个Realm验证成功即可，只返回第一个Realm身份验证成功的认证信息，其他的忽略；

AtLeastOneSuccessfulStrategy：只要有一个Realm验证成功即可，和FirstSuccessfulStrategy不同，返回所有Realm身份验证成功的认证信息；

AllSuccessfulStrategy：所有Realm验证成功才算成功，且返回所有Realm身份验证成功的认证信息，如果有一个失败就失败了。

智享未来

云和声
Unisaund

4.1 认证流程

```
ShiroTest_01.java shiro.ini
1 [users]
2 zhang=123
3 wang=123
```

1.subject.login(token)登录的时候，会调用代理的DelegatingSubject的login();

```
ShiroTest_01.java De DelegatingSubject
1 package com.shiro;
2
3 import org.apache.shiro.SecurityUtils;
4
5 /**
6  * @title: ShiroTest_01
7  * @company: 北京云和声信息技术有限公司
8  * @author: liizhen
9  * @date: 2018年12月02日
10 */
11
12 public class ShiroTest_01 {
13
14     @Test
15     public void testHelloWorld() {
16         // 1. 创建SecurityManager工厂，使用ini配置文件初始化SecurityManager
17         Factory<org.apache.shiro.mgt.SecurityManager> factory = new IniSecurityManagerFactory("classpath:shiro.ini");
18         // 2. 创建SecurityManager实例并调用SecurityUtils
19         org.apache.shiro.mgt.SecurityManager securityManager = factory.getInstance();
20         // 3. 设置SecurityManager
21         SecurityUtils.setSecurityManager(securityManager);
22         // 4. 创建Subject并设置认证令牌 (token, 用户名/密码)
23         Subject subject = SecurityUtils.getSubject();
24         UsernamePasswordToken token = new UsernamePasswordToken("zhang", "123");
25         // 5. 设置rememberMe (true);
26         token.setRememberMe(true);
27
28         try {
29             // 4. 登录，调用login
30             subject.login(token);
31         } catch (AuthenticationException e) {
32             // 5. 身份认证失败
33         }
34
35         System.out.println(subject.isAuthenticated());
36         System.out.println(subject.isPermitted("1"));
37         // 6. 退出
38         subject.logout();
39     }
40 }
41
42 }
```

4.1 认证流程

2.subject会委托给securityManager调用login(),而securityManager会把具体的操作委托给内部管理的组件来执行，即:Authenticator

```
ShiroTest_01.java De DelegatingSubject
254 public void login(AuthenticationToken token) throws AuthenticationException {
255     clearRunAsIdentityInternal();
256     Subject subject = securityManager.login(this, token);
257
258     PrincipalCollection principals;
259
260     String host = null;
261
262     if (subject instanceof DelegatingSubject) {
263         DelegatingSubject delegating = (DelegatingSubject) subject;
264         // we have to do this in case there are assumed identities - we don't want to lose the 'real'
265         principals = delegating.principals;
266         host = delegating.host;
267     } else {
268         principals = subject.getPrincipals();
269     }
270
271     if (principals == null || principals.isEmpty()) {
272         String msg = "Principals returned from securityManager.login( token ) returned a null or "
273             + "empty value. This value must be non null and populated with one or more elements."
274             + "throw new IllegalStateException(msg);";
275     }
276 }
```

4.1 认证流程

3.实际的认证过程doAuthenticate是交给子类来实现的，AbstractAuthenticator只对认证结果进行处理，认证成功时调用notifySuccess(token, info)通知所有的listener，认证失败时调用notifyFailure(token, ae)通知所有的listener。

```
AbstractAuthenticator.java De ModularRealmAuthenticator
186 *
187 * interface's JavaDoc for a more detailed explanation.
188 */
189 public final AuthenticationInfo authenticate(AuthenticationToken token) throws AuthenticationException {
190     if (token == null) {
191         throw new IllegalArgumentException("Method argument (authentication token) cannot be null.");
192     }
193
194     Log.trace("Authentication attempt received for token [{0}], token);
195
196     AuthenticationInfo info;
197     try {
198         info = doAuthenticate(token);
199     } catch (Throwable t) {
200         String msg = "No account information found for authentication token [" + token + "] by this " +
201             "Authenticator instance. Please check that it is configured correctly.";
202         throw new AuthenticationException(msg);
203     }
204
205     AuthenticationException ae = null;
206     if (!info instanceof AuthenticationException) {
207         ae = (AuthenticationException) t;
208     }
209
210     if (ae == null) {
211         //Exception thrown was not an expected AuthenticationException. Therefore it is probably a little more
212         //severe or unexpected. So, wrap in an AuthenticationException, log to warn, and propagate.
213         String msg = "Authentication failed for token submission [" + token + "]. Possible unexpected " +
214             "error? (typical or expected login exceptions should extend from AuthenticationException).";
215         ae = new AuthenticationException(msg, t);
216         if (Log.isDebugEnabled())
217             Log.warn(msg, t);
218     }
219 }
```


4.1 认证流程

```

255 ModularRealmAuthenticator.class
256 * @throws IllegalStateException if no realms have been configured at the time this method
257 * @throws AuthenticationException if the user could not be authenticated or the user is denied
258 * for the given principal and credentials.
259 */
260
261 protected AuthenticationInfo doAuthenticate(AuthenticationToken authenticationToken) throws AuthenticationException {
262     assertRealmsConfigured();
263     Collection<Realm> realms = getRealms();
264     if (realms.size() == 1) {
265         return doSingleRealmAuthentication(realms.iterator().next(), authenticationToken);
266     } else {
267         return doMultiRealmAuthentication(realms, authenticationToken);
268     }
269 }

```

```

173 ModularRealmAuthenticator.class
174 protected AuthenticationInfo doSingleRealmAuthentication(Realm realm, AuthenticationToken token) {
175     if (!realm.supports(token)) {
176         String msg = "Realm [" + realm + "] does not support authentication token [" +
177             token + "]. Please ensure that the appropriate Realm implementation is " +
178             "configured correctly or that the realm accepts AuthenticationTokens of this type."
179         throw new UnsupportedTokenException(msg);
180     }
181     AuthenticationInfo info = realm.getAuthenticationInfo(token);
182     if (info == null) {
183         String msg = "Realm [" + realm + "] was unable to find account data for the " +
184             "submitted AuthenticationToken [" + token + "].";
185         throw new UnknownAccountException(msg);
186     }
187     return info;
188 }

```

4.1 认证流程

3.最核心的是doGetAuthenticationInfo需要我们自定义realm实现该方法。

4.assertCredentialsMatch进行用户身份匹配认证

```

560 AuthenticationInfo could be found.
561 * @throws AuthenticationException if authentication failed.
562 */
563 public final AuthenticationInfo getAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
564     AuthenticationInfo info = getCachedAuthenticationInfo(token);
565     if (info == null) {
566         //otherwise not cached, perform the lookup:
567         info = doGetAuthenticationInfo(token);
568         Log.debug("Looking up authenticationInfo [{}] from doGetAuthenticationInfo", info);
569         if (token != null && info != null) {
570             cacheAuthenticationInfoIfPossible(token, info);
571         }
572     } else {
573         Log.debug("Using cached authentication info [{}] to perform credentials matching.", info);
574     }
575     if (info != null) {
576         assertCredentialsMatch(token, info);
577     } else {
578         Log.debug("No AuthenticationInfo found for submitted AuthenticationToken [{}]. Returning null.", token);
579     }
580     return info;
581 }

```

5.授权

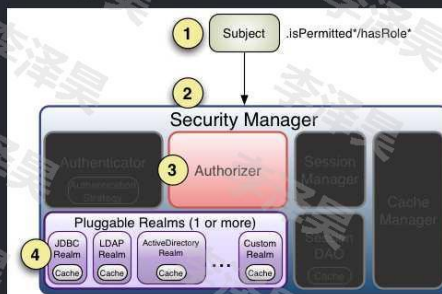
• Shiro授权分为两种类型：

- **粗粒度**：也就是代码中直接写入和角色的绑定。代码耦合度比较高
- **细粒度**：代码中写入的是和权限的绑定，而角色到权限和可配置的。

- 对于粗粒度来说，若角色对应权限有改变的话，那么则需要更改代码，很不方便。
- 对于细粒度的好处显而易见，所以一般项目中应该都采用细粒度的权限配置。

5.1 授权流程

- 1. 首先调用subject.isPermitted()/hasRole()接口，其会委托给SecurityManager，而SecurityManager接着会委托给Authorizer；
- 2. Authorizer是真正的授权者，如果我们调用如isPermitted("user:view")，其首先会通过PermissionResolver把字符串转换成相应的Permission实例；
- 3. 在进行授权之前，其会调用相应的Realm获取Subject相应的角色/权限用于匹配传入的角色/权限；
- 4. Authorizer会判断Realm的角色/权限是否和传入的匹配。



智享未来

云和声
Unisound

5.1 授权流程

```

11= /**
12=  * @title: ShiroTest_01
13=  * @company: 北京云知声信息技术有限公司
14=  * @author: lizehao
15=  * @date: 2016年12月9日
16=  */
17= public class ShiroTest_02 {
18=
19=     @Test
20=     public void testHelloWorld() {
21=         // 1. 获取SecurityManager工厂，使用Ini配置文件初始化SecurityManager
22=         Factory<org.apache.shiro.mgt.SecurityManager> factory = new IniSecurityManagerFactory(
23=             "classpath:shiro-realm.ini");
24=         // 2. 得到SecurityManager实例 并绑定给SecurityUtils
25=         org.apache.shiro.mgt.SecurityManager securityManager = factory.getInstance();
26=         // 全局设置，设置一次
27=         SecurityUtils.setSecurityManager(securityManager);
28=
29=         // 3. 得到Subject及创建用户名/密码身份验证Token（即用户身份/凭证）
30=         Subject subject = SecurityUtils.getSubject();
31=         UsernamePasswordToken token = new UsernamePasswordToken("zhang", "1213");
32=         // token.setRememberMe(true);
33=         try {
34=             // 4. 登录：即身份验证
35=             subject.login(token);
36=         } catch (AuthenticationException e) {
37=             // 5. 身份验证失败
38=         }
39=         System.out.println(subject.isPermitted("111"));
40=         System.out.println(subject.isAuthenticated() == false ? "认证失败" : "认证成功");
41=         // 6. 退出
42=         subject.logout();
43=     }
44= }
  
```

智享未来

云和声
Unisound

5.1 授权流程

- 首先调用subject.isPermitted()时，也是一个代理的subject，其内部委托给SecurityManager

```

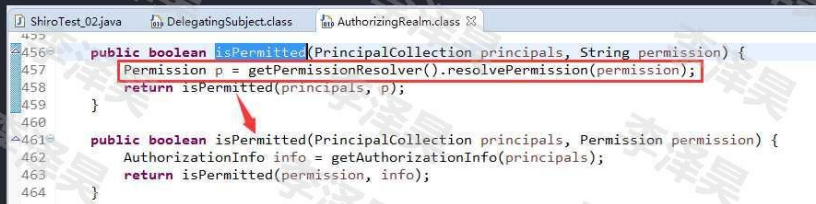
156
157= public boolean isPermitted(String permission) {
158=     return hasPrincipals() && securityManager.isPermitted(getPrincipals(), permission);
159= }
160
  
```

智享未来

云和声
Unisound

5.1 授权流程

- SecurityManager会把具体的操作交给管理的授权器组件执行授权流程。
- Authorizer是真正的授权者，PermissionResolver解析器把字符串转换成相应的Permission实例；



```

456 public boolean isPermitted(PrincipalCollection principals, String permission) {
457     Permission p = getPermissionResolver().resolvePermission(permission);
458     return isPermitted(principals, p);
459 }
460
461 public boolean isPermitted(PrincipalCollection principals, Permission permission) {
462     AuthorizationInfo info = getAuthorizationInfo(principals);
463     return isPermitted(permission, info);
464 }
465

```

智享未来

云和声
Unisound

5.1 授权流程

- protected AuthorizationInfo getAuthorizationInfo(PrincipalCollection principals) {
 - AuthorizationInfo info = null;
 - Cache<Object, AuthorizationInfo> cache = getAvailableAuthorizationCache();
 - if (cache != null) {
 - info = //从缓存中取出 角色 权限相关的信息
 - }
 - if (info == null) {
 - // 这里是授权器的核心关键，需要我们自定义realm实现该方法
 - info = doGetAuthorizationInfo(principals);
 - }
 - return info;
- }

智享未来

云和声
Unisound

5.1 授权流程

- 用于匹配传入的角色/权限(返回true说明权限验证成功)

- protected boolean isPermitted(Permission permission, AuthorizationInfo info) {
 - Collection<Permission> perms = getPermissions(info);
 - if (perms != null && !perms.isEmpty()) {
 - for (Permission perm : perms) {
 - if (perm.implies(permission)) {
 - return true ;
 - }
 - }
 - }

智享未来

云和声
Unisound

6.shiro和spring集成

- 详见:busi-admin-web项目

7.1 退出

- 退出主要执行的操作

- session的销毁
- 清理权限信息缓存

疑惑问题:

- 1.打开两个浏览器,同时登录同一个账号,退出其中一个账号,另外一个账号是否受影响?
- 答:不影响
 - session的销毁→登录成功,session和subject会记录登录的相关信息,属于session级别的,一个账号的退出,session销毁不影响另外一个浏览器账号的使用。
 - 权限缓存清理→
 - public void onLogout(PrincipalCollection principals) {
 - clearCache(principals);
 - }
 - PrincipalCollection属于会话级别的,两个浏览器不是同一个session,即一个账号对应两个会话,所以principals就不一样

7.1 退出

