## **Shiro**

### **1.Shiro简介**

A.java安全框架(同类产品Spring security)

B.有认证,授权,加密,会话管理,缓存等功能

C.不仅可以集成web,也可以用于javaee程序中

##### **认证流程**

A. controller层获取用户账号密码生成UsernamePasswordToken

UsernamePasswordToken token = new UsernamePasswordToken(username, password);

B. controller层获取当前subject

Subject subject = SecurityUtils.getSubject();

C. controller层调用shiro框架,验证账户凭证

subject.login(token);

D.自定义Realm被调用,将AuthenticationToken强转为UsernamePasswordToken

//用户输入的凭证

UsernamePasswordToken token = tokenToUser((UsernamePasswordToken)token);

E.从数据库中查找token对应的账户信息(调用service 调用Dao)

User user = loginService.doUserLogin(token);

F.根据数据库查找情况返回异常(UnknownAccountException账号不存在,LockedAccountException账号被锁定)

G.创建验证实体(和token进行凭证比对 Authentication中的信息都由数据库查找而来)

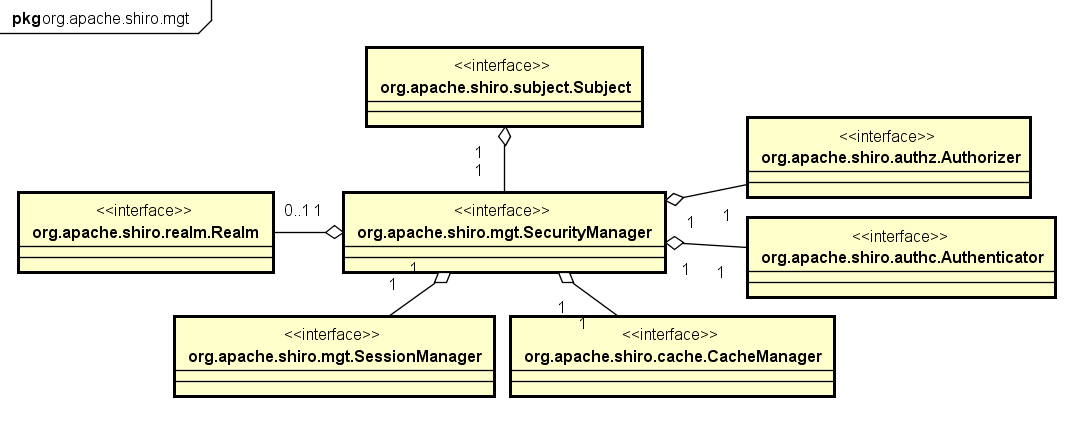
SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(principal, credentials, credentialsSalt, realmName);

H.shiro会自动进行凭证比对

### **2.shiro组件**

1.Subject:表示正在与系统交互的对象

2.securityManager:shiro核心控制器,用于协调其他组件一起完成认证授权



3.authenticator:认证管理器,用于协调一个或多个realm

4.authorizer:授权控制器

5.sessionManager:shiro独有的会话管理器,即使在web环境仍然能提供会话管理

6.cacheManager:缓存组件

7.Cryptography:加解密的命令行jar包.需要单独下载

8.credentialsMatcher:加密管理器

9.realm:认证主体

### **3.环境配置**

**3.1 POM.xml(maven依赖)**

<!-- shiro -->

<dependency>

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

<artifactId>shiro-core</artifactId>

<version>1.3.2</version>

</dependency>

<dependency>

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

<artifactId>shiro-spring</artifactId>

<version>1.2.5</version>

</dependency>

**3.2 WEB.xml(shiroFiler)**

<!-- Shiro Filter -->

<filter>

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

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

<init-param>

<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>

##### **3.3 application.xml(将shiro转化为bean)**

**3.3.1 配置ShiroFilter主过滤器.(关联WEB.xml)**

<!-- Shiro主过滤器本身功能十分强大，其强大之处就在于它支持任何基 于URL路径表达式的、自定义的过滤器的执行 Web应用中，Shiro可控制的Web请求必须经过Shiro主过滤器的拦截，Shiro对基于Spring的Web应用提供了完美的支持 -->

<!-- shiro主过滤器 -->

<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"></property>

<!-- 要求登录时的链接(登录页面地址)，非必须的属性，默认会自动寻找Web工程根目录下的"/login.jsp"页面 -->

<property name="loginUrl" value="/login.html"></property>

<!-- 登录成功后要跳转的连接(本例中此属性用不到，因为登录成功后的处理逻辑在LoginController里硬编码) -->

<!-- <property name="successUrl" value="/" ></property> -->

<!-- 用户访问未对其授权的资源时，所显示的连接 -->

<!-- <property name="unauthorizedUrl" value="/"></property> -->

<property name="filterChainDefinitions">

<value>

<!-- 对静态资源设置允许匿名访问 -->

/image/\*\* = anon

/js/\*\* = anon

/css/\*\* = anon

<!-- 可匿名访问路径，例如：短信验证码、登录连接、退出连接等 -->

/\*.html = anon

/test/\*= anon

/login/\*.do = anon

<!-- 剩余其他路径，必须认证通过才可以访问 -->

/\*\* = authc

</value>

</property>

</bean>

**3.3.2配置加密管理器(credentialsMatcher shiro内部比对时调用)**

可配对若干个,由realm调用,

也可以在realm中单独配置

<!-- 数据库保存的密码是使用MD5算法加密的，所以这里需要配置一个密码匹配对象 -->

<bean name="credentialsMatcher"

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

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

<property name="hashIterations" value="2"></property>

</bean>

**3.3.3 配置认证管理器(authenticator shiro多realm中调用)**

<bean id="authenticator"

class="org.apache.shiro.authc.pam.ModularRealmAuthenticator">

<property name="authenticationStrategy">

<bean class="org.apache.shiro.authc.pam.AtLeastOneSuccessfulStrategy"></bean>

</property>

</bean>

**3.3.4 配置缓存管理器(cacheManager)**

先不配缓存

**3.3.5 配置realm**

<!-- 使用Shiro自带的JdbcRealm类 指定密码匹配所需要用到的加密对象 指定存储用户、角色、权限许可的数据源及相关查询语句 -->

<bean id="jdbcRealm" class="com.wfly.realm.JdbcRealm">

<!-- <property name="credentialsMatcher">

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

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

<property name="hashIterations" value="2"></property>

</bean>

</property> -->

<property name="credentialsMatcher" ref="credentialsMatcher" /><!-- 使用的加密器 -->

<!-- <property name="permissionsLookupEnabled" value="true"></property> -->

</bean>

**3.3.5 配置SecurityManager**

<!-- 1. 配置 SecurityManager! -->

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

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

<property name="authenticator" ref="authenticator"></property>

<!-- 单realm配置 -->

<property name="realm" ref="jdbcRealm"></property>

<!-- 多realm配置 -->

<!-- <property name="realms">

<list>

<ref bean="jdbcRealm"/>

<ref bean="secondRealm"/>

</list>

</property> -->

<property name="rememberMeManager.cookie.maxAge" value="10"></property>

</bean>

### **4.编写自定义realm**

1.实现Realm接口或者继承其他Realm实现类(AuthorizingRealm),重写doGetAuthenticationInfo(认证)和doGetAuthorizationInfo(授权)

doGetAuthenticationInfo

2.从数据库中获取凭证相关.

3.创建SimpleAuthenticationInfo

4.将登陆信息保存到session

doGetAuthorizationInfo

5.从PrincipalCollection中获取用户登陆信息

6.查询数据库,用Set<String> roles = new HashSet<>(); 保存权限信息

7.创建SimpleAuthorizationInfo并将roles传入

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo(roles);

package com.wfly.realm;

import java.util.HashSet;

import java.util.Set;

import javax.annotation.Resource;

import org.apache.shiro.SecurityUtils;

import org.apache.shiro.authc.AuthenticationException;

import org.apache.shiro.authc.AuthenticationInfo;

import org.apache.shiro.authc.AuthenticationToken;

import org.apache.shiro.authc.SimpleAuthenticationInfo;

import org.apache.shiro.authc.UsernamePasswordToken;

import org.apache.shiro.authz.AuthorizationInfo;

import org.apache.shiro.authz.SimpleAuthorizationInfo;

import org.apache.shiro.crypto.hash.SimpleHash;

import org.apache.shiro.realm.AuthorizingRealm;

import org.apache.shiro.session.Session;

import org.apache.shiro.subject.PrincipalCollection;

import org.apache.shiro.util.ByteSource;

import com.wfly.model.SimpleUser;

import com.wfly.model.User;

import com.wfly.service.LoginService;

public class JdbcRealm extends AuthorizingRealm {

@Resource //自动注入

private LoginService loginService;

//认证

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

System.err.println("jdbcRealm:"+"AuthenticationInfo");

//用户输入的凭证

User userLogin = tokenToUser((UsernamePasswordToken)token);

//数据库中的凭证

User user = null;

try {

user = loginService.doUserLogin(userLogin);

} catch (Exception e) {

}

//若用户不存在, 则可以抛出 UnknownAccountException 异常

if(user == null){

return null;

}

//根据用户信息的情况, 决定是否需要抛出其他的 AuthenticationException 异常.

// if("monster".equals(username)){

// throw new LockedAccountException("用户被锁定");

// }

//根据用户的情况, 来构建 AuthenticationInfo 对象并返回. 通常使用的实现类为: SimpleAuthenticationInfo

//以下信息是从数据库中获取的.

//1). principal: 认证的实体信息. 可以是 username, 也可以是数据表对应的用户的实体类对象.

Object principal = user.getUsername();

//2). credentials: 密码.

Object credentials = user.getPassword(); //"fc1709d0a95a6be30bc5926fdb7f22f4";

//3). realmName: 当前 realm 对象的 name. 调用父类的 getName() 方法即可

String realmName = getName();

//4). 盐值.

ByteSource credentialsSalt = ByteSource.Util.bytes(user.getUsername());

SimpleAuthenticationInfo info = null;

//new SimpleAuthenticationInfo(principal, credentials, realmName);

info = new SimpleAuthenticationInfo(principal, credentials, credentialsSalt, realmName);

SimpleUser reUser = new SimpleUser();

reUser.setId(user.getId());

reUser.setNickname(user.getNickname());

reUser.setUsername(user.getUsername());

Session session = SecurityUtils.getSubject().getSession();

session.setAttribute("simpleUser", reUser);

// info = new SimpleAuthenticationInfo(principal, credentials, realmName);

return info;

}

public static void main(String[] args) {

String hashAlgorithmName = "MD5";

Object credentials = "bf756d99f7dd7726";

ByteSource salt = ByteSource.Util.bytes("13111919211");

// Object salt = null;

int hashIterations = 2;

System.out.println("info\_credentials : "+credentials+" info\_credentialsSalt : "+salt);

Object result = new SimpleHash(hashAlgorithmName, credentials, salt, hashIterations);

System.out.println(result);

}

//授权

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

// //1. 从 PrincipalCollection 中来获取登录用户的信息

// Object principal = principals.getPrimaryPrincipal();

//

// //2. 利用登录的用户的信息来用户当前用户的角色或权限(可能需要查询数据库)

// Set<String> roles = new HashSet<>();

// roles.add("user");

// if("admin".equals(principal)){

// roles.add("admin");

// }

//

// //3. 创建 SimpleAuthorizationInfo, 并设置其 reles 属性.

// SimpleAuthorizationInfo info = new SimpleAuthorizationInfo(roles);

//

// //4. 返回 SimpleAuthorizationInfo 对象.

// return info;

return null;

}

public static User tokenToUser(UsernamePasswordToken token) {

User user = new User();

user.setUsername(token.getUsername());

user.setPassword(String.valueOf(token.getPassword()) );

return user;

}

}