官网，<http://shiro.apache.org/>

---文档---

·10 Minute Tutorial—10分钟教程，

·Beginner's Webapp Tutorial—初学者教程；

·Reference Manual—参考手册；| 完全文档；

版本：1.2.3---2014.9.19时；

内容：

Authentication--- 验证，Authorization---授权，

Cryptography---密码，Session Management---会话管理，

integration---集成，

参考：

1，

10分钟教会你Apache Shiro---中文翻译，

<http://www.ituring.com.cn/article/details/163>

<http://www.ibm.com/developerworks/cn/java/j-lo-shiro/>

2，

shiro使用手册；---中文版；

英文参考手册；

配合公司项目代码；

-----------------------

主内容：

Web.xml配置；

Spring配置；spring-shiro.xml；

Realm类；

~

授权；

另，

缓存，*ehcache-shiro.xml*；

原理描述：

url---过滤器---守保护的资源/不受保护的资源---验证受保护资源，用户、角色通过，转入路径；不通过，返回到特定路径(登录页面/无权限页面)，并提示；

/////////////////////////////////////////////////////////////////////

Shiro中，一般权限项目中的资源描述为许可—permission。

详细架构：



实现：

Web.xml：

//shiro过滤器；

<filter>

<filter-name>shiroSecurityFilter</filter-name>

<filter-class>

org.springframework.web.filter.DelegatingFilterProxy

</filter-class>

</filter>

<filter-mapping>

<filter-name>shiroSecurityFilter</filter-name>

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

</filter-mapping>

//通过ContextLoaderListener加载contextConfigLocation指定的Spring配置文件。

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

classpath\*:/applicationContext.xml,

classpath\*:/applicationContext-shiro.xml

</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

Spring配置：写一个专门的配置文件,spring-shiro.xml；

//添加shiroFilter定义 | shiroFilter继承自web.xml中；

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

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

<property name=*"loginUrl"* value=*"${adminPath}/login"* />

<property name=*"successUrl"* value=*"${adminPath}"* />

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

<property name=*"filters"*>

<map>

<entry key=*"authc"* value-ref=*"formAuthenticationFilter"*/>

</map>

</property>

<property name=*"filterChainDefinitions"*>

<value>

/static/\*\* = anon

${adminPath}/login = authc

${adminPath}/logout = logout

${adminPath}/\*\* = user

</value>

</property>

</bean>

代码说明：

1. shiroFilter 中 loginUrl 为登录页面地址，successUrl 为登录成功页面地址（如果首先访问受保护 URL 登录成功，则跳转到实际访问页面），unauthorizedUrl 认证未通过访问的页面（前面提到的“未经授权页面”）。
2. shiroFilter 中 filters 属性，formAuthenticationFilter 配置为基于表单认证的过滤器。此处可以添加自己的filter定义；
3. shiroFilter 中 filterChainDefinitions 属性，anon 表示匿名访问（不需要认证与授权），authc 表示需要认证，perms[SECURITY\_ACCOUNT\_VIEW] 表示用户需要提供值为“SECURITY\_ACCOUNT\_VIEW”Permission 信息。由此可见，连接地址配置为 authc 或 perms[XXX] 表示为受保护资源。
4. securityManager 中 realm 属性，配置为我们自己实现的 Realm。关于 Realm，参见前面“Shiro Realm”章节。
5. myShiroRealm 为我们自己需要实现的 Realm 类，为了减小数据库压力，添加了缓存机制。
6. shiroCacheManager 是 Shiro 对缓存框架 EhCache 的配置。

//添加securityManager—安全管理

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

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

</bean>

<bean id=*"shticShiroRealm"* class=*"com.cxdai.filter.ShiroRealm"* >

//*lifecycle—*生命周期

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

//注解支持，

<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>

<!—缓存EhCache --> | *ehcache-shiro.xml文件，*

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

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

</bean>

*ehcache-shiro.xml，*

<ehcache updateCheck=*"false"* name=*"shiroCache"*>

<defaultCache maxElementsInMemory=*"10000"* eternal=*"false"*

timeToIdleSeconds=*"120"* timeToLiveSeconds=*"120"* overflowToDisk=*"false"*

diskPersistent=*"false"* diskExpiryThreadIntervalSeconds=*"120"* />

</ehcache>

realm类，---认证、授权处理；

public class ShiroRealm extends AuthorizingRealm{

//业务逻辑类，获取用户信息方法，

@Autowired

**private** PubUsersService pubUsersService;

//获取认证信息---登录时，

protected AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) throws AuthenticationException {

UsernamePasswordToken usernamePasswordToken = (UsernamePasswordToken) token;

//通过表单获取的用户名，

String username = usernamePasswordToken.getUsername();

//验证用户信息；

//用户信息，放入SimpleAuthorizationInfo中；

if( username != null && !"".equals(username) ){

LoginAccount account = pubUsersService.get( username );

if( account != null ){

return new SimpleAuthenticationInfo(

account.getLoginName(),account.getPassword(),getName() );

}

}

return null;

//获取授权信息 ---访问页面，action时；

protected AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

String username = (String) principals.fromRealm(

getName()).iterator().next();

//将用户许可信息放入AuthorizationInfo中；

if( username != null ){

// 查询用户授权信息

Collection<String> pers= pubUsersService.queryPermissions(username);

if( pers != null && !pers.isEmpty() ){

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

for( String each:pers )

info.addStringPermissions( each );

return info;

}

}

return null;

--------------------------------------------------------

登录：

//保存用户名、密码，

Subject currentUser = SecurityUtils.getSubject(); //获得当前主体；

UsernamePasswordToken token = new UsernamePasswordToken("lonestarr", "vespa");

token.setRememberMe(true);

try {

currentUser.login(token); //认证操作；

} catch (UnknownAccountException uae) {

log.info("用户名不存在 " + token.getPrincipal());

} catch (IncorrectCredentialsException ice) {

log.info("密码错误 " + token.getPrincipal() );

} catch (LockedAccountException lae) {

log.info("帐号锁定 " + token.getPrincipal() );

}

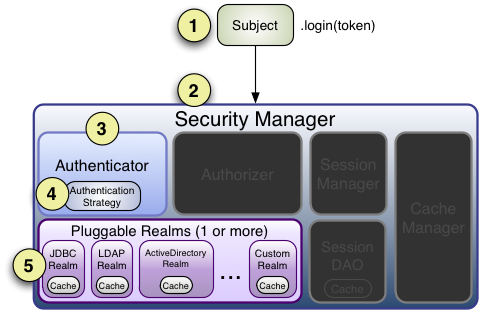
UsernamePasswordToken支持最常见的用户名/密码的认证机制。同时，由于它实现了RememberMeAuthenticationToken接口，我们可以通过令牌设置“记住我”的功能。   
但是，“已记住”和“已认证”是有区别的：   
已记住的用户仅仅是非匿名用户，你可以通过subject.getPrincipals()获取用户信息。但是它并非是完全认证通过的用户，当你访问需要认证用户的功能时，你仍然需要重新提交认证信息。   
这一区别可以参考亚马逊网站，网站会默认记住登录的用户，再次访问网站时，对于非敏感的页面功能，页面上会显示记住的用户信息，但是当你访问网站账户信息时仍然需要再次进行登录认证。 

登出：

//删除用户身份信息，无效session；

currentUser.logout();

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、Sbuject的实例通常是DelegatingSubject类（或子类）的实例对象，在认证开始时，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。   
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、判断每一个Realm是否支持提交的token，如果支持，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。

授权实现：---shiro手册6章；

1. 基于角色，

Subject currentUser = SecurityUtils.getSubject();

if (currentUser.hasRole("administrator")) {

//show the admin button

} else {

//don't show the button? Grey it out?

}

2，基于许可(即资源)，

//判断action，

Subject currentUser = SecurityUtils.getSubject();

if (currentUser.isPermitted(actionUrl.trim())) {

//show the Print button

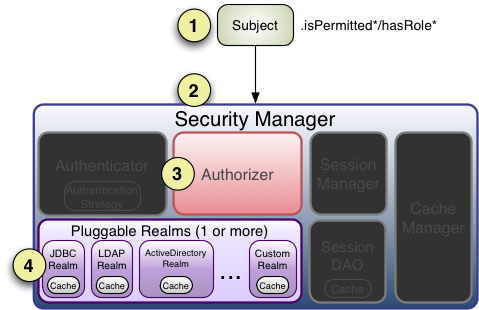
} else {

//don't show the button? Grey it out?

}

实例级访问控制，

授权实现流程，



1、在应用程序中调用授权验证方法(Subject的isPermitted\*或hasRole\*等)

2、Sbuject的实例通常是DelegatingSubject类（或子类）的实例对象，在认证开始时，会委托应用程序设置的securityManager实例调用相应的isPermitted\*或hasRole\*方法。

3、接下来SecurityManager会委托内置的Authorizer的实例（默认是ModularRealmAuthorizer 类的实例，类似认证实例，它同样支持一个或多个Realm实例认证）调用相应的授权方法。

4、每一个Realm将检查是否实现了相同的 Authorizer 接口。然后，将调用Reaml自己的相应的授权验证方法。

Realms:

Realm 认证，

调用Realm的getAuthenticationInfo(token)方法，

1、检查提交的进行认证的令牌信息   
2、根据令牌信息从数据源(通常为数据库)中获取用户信息   
3、对用户信息进行匹配验证。   
4、验证通过将返回一个封装了用户信息的AuthenticationInfo实例。   
5、验证失败则抛出AuthenticationException异常信息。

Session Management：

包括ehcached配置；

------------------------------------

<shiro:hasRole name="一级信审">