Shiro安全框架

# 第一章 认识Shiro

## 1-1 Shiro介绍

Apache Shiro是一个强大且易用的Java安全框架,执行身份验证、授权、密码和会话管理。使用Shiro的易于理解的API,您可以快速、轻松地获得任何应用程序,从最小的移动应用程序到最大的网络和企业应用程序。

## 1-2主要功能

三个核心组件：Subject, SecurityManager 和 Realms.

Subject：即“当前操作用户”。但是，在Shiro中，Subject这一概念并不仅仅指人，也可以是第三方进程、后台帐户（Daemon Account）或其他类似事物。它仅仅意味着“当前跟软件交互的东西”。  
　　Subject代表了当前用户的安全操作，SecurityManager则管理所有用户的安全操作。  
　　SecurityManager：它是Shiro框架的核心，典型的Facade模式，Shiro通过SecurityManager来管理内部组件实例，并通过它来提供安全管理的各种服务。  
　　Realm： Realm充当了Shiro与应用安全数据间的“桥梁”或者“连接器”。也就是说，当对用户执行认证（登录）和授权（访问控制）验证时，Shiro会从应用配置的Realm中查找用户及其权限信息。  
　　从这个意义上讲，Realm实质上是一个安全相关的DAO：它封装了数据源的连接细节，并在需要时将相关数据提供给Shiro。当配置Shiro时，你必须至少指定一个Realm，用于认证和（或）授权。配置多个Realm是可以的，但是至少需要一个。  
　　Shiro内置了可以连接大量安全数据源（又名目录）的Realm，如LDAP、关系数据库（JDBC）、类似INI的文本配置资源以及属性文件等。如果缺省的Realm不能满足需求，你还可以插入代表自定义数据源的自己的Realm实现。

# 第二章 SpringBoot集成Shiro

## 2-1 创建Shiro环境

创建一个普通SpringBoot的Web工程01-Shiro添加依赖包

<**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-spring</**artifactId**>  
 <**version**>1.3.2</**version**>  
</**dependency**>

## 2-2 配置Shiro

### 2-2-1定义类com.wkcto.shoro.config.ShiroConfig

*//标记当前类是一个Spring的配置类，用于模拟Spring的配置文件  
//在这里我们将要配置Shiro*@Configuration  
**public class** ShiroConfig {  
 *//配置Shiro的安全管理器* @Bean  
 **public** SecurityManager securityManager(Realm myRealm){  
 DefaultWebSecurityManager securityManager=**new** DefaultWebSecurityManager();  
 *//设置一个Realm，这个Realm是最终用于完成我们的认证号和授权操作的具体对象* securityManager.setRealm(myRealm);  
 **return** securityManager;  
 }  
 *//配置一个自定义的Realm的bean，最终将使用这个bean返回的对象来完成我们的认证和授权* @Bean  
 **public** Realm myRealm(){  
 MyRealm realm=**new** MyRealm();  
 **return** realm;  
 }  
  
 *//配置一个Shiro的过滤器bean，这个bean将配置Shiro相关的一个规则的拦截  
 //例如什么样的请求可以访问什么样的请求不可以访问等等* @Bean  
 **public** ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager){  
 *//创建Shiro的拦截的拦截器 ，用于拦截我们的用户请求* ShiroFilterFactoryBean shiroFilter=**new** ShiroFilterFactoryBean();  
 *//设置Shiro的安全管理，设置管理的同时也会指定某个Realm 用来完成我们权限分配* shiroFilter.setSecurityManager(securityManager);  
 *//用于设置一个登录的请求地址，这个地址可以是一个html或jsp的访问路径，也可以是一个控制器的路径  
 //作用是用于通知Shiro我们可以使用这里路径转向到登录页面，但Shiro判断到我们当前的用户没有登录时就会自动转换到这个路径  
 //要求用户完成成功* shiroFilter.setLoginUrl(**"/"**);  
 *//登录成功后转向页面，由于用户的登录后期需要交给Shiro完成，因此就需要通知Shiro登录成功之后返回到那个位置* shiroFilter.setSuccessUrl(**"/success"**);  
 *//用于指定没有权限的页面，当用户访问某个功能是如果Shiro判断这个用户没有对应的操作权限，那么Shiro就会将请求  
 //转向到这个位置，用于提示用户没有操作权限* shiroFilter.setUnauthorizedUrl(**"/noPermission"**);  
 *//定义一个Map集合，这个Map集合中存放的数据全部都是规则，用于设置通知Shiro什么样的请求可以访问什么样的请求不可以访问* Map<String,String> map=**new** LinkedHashMap<String,String>();  
 *// /login 表示某个请求的名字 anon 表示可以使用游客什么进行登录（这个请求不需要登录）* map.put(**"/login"**,**"anon"**);  
 *//我们可以在这里配置所有的权限规则这列数据真正是需要从数据库中读取出来  
 //或者在控制器中添加Shiro的注解  
 // /admin/\*\* 表示一个请求名字的通配， 以admin开头的任意子孙路径下的所有请求  
 // authc 表示这个请求需要进行认证（登录），只有认证（登录）通过才能访问  
 // 注意： \*\* 表示任意子孙路径  
 // \* 表示任意的一个路径  
 // ? 表示 任意的一个字符  
 map.put("/admin/\*\*","authc");  
 map.put("/user/\*\*","authc");  
 //表示所有的请求路径全部都需要被拦截登录，这个必须必须写在Map集合的最后面,这个选项是可选的  
 //如果没有指定/\*\* 那么如果某个请求不符合上面的拦截规则Shiro将方行这个请求  
 map.put("/\*\*","authc");* shiroFilter.setFilterChainDefinitionMap(map);  
 **return** shiroFilter;  
 }  
}

### 2-2-2定义com.wkcto.shiro.config.MyRealm类

*//自定义Realm永远完成具体的认证和授权操作  
// Realm的父类抽象类  
// AuthenticatingRealm 只负责认证（登录）的Realm父类  
// AuthorizingRealm 负责认证（登录）和授权 的Realm父类***public class** MyRealm **implements** Realm {  
 @Override  
 **public** String getName() {  
 **return null**;  
 }  
  
 @Override  
 **public boolean** supports(AuthenticationToken authenticationToken) {  
 **return false**;  
 }  
  
 @Override  
 **public** AuthenticationInfo getAuthenticationInfo(AuthenticationToken authenticationToken) **throws** AuthenticationException {  
 **return null**;  
 }  
}

### 2-2-3定义com.wkcto.shiro.controller.UserController

@Controller  
**public class** UserController {  
 @RequestMapping(**"/"**)  
 **public** String index(){  
 **return "login"**;  
 }  
 @RequestMapping(**"/login"**)  
 **public** String login(String username, String password, Model model){  
 **return "redirect:/success"**;  
 }  
@RequestMapping(**"/success"**)  
 **public** String success(){  
 **return "success"**;  
 }  
 @RequestMapping(**"/noPermission"**)  
 **public** String noPermission(){  
  
 **return "noPermission"**;  
 }  
  
  
 @RequestMapping(**"/user/test"**)  
 **public** @ResponseBody String userTest(){  
 **return "这是userTest请求"**;  
 }  
@RequestMapping(**"/admin/test"**)  
 **public** @ResponseBody String adminTest(){  
 **return "这是adminTest请求"**;  
 }  
 @RequestMapping(**"/admin/add"**)  
 **public** @ResponseBody String adminAdd(){  
 Subject subject= SecurityUtils.*getSubject*();  
 **return "这是adminAdd请求"**;  
 }  
}

### 2-2-4定义login.html 、nopermission.html、success.html

login.html

<**form action="login" method="post"**>  
 账号<**input type="text" name="username"**><**br**>  
 密码<**input type="text" name="password" id="password"**><**br**>  
 <**input type="submit" value="登录" id="loginBut"**>  
</**form**>

<**span style="color**: **red" th:text="${errorMessage}"**></**span**>

nopermission.html

<**h1**>没有权限请联系管理员</**h1**>

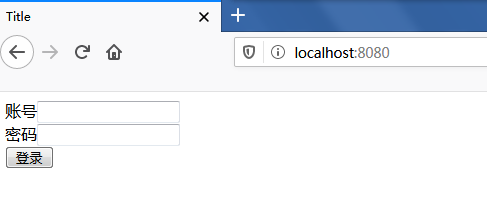
success.html

<**h1**>登录成功</**h1**>

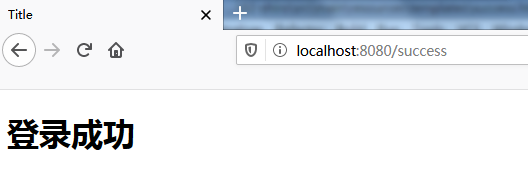
### 2-2-5测试

打开了访问 /success以及 /admin/test请求

由于配置/admin/test的请求需要认证因此无法直接访问而是转向到了登录页面



而访问/success请求因为没有配置对应的拦截所及可以访问



## 2-3 配置Shiro认证账号

### 2-3-1 修改com.wkcto.shiro.config.MyRealm类

*//自定义Realm永远完成具体的认证和授权操作  
// Realm的父类抽象类  
// AuthenticatingRealm 只负责认证（登录）的Realm父类*

**public class** MyRealm **extends** AuthenticatingRealm

{  
 */\*\*  
 \* Shiro的认证方法我们需要在这个方法中来获取用户的信息（从数据库中）  
 \** ***@param authenticationToken*** *用户登录时的Token（令牌），这个对象中将存放着我们用户在浏览器中存放的账号和密码  
 \** ***@return*** *返回一个AuthenticationInfo 对象，这个返回以后Shiro会调用这个对象中的一些方法来完成对密码的验证 密码是由Shiro  
 \* 进行验证是否合法  
 \** ***@throws*** *AuthenticationException 如果认证失败Shiro就会抛出AuthenticationException 我们也可以手动自己抛出这个AuthenticationException  
 \* 以及它的任意子异常类不通的异常类型可以认证过程中的不通错误情况我们需要根据异常类型来为用户返回特定的响应数据  
 \* AuthenticationException 异常的子类 可以我们自己抛出  
 \* AccountException 账号异常 可以我们自己抛出  
 \* UnknownAccountException 账号不存在的异常 可以我们自己抛出  
 \* LockedAccountException 账号异常锁定异常 可以我们自己抛出  
 \* IncorrectCredentialsException 密码错误异常 这个异常会在Shiro进行密码验证是抛出  
 \*/* **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) **throws** AuthenticationException {  
 *//将AuthenticationToken强转成UsernamePasswordToken 这样获取账号和密码更加的方便* UsernamePasswordToken token= (UsernamePasswordToken)authenticationToken;  
 *//获取用户在浏览器中输入的账号* String username=token.getUsername();  
 *//认证账号,正常情况我们需要这里从数据库中获取账号的信息，以及其他关键数据，例如账号是否被冻结等等* String dbusername=username;  
 **if**(!**"admin"**.equals(dbusername)&&!**"zhangsan"**.equals(dbusername)){*//判断用户账号是否正确* **throw new** UnknownAccountException(**"账号错误"**);  
 }  
**if**(**"zhangsan"**.equals(username)){  
 **throw new** LockedAccountException(**"账号被锁定"**);  
 }

*//定义一个密码这个密码应该来自数据库中*

String dbpassword=**"123456"**;

*//认证密码是否正确* **return new** SimpleAuthenticationInfo(dbusername,dbpassword,getName());  
 }  
 }

### 2-3-2 修改com.wkcto.shiro.controller.UserController

@RequestMapping(**"/login"**)  
 **public** String login(String username, String password, Model model){  
 *//创建一个shiro的Subject对象，利用这个对象来完成用户的登录认证* Subject subject= SecurityUtils.*getSubject*();  
 *//判断当前用户是否已经认证过，如果已经认证过着不需要认证如果没有认证过则进入if完成认证* **if**(!subject.isAuthenticated()){  
 *//创建一个用户账号和密码的Token对象，并设置用户输入的账号和面  
 //这个对象将在Shiro中被获取* UsernamePasswordToken token=**new** UsernamePasswordToken(username,password);  
 **try** {  
 *//例如账号不存在或密码错误等等，我们需要根据不同的异常类型来判断用户的登录状态并给与友好的信息提示  
 //调用login后Shiro就会自动执行我们自定义的Realm中的认证方法* subject.login(token);  
 } **catch** (UnknownAccountException e) {

*//进入catch 表示用户的账号错误，这个异常是我们在后台抛出的* System.***out***.println(**"---------------账号不存在"**);model.addAttribute(**"errorMessage"**,**"账号不存在"**);  
 **return "login"**;  
 }**catch** (LockedAccountException e){

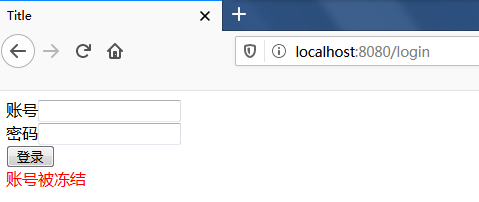
*//进入catch 表示用户的账号被锁定，这个异常是我们在后台抛出的* System.***out***.println(**"===============账号被锁定"**);model.addAttribute(**"errorMessage"**,**"账号被冻结"**);  
 **return "login"**;  
 }**catch** (IncorrectCredentialsException e){

*//进入catch 表示用户的密码，这个异常是shiro在认证密码时抛出* System.***out***.println(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*密码不匹配"**);model.addAttribute(**"errorMessage"**,**"密码错误"**);  
 **return "login"**;  
 }  
 }  
 **return "redirect:/success"**;  
 }

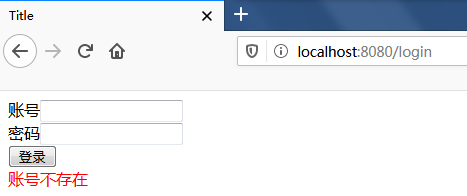
### 2-3-3测试

打开浏览器访问<http://localhost:8080> ,，，

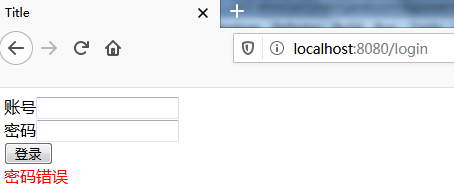
当时输入账号为shangsan 密码任意 则会显示账号冻结的错误提示



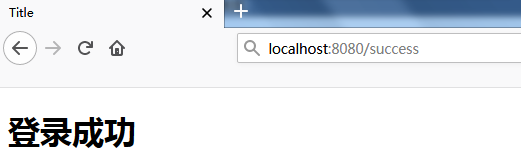
输入账号abc 密码任意则出现账号不存在的提示



输入账号admin 密码为123 则出现账号错误的提示



当输入账号为admin 密码为123456则户登录成功显示登录成功



## 2-4认证缓存

当登录成功过一次以后我可以点击后退，然后输入任意账号和密码这时无论输入什么信息Shiro都会认为认证成功，这是因为Shiro在登录成功以后会将数据写入Shiro的缓存导致，因此应该在登录请求的控制器中在判断是否认证过之前添加一个登出操作，已清空缓存这样就可以重复测试登录

### 2-4-1修改com.wkcto.shiro.controller.UserController

*//创建一个shiro的Subject对象，利用这个对象来完成用户的登录认证*Subject subject= SecurityUtils.*getSubject*();  
*//登出方法调用，用于清空登录时的缓存信息，否则无法重复登录*subject.logout();

## 2-5 密码加密

### 2-5-1修改com.wkcto.shiro.config.MyRealm类

*//定义一个密码，这个密码是数据库中的密码我们应该从数据库中获取*String dbpassword=**"123456"**;

*//密码加密码  
//参数 1 为加密算法 我们选择MD5加密  
//参数 2 为被加密的数据的数据  
//参数 3 为加密时的盐值 ，用于改变加密后数据结果  
// 通常这个盐值需要选择一个表中唯一的数据例如表中的账号  
//参数 4 为需要对数据使用指定的算法加密多少次*Object obj=**new** SimpleHash(**"MD5"**,dbpassword,**" "**,1);  
*//认证密码是否正确 使用加密后的密码登录***return new** SimpleAuthenticationInfo(dbusername,obj.toString(),getName());

注意：

1. 通常数据库中存放的数据不应该是明码123456 而是加密后的数据例如e10adc3949ba59abbe56e057f20f883e，这是使用MD5加密后的123456，如果数据库中的密码已经是加密后的那么这里可以不选择进行加密。
2. 如果数据库中的密码已经加密那么页面中传递数据前必须要对密码进行加密才能传递，否则无法可能会登录失败。
3. 如果选择加密传递那么页面和数据库中的密码加密次数以及盐必须相同，否则登录一定失败

### 2-5-2修改login.html

<**script src="/js/jquery-1.11.3.min.js"**></**script**>  
 <**script src="/js/jQuery.md5.js"**></**script**>  
 <**script**>  
 **$**(**function**(){  
 **$**(**"#loginBut"**).bind(**"click"**,**function**(){  
 **var** v\_password=**$**(**"#password"**).val()  
 **$**(**"#md5Password"**).val(**$**.md5(v\_password));  
 })  
 })  
 </**script**>  
 <**form action="/login" method="post"**>  
 账号<**input type="text" name="username"**><**br**>  
 密码<**input type="text" id="password"**><**br**>  
 <**input type="hidden" name="password" id="md5Password"**>  
 <**input type="submit" value="登录" id="loginBut"**>  
 </**form**>  
 <**span style="color**: **red" th:text="${errorMessage}"**></**span**>

## 2-6. 权限分配

需要判断用户哪个权限是否可以使用，我们就必须要先为用户分配一个权限才可以分配需要在MyRealm类中配置并且修改继承的父类

### 2-6-1 修改com.wkcto.shiro.config.MyRealm类

修改MyRealm类继承的父类为 AuthorizingRealm类，并实现抽象方法doGetAuthorizationInfo

*//Shiro用户授权的回调方法* **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {  
 *//从Shiro中获取用户名* Object username=principalCollection.getPrimaryPrincipal();  
 *//创建一个SimpleAuthorizationInfo类的对象，利用这个对象需要设置当前用户的权限信息* SimpleAuthorizationInfo simpleAuthorizationInfo=**new** SimpleAuthorizationInfo();  
 *//创建角色信息的集合* Set<String> roles=**new** HashSet<String>();  
 *//这里应该根据账号到数据库中获取用户的所对应的所有角色信息并初始化到roles集合中* **if**(**"admin"**.equals(username)){  
 roles.add(**"admin"**);  
 roles.add(**"user"**);  
 }**else if** (**"zhangsan"**.equals(username)){  
 roles.add(**"user"**);  
 }  
 Set<String>psermission=**new** HashSet<String>();  
 **if**(**"admin"**.equals(username)){  
 psermission.add(**"admin:add"**);  
 }  
 *//设置角色信息* simpleAuthorizationInfo.setRoles(roles);  
 simpleAuthorizationInfo.setStringPermissions(psermission);  
 **return** simpleAuthorizationInfo;  
 }

### 2-6-2 修改com.wkcto.shiro.config.ShiroConfig

用户拥有角色和权限以后需要配置Shiro的权限规则

@Bean  
 **public** ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager){  
 *//创建Shiro的拦截的拦截器 ，用于拦截我们的用户请求* ShiroFilterFactoryBean shiroFilter=**new** ShiroFilterFactoryBean();  
 *//设置Shiro的安全管理，设置管理的同时也会指定某个Realm 用来完成我们权限分配* shiroFilter.setSecurityManager(securityManager);  
 *//用于设置一个登录的请求地址，这个地址可以是一个html或jsp的访问路径，也可以是一个控制器的路径  
 //作用是用于通知Shiro我们可以使用这里路径转向到登录页面，但Shiro判断到我们当前的用户没有登录时就会自动转换到这个路径  
 //要求用户完成成功* shiroFilter.setLoginUrl(**"/"**);  
 *//登录成功后转向页面，由于用户的登录后期需要交给Shiro完成，因此就需要通知Shiro登录成功之后返回到那个位置* shiroFilter.setSuccessUrl(**"/success"**);  
 *//用于指定没有权限的页面，当用户访问某个功能是如果Shiro判断这个用户没有对应的操作权限，那么Shiro就会将请求  
 //转向到这个位置，用于提示用户没有操作权限* shiroFilter.setUnauthorizedUrl(**"/noPermission"**);  
 *//定义一个Map集合，这个Map集合中存放的数据全部都是规则，用于设置通知Shiro什么样的请求可以访问什么样的请求不可以访问* Map<String,String> map=**new** LinkedHashMap<String,String>();  
 *// /login 表示某个请求的名字 anon 表示可以使用游客什么进行登录（这个请求不需要登录）* map.put(**"/login"**,**"anon"**);  
 *//roles[admin] 表示 以/admin/\*\*开头的请求需要拥有admin角色才可以访问否 则返回没有权限的页面  
 //perms[admin:add] 表示 /admin/test的请求需要拥有 admin:add权限才可访问  
 //注意：admin:add仅仅是一个普通的字符串用于标记某个权限功能*

map.put(**"/admin/test"**,**"authc,perms[admin:add]"**);

map.put(**"/admin/\*\*"**,**"authc,roles[admin]"**);  
 map.put(**"/user/\*\*"**,**"authc,roles[user]"**);  
   
 *//表示所有的请求路径全部都需要被拦截登录，这个必须必须写在Map集合的最后面,这个选项是可选的  
 //如果没有指定/\*\* 那么如果某个请求不符合上面的拦截规则Shiro将方行这个请求  
// map.put("/\*\*","authc");* shiroFilter.setFilterChainDefinitionMap(map);  
 **return** shiroFilter;  
 }

## 2-7 基于注解的权限控制

Shiro不仅支持配置的权限控制还支持基于控制器注解的权限控制，如果需要使用Shiro的注解权限必须要在配置类中启动Shiro注解支持

### 2-7-1 修改com.wkcto.shiro.config.ShiroConfig

@Bean  
 **public** ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager){  
 *//创建Shiro的拦截的拦截器 ，用于拦截我们的用户请求* ShiroFilterFactoryBean shiroFilter=**new** ShiroFilterFactoryBean();  
 *//设置Shiro的安全管理，设置管理的同时也会指定某个Realm 用来完成我们权限分配* shiroFilter.setSecurityManager(securityManager);  
 *//用于设置一个登录的请求地址，这个地址可以是一个html或jsp的访问路径，也可以是一个控制器的路径  
 //作用是用于通知Shiro我们可以使用这里路径转向到登录页面，但Shiro判断到我们当前的用户没有登录时就会自动转换到这个路径  
 //要求用户完成成功* shiroFilter.setLoginUrl(**"/"**);  
 *//登录成功后转向页面，由于用户的登录后期需要交给Shiro完成，因此就需要通知Shiro登录成功之后返回到那个位置* shiroFilter.setSuccessUrl(**"/success"**);  
 *//用于指定没有权限的页面，当用户访问某个功能是如果Shiro判断这个用户没有对应的操作权限，那么Shiro就会将请求  
 //转向到这个位置，用于提示用户没有操作权限* shiroFilter.setUnauthorizedUrl(**"/noPermission"**);  
 *//定义一个Map集合，这个Map集合中存放的数据全部都是规则，用于设置通知Shiro什么样的请求可以访问什么样的请求不可以访问* Map<String,String> map=**new** LinkedHashMap<String,String>();  
 *// /login 表示某个请求的名字 anon 表示可以使用游客什么进行登录（这个请求不需要登录）* map.put(**"/login"**,**"anon"**);   
 *//表示所有的请求路径全部都需要被拦截登录，这个必须必须写在Map集合的最后面,这个选项是可选的  
 //如果没有指定/\*\* 那么如果某个请求不符合上面的拦截规则Shiro将方行这个请求  
// map.put("/\*\*","authc");* shiroFilter.setFilterChainDefinitionMap(map);  
 **return** shiroFilter;  
 }

*/\*\*  
 \* 开启Shiro的注解例如（ @RequiresRoles @RequiresUser @RequiresPermissions）  
 \* 需要借助SpringAOP来扫描这些注解  
 \*/*@Bean  
**public** DefaultAdvisorAutoProxyCreator advisorAutoProxyCreator(){  
 DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator=**new** DefaultAdvisorAutoProxyCreator();  
 defaultAdvisorAutoProxyCreator.setProxyTargetClass(**true**);  
 **return** defaultAdvisorAutoProxyCreator;  
}  
  
*/\*\*  
 \* 开启AOP的注解支持  
 \** ***@return*** *\*/*@Bean  
**public** AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager){  
 AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor=**new** AuthorizationAttributeSourceAdvisor();  
 authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);  
 **return** authorizationAttributeSourceAdvisor;  
}

注意：启动注解的权限控制以后需要删除在Shiro配置类中的权限拦截的配置规则

map.put("/admin/test","authc,perms[admin:add]");

map.put("/admin/\*\*","authc,roles[admin]");  
map.put("/user/\*\*","authc,roles[user]");

### 2-7-2 修改com.wkcto.shiro.controller.UserController

@RequiresRoles(value = {**"user"**})  
 @RequestMapping(**"/user/test"**)  
 **public** @ResponseBody String userTest(){  
 **return "这个userTest请求"**;  
 }  
  
 *//RequiresRoles Shiro的注解 表示访问这功能必须要拥有 admin角色  
 //注意如果需要支持多个角色就直接填写多个角色名称即可 例如 "admin","user"  
 //RequiresRoles 属性 logical 用于在拥有多个角色时使用 取值为Logical.AND 表示并且的意思必须同时拥有多个角色 或  
 // Logical.OR 或者的意思，只要拥有多个角色中的其中一个即可  
 //注意使用了注解以后需要配置Spring声明式异常捕获，否则将在浏览器中直接看到Shiro的错误信息而不是友好的信息提示* @RequiresRoles(value = {**"admin"**})  
 @RequestMapping(**"/admin/test"**)  
 **public** @ResponseBody String adminTest(){  
 **return "这个adminTest请求"**;  
 }  
  
 *//@RequiresPermissions 注解用于指定当前请求必须要拥有指定的权限名字为 admin:add才能访问  
 //admin:add 只是一个普通的权限名称字符串，表示admin下的add功能* @RequiresPermissions(value = {**"admin:add"**})  
 @RequestMapping(**"/admin/add"**)  
 **public** @ResponseBody String adminAdd(){  
 Subject subject= SecurityUtils.*getSubject*();  
 *//验证当前用户是否拥有这个权限  
// subject.checkPermission();  
// //验证当前用户是否拥有这个角色  
// subject.checkRole();* **return "这个adminAdd请求"**;  
 }  
  
 *//配置一个Spring的异常监控，当工程抛出了value所指定的所以异常类型以后将直接进入到当前方法中* @ExceptionHandler(value = {Exception.**class**})  
 **public** String myError(Throwable throwable){  
 *//获取异常的类型，应该根据不同的异常类型进入到不通的页面显示不同提示信息* System.***out***.println(throwable.getClass());  
 System.***out***.println(**"---------------------------------"**);  
 **return "noPermission"**;  
 }

注意：Shiro验证失败以后会抛出异常，因此这时必须要配置一个Spring的异常监控方法myError否则当前Shiro权限认证失败以后将无法转向到错误页面

## 2-8 Shiro标签

使用Thymeleaf与Shiro标签