# 01【熟悉】Shiro概述

## 1， 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。

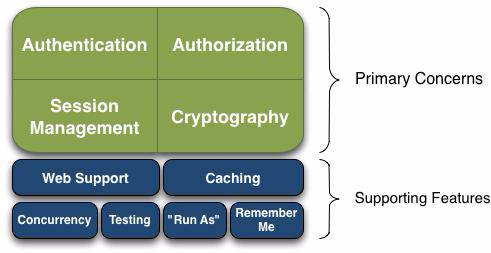
## 2，为什么要学shiro

        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。

## 3，基本功能



**1，Authentication**

身份认证/登录，验证用户是不是拥有相应的身份

**2， Authorization**

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

**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

**4，Cryptography**

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**

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；

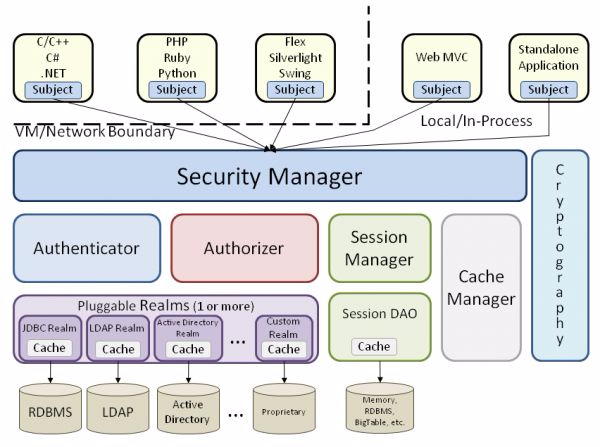
**10，Remember Me**

一次登录后，下次再来的话不用登录

**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。**

## 4，架构说明



1. **Subject**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进行认证授权，而subject通过SecurityManager安全管理器进行认证授权

**2，SecurityManager** 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。

SecurityManager是一个接口，继承了Authenticator, Authorizer, SessionManager这三个接口。

**3，Authenticator**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，可以满足大多数需求，也可以自定义认证器。

**4，Authorizer** 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** 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不仅仅是从数据源取数据，还有认证授权校验。

**6，sessionManager** 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

cookie不只是记录ID的工具而已。客户端持有ID，服务端持有session，两者一起用来保持登录状态。客户端需要用ID来作为凭证，而服务端需要用session来验证ID的有效性（ID可能过期、可能根本就是伪造的找不到对应的信息、ID下对应的客户端还没有进行登录验证等）。但是session是每个server自己独有的，豆瓣FM有自己的session、豆瓣读书有自己的session，而记录ID的cookie又是不能跨域的。所以，我们要实现一次登录一次退出，只需要想办法让各个server的共用一个session的信息，让客户端在各个域名下都能持有这个ID就好了。再进一步讲，只要各个server拿到同一个ID，都能有办法检验出ID的有效性、并且能得到ID对应的用户信息就行了，也就是能检验ID

**7，SessionDAO** 会话dao

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。

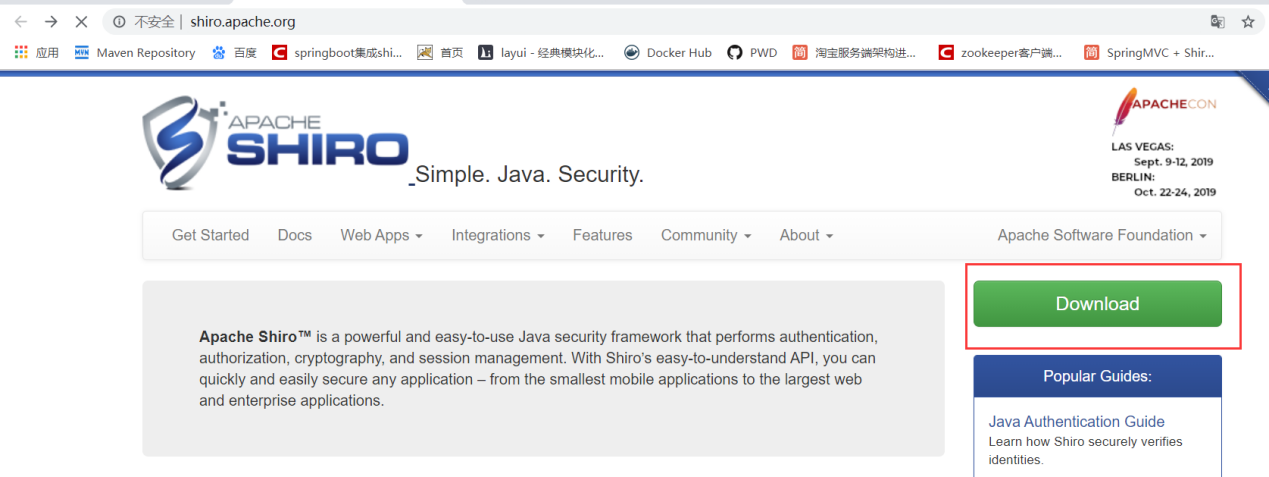
**8，CacheManager** 缓存管理

将用户权限数据存储在缓存，这样可以提高性能。

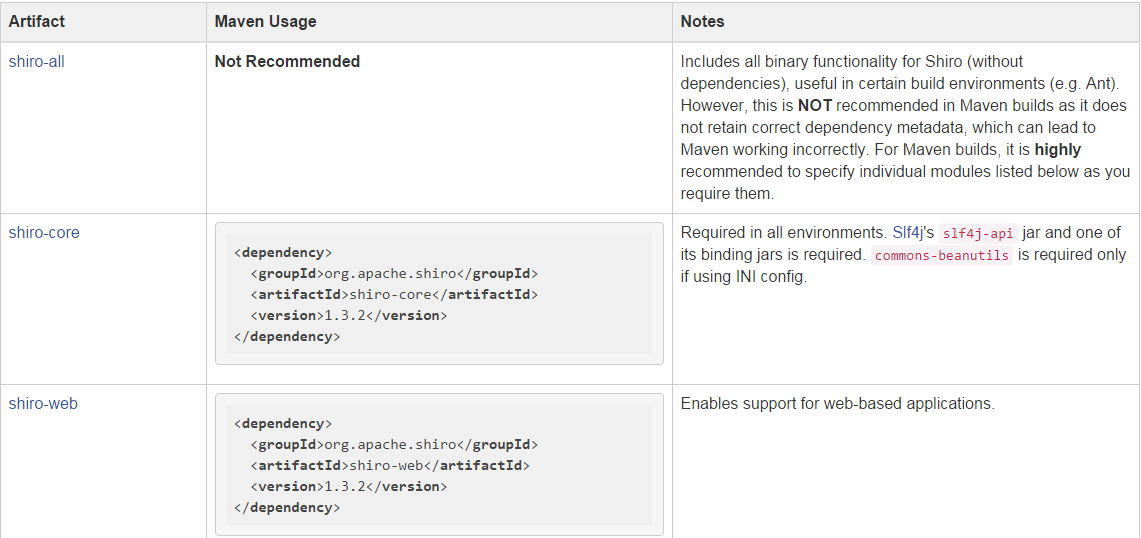
**9，Cryptography** 密码管理

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。

## 5，下载Shiro和maven的依赖







控制台程序至少要使用



# 02【掌握】Shiro.ini文件-先熟悉

## 1，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

## 2，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 <bean id=myobj class=” com.bjsxt.lei”>  securityManager.对象属性=$myobj <bean id=” securityManager” class=””>  <property name=” 对象属性” ref=” 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

如

[roles]

role1=user:query,user:add,user:update,user:delete,user:export

role2=user:query,user:add

### 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”]

authc 代表必须认证之后才能访问的路径

anon 任意的url都不需要进行认证等功能

user 所有的内容都必须保证用户已经登录.

logout 注销



# 03【掌握】Shiro实现认证\_ini

## 1，基本概念

**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。

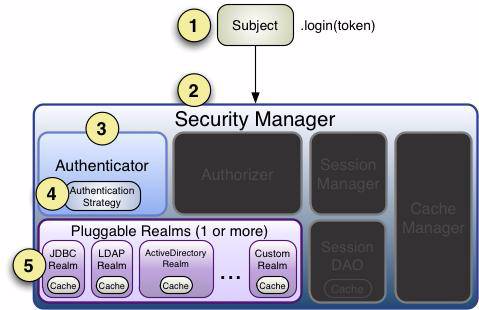
一个主体可以有多个principals，但只有一个Primary principals，一般是用户名/密码/手机号。

**3，credentials 相当于密码**

        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。

## 2，认证流程



1. 将身份和密码封装在subject中
2. 交给security manager 安全管理器
3. security manager调用authenticator认证器
4. 认证器去匹配realm

## 3，使用shiro的ini文件完成认证功能

### 创建项目并引入shiro-core的依赖

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.sxt.shiro</groupId>  <artifactId>01\_shiro\_Authentication\_ini</artifactId>  <version>1.0</version>  <dependencies>  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-core</artifactId>  <version>1.4.1</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-simple</artifactId>  <version>1.7.21</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>jcl-over-slf4j</artifactId>  <version>1.7.21</version>  <scope>test</scope>  </dependency>  </dependencies>  </project> |

### 创建log4j.properties

|  |
| --- |
| # Global logging configuration  log4j.rootLogger=DEBUG, stdout  # MyBatis logging configuration...  log4j.logger.org.mybatis.example.BlogMapper=TRACE  # Console output...  log4j.appender.stdout=org.apache.log4j.ConsoleAppender  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n |

### 创建shiro.ini

|  |
| --- |
| #配置用户  [users]  zhangsan=123456  lisi=123456 |

### 创建TestAuthenticationApp测试

|  |
| --- |
| **public** **class** TestAuthenticationApp {  // 日志输出工具  **private** **static** **final** **transient** Logger ***log*** = LoggerFactory.*getLogger*(TestAuthenticationApp.**class**);  @SuppressWarnings("deprecation")  **public** **static** **void** main(String[] args) {  String username = "zhangsan1";  String password = "123456";  ***log***.info("My First Apache Shiro Application");  // 1，创建安全管理器的工厂对象 org.apache.shiro.mgt.SecurityManager; 不能使用java.lang.SecurityManager  Factory<SecurityManager> factory = **new** ~~IniSecurityManagerFactory~~("classpath:shiro.ini");  // 2,使用工厂创建安全管理器  SecurityManager securityManager = factory.getInstance();  // 3,把当前的安全管理器绑定当到线的线程  SecurityUtils.*setSecurityManager*(securityManager);  // 4,使用SecurityUtils.getSubject得到主体对象  Subject subject = SecurityUtils.*getSubject*();  // 5，封装用户名和密码  AuthenticationToken token = **new** UsernamePasswordToken(username, password);  // 6,得到认证  **try** {  subject.login(token);  System.***out***.println("认证通过");  } **catch** (AuthenticationException e) {  System.***out***.println("用户名或密码不正确");  }  /\*} catch (IncorrectCredentialsException e) {  System.out.println("密码不正确");  } catch (UnknownAccountException e) {  System.out.println("用户名不存在");  }\*/    Subject subject2 = SecurityUtils.*getSubject*();    System.***out***.println(subject);  System.***out***.println(subject2);    *aaaa*();  }    **public** **static** **void** aaaa() {  Subject subject2 = SecurityUtils.*getSubject*();  System.***out***.println(subject2);  }  } |

# 04【掌握】shiro.ini实现授权

关键对象：主体（Subject）、资源（Resource）、权限（Permission）、角色（Role）。

## 2，关键对象介绍

1，主体

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。

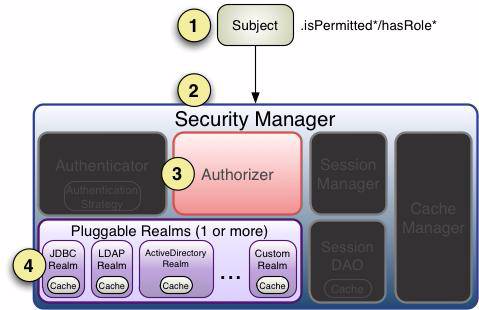
3，权限

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

## 3，授权流程



## 4，相关方法说明

1 subject.hasRole(“”); 判断是否有角色

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

3 subject.hasAllRoles(Collection); 判断是否拥有参数中所有角色

4 subject.isPermitted(“”);判断是否具有权限.

## 5，修改shiro.ini

|  |
| --- |
| #配置用户  [users]  zhangsan=123456,role1  lisi=123456,role2  wangwu=123456,role3  zhaoliu=123456,role2,role3  #声明角色  [roles]  role1=user:query,user:add,user:update,user:delete,user:export  role2=user:query,user:add  role3=user:query,user:export  role4=\*:\* 所有权限 |

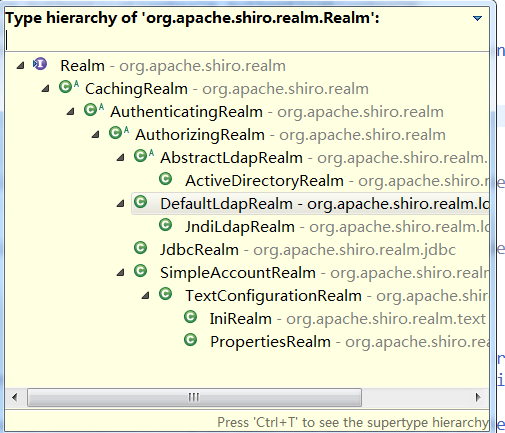
## 6，测试

|  |
| --- |
| public class TestAuthorizationApp {  // 日志输出工具  private static final transient Logger *log* = LoggerFactory.*getLogger*(TestAuthorizationApp.class);   public static void main(String[] args) {   String username = "sunqi";  String password = "123456";   *log*.info("My First Apache Shiro Application");  // 1，创建安全管理器的工厂对象 org.apache.shiro.mgt.SecurityManager; 不能使用java.lang.SecurityManager  Factory<SecurityManager> factory = new IniSecurityManagerFactory("classpath:shiro.ini");  // 2,使用工厂创建安全管理器  SecurityManager securityManager = factory.getInstance();  // 3,把当前的安全管理器绑定当到线的线程  SecurityUtils.*setSecurityManager*(securityManager);  // 4,使用SecurityUtils.getSubject得到主体对象  Subject subject = SecurityUtils.*getSubject*();  // 5，封装用户名和密码  AuthenticationToken token = new UsernamePasswordToken(username, password);  // 6,得到认证  try {  subject.login(token);  System.*out*.println("认证通过");  } catch (AuthenticationException e) {  System.*out*.println("用户名或密码不正确");  }     //subject.logout();//退出的方法  //判断用户是否认证通过  boolean authenticated = subject.isAuthenticated();  System.*out*.println("是否认证通过:"+authenticated);  //角色判断  boolean hasRole1 = subject.hasRole("role1");  System.*out*.println("是否有role1的角色:"+hasRole1);  //分别判断集合里面的角色 返回数组  //Arrays.asList 将数组转化为字符  List<String> roleIdentifiers=Arrays.*asList*("role1","role2","role3");  boolean[] hasRoles = subject.hasRoles(roleIdentifiers);  for (boolean b : hasRoles) {  System.*out*.println(b);  }  //判断当前用户是否有参数配置里面的所有角色  boolean hasAllRoles = subject.hasAllRoles(roleIdentifiers);  System.*out*.println(hasAllRoles);    //权限判断  boolean permitted = subject.isPermitted("user:query");  System.*out*.println("判断当前用户是否有user:query的权限 "+permitted);   //权限分别判断  boolean[] permitted2 = subject.isPermitted("user:query","user:add","user:export");  for (boolean b : permitted2) {  System.*out*.println(b);  }   //所有权限判断  boolean permittedAll = subject.isPermittedAll("user:query","user:add","user:export");  System.*out*.println(permittedAll);  } } |

# 05【掌握】自定义Realm实现认证

*Shiro默认使用自带的IniRealm，IniRealm从ini配置文件中读取用户的信息，大部分情况下需要从系统的数据库中读取用户信息，所以需要自定义realm。*

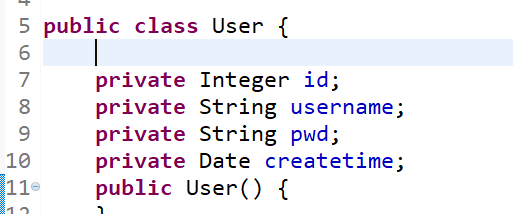
## 1，Realm接口



CachingRealm负责缓存处理，AuthenticationRealm负责认证，AuthorizingRealm负责授权，通常自定义的realm继承AuthorizingRealm,既能认证又能授权

## 2，实现步骤

### 创建User



### 创建UserService



### 创建UserServiceImpl



### 创建UserRealm

|  |
| --- |
| **public** **class** UserRealm **extends** AuthenticatingRealm {  **private** UserService userService=**new** UserServiceImpl();  /\*\*  \* 做认证  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  String username=token.getPrincipal().toString();  token.getCredentials();  System.***out***.println(username);  /\*\*  \* 以前登陆的逻辑是 把用户和密码全部发到数据库 去匹配  \* 在shrio里面是先根据用户名把用户对象查询出来，再来做密码匹配  \*/  User user=userService.queryUserByUserName(username);  **if**(**null**!=user) {  /\*\*  \* 参数说明  \* 参数1:可以传到任意对象  \* 参数2:从数据库里面查询出来的密码  \* 参数3:当前类名  \*/  SimpleAuthenticationInfo info=**new** SimpleAuthenticationInfo(user, user.getPwd(), **this**.getName());  **return** info;  }**else** {  //用户不存在 shiro会抛 UnknowAccountException  **return** **null**;  }  }  } |

### 修改shiro.ini

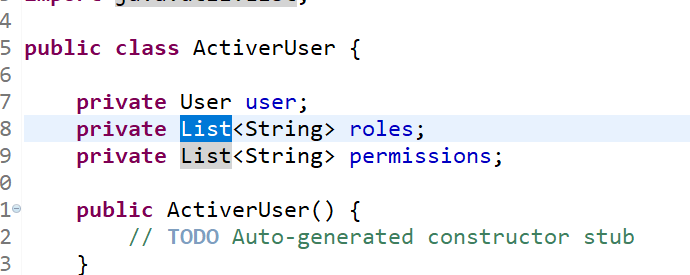
|  |
| --- |
| [main]  #创建userRealm对象  userRealm=com.sxt.realm.UserRealm  #把当前对象给安全管理器  #securityManager=org.apache.shiro.mgt.DefaultSecurityManager  securityManager.realm=$userRealm |

### 测试

|  |
| --- |
| /\*\*  \* shiro的认证使用shiro.ini文件  \*  \* **@author** LJH  \*  \*/  @SuppressWarnings("deprecation")  **public** **class** TestAuthenticationApp {  // 日志输出工具  **private** **static** **final** **transient** Logger ***log*** = LoggerFactory.*getLogger*(TestAuthenticationApp.**class**);  **public** **static** **void** main(String[] args) {  String username = "zhangsan";  String password = "123456";  ***log***.info("My First Apache Shiro Application");  // 1，创建安全管理器的工厂对象 org.apache.shiro.mgt.SecurityManager;  // 不能使用java.lang.SecurityManager  Factory<SecurityManager> factory = **new** ~~IniSecurityManagerFactory~~("classpath:shiro.ini");  // 2,使用工厂创建安全管理器  DefaultSecurityManager securityManager = (DefaultSecurityManager) factory.getInstance();  // 3,创建UserRealm  // UserRealm realm = new UserRealm();  // 4,给securityManager注入userRealm  // securityManager.setRealm(realm);  // 6,把当前的安全管理器绑定当到线的线程  SecurityUtils.*setSecurityManager*(securityManager);  // 7,使用SecurityUtils.getSubject得到主体对象  Subject subject = SecurityUtils.*getSubject*();  // 8，封装用户名和密码  AuthenticationToken token = **new** UsernamePasswordToken(username, password);  // 9,得到认证  **try** {  subject.login(token);  System.***out***.println("认证通过");    Object principal = subject.getPrincipal();    System.***out***.println(principal);    } **catch** (IncorrectCredentialsException e) {  System.***out***.println("密码不正确");  } **catch** (UnknownAccountException e) {  System.***out***.println("用户名不存在");  }  }  } |

# 06【掌握】自定义Realm实现授权

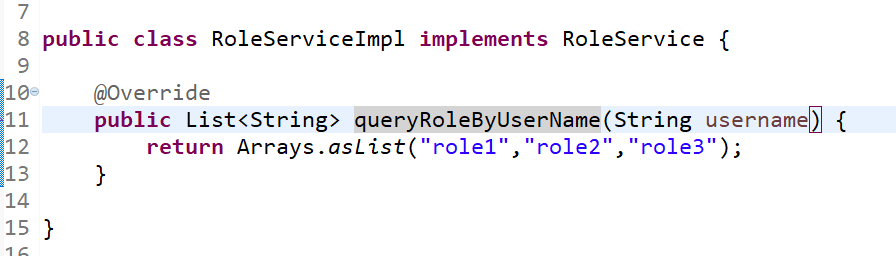
## 创建ActiverUser



## 创建RoleService



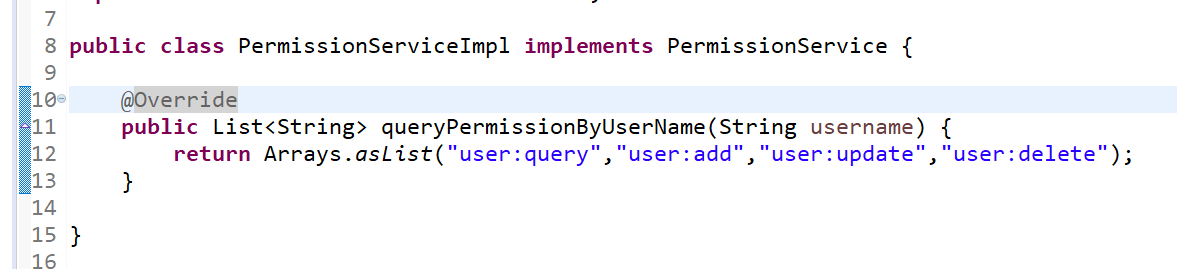
## 创建RoleServiceImpl



## 创建PermissionService



## 创建PermissionServiceImpl



## 修改UserRealm

|  |
| --- |
| **public** **class** UserRealm **extends** AuthorizingRealm {      **private** UserService userService=**new** UserServiceImpl();    **private** RoleService roleService =**new** RoleServiceImpl();    **private** PermissionService permissionService=**new** PermissionServiceImpl();      /\*\*  \* 做认证  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  String username=token.getPrincipal().toString();  token.getCredentials();  System.***out***.println(username);  /\*\*  \* 以前登陆的逻辑是 把用户和密码全部发到数据库 去匹配  \* 在shrio里面是先根据用户名把用户对象查询出来，再来做密码匹配  \*/  User user=userService.queryUserByUserName(username);  **if**(**null**!=user) {  List<String> roles=roleService.queryRoleByUserName(user.getUsername());    List<String> permissions=permissionService.queryPermissionByUserName(user.getUsername());  ActiverUser activerUser=**new** ActiverUser(user, roles, permissions);  /\*\*  \* 参数说明  \* 参数1:可以传到任意对象  \* 参数2:从数据库里面查询出来的密码  \* 参数3:当前类名  \*/  SimpleAuthenticationInfo info=**new** SimpleAuthenticationInfo(activerUser, user.getPwd(), **this**.getName());  **return** info;  }**else** {  //用户不存在 shiro会抛 UnknowAccountException  **return** **null**;  }  }  /\*\*  \* 作授权  \*  \* 参数说明  \*/  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  ActiverUser activerUser = (ActiverUser) principals.getPrimaryPrincipal();  SimpleAuthorizationInfo info=**new** SimpleAuthorizationInfo();  //添加角色  Collection<String> roles=activerUser.getRoles();  **if**(**null**!=roles&&roles.size()>0) {  info.addRoles(roles);  }  Collection<String> permissions=activerUser.getPermissions();  //添加权限  **if**(**null**!=permissions&&permissions.size()>0) {  info.addStringPermissions(permissions);  }  // if(activerUser.getUser().getType()==0) {  // info.addStringPermission("\*:\*");  // }  **return** info;  }  } |

## 测试

# 07【掌握】散列算法+凭证配置

散列算法一般用于生成数据的摘要信息,是一种不可逆的算法,如md5.

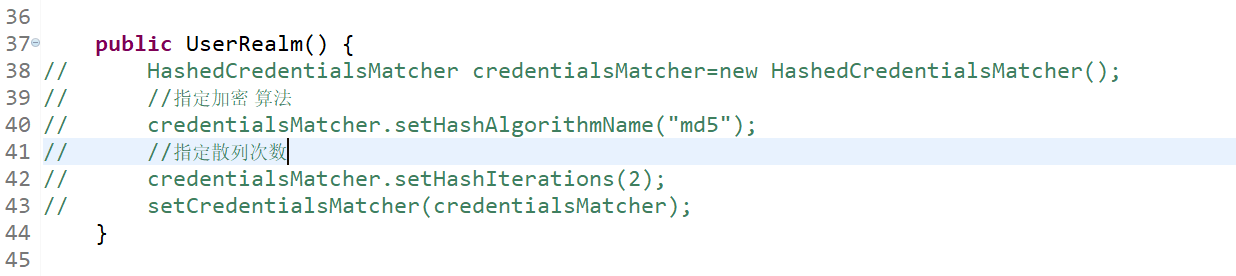
在进行散列算法时需要加入salt(盐),作为干扰

## 散列算法

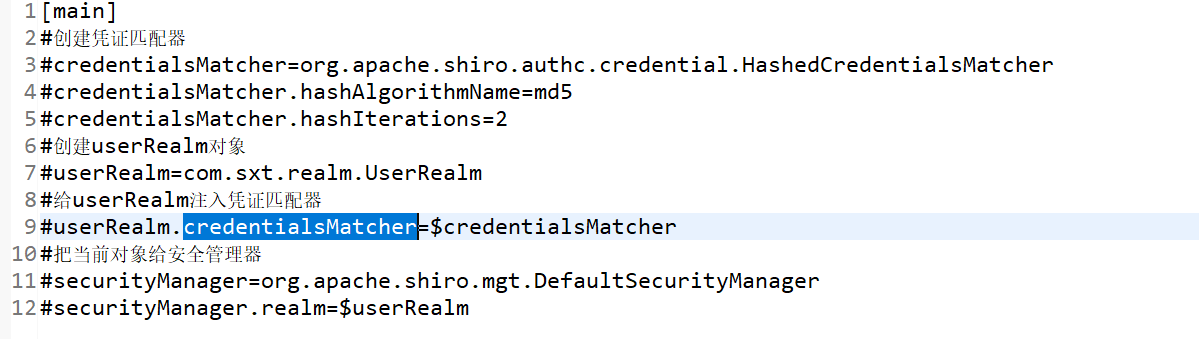
|  |
| --- |
| **public** **class** MD5Utils {    **public** **static** **void** main(String[] args) {  String source="123456";  Md5Hash hash1=**new** Md5Hash(source);  System.***out***.println("使用MD5加密后的结果:"+hash1.toString());    Md5Hash hash2=**new** Md5Hash(source, "北京武汉");  System.***out***.println("使用MD5加密并加盐后的结果:"+hash2.toString());  Md5Hash hash3=**new** Md5Hash(source, "北京武汉", 2);  System.***out***.println("使用MD5加密加盐并散列两次后的结果:"+hash3.toString());  }    /\*\*  \* 对密码加密 md5  \* **@param** source 要加密的明文  \* **@param** salt 盐  \* **@param** hashIterations 散列次数  \* **@return**  \*/  **public** **static** String md5(String source, Object salt, Integer hashIterations) {  **return** **new** Md5Hash(source, salt, hashIterations).toString();  }      /\*\*  \* 对密码加密sha1  \* **@param** source 要加密的明文  \* **@param** salt 盐  \* **@param** hashIterations 散列次数  \* **@return**  \*/  **public** **static** String sha1(String source, Object salt, Integer hashIterations) {  **return** **new** Sha1Hash(source, salt, hashIterations).toString();  }    } |

## 凭证配置

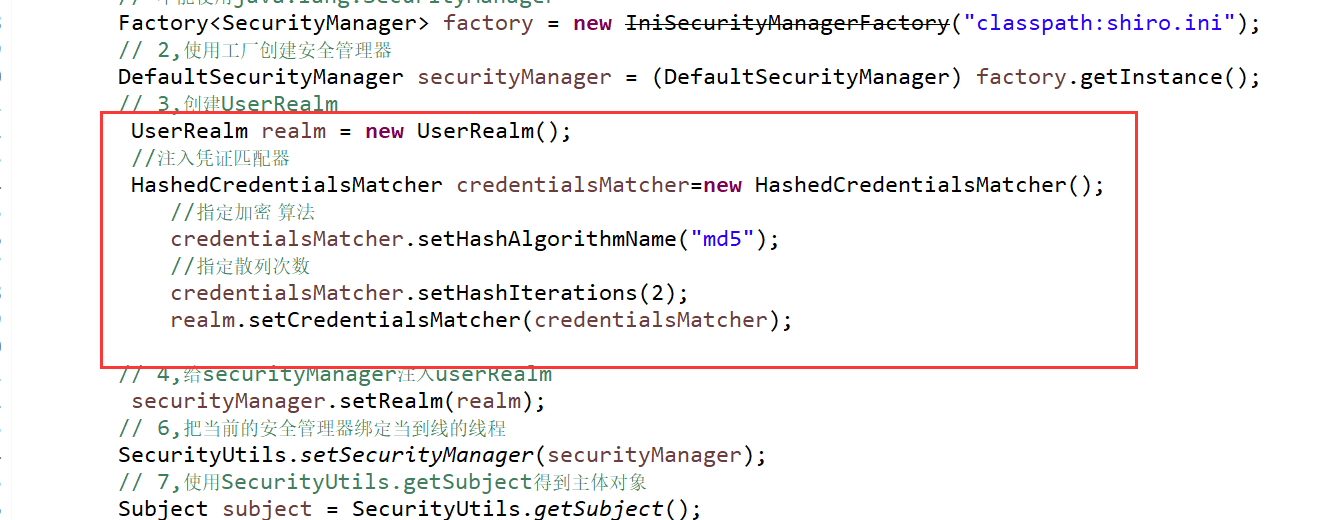
### 方式1



### 方式2



### 方式3



## 凭证的认证修改

### 修改UserServiceImpl



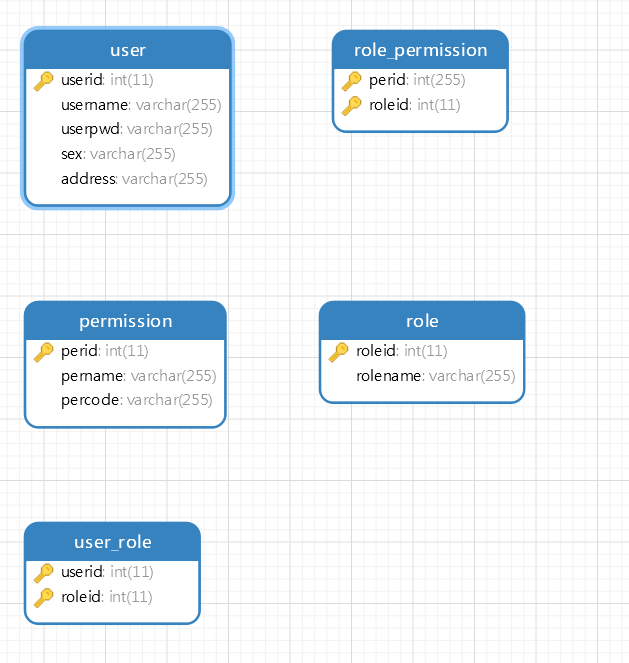
### 修改UserRealm



### 测试

# 08【掌握】ssm+shiro+Realm实现集成

## 准备数据库



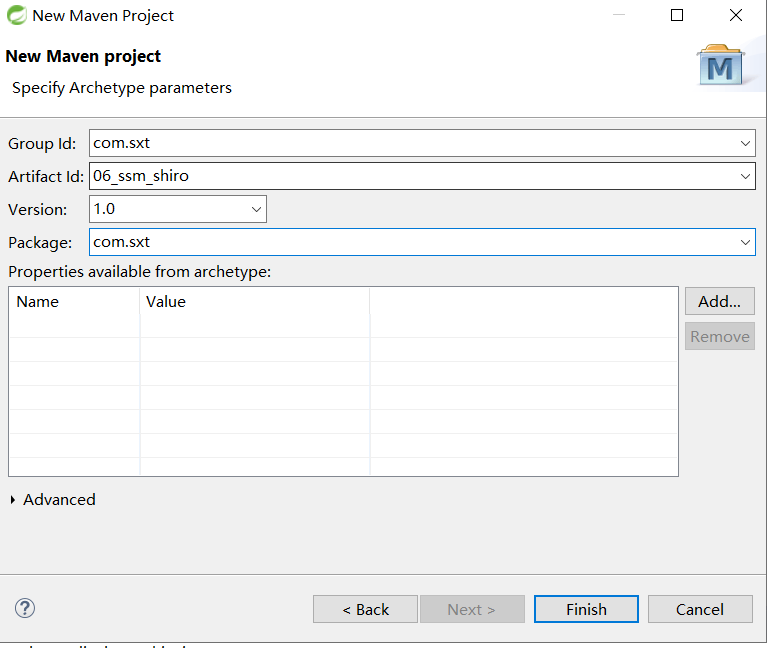
数据结构

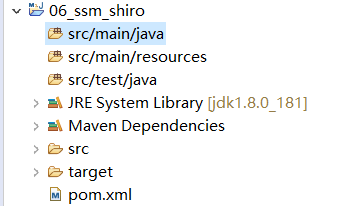
张三 -🡪user:query user:add user:update user:delete

李四 ---》user:query user:add user:update  
王五-----》user:query user:export

## 搭建ssm的环境

### 创建项目

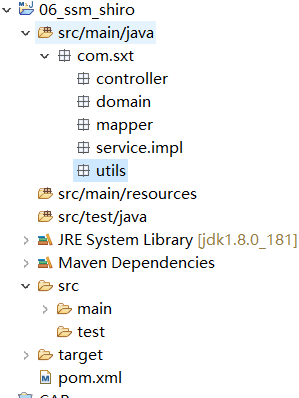




### 修改pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.sxt</groupId>  <artifactId>06\_ssm\_shiro</artifactId>  <packaging>war</packaging>  <version>1.0</version>  <name>06\_ssm\_shiro Maven Webapp</name>  <url>http://maven.apache.org</url>  <!-- 声明常量 -->  <properties>  <servlet.version>3.1.0</servlet.version>  <jsp.version>2.3.1</jsp.version>  <spring.version>4.3.24.RELEASE</spring.version>  <mybatis.version>3.5.1</mybatis.version>  <mybatis.spring.version>2.0.1</mybatis.spring.version>  <mysql.version>5.1.47</mysql.version>  <pagehelper.version>5.1.10</pagehelper.version>  <druid.version>1.1.19</druid.version>  <log4j.version>1.2.17</log4j.version>  <slf4j.version>1.7.26</slf4j.version>  <jackson.version>2.9.9</jackson.version>  <shiro.version>1.4.1</shiro.version>  </properties>  <dependencies>  <!--servlet -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>${servlet.version}</version>  <scope>provided</scope>  </dependency>  <!-- javax.servlet.jsp -->  <dependency>  <groupId>javax.servlet.jsp</groupId>  <artifactId>javax.servlet.jsp-api</artifactId>  <version>${jsp.version}</version>  <scope>provided</scope>  </dependency>  <!--spring-core -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-orm</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-oxm</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <!-- mybatis -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <!-- mybatis-spring -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${mybatis.spring.version}</version>  </dependency>  <!-- mysql-connector-java -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <!-- pagehelper -->  <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>${pagehelper.version}</version>  </dependency>  <!-- druid -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>${druid.version}</version>  </dependency>  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>${log4j.version}</version>  </dependency>  <!-- slf4j-api -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${slf4j.version}</version>  </dependency>  <!-- jackson-core -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>${jackson.version}</version>  </dependency>  <!-- 引入shiro的包 -->  <!-- <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-core</artifactId>  <version>${shiro.version}</version>  </dependency> -->  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-spring</artifactId>  <version>${shiro.version}</version>  </dependency>  </dependencies>  <build>  <finalName>06\_ssm\_shiro</finalName>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  <configuration>  <!-- 配置urlencoding -->  <uriEncoding>UTF-8</uriEncoding>  <!-- 配置端口 -->  <port>8080</port>  <!-- 配置访问路径 -->  <path>/bjsxt</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

### 创建包



### 创建db.properties

|  |
| --- |
| driver=com.mysql.jdbc.Driver  url=jdbc:mysql://127.0.0.1:3306/shiro?useUnicode=true&characterEncoding=utf8&useSSL=true  user=root  password=123456  #maxActive \u6700\u5927\u7684\u8FDE\u63A5\u6570  maxActive=20  #initialSize \u521D\u59CB\u5316\u8FDE\u63A5\u7684\u4E2A\u6570  initialSize=1  #maxWait \u6700\u5927\u7B49\u5F85\u65F6\u95F4  maxWait=60000  #minIdle \u6700\u5C0F\u6D3B\u8DC3\u8FDE\u63A5  minIdle=1  filters=stat,log4j,wall |

### 创建log4j.properties

|  |
| --- |
| driver=com.mysql.jdbc.Driver  url=jdbc:mysql://127.0.0.1:3306/shiro?useUnicode=true&characterEncoding=utf8&useSSL=true  user=root  password=123456  #maxActive \u6700\u5927\u7684\u8FDE\u63A5\u6570  maxActive=20  #initialSize \u521D\u59CB\u5316\u8FDE\u63A5\u7684\u4E2A\u6570  initialSize=1  #maxWait \u6700\u5927\u7B49\u5F85\u65F6\u95F4  maxWait=60000  #minIdle \u6700\u5C0F\u6D3B\u8DC3\u8FDE\u63A5  minIdle=1  filters=stat,log4j,wall |

### 创建application-dao.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>  <!-- 解析配置文件 -->  <context:property-placeholder location=*"classpath:db.properties"* system-properties-mode=*"FALLBACK"*/>  <!-- 声明数据源 -->  <bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*>  <property name=*"driverClassName"* value=*"${driver}"*></property>  <property name=*"url"* value=*"${url}"*></property>  <property name=*"username"* value=*"${user}"*></property>  <property name=*"password"* value=*"${password}"*></property>  <property name=*"maxActive"* value=*"${maxActive}"*></property>  <property name=*"initialSize"* value=*"${initialSize}"*></property>  <property name=*"maxWait"* value=*"${maxWait}"*></property>  <property name=*"minIdle"* value=*"${minIdle}"*></property>  <property name=*"filters"* value=*"${filters}"*></property>  </bean>    <!-- 创建mybatis的 configuration对象-->  <bean id=*"configuration"* class=*"org.apache.ibatis.session.Configuration"*>  <property name=*"logImpl"* value=*"org.apache.ibatis.logging.stdout.StdOutImpl"*></property>  </bean>    <!-- 创建sqlSessionFactory -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <!-- 注入数据源 -->  <property name=*"dataSource"* ref=*"dataSource"*></property>    <!-- 注入配置类 -->  <property name=*"configuration"* ref=*"configuration"*></property>    <!-- 扫描mapper.xml -->  <property name=*"mapperLocations"*>  <array>  <value>classpath:mapper/\*Mapper.xml</value>  </array>  </property>  <!-- 配置分页插件 -->  <property name=*"plugins"*>  <array>  <bean class=*"com.github.pagehelper.PageInterceptor"*></bean>  </array>  </property>  </bean>    <!-- 配置mapper接口的扫描 -->  <bean id=*"mapperScannerConfigurer"* class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.sxt.mapper"*></property>  <!-- <property name="basePackage">  <value>  com.sxt.mapper  com.bjsxt.mapper  </value>  </property> -->    <!-- 注入sqlSessionFactory -->  <property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"*></property>    </bean>  </beans> |

### 创建application-service.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.3.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd"*>  <!-- 扫描 -->  <context:component-scan base-package=*"com.sxt.service.impl"*></context:component-scan>    <!-- 配置事务管理器 -->  <bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 注入数据源 -->  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>    <!-- 配置事务的传播特性 -->  <tx:advice id=*"advise"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <tx:method name=*"add\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"save\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"insert\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"update\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"del\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"reset\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"change\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"\*"* read-only=*"true"*/>  </tx:attributes>  </tx:advice>    <!-- 配置aop -->  <aop:config>  <aop:pointcut expression=*"execution(\* com.sxt.service.impl.\*.\*(..))"* id=*"pc"*/>  <!-- 织入操作 -->  <aop:advisor advice-ref=*"advise"* pointcut-ref=*"pc"*/>  </aop:config>  </beans> |

### 创建applicationContext.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.3.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd"*>  <import resource=*"classpath:application-dao.xml"* />  <import resource=*"classpath:application-service.xml"* />  </beans> |

### 创建springmvc.xml

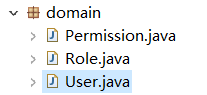
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd*  *http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd"*>  <!-- 扫描 -->  <context:component-scan base-package=*"com.sxt.controller"*></context:component-scan>  <!-- 配置适配器和映射器 -->  <mvc:annotation-driven></mvc:annotation-driven>    <!-- 配置视图解析器 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <!-- 前缀 -->  <property name=*"prefix"* value=*"/WEB-INF/view/"*></property>  <!-- 后缀 -->  <property name=*"suffix"* value=*".jsp"*></property>  </bean>  <!-- 配置文件上传 -->  <!-- 配置拦截器 -->  <!-- 配置静态资源放行-->  <mvc:default-servlet-handler/>  </beans> |

### 配置web.xml

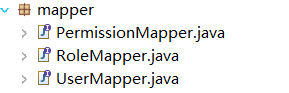
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*  xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*  id=*"WebApp\_ID"* version=*"3.1"*>  <display-name>06\_ssm\_shiro</display-name>  <!-- 配置编码过滤器 开始 -->  <filter>  <filter-name>encodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>encodingFilter</filter-name>  <servlet-name>springmvc</servlet-name>  </filter-mapping>  <!-- 配置编码过滤器 结束 -->  <!-- 配置spring的监听器加载 applicationContext.xml开始 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath\*:applicationContext.xml</param-value>  </context-param>  <!-- 配置spring的监听器加载 applicationContext.xml结束 -->  <!-- 配置前端控制器开始 -->  <servlet>  <servlet-name>springmvc</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- 注入springmvc.xml -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath\*:springmvc.xml</param-value>  </init-param>  <!-- 启动创建 -->  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>\*.action</url-pattern>  </servlet-mapping>  <!-- 配置前端控制器结束 -->  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  </web-app> |

## 集成shiro

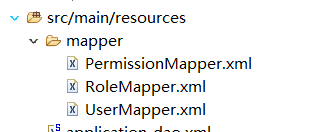
### 生成User Role Permission



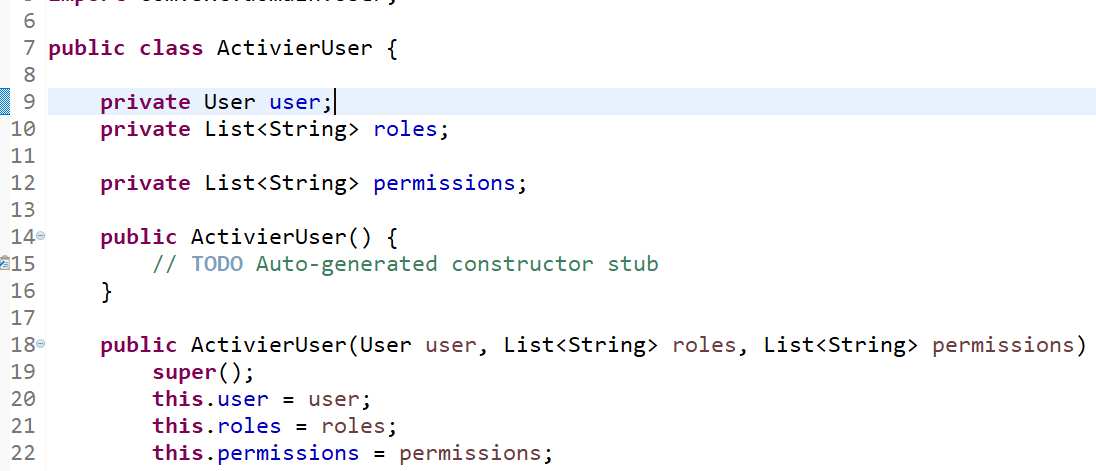
### 生成UserMapper RoleMapper PermissionMapper



### 生成UserMapper.xml RoleMapper.xml PermissionMapper.xml



### 创建ActivierUser



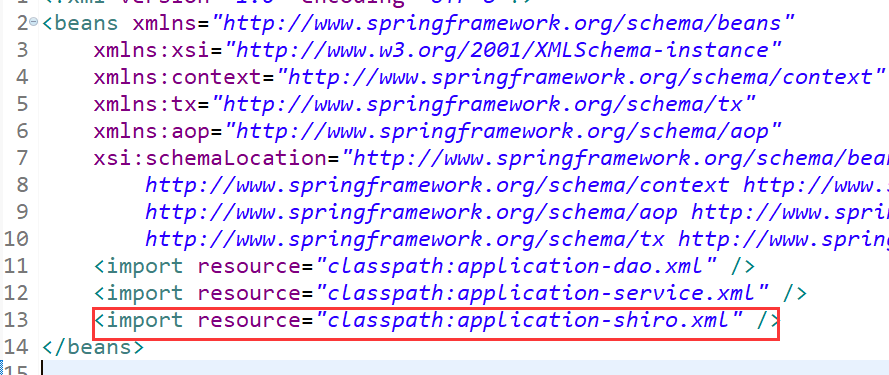
### 创建UserRealm

|  |
| --- |
| **public** **class** UserRealm **extends** AuthorizingRealm {    /\*\*  \* 认证  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {    **return** **null**;  }    /\*\*  \* 授权  \*/  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  // **TODO** Auto-generated method stub  **return** **null**;  }  } |

### 创建application-shrio.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>  <!-- 声明凭证匹配器 -->  <bean id=*"credentialsMatcher"* class=*"org.apache.shiro.authc.credential.HashedCredentialsMatcher"*>  <property name=*"hashAlgorithmName"* value=*"md5"*></property>  <property name=*"hashIterations"* value=*"2"*></property>  </bean>  <!-- 声明userRealm -->  <bean id=*"userRealm"* class=*"com.sxt.realm.UserRealm"*>  <!-- 注入凭证匹配器 -->  <property name=*"credentialsMatcher"* ref=*"credentialsMatcher"*></property>  </bean>  <!-- 配置SecurityManager -->  <bean id=*"securityManager"* class=*"org.apache.shiro.web.mgt.DefaultWebSecurityManager"*>  <!-- 注入realm -->  <property name=*"realm"* ref=*"userRealm"*></property>  </bean>    <!-- 配置shiro的过滤器 这里面的id必须和web.xml里面的配置一样 -->  <bean id=*"shiroFilter"* class=*"org.apache.shiro.spring.web.ShiroFilterFactoryBean"* >  <!-- 注入安全管理器 -->  <property name=*"securityManager"* ref=*"securityManager"*></property>  <!-- 注入未登陆的跳转页面 默认的是webapp/login.jsp-->  <property name=*"loginUrl"* value=*"/index.jsp"*></property>  <!-- 注入未授权的访问页面 -->  <property name=*"unauthorizedUrl"* value=*"/unauthorized.jsp"*></property>  <!-- 配置过滤器链 -->  <property name=*"filterChainDefinitions"*>  <value>  <!-- 放行index.jsp -->  /index.jsp\*=anon  <!-- 放行跳转到登陆页面的路径 -->  /login/toLogin\*=anon  <!-- 放行登陆的请求 -->  /login/login\*=anon  <!-- 设置登出的路径 -->  /login/logout\*=logout  <!-- 设置其它路径全部拦截 -->  /\*\*=authc  </value>  </property>    </bean>  </beans> |

### 修改applicationContext.xml



### 修改web.xml

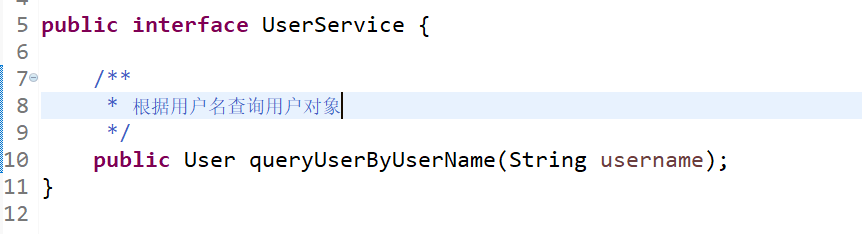
|  |
| --- |
| <!-- 配置shiro的集成开始 -->  <filter>  <filter-name>shiroFilter</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  <init-param>  <param-name>targetFilterLifecycle</param-name>  <param-value>true</param-value>  </init-param>  <init-param>  <!-- 这里面的shiroFilter必须和application-shiro.xml里面的  <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean" >id 一样 -->  <param-name>targetBeanName</param-name>  <param-value>shiroFilter</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>shiroFilter</filter-name>  <servlet-name>springmvc</servlet-name>  </filter-mapping>  <!-- 配置shiro的集成结束 --> |

## 使用shiro完成用户权限的开发

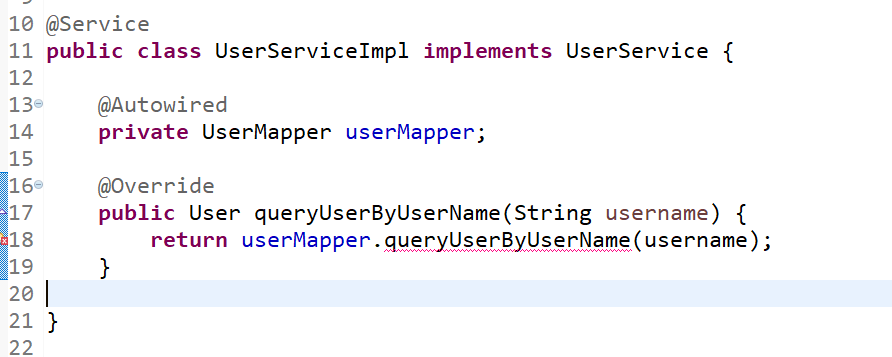
### 修改UserRealm

|  |
| --- |
| **public** **class** UserRealm **extends** AuthorizingRealm {  @Autowired  **private** UserService userService;  @Autowired  **private** RoleService roleService;  @Autowired  **private** PermissionService permissionService;      @Override  **public** String getName() {  **return** **this**.getClass().getSimpleName();  }  /\*\*  \* 认证  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  String username = token.getPrincipal().toString();  // 根据用户名查询用户  User user = **this**.userService.queryUserByUserName(username);  **if** (**null** != user) {  //查询角色  List<String> roles = **this**.roleService.queryRolesByUserId(user.getUserid());  //查询权限  List<String> permissions = **this**.permissionService.queryPermissionByUserId(user.getUserid());  //构造ActiverUser  ActivierUser activierUser=**new** ActivierUser(user, roles, permissions);  //创建盐  ByteSource credentialsSalt=ByteSource.Util.*bytes*(user.getUsername()+user.getAddress());  SimpleAuthenticationInfo info=**new** SimpleAuthenticationInfo(activierUser, user.getUserpwd(), credentialsSalt, **this**.getName());  **return** info;  } **else** {  **return** **null**;  }  }  /\*\*  \* 授权  \*/  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  ActivierUser activierUser = (ActivierUser) principals.getPrimaryPrincipal();  SimpleAuthorizationInfo info=**new** SimpleAuthorizationInfo();    List<String> roles = activierUser.getRoles();  List<String> permissions = activierUser.getPermissions();  **if**(**null**!=roles&&roles.size()>0) {  info.addRoles(roles);  }  **if**(**null**!=permissions&&permissions.size()>0) {  info.addStringPermissions(permissions);  }  **return** info;  }  } |

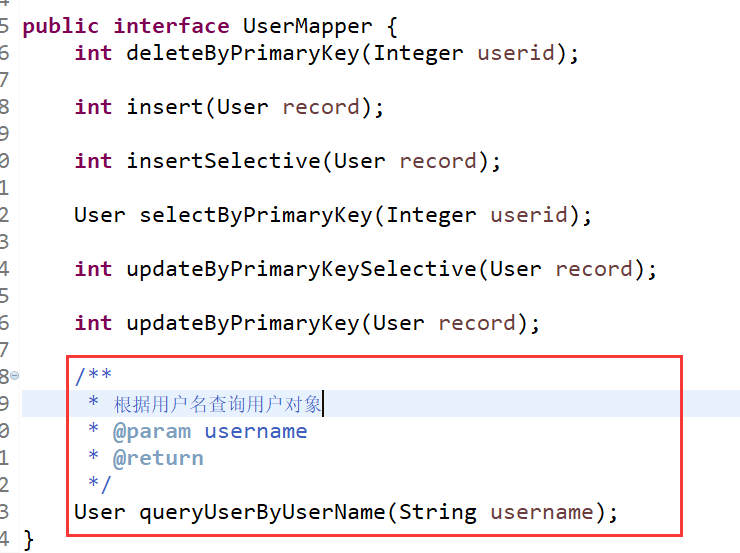
### 创建UserService



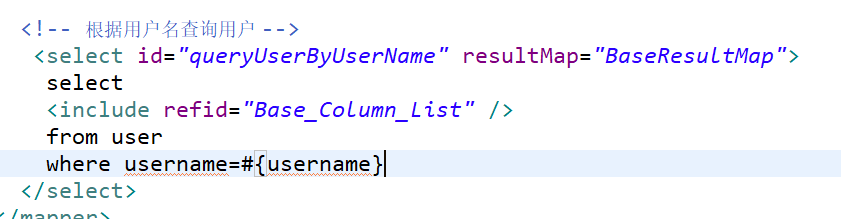
### 创建UserServiceImpl



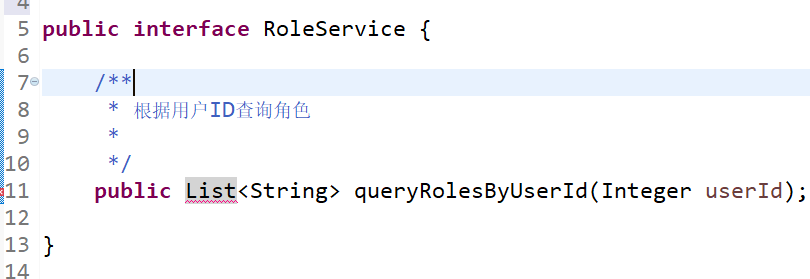
### 修改UserMapper



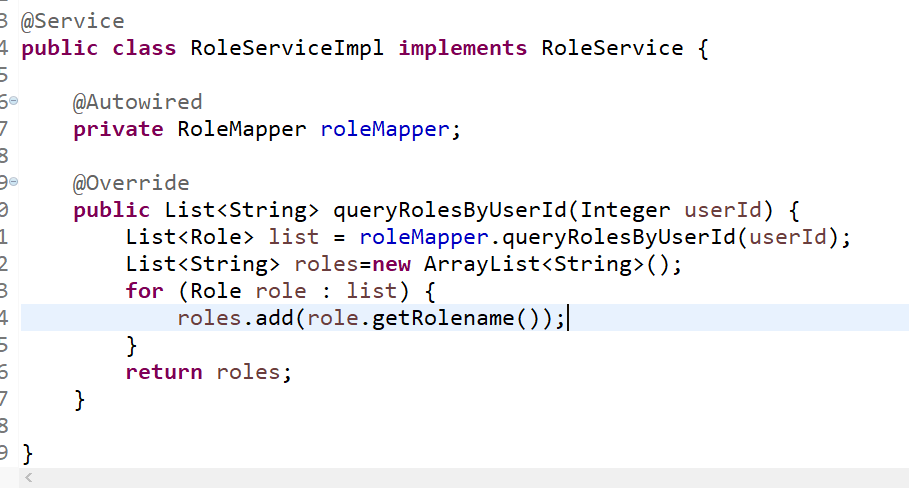
### 修改UserMapper.xml



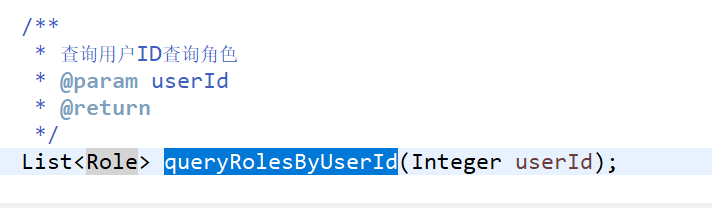
### 创建RoleService



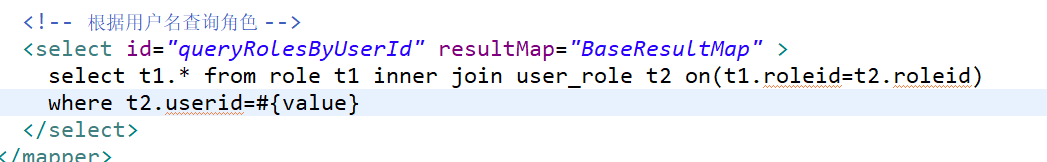
### 创建RoleServiceImpl



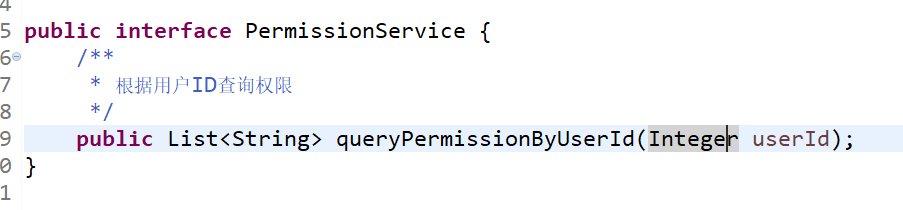
### 修改RoleMapper



### 修改RoleMapper.xml



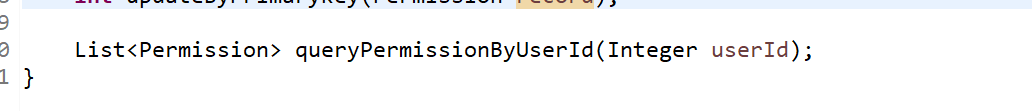
### 创建PermissionService



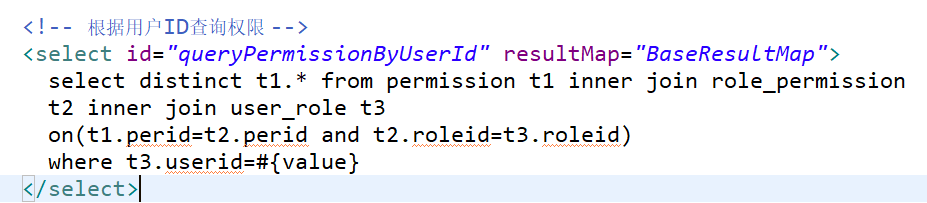
### 创建PermissionServiceImpl



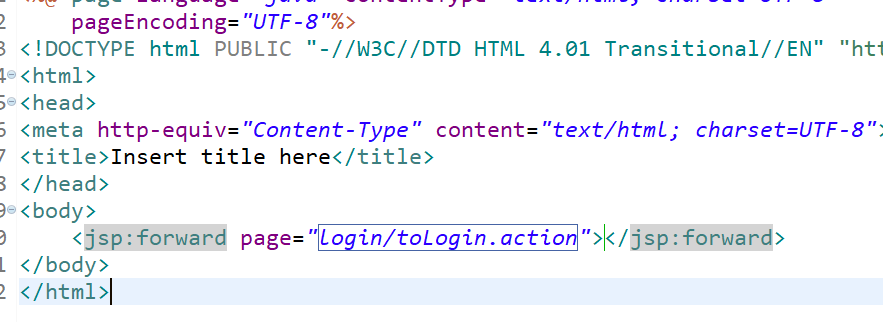
### 修改PermissionMapper



### 修改PermissionMapper.xml



### 创建index.jsp



### 创建WEB-INF/view/login.jsp



### 创建LoginController

|  |
| --- |
| /\*\*  \* 用户登陆控制器  \* **@author** LJH  \*  \*/  @RequestMapping("login")  @Controller  **public** **class** LoginController {    /\*\*  \* 跳转到登陆页面  \*/  @RequestMapping("toLogin")  **public** String toLogin() {  **return** "login";  }      /\*\*  \* 做登陆  \*/  @RequestMapping("login")  **public** String login(String username,String pwd,HttpSession session) {  //得到主体  Subject subject = SecurityUtils.*getSubject*();  UsernamePasswordToken token=**new** UsernamePasswordToken(username, pwd);  **try** {  subject.login(token);  System.***out***.println("登陆成功");  ActivierUser activierUser = (ActivierUser) subject.getPrincipal();  session.setAttribute("user", activierUser.getUser());  **return** "redirect:/user/toUserManager.action";  } **catch** (AuthenticationException e) {  e.printStackTrace();  **return** "redirect:/index.jsp";  }  }  } |

### 创建UserController

|  |
| --- |
| @Controller  @RequestMapping("user")  **public** **class** UserController {  /\*\*  \* 跳转到用户管理的页面  \*/  @RequestMapping("toUserManager")  **public** String toUserManager() {  **return** "list";  }  @RequestMapping("query")  @ResponseBody  **public** Map<String,Object> query(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "query");  **return** map;  }  @RequestMapping("add")  @ResponseBody  **public** Map<String,Object> add(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "add");  **return** map;  }  @RequestMapping("update")  @ResponseBody  **public** Map<String,Object> update(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "update");  **return** map;  }  @RequestMapping("delete")  @ResponseBody  **public** Map<String,Object> delete(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "delete");  **return** map;  }  @RequestMapping("export")  @ResponseBody  **public** Map<String,Object> export(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "export");  **return** map;  }  } |

### 前台jsp页面

<%@ **page** language="java" contentType="text/html; charset=UTF-8"  
 pageEncoding="UTF-8"%>  
<%@**taglib** prefix="shiro" uri="http://shiro.apache.org/tags" %>

<%--加上标签--%>  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
<title>Insert title here</title>  
</head>  
<body>  
 <**shiro:hasPermission** name="user:query">

<h1><a href="user/query.action">查询用户</a></h1>  
 </**shiro:hasPermission**>

<%--判断是否有这个权限,有才会在页面显示--%>  
</body>  
</html>

# 09【掌握】ssm+shrio前后端分离的使用方法

## 什么是前后端分离

后端只给前端提供数据接口

## 接文档的说明【以登陆为例】

接口名:用户登陆

接口地址: <http://127.0.0.1:8080/login/login.action>

参数说明

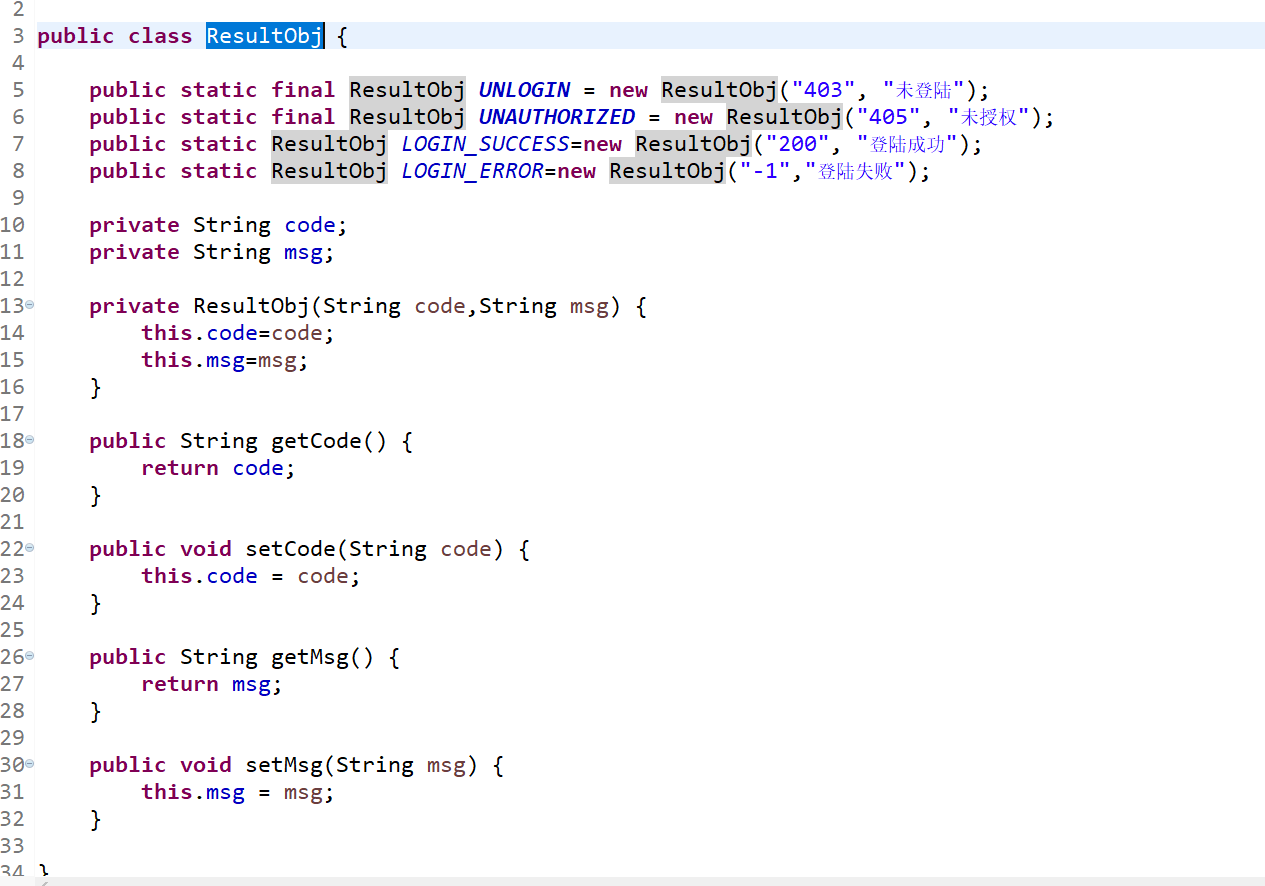
|  |  |  |
| --- | --- | --- |
| 参数名 | 参数说明 | 要求 |
| Username | 登陆名 | 必填 |
| Pwd | 登陆密码 | 必填 |

返回示例

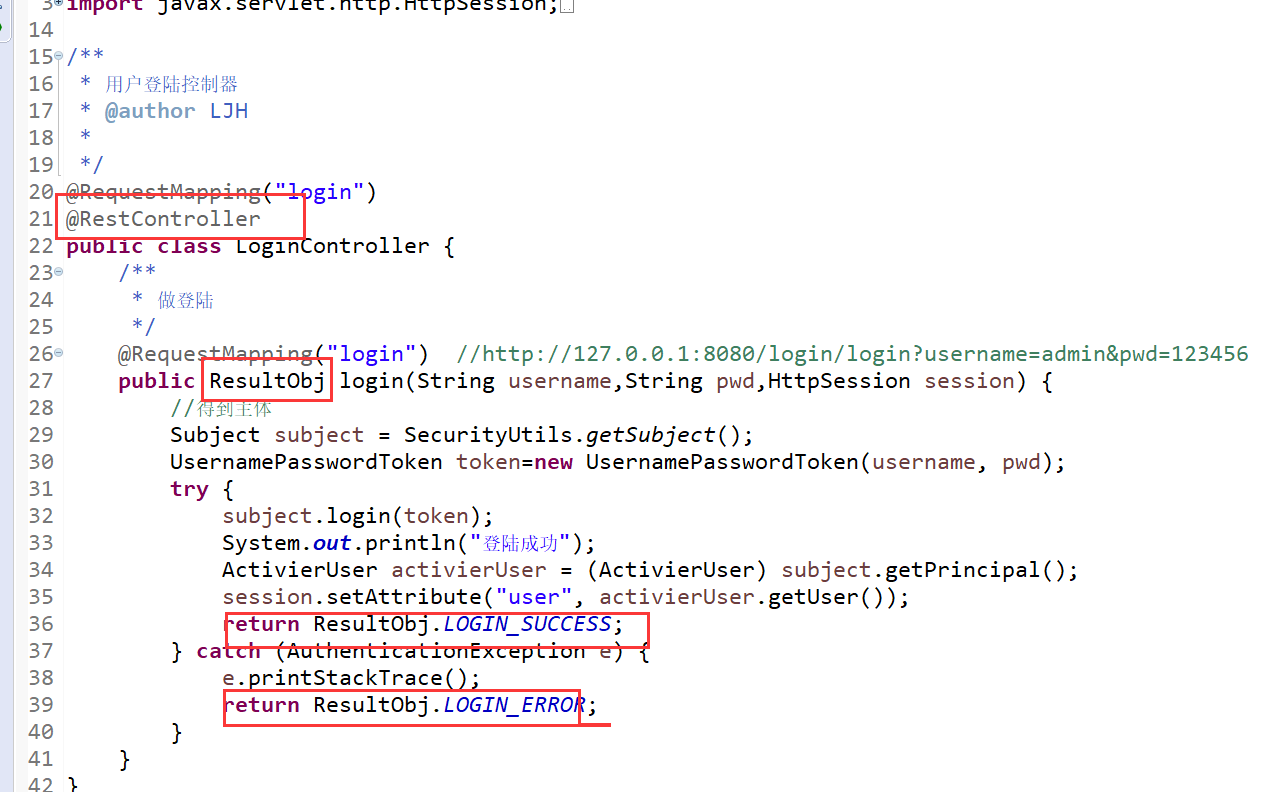
|  |
| --- |
| {  “code”：200，  “msg”:”登陆成功”  } |

## 复制项目

## 创建ResultObj



## 修改loginController



## 创建ShiroLoginFilter[解决未登陆时返回json]



## 修改UserController

|  |
| --- |
| @Controller  @RequestMapping("user")  **public** **class** UserController {  @RequiresPermissions(value="user:query")  @RequestMapping("query")  @ResponseBody  **public** Map<String,Object> query(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "query");  **return** map;  }  @RequiresPermissions(value="user:add")  @RequestMapping("add")  @ResponseBody  **public** Map<String,Object> add(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "add");  **return** map;  }    @RequiresPermissions(value="user:update")  @RequestMapping("update")  @ResponseBody  **public** Map<String,Object> update(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "update");  **return** map;  }  @RequiresPermissions(value="user:delete")  @RequestMapping("delete")  @ResponseBody  **public** Map<String,Object> delete(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "delete");  **return** map;  }  @RequiresPermissions(value= {"user:export"})  @RequestMapping("export")  @ResponseBody  **public** Map<String,Object> export(){  Map<String,Object> map=**new** HashMap<>();  map.put("msg", "export");  **return** map;  }  } |

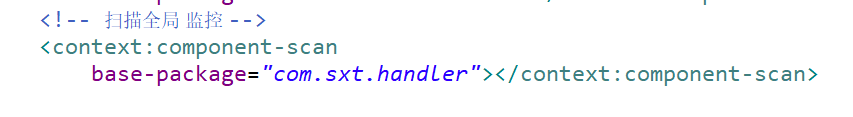
## 修改application-shrio.xml



## 创建GlobalExceptionHanderAdvise

|  |
| --- |
| /\*\*  \* 全局异常监控  \* **@author** LJH  \*/  @RestControllerAdvice  **public** **class** GlobalExceptionHanderAdvise {  /\*\*  \* 未授权  \* 只要当前项目的代码抛出UnauthorizedException就会回调  \*/  @ExceptionHandler(value= {UnauthorizedException.**class**})  **public** ResultObj unauthorized() {  **return** ResultObj.***UNAUTHORIZED***;  }  } |

## 修改springmvc.xml



|  |
| --- |
| <!-- 启动Shrio的注解 -->  <bean id=*"lifecycleBeanPostProcessor"*  class=*"org.apache.shiro.spring.LifecycleBeanPostProcessor"* />  <bean  class=*"org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"*  depends-on=*"lifecycleBeanPostProcessor"* />  <bean  class=*"org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor"*>  <property name=*"securityManager"* ref=*"securityManager"* />  </bean> |

## 测试

未登陆时调用 <http://127.0.0.1:8080/bjsxt/user/query.action>

|--显示 未登陆

使用张三已登陆 调用<http://127.0.0.1:8080/bjsxt/user/export.action>

|--显示 未授权

## 关于CAS

<https://www.jianshu.com/p/3c6680daab0e>

689370395