动吧权限管理子系统

Shiro安全框架

[1. Shiro安全框架简介](#_Toc28318)

[1.1. Shiro 概述](#_Toc28599)

[1.2. Shiro 概要架构](#_Toc4410)

[1.3. Shiro 详细架构](#_Toc6201)

[2. Shiro 框架认证拦截实现](#_Toc29400)

[2.1. 添加shiro依赖](#_Toc15766)

[2.2. Shiro核心对象配置](#_Toc4405)

[2.3. 服务端登录页面呈现编写及配置](#_Toc28606)

[2.4. 客户端登录页面编写及测试](#_Toc10528)

[3. Shiro 框架认证过程实现](#_Toc6775)

[3.1. 认证流程分析](#_Toc22706)

[3.2. 认证服务端实现](#_Toc26418)

[3.2.1. Dao接口实现](#_Toc8265)

[3.2.2. Mapper元素定义](#_Toc18089)

[3.2.3. Service 接口实现](#_Toc8267)

[3.2.4. Controller类实现](#_Toc10255)

[3.3. 认证客户端实现](#_Toc8694)

[3.3.1. 编写用户登陆页面](#_Toc21275)

[3.3.2. 异步登陆操作实现](#_Toc11214)

[3.4. 退出操作配置实现](#_Toc17220)

[4. Shiro 框架授权过程实现](#_Toc9832)

[4.1. 授权流程分析](#_Toc5591)

[4.2. 添加授权配置](#_Toc31909)

[4.3. 授权服务端实现](#_Toc27097)

[4.3.1. Dao 实现](#_Toc14996)

[4.3.2. Mapper实现](#_Toc18381)

[4.3.3. Service实现](#_Toc2197)

[4.4. 授权拦截实现实现](#_Toc6357)

[5. Shiro 功能应用增强](#_Toc3193)

[5.1. Shiro缓存配置(扩展实现)](#_Toc12242)

[5.2. Shiro 记住我（扩展）](#_Toc18604)

[5.2.1. 客户端业务实现](#_Toc15456)

[5.2.2. 服务端业务实现](#_Toc4528)

[5.3. Shiro 会话时长配置（扩展）](#_Toc32488)

[6. 总结](#_Toc13428)

[6.1. 重点和难点分析](#_Toc3621)

[6.2. 常见FAQ](#_Toc31813)

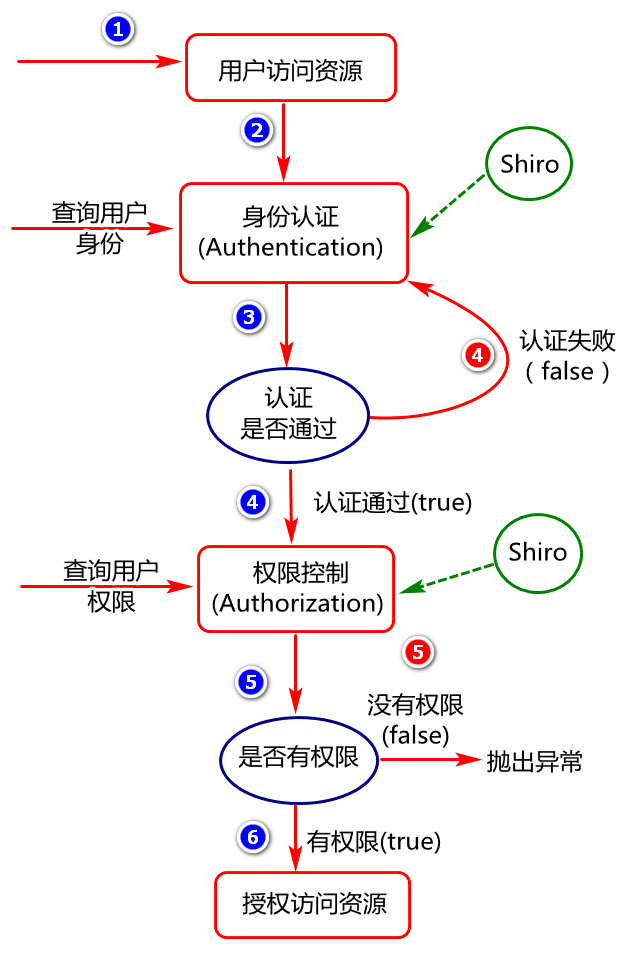
[6.3. Bug分析](#_Toc16832)

# Shiro安全框架简介

## Shiro 概述

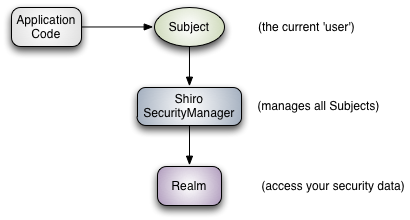
Shiro是apache旗下一个开源安全框架，它将软件系统的安全认证相关的功能抽取出来，实现用户身份认证，权限授权、加密、会话管理等功能，组成了一个通用的安全认证框架，使用shiro就可以非常快速的完成认证、授权等功能的开发，降低系统成本。

用户资源访问控制,流程分析:



## Shiro 概要架构

在概念层，Shiro 架构包含三个主要的理念：Subject,SecurityManager和 Realm。



## Shiro 详细架构

Shiro 的核心架构思想如下图所示：



通过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:

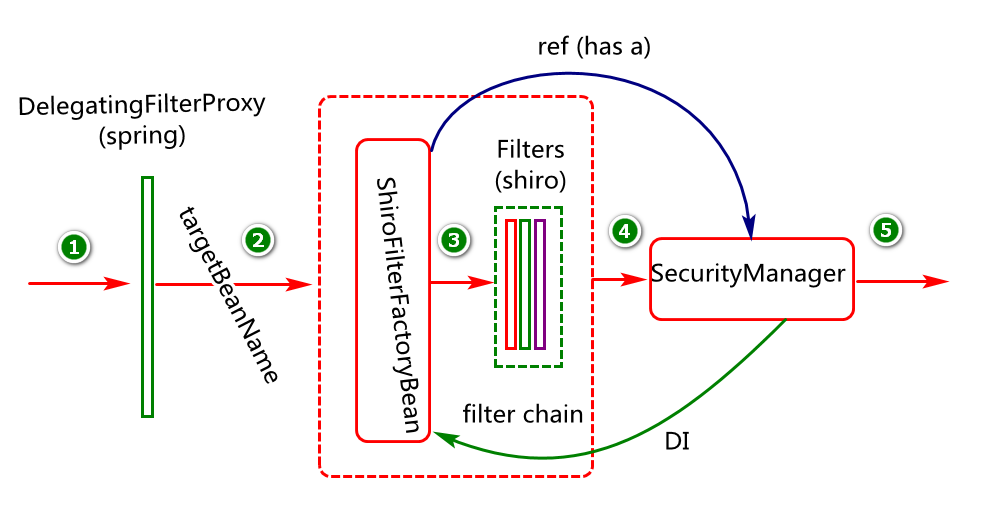
认证管理对象,授权管理对象,会话管理对象,缓存管理对象,加密管理对象

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)

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

## API 架构分析



## 添加shiro依赖

添加shiro框架依赖（spring整合shiro）

<dependency>

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

<artifactId>shiro-spring</artifactId>

<version>1.3.2</version>

</dependency>

## Shiro核心对象配置

创建SpringShiroConfig配置类,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 {

}

在SpringShiroConfig中添加如下配置：

配置SecurityManager对象(Shiro框架的核心管理器对象)

/\*\***@Bean** 描述的方法,其返回值会交给spring管理

\* **@Bean** 一般应用在整合第三bean资源时\*/

@Bean

**public** SecurityManager newSecurityManager() {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

**return** sManager;

}

配置ShiroFilterFactoryBean对象(通过此对象创建shiro中的过滤器对象)

@Bean("shiroFilterFactory")

**public** ShiroFilterFactoryBean newShiroFilterFactoryBean(

@Autowired 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;

}

创建WebFilterConfig类,并进行spring过滤器配置,通过此过滤器完成shiro的初始化操作(假如是xml配置则会在web.xml配置).

@Configuration

**public** **class** WebFilterConfig {//取代web.xml中filter配置

//注册filter对象

@SuppressWarnings({ "rawtypes", "unchecked" })

@Bean

**public** FilterRegistrationBean

newFilterRegistrationBean() {

//1.构建过滤器的注册器对象

FilterRegistrationBean fBean=

**new** FilterRegistrationBean();

//2.注册过滤器对象

DelegatingFilterProxy filter=

**new** DelegatingFilterProxy("shiroFilterFactory");

fBean.setFilter(filter);

//3.进行过滤器配置

//配置过滤器的生命周期管理(可选)由ServletContext对象负责

//fBean.setEnabled(true);//默认值就是true

fBean.addUrlPatterns("/\*");

//....

**return** fBean;

}

}

说明：shiroFilterFactory 名字要与ShiroFilterFactoryBean配置的id相同。

## 服务端登录页面呈现编写及配置

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.

在PageController中添加一个呈现登录页面的方法：

@RequestMapping("doLoginUI")

**public** String doLoginUI(){

**return** "login";

}

在SpringShiroConfig类中的shiroFilterFactorybean的配置中添加如下配置(设置shiroFilterFactorybean的登陆url):

sfBean.setLoginUrl("/doLoginUI");

见如下代码的见黄色区域部分:

@Bean("shiroFilterFactory")

**public** ShiroFilterFactoryBean newShiroFilterFactoryBean(

@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");

//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;

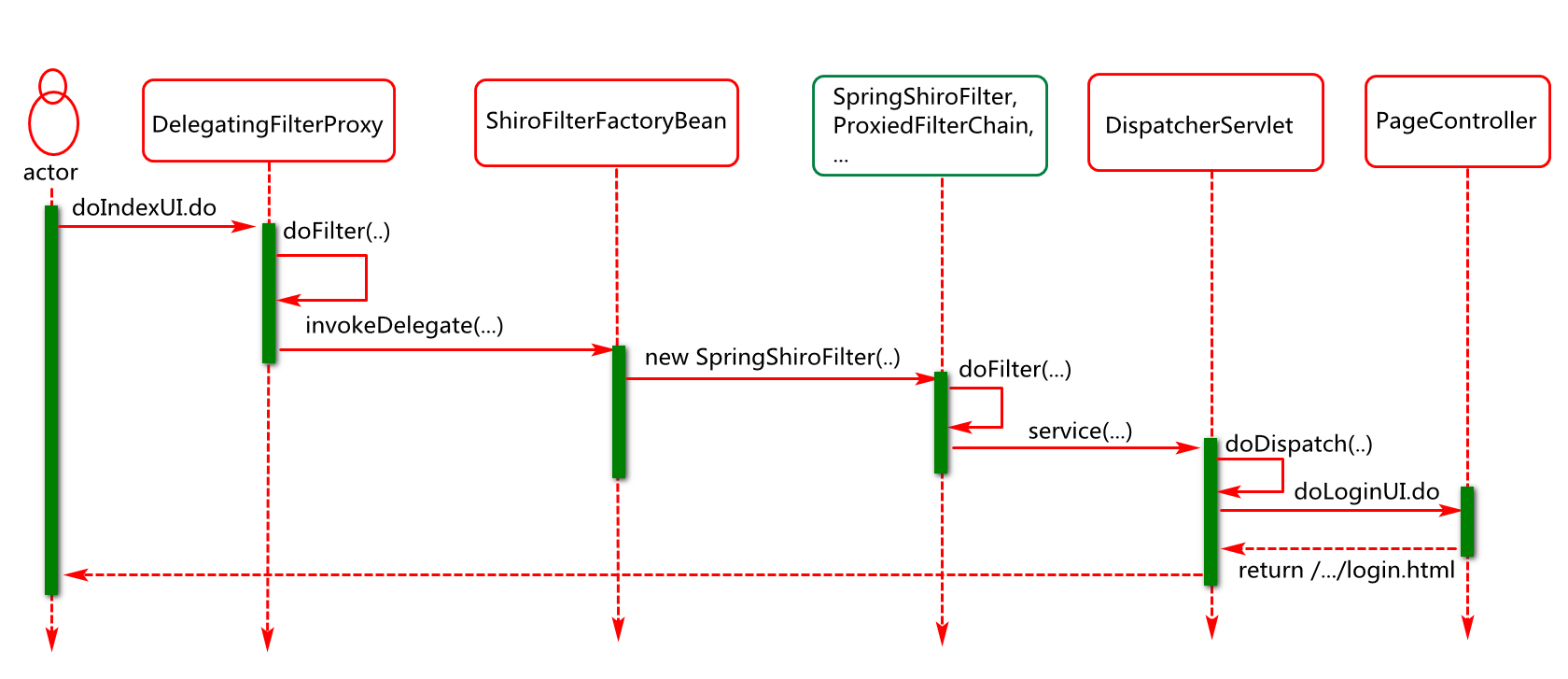
}

说明:如上配置中表示为shiroFilterFactorybean的LoginUrl属性注入值.

## 客户端登录页面编写及测试

在/WEB-INF/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.

其中:页面呈现流程分析的时序图如下:



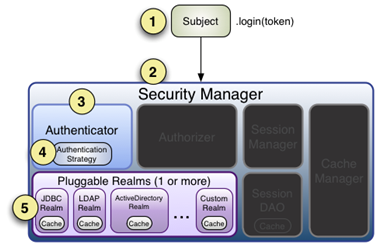
# Shiro 框架认证过程实现

本讲的shiro应用主要讲解shiro是集成到spring如何实现权限控制

## 认证流程分析

身份认证：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）流程如下:



具体流程分析如下:

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。

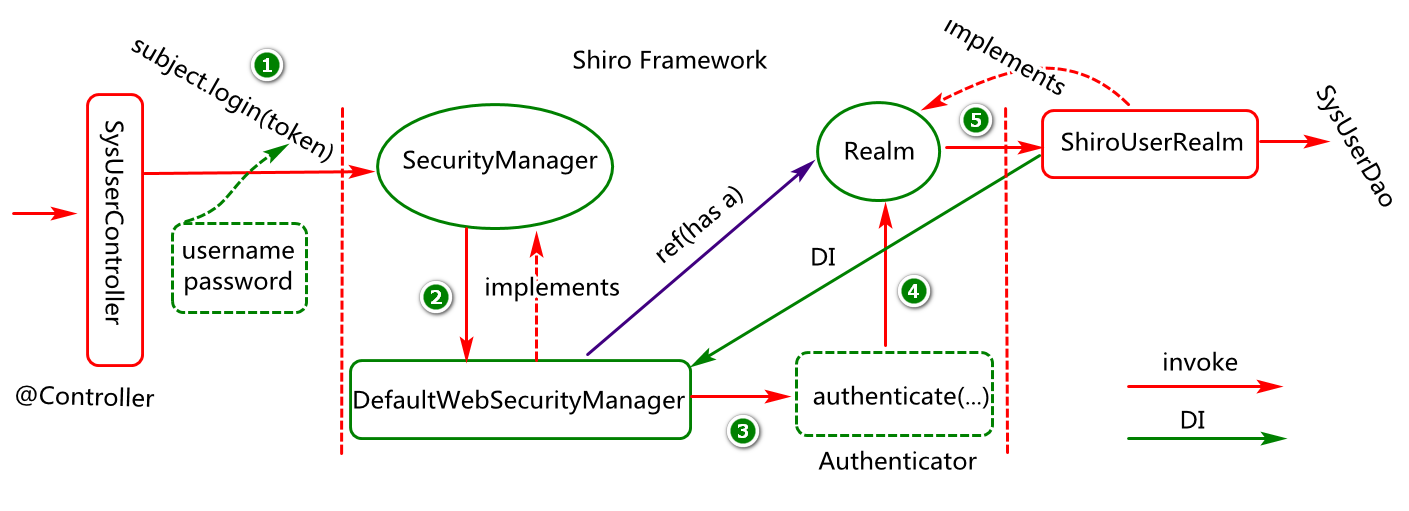
4)Realm访问数据库获取用户信息然后对信息进行封装并返回。

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

### API架构分析



### Dao接口实现

业务描述

在SysUserDao中根据用户名获取用户对象

业务实现(根据用户名查询用户对象的方法定义)

1. 返回值SysUser
2. 方法名findUserByUserName
3. 参数列表（String username）

代码实现：

SysUser findUserByUserName(String username);

### Mapper元素定义

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

<select id=*"findUserByUserName"*

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

select \*

from sys\_users

where username=#{username}

</select>

### Service 接口实现

业务描述

本模块的service可以借助realm实现，我们编写realm时可以继承AuthorizingRealm并重写相关方法完成相关业务的实现。

业务实现：(创建realm类并重写相关方法)

1. 包名:com.cy.pj.sys.service.realm
2. 类名:ShiroUserRealm
3. 方法:AuthenticationInfo （完成认证信息的获取与封装）

@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)

}

....

}

对此realm在SpringShiroConfig类中注入给SecurityManager对象,例如:

@Bean

**public** SecurityManager newSecurityManager(@Autowired Realm realm) {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

sManager.setRealm(realm);

**return** sManager;

}

### Controller类实现

业务描述

1. 在SysUserController添加登录方法doLogin
2. 接收用户名及密码参数，并对其进行有效验证
3. 执行登录认证

代码实现

@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");

}

说明：此控制层的方法映射必须允许匿名访问。需要在spring-shiro.xml配置文件中对/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("shiroFilterFactory")

**public** ShiroFilterFactoryBean newShiroFilterFactoryBean(

@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;

}

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");

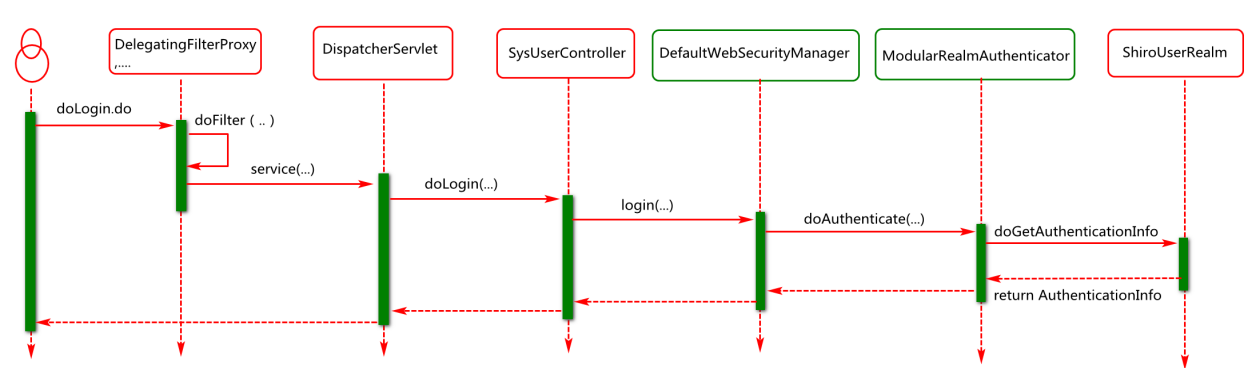
}

e.printStackTrace();

**return** r;

}

认证操作时序图分析：



## 认证客户端实现

### 编写用户登陆页面

在WEB-INF/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.do?t="+Math.random();

}**else**{

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

}

});

}

## 退出操作配置实现

在SpringShiroConfig类的过滤规则中添加黄色标记部分的配置,请看如下代码:

@Bean("shiroFilterFactory")

**public** ShiroFilterFactoryBean newShiroFilterFactoryBean(

@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;

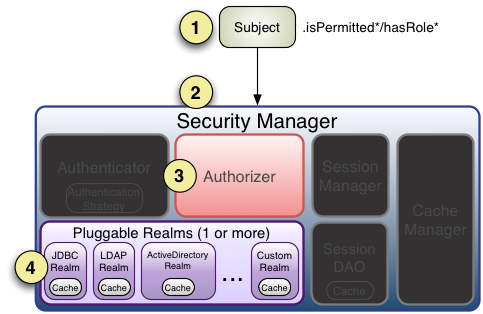
}

# Shiro 框架授权过程实现

## 授权流程分析

授权：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

Authorizer将用户信息委托给realm.

1. Realm访问数据库获取用户权限信息并封装。
2. 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中追加如下配置：

配置shiro框架中一些bean对象的生命周期管理器

@Bean("lifecycleBeanPostProcessor")

**public** LifecycleBeanPostProcessor

newLifecycleBeanPostProcessor() {

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

}

配置代理对象创建器,通过此对象为目标业务对象创建代理对象

@DependsOn("lifecycleBeanPostProcessor")

@Bean

**public** DefaultAdvisorAutoProxyCreator newDefaultAdvisorAutoProxyCreator() {

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

}

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();

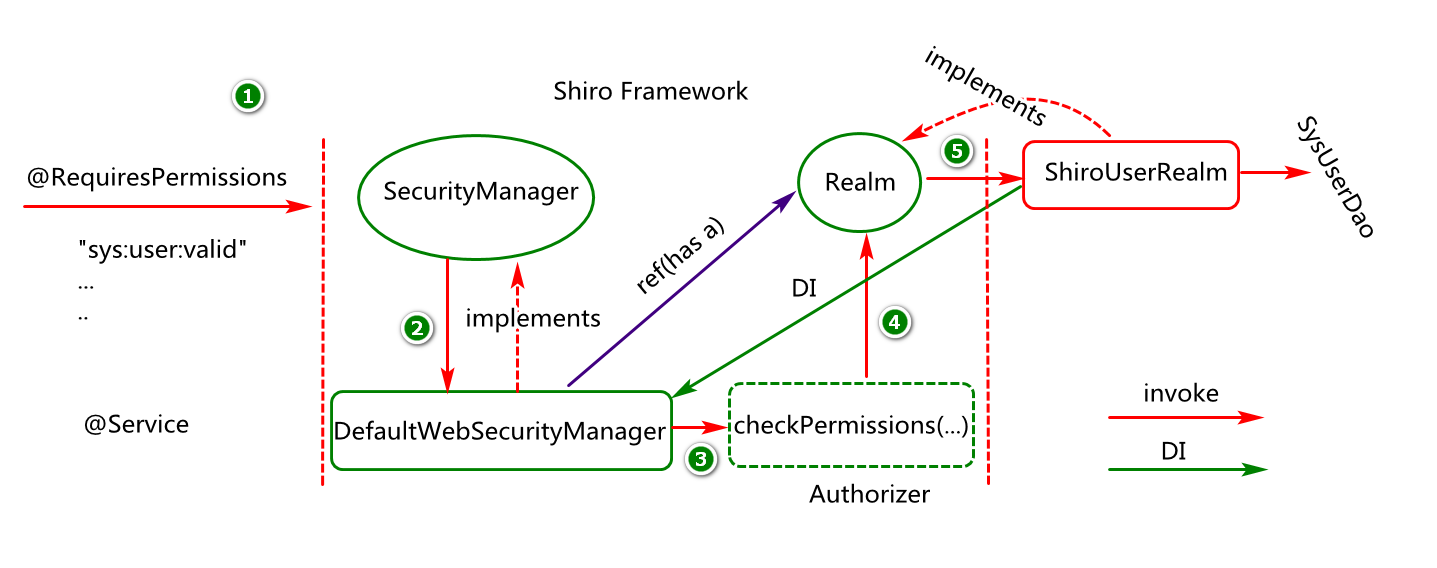
**return** advisor;

}

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

### API架构分析



### Dao 实现

业务描述：(核心业务是基于用户id获取用户对应的权限)

1. 基于用户id查找角色id信息
2. 基于角色id查找菜单id信息
3. 基于菜单id查找权限标识信息

业务实现：(在SysUserRoleDao中基于用户id查找角色id信息)

1. 返回值 List<Integer>
2. 方法名 findRoleIdsByUserId
3. 参数列表（Integer id）

业务实现：(在SysRoleMenuDao中基于用户id查找菜单id信息)

1. 返回值 List<Integer>
2. 方法名 findMenuIdsByRoleIds
3. 参数列表（Integer[] id）

业务实现：(在SysMenuDao中基于菜单id查找权限标识信息)

1. 返回值 List<String>
2. 方法名 findPermisssions
3. 参数列表（Integer[] id）

代码实现：

SysUserRoleDao 中方法定义(已有则无须定义)

List<Integer> findRoleIdsByUserId(

Integer id);

SysRoleMenuDao中方法定义

List<Integer> findMenuIdsByRoleIds(

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

SysMenuDao中方法定义

List<String> findPermissions(

@Param("menuIds")

Integer[] menuIds);

### Mapper实现

业务描述

基于Dao中方法，定义映射元素

代码实现：

SysUserRoleMapper中元素定义

<select id=*"findRoleIdsByUserId"*

resultType=*"int"*>

select role\_id

from sys\_user\_roles

where user\_id=#{userId}

</select>

SysRoleMenuMapper中元素定义

<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>

SysMenuMapper中元素定义

<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

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。

@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

}

}

## 授权拦截实现实现

在需要进行授权访问的业务方法(业务层)上添加执行此方法需要的权限标识

例如

@RequiresPermissions(“sys:user:valid”)

# Shiro 功能应用增强

## Shiro缓存配置(扩展实现)

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:

Step01:在SpringShiroConfig中配置缓存Bean对象(Shiro框架提供)

@Bean

**public** CacheManager newCacheManager(){

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

}

Step02:将缓存对象注入给SecurityManager对象,见黄色部分

@Bean

**public** SecurityManager newSecurityManager(

@Autowired Realm realm,

@Autowired CacheManager cacheManager) {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

sManager.setRealm(realm);

sManager.setCacheManager(cacheManager);

**return** sManager;

}

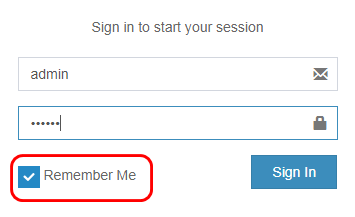
说明:shiro框架还可以集成第三方缓存,例如ehcache,Redis等

## Shiro 记住我（扩展）

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.do)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.



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.do";

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.do?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:

Step01:控制层方法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");

}

Step03:SpringShiroConfig中添加cookie置

**public** SimpleCookie newCookie() {

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

c.setMaxAge(10\*60);

**return** c;

}

//@Bean

**public** CookieRememberMeManager newRememberMeManager() {

CookieRememberMeManager cManager=

**new** CookieRememberMeManager();

cManager.setCookie(newCookie());

**return** cManager;

}

Step03:SpringShiroConfig中为securityManager的rememberManager属性注入值,见黄色标记部分.

@Bean

**public** SecurityManager newSecurityManager(

@Autowired Realm realm,

@Autowired CacheManager cacheManager) {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

sManager.setRealm(realm);

sManager.setCacheManager(cacheManager);

sManager.setRememberMeManager(newRememberMeManager());

**return** sManager;

}

Step04:修改shiro的过滤认证级别：

将/\*\*=author修改为/\*\*=users,查看黄色背景部分

@Bean("shiroFilterFactory")

**public** ShiroFilterFactoryBean newShiroFilterFactoryBean(

@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");//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（扩展）

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:

Step01：在SpringShiroConfig添加会话管理器配置

**public** DefaultWebSessionManager newSessionManager() {

DefaultWebSessionManager sManager=

**new** DefaultWebSessionManager();

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

**return** sManager;

}

Step02：对安全管理器  securityManager  增加  sessionManager值的注入,见红色标记部分.

@Bean

**public** SecurityManager newSecurityManager(

@Autowired Realm realm,

@Autowired CacheManager cacheManager) {

DefaultWebSecurityManager sManager=

**new** DefaultWebSecurityManager();

sManager.setRealm(realm);

sManager.setCacheManager(cacheManager);

sManager.setRememberMeManager(newRememberMeManager());

sManager.setSessionManager(newSessionManager());

**return** sManager;

}

扩展:实现用户密码修改.

# 总结

## 重点和难点分析

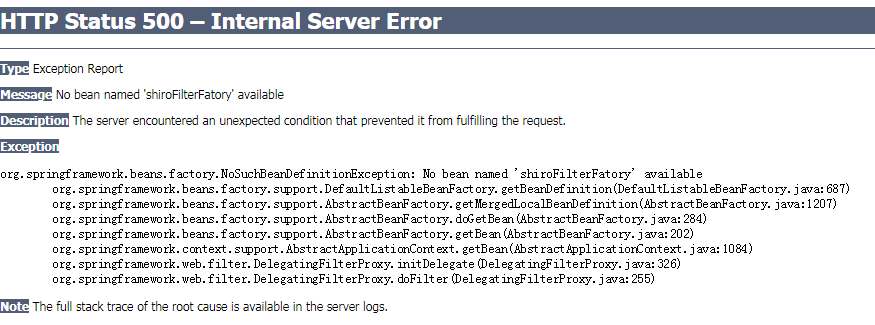
1. shiro 认证过程分析及实现(判定用户身份的合法性)
2. Shiro 授权过程分析及实现(对资源访问进行权限检测和授权)

## 常见FAQ

1. 说说shiro的核心组件?(Subject,SecurityManager,Realm,...)
2. 说说shiro的认证流程,你如何知道的,为什么要认证?
3. 资源访问认证拦截(访问资源时判定用户是否是认证用户)
4. 用户认证实现?(判定用户身份合法性)
5. 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?
6. 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?
7. Shiro中的记住我功能如何实现?为什么要使用这个功能?
8. Shiro中会话session的默认时长是多少,你怎么知道的?

## Bug分析

Bug-01



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

Bug-02

org.springframework.beans.factory.parsing.BeanDefinitionParsingException: Configuration problem: Bean name 'securityManager' is already used in this <beans> element

Offending resource: class path resource [spring-shiro.xml]

问题分析:

在spring-shiro.xml中可能存在多个'securityManager' 对象的配置

Bug-03

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'shiroFilterFactory' defined in class path resource [spring-shiro.xml]: Cannot resolve reference to bean 'securityManager' while setting bean property 'SecurityManager'; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'securityManager' defined in class path resource [spring-shiro.xml]: Cannot resolve reference to bean 'shiroUserRealm' while setting bean property 'Realm'; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No bean named 'shiroUserRealm' available

问题分析:

1. spring-shiro.xml中'shiroUserRealm' 名字对应的bean的配置不存在
2. ShiroUserRealm 对象上没有使用@Service这样的注解修饰