# shiro

## shiro概述

### 什么是shiro

Apache Shiro 是Java 的一个安全框架。Shiro 可以非常容易的开发出足够好的应用，其不仅可以用在JavaSE 环境，也可以用在JavaEE 环境。Shiro 可以帮助我们完成：认证、授权、加密、会话管理、与Web 集成、缓存等。

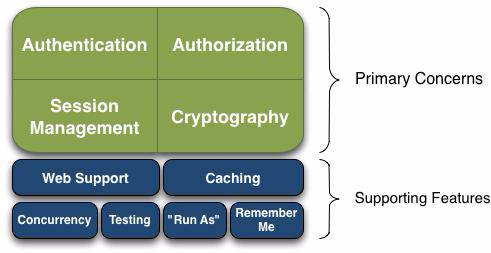
### 引入shiro

1，既然shiro将安全认证相关的功能抽取出来组成一个框架，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。

2，shiro使用广泛，shiro可以运行在web应用，非web应用，集群分布式应用中越来越多的用户开始使用shiro。

java领域中spring security(原名Acegi)也是一个开源的权限管理框架，但是spring security依赖spring运行，而shiro就相对独立，最主要是因为shiro使用简单、灵活，所以现在越来越多的用户选择shiro。

### shiro功能



1，Authentication 【??,θent?'ke???n/】

身份认证/登录，验证用户是不是拥有相应的身份；

2， Authorization【/??θ?ra?'ze??(?)n/】

授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情，常见的如：验证某个用户是否拥有某个角色。或者细粒度的验证某个用 户对某个资源是否具有某个权限；

3， Session Manager

会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通JavaSE环境的，也可以是如Web环境的；

4，Cryptography【/kr?p't?gr?f?/】

加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；

5，Web Support

Web 支持，可以非常容易的集成到Web 环境；

6，Caching

缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；

7，Concurrency【/k?n'k?r?nsi】

shiro 支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能 把权限自动传播过去；

8，Testing

提供测试支持；

9，Run As

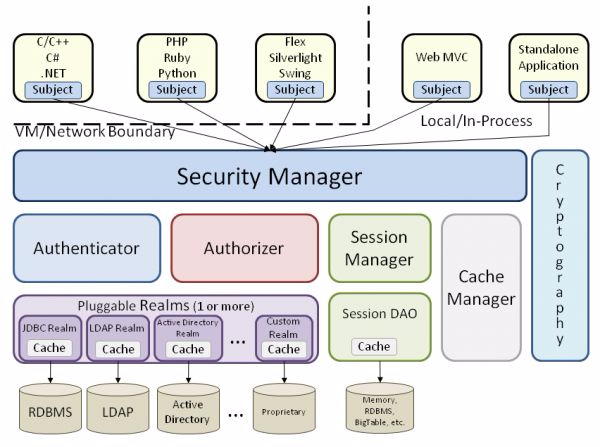
允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；

10，Remember Me

记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录了。

注意：Shiro 不会去维护用户、维护权限；这些需要我们自己去设计/提供；然后通过相应的接口注入给Shiro即可。关于设计，后面的ssm集成shiro里面去说哦

### shiro架构



1.subject

Subject即主体，外部应用与subject进行交互，subject记录了当前操作用户，将用户的概念理解为当前操作的主体，可能是一个通过浏览器请求的用户，也可能是一个运行的程序。 Subject在shiro中是一个接口，接口中定义了很多认证授相关的方法，外部程序通过subject进行认证授，而subject是通过SecurityManager安全管理器进行认证授权.

2.SecurityManager  
 SecurityManager即安全管理器，对全部的subject进行安全管理，它是shiro的核心，负责对所有的subject进行安全管理。通过SecurityManager可以完成subject的认证、授权等，实质上SecurityManager是通过Authenticator进行认证，通过Authorizer进行授权，通过SessionManager进行会话管理等。

3，Authenticator

Authenticator即认证器，对用户身份进行认证，Authenticator是一个接口，shiro提供ModularRealmAuthenticator实现类，通过ModularRealmAuthenticator基本上可以满足大多数需求，也可以自定义认证器。

4，Authorizer

Authorizer即授权器，用户通过认证器认证通过，在访问功能时需要通过授权器判断用户是否有此功能的操作权限。

5，realm

Realm即领域，相当于datasource数据源，securityManager进行安全认证需要通过Realm获取用户权限数据，比如：如果用户身份数据在数据库那么realm就需要从数据库获取用户身份信息。

注意：不要把realm理解成只是从数据源取数据，在realm中还有认证授权校验的相关的代码。

6，sessionManager

sessionManager即会话管理，shiro框架定义了一套会话管理，它不依赖web容器的session，所以shiro可以使用在非web应用上，也可以将分布式应用的会话集中在一点管理，此特性可使它实现单点登录。

7，SessionDAO

SessionDAO即会话dao，是对session会话操作的一套接口，比如要将session存储到数据库，可以通过jdbc将会话存储到数据库。

8，CacheManager

CacheManager即缓存管理，将用户权限数据存储在缓存，这样可以提高性能。

9，Cryptography

Cryptography即密码管理，shiro提供了一套加密/解密的组件，方便开发。比如提供常用的散列、加/解密等功能。

### shrio的jar包

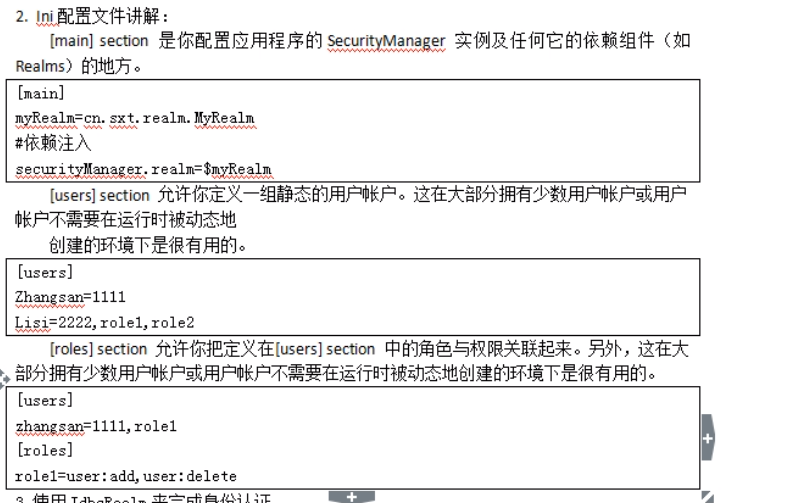
shiro-all 是shiro的所有功能jar包

shiro-core  是shiro的基本功能包

shiro-web  和web集成的包

shiro-spring  shrio和spring集成的包

## Shiro.ini文件



### Shiro.ini文件的说明

1. ini (InitializationFile) 初始文件.Window系统文件扩展名.

2. Shiro 使用时可以连接数据库,也可以不连接数据库.

        2.1 如果不连接数据库,可以在shiro.ini中配置静态数据

### Shrio.ini文件的组成部分

1，[main] :定义全局变量

    1 内置securityManager对象.

    2 操作内置对象时,在[main]里面写东西

|  |
| --- |
| [main]  securityManager.属性=值  myobj=com.bjsxt.lei  securityManager.对象属性=$myobj |

2，[users] :定义用户名和密码

|  |
| --- |
| [users]  # 定义用户名为zhangsan 密码为zs  zhangsan=zs  # 定义用户名lisi密码为lisi同时具有role1和role2两个角色  lisi=lisi,role1,role2 |

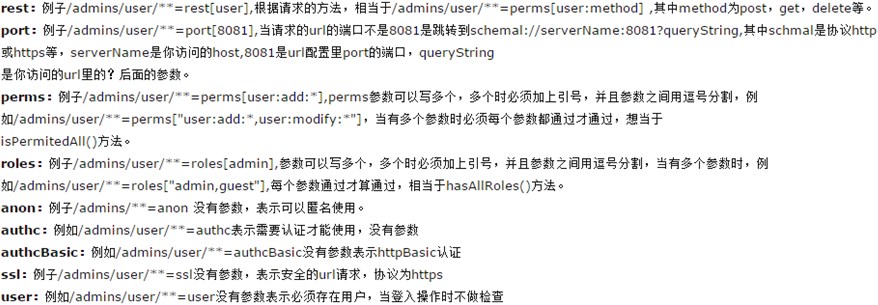
3[roles]: 定义角色

|  |
| --- |
| [roles]  role1=权限名1,权限名2  role2=权限3,权限4 |

4，[urls] : 定义哪些内置urls生效.在web应用时使用.

|  |
| --- |
| [urls]  #url地址=内置filter或自定义filter  # 访问时出现/login的url必须去认证.支持authc对应的Filter  /login=authc  # 任意的url都不需要进行认证等功能.  /\*\* = anon  # 所有的内容都必须保证用户已经登录.  /\*\*=user  # url abc 访问时必须保证用户具有role1和role2角色.  /abc=roles[“role1,role2”] |





## 实现认证

### 概念

1.身份验证

 即在应用中谁能证明他就是他本人。一般提供如他们的身份ID 一些标识信息来表明他就是他本人，如提供身份证，用户名/密码来证明。

在 shiro 中，用户需要提供principals （身份）和credentials（证明）给shiro，从而应用能验证用户身份：

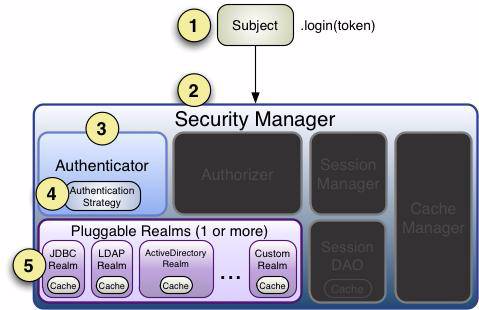
2.principals

身份，即主体的标识属性，可以是任何东西，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/手机号。

3.credentials

证明/凭证，即只有主体知道的安全值，如密码/数字证书等。 最常见的principals和credentials组合就是用户名/密码了。接下来先进行一个基本的身份认证。

### 认证流程



5.执行流程：

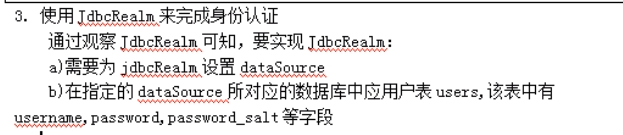
a）通过shiro 相关api,创建 securityManager及获取 Subject实例

b）封装token信息

c）通过subject.loginftoken)进行用户认证i.Subject 接收token，通过其实现类Delegatingsubject 将 token委托给SecurityManager来完成认证。SecurityManager是接口通过DefaultSecurityManager 来完成相关的功能。由DefaultSecurityManager中login来完成认证过程。在login 中调用了该类authenticatel（）来完成认证。该方法是由AuthenticatingsecurityManager 来完成的。在该类的authenticate()中，通过调用authenticator（认证器）来完成认证工作。Authenticator是由其黑默认实现类ModularRealmAuthenticator.来完成认证。通过ModularRealmAuthenticator中的doAuthenticate 来获取Realms信息。如果是单realm直接将token 和ream中的数据进行比较，判断是否认证成功。如果是多realm 那么需要通过Authentication Strategy来完成对应的认证工作。

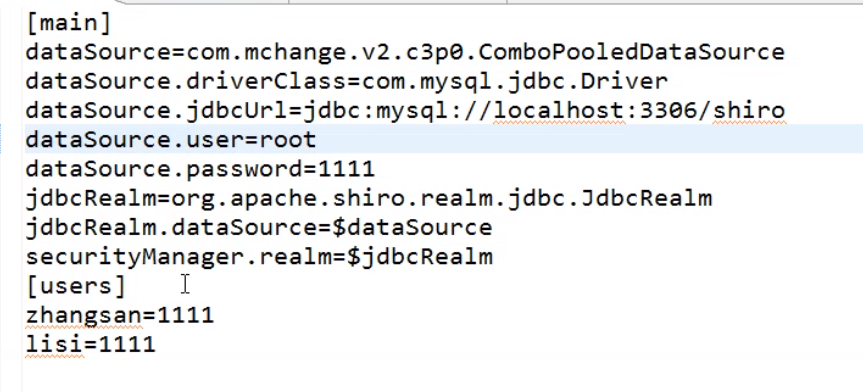
d）通过subject.isAuthenticated来判断是否认证成功。

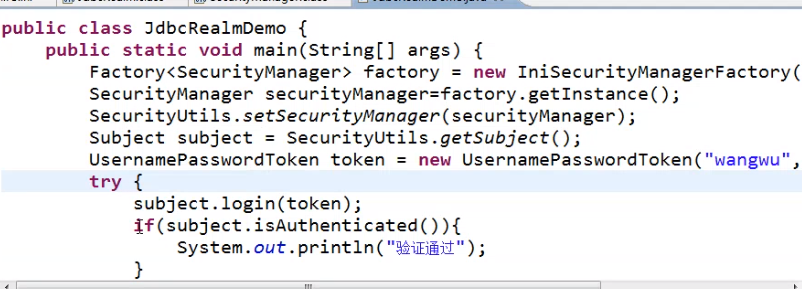
### JDBCRealm





1、配置





### 自定义realm

1.jdbcRealm已经实现了从数据库中获取用户的验证信息，但是jdbcRealm灵活性太差。如果要实现自己的一些特殊应用时将不能支持。这个时候可以通过自定义Realm来实现身份的认证功能。

2.Realm是一个接口，在接口中定义了根据 token获得认证信息的方法。Shiro内容实现了一系列的realm。这些不同Realm实现类提供了不同的功能。AuthenticatingRealm实现了获取身份信息的功能，AuthorizingRealm实现了获取权限信息的功能。通常自定义Realm 需要继承AuthorizingBealm，既这样可以提供了身份认证的自定义方法也可以实现授权的自定义方法。

实现自定义Realm

|  |
| --- |
| **public** **class** MyRealm **extends** AuthorizingRealm {  @Override  **public** String getName() {  **return** **this**.getClass().getSimpleName();  }  //完成身份认证，并且返回认证信息，如果认证失败，返回null  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  //1获取页面用户名和密码  String username =(String)token.getPrincipal();  //2根据用户名从数据库中获取密码信息。  String passwprd = "admin";  AuthenticationInfo info = **new** SimpleAuthenticationInfo(username, passwprd,**this**.getName());  System.***out***.println(info);  **return** info;  }  //获取权限信息  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  **return** **null**;  }  } |

shiro配置

|  |
| --- |
| #自定义realm  [main]  myrealm=com.sx.login.MyRealm  securityManager=org.apache.shiro.mgt.DefaultSecurityManager  securityManager.realms=$myrealm  #配置用户  [users]  admin=admin  sx=123456 |

登录：

|  |
| --- |
| **public** **class** login {  **public** **static** **void** main(String[] args) {  Factory<SecurityManager> factory = **new** IniSecurityManagerFactory("classpath:shiro.ini");  SecurityManager securityManager = factory.getInstance();  SecurityUtils.*setSecurityManager*(securityManager);  Subject subject = SecurityUtils.*getSubject*();  String username = "admin";  String password = "admin";  UsernamePasswordToken token = **new** UsernamePasswordToken(username, password);  **try** {  subject.login(token);  System.***out***.println("登录成功");  } **catch** (IncorrectCredentialsException e) {  System.***out***.println("登录失败");  }**catch** (UnknownAccountException e){  System.***out***.println("账号不存在");  }  }  } |

## 散列算法

## 授权

1.授权：给身份认证通过的人，授予他可以访问某些资源的权限。

2.权限粒度：分为粗粒度和细粒度。粗粒度：对user的crud。也就是说通常对表的操作。细粒度：是对记录的操作。如：只允许查询id为1的user的工资。Shiro一般管理的是粗粒度的权限。比如：菜单，按钮，url。一般细粒度的权限是通过业务来控制的。|

### 授权概述

授权，也叫访问控制，即在应用中控制谁能访问哪些资源（如访问页面/编辑数据/页面操作 等）。在授权中需了解的几个关键对象：主体（Subject）、资源（Resource）、权限（Permission）、 角色（Role）。

### 关键对象介绍

1.主体

主体，即访问应用的用户，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。

2，资源

在应用中用户可以访问的任何东西，比如访问JSP 页面、查看/编辑某些数据、访问某个业 务方法、打印文本等等都是资源。用户只要授权后才能访问。

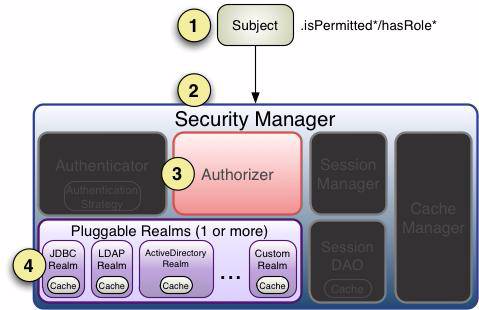
3，权限

安全策略中的原子授权单位，通过权限我们可以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：访问用户列表页面查看/新增/修改/删除用户数据（即很多时候都是CRUD（增查改删）式权限控制）打印文档等等。

4，角色

角色代表了操作集合，可以理解为权限的集合，一般情况下我们会赋予用户角色而不是权 限，即这样用户可以拥有一组权限，赋予权限时比较方便。典型的如：项目经理、技术总监、CTO、开发工程师等都是角色，不同的角色拥有一组不同的权限。

### 授权流程



### 相关方法说明

1 subject.hasRole(“”); 判断是否有角色

2 subject.hashRoles(List);分别判断用户是否具有List中每个内容

3 subject.hasAllRoles(Collection);返回boolean,要求参数中所有角色用户都需要具有.

4 subject.isPermitted(“”);判断是否具有权限.

### 配置

|  |
| --- |
| #配置用户  [users]  #定义张三的用户它拥有role1和role2的角色和 role1 role2的所有权限  zhangsan=123456,role1,role2  lisi=123456,role2  wangwu=123456,role3  zhangliu=123456,role5  #配置角色  [roles]  role1=user:add,user:update,user:delete,user:query,user:export  role2=user:query,user:add  role3=user:query,user:export  role4=user:query,user:update  role5=\*:\* |

登录

|  |
| --- |
| **public** **class** login {  **public** **static** **void** main(String[] args) {  Factory<SecurityManager> factory = **new** IniSecurityManagerFactory("classpath:shiro.ini");  SecurityManager securityManager = factory.getInstance();  SecurityUtils.*setSecurityManager*(securityManager);  Subject subject = SecurityUtils.*getSubject*();  String username = "admin";  String password = "admin";  UsernamePasswordToken token = **new** UsernamePasswordToken(username, password);  **try** {  subject.login(token);  System.***out***.println("登录成功");  } **catch** (IncorrectCredentialsException e) {  System.***out***.println("登录失败");  }**catch** (UnknownAccountException e){  System.***out***.println("账号不存在");  }  //判断用户是否含有role1角色  System.***out***.println(subject.hasRole("role1"));//true  //判断用户是否含有多种角色  List<String> roleIdentifiers = **new** ArrayList<>();  roleIdentifiers.add("role1");  roleIdentifiers.add("role2");  **boolean**[] hasRoles = subject.hasRoles(roleIdentifiers);  **for** (**boolean** b : hasRoles) {  System.***out***.println(b);//true,true  }  //判断用户是否有权限  System.***out***.println(subject.isPermitted("user:add"));//true  System.***out***.println(subject.isPermittedAll("user:add","user:update"));//false  String[] permissions= {"user:add","user:update"};  System.***out***.println("判断当前用户是否同时有user:add user:update权限:"+subject.isPermittedAll(permissions));  }  } |

# shrio拦截页面

## 依赖

|  |
| --- |
| <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-core</artifactId>  <version>1.4.0</version>  </dependency>  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-spring</artifactId>  <version>1.4.0</version>  </dependency> |

## 自动以realm

|  |
| --- |
| **public** **class** UserRealm **extends** AuthorizingRealm {  //授权逻辑  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  **return** **null**;  }  //创建defaultWebSecuirtyManager  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  **return** **null**;  }  } |

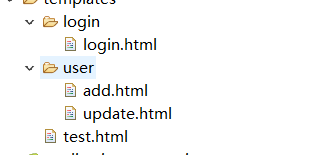
## ShrioConfig

|  |
| --- |
| @Configuration  **public** **class** ShrioConfig {  //1、常见shrioFilterFactoryBean  @Bean  **public** ShiroFilterFactoryBean getShrioFilterFactoryBean(@Qualifier("defaultWebSecurityManager")DefaultWebSecurityManager securityManager) {  ShiroFilterFactoryBean bean = **new** ShiroFilterFactoryBean();  bean.setSecurityManager(securityManager);  //添加shiro的内置过滤器  //1、常用的过滤器:  /\*anon:无需认证可以访问  authc:必须认证才可以访问  user:如果使用remeberme的功能可以直接访问  perms:该资源必须得到资源权限才可以访问  role:该资源必须得到角色权限才可以访问\*/  //1、设置访问的权限级别  Map<String, String> filterMap = **new** HashMap<>();  /\* filterMap.put("/user/add", "authc");  filterMap.put("/user/update", "authc");\*/  bean.setFilterChainDefinitionMap(filterMap);  //2、使用通配的原则  filterMap.put("/user/\*", "authc");  //如果没有验证通过访问跳转到登录页面  bean.setLoginUrl("/toLogin");  **return** bean;  }  //2、创建DefaultWebSecuurityManager  @Bean(name="defaultWebSecurityManager")  **public** DefaultWebSecurityManager getDefaultWebSecurityManager(@Qualifier("userRealm")UserRealm realm) {  DefaultWebSecurityManager defaultWebSecurityManager = **new** DefaultWebSecurityManager();  defaultWebSecurityManager.setRealm(realm);  System.***out***.println("sss");  **return** defaultWebSecurityManager;  }  @Bean("userRealm")  **public** UserRealm getRealm() {  **return** **new** UserRealm();  }  } |

## 路由

|  |
| --- |
| @Controller  **public** **class** IndexController {  @RequestMapping("testThymeleaf")  **public** String testThymeleaf(Model model) {  model.addAttribute("name", "黑马程序员");  **return** "test";  }    @RequestMapping("user/add")  **public** String add(Model model) {  model.addAttribute("name", "黑马程序员");  **return** "user/add";  }    @RequestMapping("user/update")  **public** String update(Model model) {  model.addAttribute("name", "黑马程序员");  **return** "user/update";  }    @RequestMapping("toLogin")  **public** String login(Model model) {  //用户没有权限访问，跳转到login页面  **return** "login/login";  }  } |

## 页面



# shrio登录操作

## ShrioConfig

|  |
| --- |
| @Configuration  **public** **class** ShrioConfig {  //1、常见shrioFilterFactoryBean  @Bean  **public** ShiroFilterFactoryBean getShrioFilterFactoryBean(@Qualifier("defaultWebSecurityManager")DefaultWebSecurityManager securityManager) {  ShiroFilterFactoryBean bean = **new** ShiroFilterFactoryBean();  bean.setSecurityManager(securityManager);  //添加shiro的内置过滤器  //1、常用的过滤器:  /\*anon:无需认证可以访问  authc:必须认证才可以访问  user:如果使用remeberme的功能可以直接访问  perms:该资源必须得到资源权限才可以访问  role:该资源必须得到角色权限才可以访问\*/  //1、设置访问的权限级别  Map<String, String> filterMap = **new** HashMap<>();  /\* filterMap.put("/user/add", "authc");  filterMap.put("/user/update", "authc");\*/  bean.setFilterChainDefinitionMap(filterMap);  //2、使用通配的原则  filterMap.put("/user/\*", "authc");  //如果没有验证通过访问跳转到登录页面  bean.setLoginUrl("/toLogin");  **return** bean;  }  //2、创建DefaultWebSecuurityManager  @Bean(name="defaultWebSecurityManager")  **public** DefaultWebSecurityManager getDefaultWebSecurityManager(@Qualifier("userRealm")UserRealm realm) {  DefaultWebSecurityManager defaultWebSecurityManager = **new** DefaultWebSecurityManager();  defaultWebSecurityManager.setRealm(realm);  System.***out***.println("sss");  **return** defaultWebSecurityManager;  }  @Bean("userRealm")  **public** UserRealm getRealm() {  **return** **new** UserRealm();  }  } |

## UserRealm

|  |
| --- |
| **public** **class** UserRealm **extends** AuthorizingRealm {  //授权逻辑  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  System.***out***.println("验证逻辑");  **return** **null**;  }  //创建defaultWebSecuirtyManager  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  System.***out***.println("登陆的验证");  //假设数据库的用户名和密码  String username = "root";  String password="123456";  //shiro的验证逻辑  //1、判断用户名  UsernamePasswordToken passwordToken = (UsernamePasswordToken)token;  **if**(!passwordToken.getUsername().equals(username)) {  //shiro地城会派出unknowAccountExecption  **return** **null**;  }  //判断密码  AuthenticationInfo authenticationInfo = **new** SimpleAuthenticationInfo("",password,"");  **return** authenticationInfo;  }  } |

## 登录的方法

|  |
| --- |
| @RequestMapping("login")  **public** String login(String username ,String password ,Model model) {  System.***out***.println("==============");  //使用shiro来编写认证的操作  Subject subject = SecurityUtils.*getSubject*();  //2、封装用户数据  UsernamePasswordToken token = **new** UsernamePasswordToken(username,password);  //用户没有权限访问，跳转到login页面  //3、执行登陆方法  **try** {  //会进入UserRealm进行判断  subject.login(token);  //登陆成功  **return** "redirect:/testThymeleaf";  } **catch** (UnknownAccountException e) {  //登陆不成功返回的异常  model.addAttribute("msg", "登陆失败");  **return** "login/login";  }  } |