**Shiro概述**

**1.什么是Shiro？**

Apache shiro是Java安全框架,可以完成认证、授权、加密、会话管理、与web集成、缓存等功能。

官方文档：<https://shiro.apache.org/reference.html>

## 2.为什么使用shiro？

使用shiro构建系统安全框架非常简单。

Shiro保护系统安全框架需要的功能，满足安全需求的“一站式服务“。

兼容性强，shiro的设计模式使其易于与其他框架集成。

## 3.Shiro和SpringSecurity对比

1.spirngSecurity基于spring开发，若项目使用Spring为基础，配合SpringSecurity做权限更加方便，shiro需要与spring整合开发。

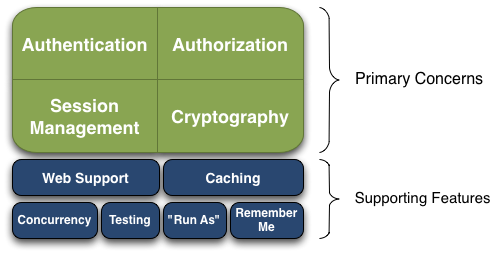
2.SpringSecurity功能比shiro更加丰富；

3.shiro的配置和使用比较简单，SpringSecurity上手辅助一些；

4.shiro依赖性低，不需要任何框架和容器，可以独立运行，SpringSecurity依赖与Spring容器

5.Shiro不仅可以使用在web中，它可以工作在任何应用环境中。在集群会话时shiro最重要的一个好处是它的会话是独立于容器的；

**3.Shiro的主要功能**

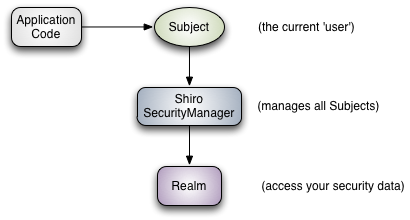


Authentication：身份认证 / 登录（确定用户身份）;  
Authorization：授权，即权限验证（对用户访问系统资源的行为做控制）;  
Session Management：会话管理，即用户登录后就是一次会话;  
Cryptography：加密;

更多信息：[https://www.w3cschool.cn/shiro/co4m1if2.html](https://www.w3cschool.cn/shiro/co4m1if2.html" \t "_blank" \o "https://www.w3cschool.cn/shiro/co4m1if2.html)

**3.Shiro是如何工作的？**

Shiro架构：（shiro外部来看）

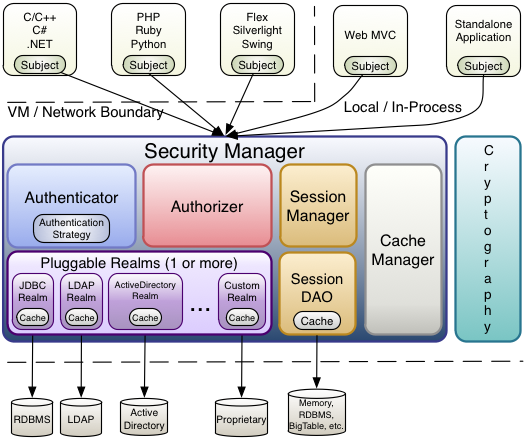


Subject：主体，代表了当前 “用户”；

SecurityManager：安全管理器，它管理着所有 Subject，是shiro的核心；

Realm：域，Shiro 从 Realm 获取安全数据（如用户、角色、权限）， Shiro 不提供维护用户 / 权限，而是通过 Realm 让开发人员自己注入；

Shiro架构：（shiro内部来看）



官方文档：

<https://shiro.apache.org/reference.html>

# Shiro的使用

## 1.登录认证

1.省份认证：一般需要提供如省份ID等一些标识信息来表明登录着的身份，如果提供email,用户名/密码来证明。

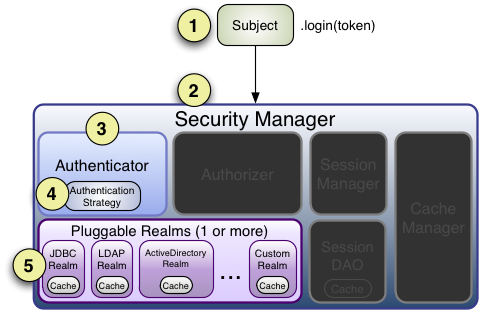
2.在shiro中，用户需要提供principals(身份)和credentials(证明)给shiro，从而实现应用能验证用户身份。

3.Principals：身份即主体的标识属性，可以是任何属性，如用户名、邮箱等，唯一即可。一个主体可以有多个Principals，但只有一个Primary Principals，一般是用户名/密码。

4.credentials：证明/凭证，即只有主体知道的安全值，如密码/数字证书等。

5.最常见的principals 和crgdentials组合就是用户名/密码

## 2.登录认证的基本流程



(1）收集用户身份/凭证，即如用户名/密码

(2）调用Subiect.login进行登录，如果失败将得到相应的AuthenticationException

异常，根据异常提示用户错误信息；否则登录成功

(3）创建自定义的Realm类，继承 arg.apache.shiro.realm.AuthenticatingRealm类，

实现doGetAuthenticatsianInfo()方法；

## 3.代码实现

//1.初始化获取securityManager

//2.获取Subject对象

//3.创建token对象，web应用用户名和密码从页面传递。

//4.完成登录

try {

    Subject currentUser = SecurityUtils.getSubject();

    UsernamePasswordToken token = new UsernamePasswordToken(username, password);

    currentUser.login(token);

} catch ( UnknownAccountException uae ) { ...

} catch ( IncorrectCredentialsException ice ) { ...

} catch ( LockedAccountException lae ) { ...

} catch ( ExcessiveAttemptsException eae ) { ...

} ... catch your own ...

} catch ( AuthenticationException ae ) {

*//unexpected error?*

}

## 4.角色、授权

1.授权：授权也叫访问控制，即在应用中控制谁访问哪些资源，在授权中需要了解几个关键对象：主体(Subject)、资源(resource)、权限（Permission）、角色(Role).

2.主体（Subject）：访问应用的用户，在shiro中使用Subject代表该用户。用户只有授权后才允许访问相应的资源。

3.资源（Resource）:在应用中用户可以访问的URL，比如页面、查看/编辑某些数据、访问某个业务方法等等都是资源。用户只有授权后才能访问。

4.权限(Permission)：安全策略中的原子授权单位，通过权限表示在应用中用户有没有操作某个资源的权力。即权限表示在应用中用户能不能访问某个资源，如CRUD（增删改查）式权限控制等。权限代表了用户有没有操作某个资源的权力，即反映在某个资源上的操作允不允许。

5.shiro支持粗粒度权限 （如用户模块的所以权限）和细粒度权限(操作某个用户的权限)

6. 角色(Role)：权限的集合，一般情况下回赋予用户角色而不是权限，即用户可以拥有一组权限，赋予权限比较方便。

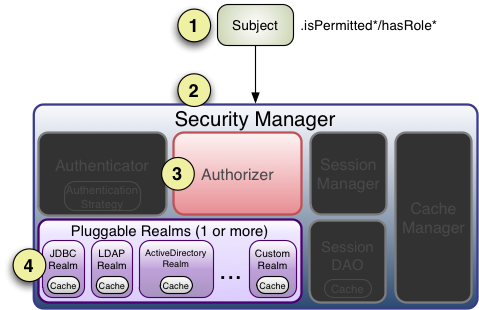
## 5.授权方式

1.编程式：通过Java代码硬编码方式完成

2.注解式：通过在执行的java方法上放置对于的注解完成，没有权限抛出对于的异常。

3.标签方式:在页面上通过相应的标签完成。

## 6.授权的基本流程



<https://shiro.apache.org/authorization.html>

(1）首先调用Subiect.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

(4） Authorizer会判断Realm的角色/权限是否和传入的匹配，如果有多个Realm，会委托

给ModularRealmAuthorizer进行循环判断，如果匹配如isPermitted\*/hasRole\*：会返回

true，否则返回false表示授杈失败。

## 7.shiro的加密

实际系统开发中，一些敏感信息需要加密处理，比如用户密码。Shiro内嵌入了许多加密算法，如果Md5加密，shiro可以简单的使用信息加密。

使用shiro进行加密

public static void main(String[] args) {

*//密码明文*

        String password = "123";

*//使用MD5加密*

        String encodedPassword = new Md5Hash(password).toString();

        System.out.println(encodedPassword);

*//带盐的MD5加密，盐式在密码明文后拼接新字符串，然后再进行加密；*

         String salt =”3F2504E0-4F89-11D3-9A0C-0305E82C3301”；

         String encodedPassword2 = new Md5Hash(password,salt).toString();

        System.out.println(encodedPassword2);

*//为了保证安全，避免破解还可以多次迭代加密，保证数据安全*

        String salt1 =”3F2504E0-4F89-11D3-9A0C-0305E82C3301”；

         String encodedPassword3 = new Md5Hash(password,salt1,1024).toString();

        System.out.println(encodedPassword3);

    }

使用父类进行加密

public static void main(String[] args) {

*//密码明文*

        String password = "123";

*//盐值*

        String salt = new SecureRandomNumberGenerator().nextBytes().toString();

*//计算次数*

        int times = 2;

        String algorithmName = "md5";

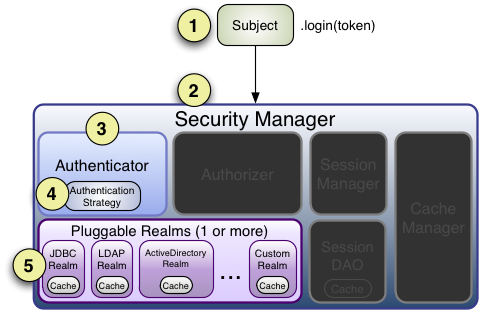
        String encodedPassword = new SimpleHash(algorithmName,password,salt,times).toString();

        System.out.printf("原始密码是 %s , 盐是： %s, 运算次数是： %d, 运算出来的密文是：%s ",password,salt,times,encodedPassword);

}

## 8.自定义登录认证

Shiro默认登录认证式不带加密的，如果想要实现加密认证需要自定义登录认证，自定义Realm。



**登录认证的基本流程**

1. 收集用户身份/凭证，即如用户名/密码
2. 调用Subject.login进行登录，如果失败将得到相应的AuthenticationException异常，根据异常提示用户错误信息；否则登录成功；
3. 创建自定义的Reaml类，继承AuthenticatingReaml类实现doGetAuthenticationInfo()方法

public class MyRealm extends AuthenticatingRealm {

*//认证方法，shiro的login方法底层会调用该类的认证方法进行认证*

*//需要配置的自定义的Realm生效，需要配置ini文件中配置或在Springboot中配置*

*//该方法只是获取进行对比的信息。认证的逻辑还是按照shiro底层认证逻辑完成。*

    @Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

//1.获取身份信息

String principal = authenticationToken.getPrincipal().toString();

//2.获取凭证信息

String passwd = new Stirng(char[])authenticationToken.getCtedentials());

//3.获取数据库中存储的用户信息

User user = useApi.selectOne(user.getUserName());

//4.创建封装校验逻辑对象，封装数据返回

String salt =ByteSource.Util.bytes(principal)

AuthenticationInfo info = new SimpleAuthenticationInfo(principal , user.getPasswd , salt,passwd );

        return info;

    }

在ini文件或spring中配置加密方式，告诉shiro如何加密

# Springboot集成Shiro

## 1.步基本骤

1）引入依赖  
2）配置Shiro

## 3.代码实现

### 1.引入依赖

<properties>

<shiro-spring-boot-web-starter.version>1.6.0</shiro-spring-boot-web-starter.version>

</properties>

<dependency>

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

<artifactId>shiro-spring-boot-web-starter</artifactId>

<version>${shiro-spring-boot-web-starter.version}</version>

</dependency>

### 2.构建Realm

public class MyRealm extends AuthorizingRealm {

private static final Logger logger = LoggerFactory.getLogger(MyRealm.class);

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

// TODO Auto-generated method stub

logger.info("doGetAuthorizationInfo---------------------------->>>");

return null;

}

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

// TODO Auto-generated method stub

logger.info("doGetAuthenticationInfo---------------------------->>>");

//1.获取用户身份信息

Stirng username= principals.getPrimaryPrincipal().toStirng();

String passwd = new Stirng(char[])authenticationToken.getCtedentials());

//2.调用业务逻辑获取用户信息(数据库)

User user =userService.seleOne(username);

//3.判断用户式否存在，存在将数据封装返回，否则抛出异常；

if(user!=null){

//3.0 principal

Object principal = principals.getPrimaryPrincipal();

//3.1Realm对象的name，调用父类的getName()方法即可

String realmName = getName();

//3.2盐

ByteSource credentialsSalt = ByteSource.Util.bytes(salt);

//3.3构建SimpleAuthenticationInfo

SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(principal, user.getPasswd, credentialsSalt, realmName);

}

return null;

}

}

### 3. 配置Shiro

@Configuration

public class ShiroConfig {

//1.Realm 代表系统资源

@Bean

public Realm myRealm() {

MyRealm myRealm = new MyRealm();

return myRealm;

}

//2.SecurityManager 流程控制

@Bean

public DefaultWebSecurityManager mySecurityManager(Realm myRealm) {

//创建SecurityManager对象

DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();

//创建相加密对象、相关属性

HashedCredentialsMatcher matcher = new HashedCredentialsMatcher();

matcher.setHashItearations(1024);//迭代加密次数

matcher.setHashAlgorithmName(“MD5”);//加密方式

//将加密对象存储到Realm中

myRealm.setCreadentialsMatcher(matcher);

//将Realm存入到SecurityManager中

securityManager.setRealm(myRealm);

return securityManager;

}

//3.ShiroFilterFactoryBean 请求过滤

//配置shiro内置过滤器拦截页面

@Bean

public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager mySecurityManager) {

ShiroFilterFactoryBean factoryBean = new ShiroFilterFactoryBean();

factoryBean.setSecurityManager(mySecurityManager);

return factoryBean;

}

# Shiro的基础认证授权

[Shiro](http://localhost:9093/itour/work/detail?id=65)

阅读  最后发布于 2022-06-20 19:30

## 1.Shiro认证思路

1.获取当前的 Subject，调用SecurityUtils.getSubject()；

2.校验用户是否已经被认证，即是否已经登录，调用Subject 的isAuthenticated()；

3.若没有认证，则根据前端传入的用户名和密码,把用户名和密码封装为UsernamePasswordToken对象；

1）前台创建1个表单；

2）把请求提交springmvc的Handler；

3）在controller中获取用户名和密码；

4.执行登录,调用Subject中的login(AuthenticationToken token)方法；

5.自定义Realm的方法，从数据库中获取对应的记录给Shiro。

1）需要继承org.apache.shiro.realm.AuthorizingRealm类；

2）使用shiro加密密码

a.配置shiro加密方式

b.将加密方式注入Realm

3）实现doGetAuthenticationInfo (AuthenticationToken token)方法；

6.由shiro完成对密码的比对；

## 2.shiroFilter工作原理：

如果请求的url在ShiroFilterFactoryBean中没有配置或者配置了是可以匿名访问的可以直接访问对应的页面，否则会被重定向到loginurl页面。

## 3.如何配置ShiroFilter

3.1配置路径过滤key：是ant路径,支持\*,\*\*,?;value配置shiro的默认过滤器, anon:匿名访问;authc:需要认证(登录)才能访问;

| **key** | **value** | **说明** |
| --- | --- | --- |
| ？ | 匹配一个字符 | /admin?匹配/admin1但是不匹配/admin ,/admin/ |
| \* | 匹配零个或多个字符串 | /admin\*匹配/admin1,/admin,但不匹配/admin/1 |
| \*\* | 匹配路径中的零个或多个路基 | /admin/\*\* 匹配/admin/a,/admin/a/b |

3.2 URL权限采取第一次匹配优先的方式，即从头开始使用第一个匹配的url模式对应的拦截器链；

Map<String,String> filterMap = new HashedMap<String, String>();

filterMap.put("/account/\*\*","anon");

filterMap.put("/account/queryAccout","authc");

filterMap.put("/\*\*","authc");

如果请求的url是“/account/queryAccout”,因为按照声明顺序进行匹配，那么将使用anon进行拦截；

## 4.代码实现

### 1.配置Shiro

@Configuration

public class ShiroConfig {

@Autowired

AccountConnector accountApi;

//1.Realm 代表系统资源

@Bean

public Realm myRealm() {

MyRealm myRealm = new MyRealm();

return myRealm;

}

//2.SecurityManager 流程控制

@Bean

public DefaultWebSecurityManager mySecurityManager(Realm myRealm) {

DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();

securityManager.setRealm(myRealm);

securityManager.setCacheManager(memoryConstrainedCacheManager());

return securityManager;

}

//3.ShiroFilterFactoryBean 请求过滤

@Bean

public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager mySecurityManager) {

ShiroFilterFactoryBean factoryBean = new ShiroFilterFactoryBean();

factoryBean.setSecurityManager(mySecurityManager);

//设置登录页面,如果不设置会默认去找项目根目录下的login.jsp

factoryBean.setLoginUrl("/login");

//配置路径过滤key：是ant路径,支持\*,\*\*,?;value配置shiro的默认过滤器, anon:匿名访问;authc:需要认证(登录)才能访问

Map<String,String> filterMap = new LinkedHashMap<String, String>();

//放行静态资源

filterMap.put("/css/\*\*","anon");

filterMap.put("/md/\*\*","anon");

filterMap.put("/js/\*\*","anon");

filterMap.put("/img/\*\*","anon");

//从数据库中获取权限

List<Map<String, String>> accountRight = accountApi.getAccountRight();

for (Map<String, String> map : accountRight) {

String url = map.get("url");

String isLogin = map.get("isLongin");

filterMap.put(url, isLogin);

}

filterMap.put("/loginSub", "anon");

//退出

filterMap.put("/shiro/logout", "logout");

filterMap.put("/\*\*", "authc");

factoryBean.setFilterChainDefinitionMap(filterMap);

return factoryBean;

}

//Shiro自带的缓存 (缓存只能用于本机，那么在集群时就无法使)

@Bean

public CacheManager memoryConstrainedCacheManager() {

MemoryConstrainedCacheManager cacheManager = new MemoryConstrainedCacheManager();

return cacheManager;

}

/\*\*

\* 开启Shiro的注解，

\* (如@RequiresRoles,@RequiresPermissions),需借助SpringAOP扫描使用Shiro注解的类,

\* 并在必要时进行安全逻辑验证 \* 配置以下两个bean(DefaultAdvisorAutoProxyCreator(可选)

\* 和AuthorizationAttributeSourceAdvisor)即可实现此功能

\* 与springboot集成时，如果不配置，且使用了相关注解会导致页面404无法访问;

\* @return

\*/

@Bean

public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {

DefaultAdvisorAutoProxyCreator proxyCreator = new DefaultAdvisorAutoProxyCreator();

proxyCreator.setProxyTargetClass(true);

return proxyCreator;

}

/\*\*

\* 开启 shiro aop注解支持.

\*

\* @param securityManager

\* @return

\*/

@Bean

public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(DefaultWebSecurityManager securityManager) {

AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdvisor();

advisor.setSecurityManager(securityManager);

return advisor;

}

}

### 2.编写认证方法

@RequestMapping("/login")

public Map login(@RequestBody JSONObject jsonObject,HttpServletRequest request) {

Map<String,Object> resultMap = new HashMap<String, Object>();

String password = jsonObject.getString("password");

String userName = jsonObject.getString("userName");

try {

//1.获取subject对象

Subject subject = SecurityUtils.getSubject();

if(!subject.isAuthenticated()) {

//2.封装请求数据到UsernamePasswordToken

UsernamePasswordToken token = new UsernamePasswordToken(userName, password);

//3.调用login方法进行登录

subject.login(token);

}

} catch (UnknownAccountException e) {//用户不存在

// TODO: handle exception

e.printStackTrace();

resultMap.put("code", Constant.FAILED\_CODE);

resultMap.put("msg", ExceptionInfo.EXCEPTION\_ACCOUNTINFO);

}catch (IncorrectCredentialsException e) {//用户存在，但密码不匹配

// TODO: handle exception

e.printStackTrace();

resultMap.put("code", Constant.FAILED\_CODE);

resultMap.put("msg", ExceptionInfo.EXCEPTION\_ACCOUNTINFO);

}catch (LockedAccountException e) {//用户被锁定

// TODO: handle exception

e.printStackTrace();

resultMap.put("code", Constant.FAILED\_CODE);

resultMap.put("msg", ExceptionInfo.EXCEPTION\_STATUS);

}catch (AuthenticationException e) {

// TODO: handle exception

e.printStackTrace();

resultMap.put("code", Constant.FAILED\_CODE);

resultMap.put("msg", Constant.FAILED\_SYSTEM\_ERROR);

}catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

resultMap.put("msg", Constant.FAILED\_SYSTEM\_ERROR);

}

return resultMap;

}

### 3.自定义Realm

认证基本思路

1.把AuthenticationToken转换为ExUsernamePasswordToken

2.从ExUsernamePasswordToken中获取Username

3.调用数据库方法从数据库中查询Username对应的记录

4.若用户不存在则可以抛出UnknownAccountException异常

5.根据用户情况,决定是否抛出其他的AuthenticationException异常

6.根据用户信息来构建AuthenticationInfo并返回，通常使用的是SimpleAuthenticationInfo，并使用盐值加密；

盐值加密

1).如何把一个字符串加密为MD5或SHA-1

/\*\*

\* MD5盐值加密

\* @param credential

\* @param salt

\* @return

\*/

public static String simpleHashMd5(String credential,String salt) {

String result = new SimpleHash("MD5", credential, ByteSource.Util.bytes(salt), 1024).toHex();

return result;

}

/\*\*

\* SHA-1加密

\* @param credential 密码

\* @param salt 盐

\* @return

\*/

public static String simpleHashSHA\_1(String credential,String salt) {

String result = new SimpleHash("SHA-1", credential, ByteSource.Util.bytes(salt), 1024).toHex();

return result;

}

授权思路

1.从PrincipalCollection获取登录用户的信息

2.利用登录的用户信息来获取当前用户的角色或权限(可能需要查询数据库)

3.创建SimpleAuthorizationInfo并设置roles属性和StringPermissions属性

具体实现

public class MyRealm extends AuthorizingRealm {

private static final Logger logger = LoggerFactory.getLogger(MyRealm.class);

@Autowired

AccountConnector accountApi;

//授权

//1.从PrincipalCollection获取登录用户的信息

//2.利用登录的用户信息来获取当前用户的角色或权限(可能需要查询数据库)

//3.创建SimpleAuthorizationInfo并设置roles属性

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

// TODO Auto-generated method stub

logger.info("doGetAuthorizationInfo---------------------------->>>");

//1.从PrincipalCollection获取登录用户的信息

Oauth primaryPrincipal =(Oauth) principals.getPrimaryPrincipal();

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

Set<String> permissions = new HashSet<String>();

JSONObject jsonObject = new JSONObject();

jsonObject.put("uid", primaryPrincipal.getuId());

List<Role> roles= accountApi.getAccountRoleName(jsonObject, null);

for (Role role : roleList) {

roles.add(role.getRoleName());

}

//2.3.获取当前用户下的权限

List<Rigth> queryAccountRight = accountConnector.getAccountRightDetial(jsonObject, null);

for (Rigth rightDetail : rightDetailList) {

permissions.add(rightDetail.getUrl());

}

//3.创建SimpleAuthorizationInfo

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

info.setRoles(roles);

info.setStringPermissions(permissions);

return info;

}

//认证

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

// TODO Auto-generated method stub

//1.把AuthenticationToken转换为UsernamePasswordToken

UsernamePasswordToken upt = (UsernamePasswordToken)token;

//2.获取前端传入的用户名

String username = upt.getUsername();

//3.从数据库获取对应记录

JSONObject jsonObject = new JSONObject();

jsonObject.put("username", username);

Account account = accountApi.selectOne(jsonObject);

//4.若用户不存在则可以抛出UnknownAccountException异常

if (account == null) {

throw new UnknownAccountException(ExceptionInfo.EXCEPTION\_USRNAME);

}

//5.根据用户情况,决定是否抛出其他的AuthenticationException异常

if ("1".equals(account.getStatus())) {

//如果用户被锁定了抛出LockedAccountException

throw new LockedAccountException(ExceptionInfo.EXCEPTION\_STATUS);

}

//6.使用盐值加密;

//6.1认证的实体信息，可以是username，也可以是数据库表对应的用户的实体对

Object principal = account;

//6.2加密后的密码（从数据库中取的密码）

Object hashedCredentials = account.getCredentials();

//6.3Realm对象的name，调用父类的getName()方法即可

String realmName = getName();

//6.4盐

String salt = username;

ByteSource credentialsSalt = ByteSource.Util.bytes(salt);

7.构建SimpleAuthenticationInfo

SimpleAuthenticationInfo simpleAuthenticationInfo = new SimpleAuthenticationInfo(principal, hashedCredentials, credentialsSalt, realmName);

return simpleAuthenticationInfo;

}

}

# 多Realm实现

1. 多个Realm实现原理

当应用程序配置多个Realm时，例如用户名密码校验，手机号验证码校验等等。

Shiro的ModularRealmAuthenticator会使用内部的AuthenticationStrategy组件判断认证是成功还是失败。

AuthenticationStrategy是一个无状态的组件，它在身份认证中被询问四次（这四次交互所需的任何必要的状态将被作为方法参数）:

1. 在所有的Realm被调用之前
2. 在调用Realm的getAuthenticatjionInfo方法之前
3. 在调用Realm的getAuthenticationInfo方法之后
4. 在所有的Realm被调用之后

认证策略的另外一项工作就是聚合索引的Realm的结果信息封装至一个AuthenticationInfo实例中，并将此信息返回，以此作为Subject的身份信息

Shiro中定义了三种认知策略实现：

|  |  |
| --- | --- |
| AuthenticationStrategy.class | 描述 |
| AtLeastOneSuccessfulStrategy | 只要有一个（或更多）的Realm验证成功，那么认证视为成功 |
| FirstSuccessfulStrategy | 第一个Realm验证成功，整体认证视为成功，且后续Realm会被忽略 |
| AllSuccessfulStrategy | 所有的Realm成功，认证才视为成功。 |

ModularRealmAuthenticator内置的认证策略默认是AtLeastOneSuccessfulStrategy方式，可以通过配置修改策略。

**多个realm的实现代码**

# Remember me

Shiro提供了记住我(remember me)的功能，比如访问一些网站时，关闭了浏览器，下次再打开时还是能记住你是谁，下次访问无需再登录即可访问。

**基本流程**

1. 首先在登录页面选中remember me 然后登录成功，如果时浏览器登录，一般会把RememberMe的Cookie写到客户端并保存下来。
2. 关闭浏览器再重新打开，会发现浏览器还是记住你的；
3. 访问一般的网页服务器端，仍然知道你是谁，且能正常访问。
4. 但是，如果我们访问电商平台时，如果要查看我们的订单或进行支付时，此时还是需要再进行身份认证，以确保当前用户还是你。

**代码实现**

**Shiro登出**

[Shiro](http://localhost:9093/itour/work/detail?id=71)

阅读  最后发布于 2022-06-02 17:14

**1.自己实现**

@RequestMapping("/shiro/logout")

public String logout() {

Subject subject = SecurityUtils.getSubject();

subject.logout();

return "/login";

}

**2.使用Shiro过滤器实现**

shiro帮我们实现了退出功能，在ShiroFilter中配置filterMap.put("/shiro/logout", "logout")可以实现退出，跳转登录页面.

@Bean

public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager mySecurityManager) {

ShiroFilterFactoryBean factoryBean = new ShiroFilterFactoryBean();

factoryBean.setSecurityManager(mySecurityManager);

//配置登录页面地址，如果没有配置会去项目根目录下找login.jsp

factoryBean.setLoginUrl("/login");

Map<String,String> filterMap = new HashMap<String, String>();

//放行静态资源

filterMap.put("/css/\*\*","anon");

filterMap.put("/md/\*\*","anon");

filterMap.put("/js/\*\*","anon");

filterMap.put("/img/\*\*","anon");

filterMap.put("/loginSub", "anon");

filterMap.put("/shiro/logout", "logout");

filterMap.put("/\*\*", "authc");

factoryBean.setFilterChainDefinitionMap(filterMap);

return factoryBean;

}