# Shiro安全框架简介

## Shiro概述

Shiro是apache旗下一个开源安全框架，它将软件系统的安全认证相关的功能抽取出来，实现用户身份认证，权限授权、加密、会话管理等功能，组成了一个通用的安全认证框架。使用shiro就可以非常快速的完成认证、授权等功能的开发，降低系统成本。

用户在进行资源访问时，要求系统要对用户进行权限控制,其具体流程如图-1所示：

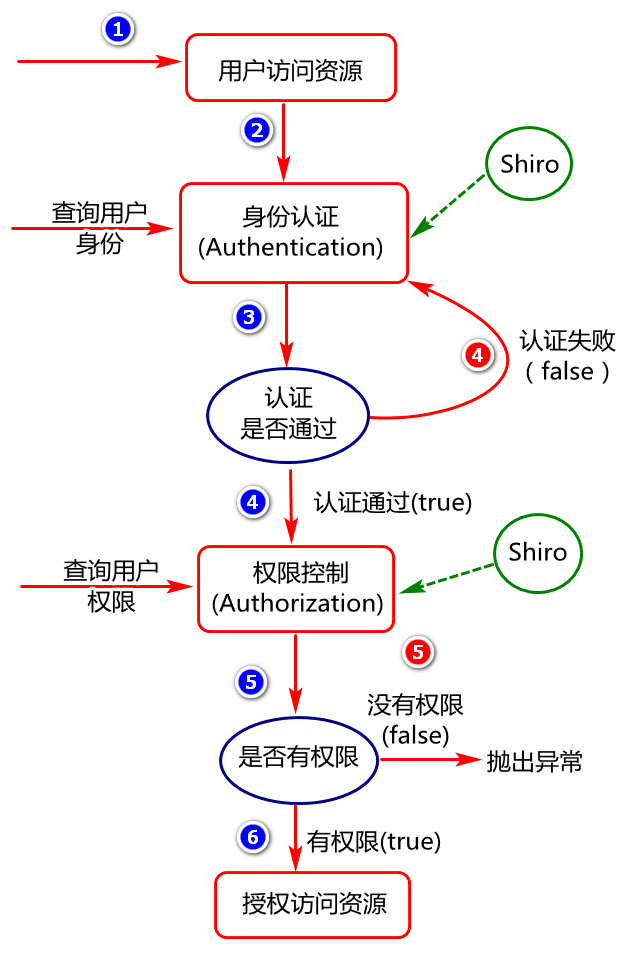


图-1

## Shiro概要架构

在概念层面，Shiro 架构包含三个主要的理念，如图-2所示：

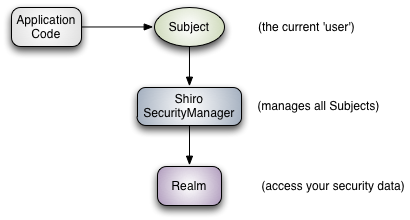


图-2

其中：

1. Subject :主体对象，负责提交用户认证和授权信息。
2. SecurityManager：安全管理器，负责认证，授权等业务实现。
3. Realm：领域对象，负责从数据层获取业务数据。

## Shiro详细架构

Shiro框架进行权限管理时,要涉及到的一些核心对象,主要包括:认证管理对象,授权管理对象,会话管理对象,缓存管理对象,加密管理对象以及Realm管理对象(领域对象:负责处理认证和授权领域的数据访问题)等，其具体架构如图-3所示：



图-3

其中：

1. Subject（主体）:与软件交互的一个特定的实体（用户、第三方服务等）。
2. SecurityManager(安全管理器) :Shiro 的核心，用来协调管理组件工作。
3. Authenticator(认证管理器):负责执行认证操作。
4. Authorizer(授权管理器):负责授权检测。
5. SessionManager(会话管理):负责创建并管理用户 Session 生命周期，提供一个强有力的 Session 体验。
6. SessionDAO:代表 SessionManager 执行 Session 持久（CRUD）动作，它允许任何存储的数据挂接到 session 管理基础上。
7. CacheManager（缓存管理器）:提供创建缓存实例和管理缓存生命周期的功能。
8. Cryptography(加密管理器):提供了加密方式的设计及管理。
9. Realms(领域对象):是shiro和你的应用程序安全数据之间的桥梁。

# Shiro框架认证拦截实现（filter）

## Shiro基本环境配置

### 添加shiro依赖

实用spring整合shiro时，需要在pom.xml中添加如下依赖：

<dependency>

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

<artifactId>shiro-spring</artifactId>

<version>1.4.1</version>

</dependency>

### Shiro核心对象配置

第一步:创建SpringShiroConfig类。关键代码如下：

**package** com.cy.pj.common.config;

/\*\*

\* **@Configuration** 注解描述的类为一个配置对象,

\* 此对象也会交给spring管理

\*/

@Configuration //bean

**public** **class** SpringShiroConfig {

}

第二步：在Shiro配置类中添加SecurityManager配置，关键代码如下：

@Bean

**public** SecurityManager securityManager() {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

**return** sManager;

}

第三步: 在Shiro配置类中添加ShiroFilterFactoryBean对象的配置。通过此对象设置资源匿名访问、认证访问。关键代码如下：

@Bean

**public** ShiroFilterFactoryBean shiroFilterFactory (

SecurityManager securityManager) {

ShiroFilterFactoryBean sfBean=

**new** ShiroFilterFactoryBean();

sfBean.setSecurityManager(securityManager);

//定义map指定请求过滤规则(哪些资源允许匿名访问,哪些必须认证访问)

LinkedHashMap<String,String> map=

**new** LinkedHashMap<>();

//静态资源允许匿名访问:"anon"

map.put("/bower\_components/\*\*","anon");

map.put("/build/\*\*","anon");

map.put("/dist/\*\*","anon");

map.put("/plugins/\*\*","anon");

//除了匿名访问的资源,其它都要认证("authc")后访问

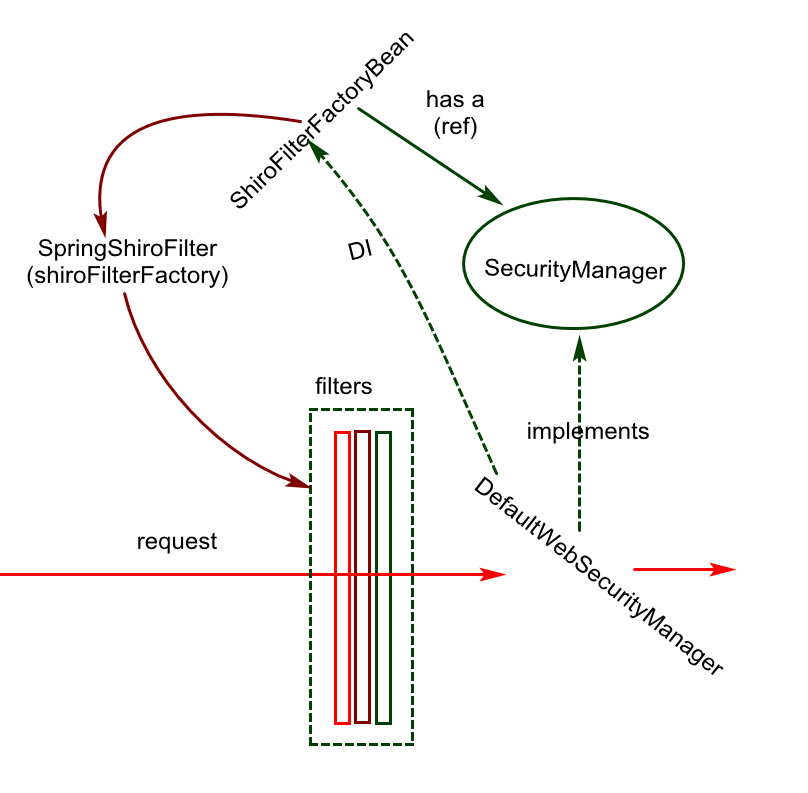
map.put("/\*\*","authc");

sfBean.setFilterChainDefinitionMap(map);

**return** sfBean;

}

其配置过程中,对象关系如下图所示:



## Shiro登陆页面呈现

### 服务端Controller实现

* 业务描述及设计实现

当服务端拦截到用户请求以后,判定此请求是否已经被认证,假如没有认证应该先跳转到登录页面。

* 关键代码分析及实现.

第一步：在PageController中添加一个呈现登录页面的方法,关键代码如下：

@RequestMapping("doLoginUI")

**public** String doLoginUI(){

**return** "login";

}

第二步：修改SpringShiroConfig类中shiroFilterFactorybean的配置过，添加登陆url的设置。关键代码见如下代码中的黄色部分。

@Bean

**public** ShiroFilterFactoryBean shiroFilterFactory (

@Autowired SecurityManager securityManager) {

ShiroFilterFactoryBean sfBean=

**new** ShiroFilterFactoryBean();

sfBean.setSecurityManager(securityManager);

sfBean.setLoginUrl("/doLoginUI");

//定义map指定请求过滤规则(哪些资源允许匿名访问,哪些必须认证访问)

LinkedHashMap<String,String> map=

**new** LinkedHashMap<>();

//静态资源允许匿名访问:"anon"

map.put("/bower\_components/\*\*","anon");

map.put("/build/\*\*","anon");

map.put("/dist/\*\*","anon");

map.put("/plugins/\*\*","anon");

//除了匿名访问的资源,其它都要认证("authc")后访问

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

sfBean.setFilterChainDefinitionMap(map);

**return** sfBean;

}

### 客户端页面实现

* 业务描述及设计实现。

在/templates/pages/添加一个login.html页面,然后将项目部署到web服务器,并启动测试运行。

* 关键代码分析及实现。

具体代码见项目中login.html。

# Shiro框架认证业务实现

## 认证流程分析

身份认证即判定用户是否是系统的合法用户，用户访问系统资源时的认证（对用户身份信息的认证）流程图-5所示:

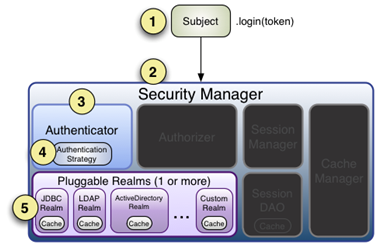


图-5

其中认证流程分析如下:

1. 系统调用subject的login方法将用户信息提交给SecurityManager
2. SecurityManager将认证操作委托给认证器对象Authenticator
3. Authenticator将用户输入的身份信息传递给Realm。
4. Realm访问数据库获取用户信息然后对信息进行封装并返回。
5. Authenticator 对realm返回的信息进行身份认证。

思考：不使用shiro框架如何完成认证操作？filter，intercetor。

## 认证服务端实现

### 核心业务分析

认证业务API处理流程分析，如图-6所示：

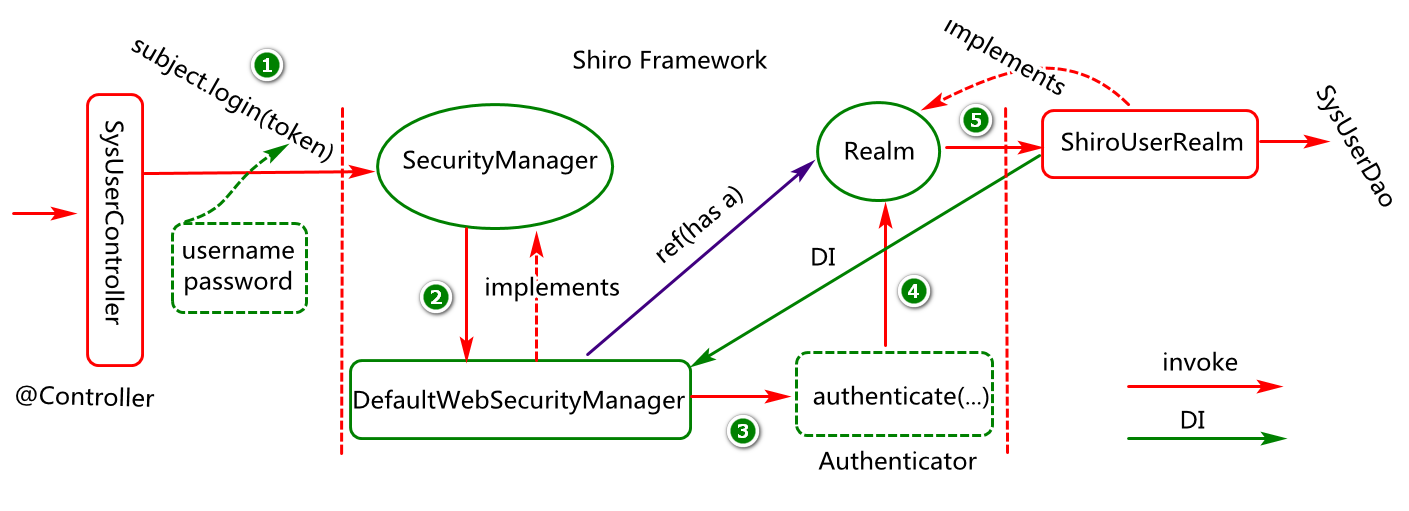


图-6

### DAO接口定义

* 业务描述及设计实现。

在用户数据层对象SysUserDao中，按特定条件查询用户信息，并对其进行封装。

* 关键代码分析及实现。

在SysUserDao接口中，添加根据用户名获取用户对象的方法，关键代码如下：

SysUser findUserByUserName(String username)。

### Mapper元素定义

* 业务描述及设计实现。

根据SysUserDao中定义的方法，在SysUserMapper文件中添加元素定义。

* 关键代码分析及实现。

基于用户名获取用户对象的方法，关键代码如下：

<select id="findUserByUserName"

resultType="com.cy.pj.sys.entity.SysUser">

select \*

from sys\_users

where username=#{username}

</select>

### Service接口及实现

* 业务描述及设计实现。

本模块的业务在Realm类型的对象中进行实现，我们编写realm时，要继承

AuthorizingRealm并重写相关方法，完成认证及授权业务数据的获取及封装。

* 关键代码分析及实现。

第一步：定义ShiroUserRealm类，关键代码如下：

**package** com.cy.pj.sys.service.realm;

@Service

**public** **class** ShiroUserRealm **extends** AuthorizingRealm {

@Autowired

**private** SysUserDao sysUserDao;

/\*\*

\* 设置凭证匹配器(与用户添加操作使用相同的加密算法)

\*/

@Override

**public** **void** setCredentialsMatcher(

CredentialsMatcher credentialsMatcher) {

//构建凭证匹配对象

HashedCredentialsMatcher cMatcher=

**new** HashedCredentialsMatcher();

//设置加密算法

cMatcher.setHashAlgorithmName("MD5");

//设置加密次数

cMatcher.setHashIterations(1);

**super**.setCredentialsMatcher(cMatcher);

}

/\*\*

\* 通过此方法完成认证数据的获取及封装,系统

\* 底层会将认证数据传递认证管理器，由认证

\* 管理器完成认证操作。

\*/

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token)

**throws** AuthenticationException {

//1.获取用户名(用户页面输入)

UsernamePasswordToken upToken=

(UsernamePasswordToken)token;

String username=upToken.getUsername();

//2.基于用户名查询用户信息

SysUser user=

sysUserDao.findUserByUserName(username);

//3.判定用户是否存在

**if**(user==**null**)

**throw** **new** UnknownAccountException();

//4.判定用户是否已被禁用。

**if**(user.getValid()==0)

**throw** **new** LockedAccountException();

//5.封装用户信息

ByteSource credentialsSalt=

ByteSource.Util.bytes(user.getSalt());

//记住：构建什么对象要看方法的返回值

SimpleAuthenticationInfo info=

**new** SimpleAuthenticationInfo(

user,//principal (身份)

user.getPassword(),//hashedCredentials

credentialsSalt, //credentialsSalt

getName());//realName

//6.返回封装结果

**return** info;//返回值会传递给认证管理器(后续

//认证管理器会通过此信息完成认证操作)

}

....

}

第二步：对此realm，需要在SpringShiroConfig类中，注入给SecurityManager对象,见黄色背景部分，例如:

@Bean

**public** SecurityManager securityManager(Realm realm) {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

sManager.setRealm(realm);

**return** sManager;

}

### Controller 类实现

* 业务描述及设计实现。

在此对象中定义相关方法，处理客户端的登陆请求，例如获取用户名，密码等然后提交该shiro框架进行认证。

* 关键代码分析及实现。

第一步：在SysUserController中添加处理登陆的方法。关键代码如下：

@RequestMapping("doLogin")

@ResponseBody

**public** JsonResult doLogin(String username,String password){

//1.获取Subject对象

Subject subject=SecurityUtils.getSubject();

//2.通过Subject提交用户信息,交给shiro框架进行认证操作

//2.1对用户进行封装

UsernamePasswordToken token=

**new** UsernamePasswordToken(

username,//身份信息

password);//凭证信息

//2.2对用户信息进行身份认证

subject.login(token);

//分析:

//1)token会传给shiro的SecurityManager

//2)SecurityManager将token传递给认证管理器

//3)认证管理器会将token传递给realm

**return** **new** JsonResult("login ok");

}

第二步：修改shiroFilterFactory的配置，对/user/doLogin.do这个路径进行匿名访问的配置，查看如下黄色标记部分的代码:

@Bean

**public** ShiroFilterFactoryBean shiroFilterFactory (

@Autowired SecurityManager securityManager) {

ShiroFilterFactoryBean sfBean=

**new** ShiroFilterFactoryBean();

sfBean.setSecurityManager(securityManager);

//假如没有认证请求先访问此认证的url

sfBean.setLoginUrl("/doLoginUI");

//定义map指定请求过滤规则(哪些资源允许匿名访问,哪些必须认证访问)

LinkedHashMap<String,String> map=

**new** LinkedHashMap<>();

//静态资源允许匿名访问:"anon"

map.put("/bower\_components/\*\*","anon");

map.put("/build/\*\*","anon");

map.put("/dist/\*\*","anon");

map.put("/plugins/\*\*","anon");

map.put("/user/doLogin","anon");

//除了匿名访问的资源,其它都要认证("authc")后访问

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

sfBean.setFilterChainDefinitionMap(map);

**return** sfBean;

}

第三步：当我们在执行登录操作时,为了提高用户体验,可对系统中的异常信息进行处理,例如,在统一异常处理类中添加如下方法:

@ExceptionHandler(ShiroException.**class**)

@ResponseBody

**public** JsonResult doHandleShiroException(

ShiroException e) {

JsonResult r=**new** JsonResult();

r.setState(0);

**if**(e **instanceof** UnknownAccountException) {

r.setMessage("账户不存在");

}**else** **if**(e **instanceof** LockedAccountException) {

r.setMessage("账户已被禁用");

}**else** **if**(e **instanceof** IncorrectCredentialsException) {

r.setMessage("密码不正确");

}**else** **if**(e **instanceof** AuthorizationException) {

r.setMessage("没有此操作权限");

}**else** {

r.setMessage("系统维护中");

}

e.printStackTrace();

**return** r;

}

## 认证客户端实现

### 编写用户登陆页面

在/templates/pages/目录下添加登陆页面(login.html)。

### 异步登陆操作实现

点击登录操作时,将输入的用户名,密码异步提交到服务端。

$(function () {

$(".login-box-body").on("click",".btn",doLogin);

});

function doLogin(){

var params={

username:$("#usernameId").val(),

password:$("#passwordId").val()

}

var url="user/doLogin.do";

$.post(url,params,function(result){

if(result.state==1){

//跳转到indexUI对应的页面

location.href="doIndexUI?t="+Math.random();

}else{

$(".login-box-msg").html(result.message);

}

});

}

## 退出操作配置实现

在SpringShiroConfig配置类中，修改过滤规则，添加黄色标记部分代码的配置,请看如下代码:

@Bean

**public** ShiroFilterFactoryBean shiroFilterFactory(

@Autowired SecurityManager securityManager) {

ShiroFilterFactoryBean sfBean=

**new** ShiroFilterFactoryBean();

sfBean.setSecurityManager(securityManager);

//假如没有认证请求先访问此认证的url

sfBean.setLoginUrl("/doLoginUI");

//定义map指定请求过滤规则(哪些资源允许匿名访问,哪些必须认证访问)

LinkedHashMap<String,String> map=**new** LinkedHashMap<>();

//静态资源允许匿名访问:"anon"

map.put("/bower\_components/\*\*","anon");

map.put("/build/\*\*","anon");

map.put("/dist/\*\*","anon");

map.put("/plugins/\*\*","anon");

map.put("/user/doLogin","anon");

map.put("/doLogout","logout");

//除了匿名访问的资源,其它都要认证("authc")后访问

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

sfBean.setFilterChainDefinitionMap(map);

**return** sfBean;

}

# Shiro框架授权过程实现

## 授权流程分析

授权即对用户资源访问的授权（是否允许用户访问此资源），用户访问系统资源时的授权流程如图-8所示:

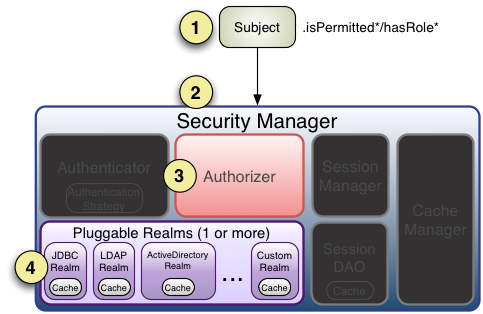


图-8

其中授权流程分析如下：

1. 系统调用subject相关方法将用户信息(例如isPermitted)递交给SecurityManager。
2. SecurityManager将权限检测操作委托给Authorizer对象。
3. Authorizer将用户信息委托给realm。
4. Realm访问数据库获取用户权限信息并封装。
5. Authorizer对用户授权信息进行判定。

思考：思考不使用shiro如何完成授权操作？intercetor，aop。

## 添加授权配置

在SpringShiroConfig配置类中，添加授权时的相关配置：

第一步:配置bean对象的生命周期管理。

@Bean

**public** LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {

**return** **new** LifecycleBeanPostProcessor();

}

第二步: 通过如下配置要为目标业务对象创建代理对象（SpringBoot中可省略）。

@DependsOn("lifecycleBeanPostProcessor")

@Bean

**public** DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {

**return** **new** DefaultAdvisorAutoProxyCreator();

}

第三步:配置advisor对象,shiro框架底层会通过此对象的matchs方法返回值决定是否创建代理对象,进行权限控制。

@Bean

**public** AuthorizationAttributeSourceAdvisor

newAuthorizationAttributeSourceAdvisor(

@Autowired SecurityManager securityManager) {

AuthorizationAttributeSourceAdvisor advisor=

**new** AuthorizationAttributeSourceAdvisor();

advisor.setSecurityManager(securityManager);

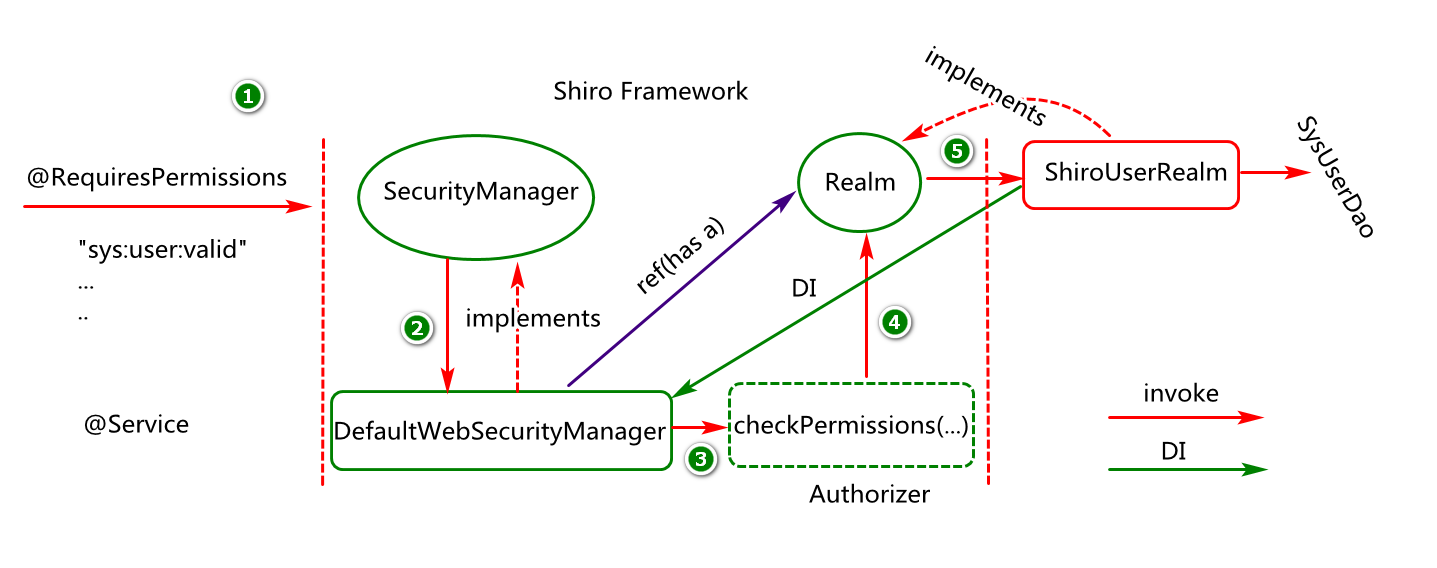
**return** advisor;

}

说明:使用框架最重要的尊重规则,框架规则指定了什么方式就使用什么方式。

## 授权服务端实现

### 核心业务分析



### Dao实现

* 业务描述及设计实现。

基于登陆用户ID，认证信息获取登陆用户的用权限信息，并进行封装。

* 关键代码分析及实现。

第一步：在SysUserRoleDao中定义基于用户id查找角色id的方法，关键代码如下：

List<Integer> findRoleIdsByUserId(Integer id);

第二步：在SysRoleMenuDao中定义基于角色id查找菜单id的方法，关键代码如下：

List<Integer> findMenuIdsByRoleIds(

@Param("roleIds")Integer[] roleIds);

第三步：在SysMenuDao中基于菜单id查找权限标识的方法，关键代码如下：

List<String> findPermissions(

@Param("menuIds")

Integer[] menuIds);

### Mapper实现

* 业务描述及设计实现。

基于Dao中方法，定义映射元素。

* 关键代码分析及实现。

第一步：在SysUserRoleMapper中定义findRoleIdsByUserId元素。关键代码如下：

<select id="findRoleIdsByUserId"

resultType="int">

select role\_id

from sys\_user\_roles

where user\_id=#{userId}

</select>

第二步:在SysRoleMenuMapper中定义findMenuIdsByRoleIds元素。关键代码如下：

<select id="findMenuIdsByRoleIds"

resultType="int">

select menu\_id

from sys\_role\_menus

where role\_id in

<foreach collection="roleIds"

open="("

close=")"

separator=","

item="item">

#{item}

</foreach>

</select>

第三步:在SysMenuMapper中定义findPermissions元素，关键代码如下：

<select id="findPermissions"

resultType="string">

select permission <!-- sys:user:update -->

from sys\_menus

where id in

<foreach collection="menuIds"

open="("

close=")"

separator=","

item="item">

#{item}

</foreach>

</select>

### Service实现

* 业务描述及设计实现。

在ShiroUserReam类中，重写对象realm的doGetAuthorizationInfo方法，并完成用户权限信息的获取以及封装，最后将信息传递给授权管理器完成授权操作。

* 关键代码分析及实现。

修改ShiroUserRealm类中的doGetAuthorizationInfo方法，关键代码如下：

@Service

**public** **class** ShiroUserRealm **extends** AuthorizingRealm {

@Autowired

**private** SysUserDao sysUserDao;

@Autowired

**private** SysUserRoleDao sysUserRoleDao;

@Autowired

**private** SysRoleMenuDao sysRoleMenuDao;

@Autowired

**private** SysMenuDao sysMenuDao;

/\*\*通过此方法完成授权信息的获取及封装\*/

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

//1.获取登录用户信息，例如用户id

SysUser user=(SysUser)principals.getPrimaryPrincipal();

Integer userId=user.getId();

//2.基于用户id获取用户拥有的角色(sys\_user\_roles)

List<Integer> roleIds=

sysUserRoleDao.findRoleIdsByUserId(userId);

**if**(roleIds==**null**||roleIds.size()==0)

**throw** **new** AuthorizationException();

//3.基于角色id获取菜单id(sys\_role\_menus)

Integer[] array={};

List<Integer> menuIds=

sysRoleMenuDao.findMenuIdsByRoleIds(

roleIds.toArray(array));

**if**(menuIds==**null**||menuIds.size()==0)

**throw** **new** AuthorizationException();

//4.基于菜单id获取权限标识(sys\_menus)

List<String> permissions=

sysMenuDao.findPermissions(

menuIds.toArray(array));

//5.对权限标识信息进行封装并返回

Set<String> set=**new** HashSet<>();

**for**(String per:permissions){

**if**(!StringUtils.isEmpty(per)){

set.add(per);

}

}

SimpleAuthorizationInfo info=

**new** SimpleAuthorizationInfo();

info.setStringPermissions(set);

**return** info;//返回给授权管理器

}

}

## 授权访问实描述现

在需要进行授权访问的业务层方法上添加执行此方法需要的权限标识，例如

@RequiresPermissions(“sys:user:update”)

说明：此要注解一定要添加到业务层方法上。

# Shiro扩展功能应用

## Shiro缓存配置

当我们进行授权操作时,每次都会从数据库查询用户权限信息,为了提高授权性能,可以将用户权限信息查询出来以后进行缓存,下次授权时从缓存取数据即可。

Shiro中内置缓存应用实现,其步骤如下:

第一步:在SpringShiroConfig中配置缓存Bean对象(Shiro框架提供)。

@Bean

**public** CacheManager shiroCacheManager(){

**return** **new** MemoryConstrainedCacheManager();

}

第二步:修改securityManager的配置，将缓存对象注入给SecurityManager对象。

@Bean

**public** SecurityManager securityManager(

Realm realm,

CacheManager cacheManager) {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

sManager.setRealm(realm);

sManager.setCacheManager(cacheManager);

**return** sManager;

}

## Shiro记住我

记住我功能是要在用户登录成功以后,假如关闭浏览器,下次再访问系统资源(例如首页doIndexUI)时,无需再执行登录操作。

### 客户端业务实现

在页面上选中记住我,然后执行提交操作,将用户名,密码,记住我对应的值提交到控制层，如图-8所示：

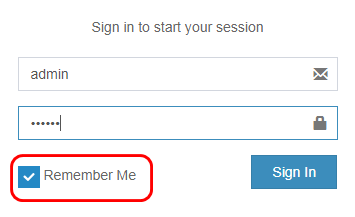


图-8

其客户端login.html中关键JS实现:

**function** doLogin(){

**var** params={

username:$("#usernameId").val(),

password:$("#passwordId").val(),

isRememberMe:$("#rememberId").prop("checked"),

}

**var** url="user/doLogin";

console.log("params",params);

$.post(url,params,**function**(result){

**if**(result.state==1){

//跳转到indexUI对应的页面

location.href="doIndexUI?t="+Math.random();

}**else**{

$(".login-box-msg").html(result.message);

}

**return** **false**;//防止刷新时重复提交

});

}

### 服务端业务实现

服务端业务实现的具体步骤如下:

第一步:在SysUserController中的doLogin方法中基于是否选中记住我，设置token的setRememberMe方法。

@RequestMapping("doLogin")

@ResponseBody

**public** JsonResult doLogin(

**boolean** isRememberMe,

String username,

String password) {

//1.封装用户信息

UsernamePasswordToken token=

**new** UsernamePasswordToken(username, password);

**if**(isRememberMe) {

token.setRememberMe(**true**);

}

//2.提交用户信息

Subject subject=SecurityUtils.*getSubject*();

subject.login(token);//token会提交给securityManager

**return** **new** JsonResult("login ok");

}

第二步:在SpringShiroConfig配置类中添加记住我配置，关键代码如下：

@Bean

**public** CookieRememberMeManager rememberMeManager() {

CookieRememberMeManager cManager=

**new** CookieRememberMeManager();

SimpleCookie cookie=**new** SimpleCookie("rememberMe");

cookie.setMaxAge(10\*60);

cManager.setCookie(cookie);

**return** cManager;

}

第三步:在SpringShiroConfig中修改securityManager的配置，为

securityManager注入rememberManager对象。参考黄色部分代码。

@Bean

**public** SecurityManager securityManager(

Realm realm,CacheManager cacheManager

CookieRememberMeManager rememberManager) {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

sManager.setRealm(realm);

sManager.setCacheManager(cacheManager);

sManager.setRememberMeManager(rememberManager);

**return** sManager;

}

第四步:修改shiro的过滤认证级别，将/\*\*=author修改为/\*\*=user,查看黄色背景部分。

@Bean

**public** ShiroFilterFactoryBean shiroFilterFactory(

SecurityManager securityManager) {

ShiroFilterFactoryBean sfBean=

**new** ShiroFilterFactoryBean();

sfBean.setSecurityManager(securityManager);

//假如没有认证请求先访问此认证的url

sfBean.setLoginUrl("/doLoginUI");

//定义map指定请求过滤规则(哪些资源允许匿名访问,哪些必须认证访问)

LinkedHashMap<String,String> map=

**new** LinkedHashMap<>();

//静态资源允许匿名访问:"anon"

map.put("/bower\_components/\*\*","anon");

map.put("/build/\*\*","anon");

map.put("/dist/\*\*","anon");

map.put("/plugins/\*\*","anon");

map.put("/user/doLogin","anon");

map.put("/doLogout", "logout");//自动查LoginUrl

//除了匿名访问的资源,其它都要认证("authc")后访问

map.put("/\*\*","user");//authc

sfBean.setFilterChainDefinitionMap(map);

**return** sfBean;

}

说明:查看浏览器cookie设置,可在浏览器中输入如下语句。

<chrome://settings/content/cookies>

## Shiro会话时长配置

使用shiro框架实现认证操作,用户登录成功会将用户信息写入到会话对象中,其默认时长为30分钟,假如需要对此进行配置,可参考如下配置:

第一步：在SpringShiroConfig类中，添加会话管理器配置。关键代码如下：

@Bean

**public** DefaultWebSessionManager sessionManager() {

DefaultWebSessionManager sManager=

**new** DefaultWebSessionManager();

sManager.setGlobalSessionTimeout(60\*60\*1000);

**return** sManager;

}

第二步：在SpringShiroConfig配置泪中，对安全管理器  securityManager

增加 sessionManager值的注入，关键代码如下：

@Bean

**public** SecurityManager securityManager(

Realm realm,CacheManager cacheManager,

CookieRememberMeManager rememberManager,

DefaultWebSessionManager sessionManager) {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

sManager.setRealm(realm);

sManager.setCacheManager(cacheManager);

sManager.setRememberMeManager(newRememberMeManager());

sManager.setSessionManager(sessionManager);

**return** sManager;

}

# Shiro总结

## 重点和难点分析

1. shiro 认证过程分析及实现(判定用户身份的合法性)。
2. Shiro 授权过程分析及实现(对资源访问进行权限检测和授权)。
3. Shiro 缓存，会话时长，记住我等功能实现。

## 常见FAQ

1. 说说shiro的核心组件?
2. 说说shiro的认证流程,你如何知道的,为什么要认证?
3. 说说shiro的授权流程,你如何知道流程是这样的,为什么要进行授权?
4. Shiro中内置缓存应用实现?为什么使用此缓存?是否可以使用第三方缓存?
5. Shiro中的记住我功能如何实现?为什么要使用这个功能?
6. Shiro中会话session的默认时长是多少,你怎么知道的?

## Bug分析

Bug-01:如图-9所示。

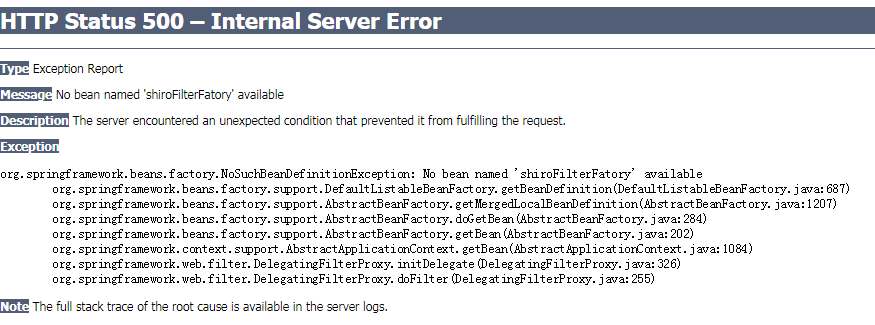


图-9

问题分析:

1. 检查web.xml中DelegatingFilterProxy 加载的参数targetBeanName对应的值是否在spring-shiro.xml中有配置。
2. 检查spring-shiro.xml文件是否在spring-service.xml中被引入。