# 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。

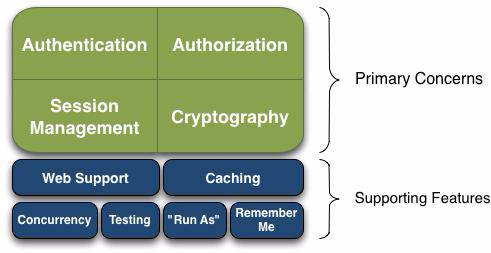
## 2，为什么要学shiro

        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 【**ɔː,θentɪ'keɪʃən/】

身份认证/登录，验证用户是不是拥有相应的身份；

**2， Authorization【/ɔːθəraɪ'zeɪʃ(ə)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【/krɪp'tɒgrəfɪ/】**

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【/kən'kɚrənsi】**

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**

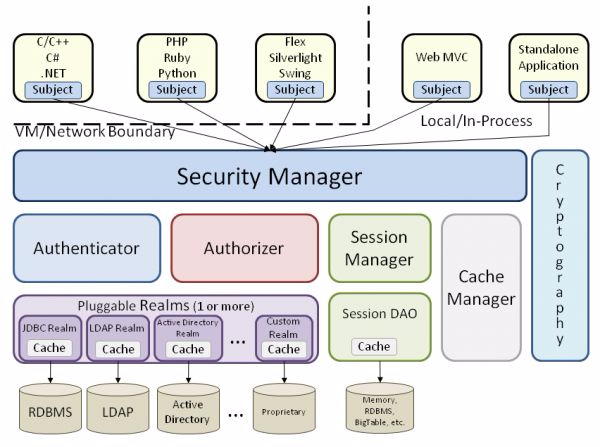
记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录

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**

## 4，架构说明



**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【/sɪ'kjʊərətɪ/】**

 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。

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

**3，Authenticator【/ɔ'θɛntɪ,ketɚ/】**

 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。

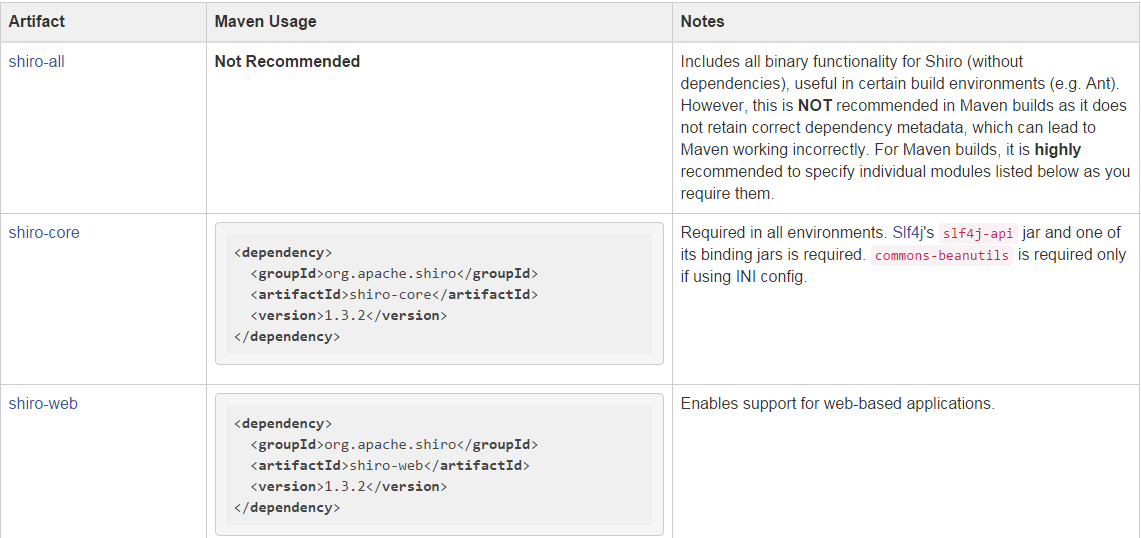
**9，Cryptography【/krɪp'tɒgrəfɪ/】**

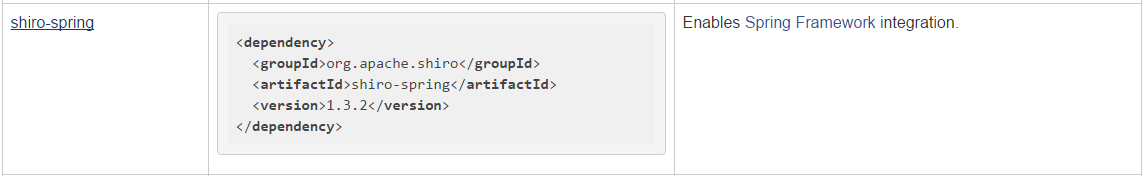
 Cryptography即密码管理，shiro提供了一套加密/解密的组件，方便开发。比如提供常用的散列、加/解密等功能。

## 5，下载Shiro和maven的依赖

<http://shiro.apache.org/download.html>







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

# 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 <property name=”属性名” value=””

myobj=com.bjsxt.Obj <bean id=”myobj” class=”com.bjsxt.Obj”

securityManager.对象属性=$myobj

<property name=”属性名” ref=” myobj”WW

### 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.

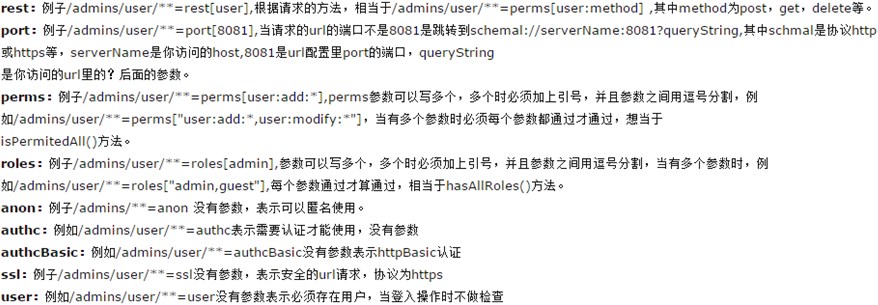
/abc=roles[“role1,role2”]

/login.html\*=anon

/loginOut\*=logout

/\*\*= authc 所有URL必须认证通过之后才能放行 一般放到最后

|  |  |  |
| --- | --- | --- |
| 过滤器名称 | 过滤器类 | 描述 |
| anon | [org.apache.shiro.web.filter.authc.AnonymousFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/AnonymousFilter.html) | 匿名过滤器 |
| authc | [org.apache.shiro.web.filter.authc.FormAuthenticationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/FormAuthenticationFilter.html) | 如果继续操作，需要做对应的表单验证否则不能通过 |
| authcBasic | [org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/BasicHttpAuthenticationFilter.html) | 基本http验证过滤，如果不通过，跳转屋登录页面 |
| logout | [org.apache.shiro.web.filter.authc.LogoutFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/LogoutFilter.html) | 登录退出过滤器 |
| noSessionCreation | [org.apache.shiro.web.filter.session.NoSessionCreationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/session/NoSessionCreationFilter.html) | 没有session创建过滤器 |
| perms | [org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/PermissionsAuthorizationFilter.html) | 权限过滤器 |
| port | [org.apache.shiro.web.filter.authz.PortFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/PortFilter.html) | 端口过滤器，可以设置是否是指定端口如果不是跳转到登录页面 |
| rest | [org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/HttpMethodPermissionFilter.html) | http方法过滤器，可以指定如post不能进行访问等 |
| roles | [org.apache.shiro.web.filter.authz.RolesAuthorizationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/RolesAuthorizationFilter.html) | 角色过滤器，判断当前用户是否指定角色 |
| ssl | [org.apache.shiro.web.filter.authz.SslFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/SslFilter.html) | 请求需要通过ssl，如果不是跳转回登录页 |
| user | [org.apache.shiro.web.filter.authc.UserFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/UserFilter.html) | 如果访问一个已知用户，比如记住我功能，走这个过滤器 |



# 03【掌握】Shiro实现认证

## 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 【/'prɪnsəpl】**

        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。

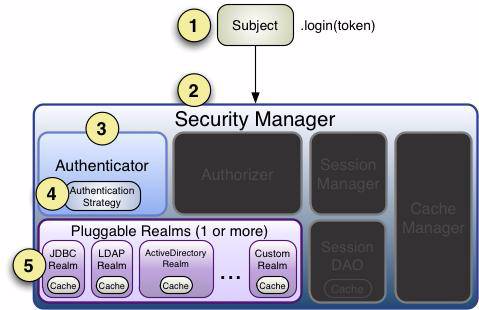
一个主体可以有多个principals，但只有一个Primary principals，一般是用户名/密码/手机号。

**3，credentials 【/krə'dɛnʃlz/】**

        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。

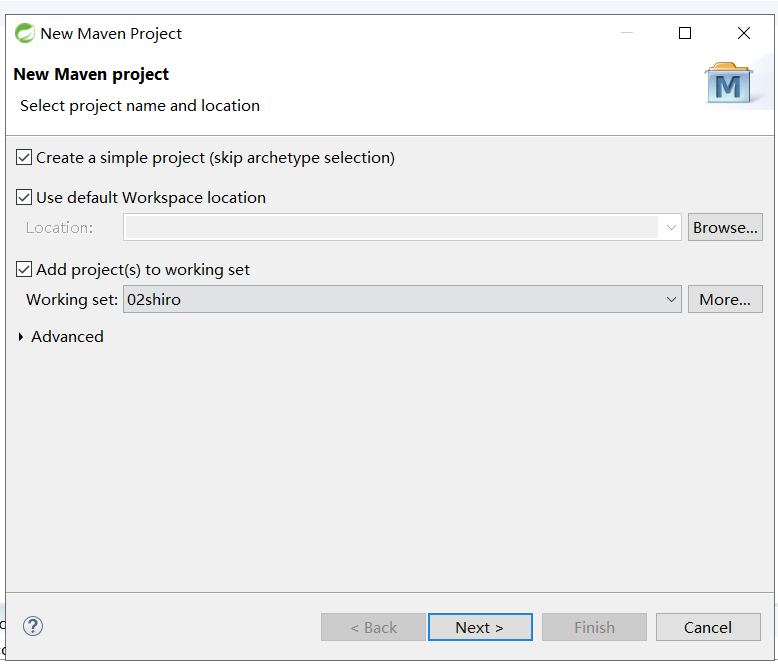
## 2，认证流程

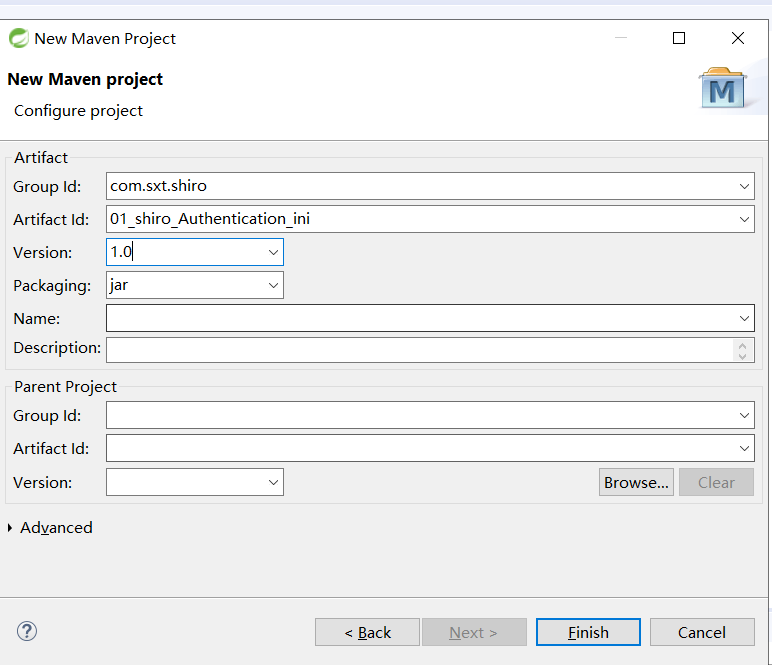


其它就是使用Shrio的认证来取代我们传统的登陆方式

## 3，入门程序实现认证

### 创建项目





### 创建shiro.ini

|  |
| --- |
| #配置用户  [users]  zhangsan=123456  lisi=123456  wangwu=1234567890 |

### 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/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>  <!-- 版本号 -->  <properties>  <shiro.version>1.4.1</shiro.version>  <logging.version>1.2</logging.version>  </properties>  <dependencies>  <!-- 依赖shiro -->  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-core</artifactId>  <version>${shiro.version}</version>  </dependency>    <dependency>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  <version>${logging.version}</version>  </dependency>  </dependencies>  </project> |

### 写代码测试

|  |
| --- |
| /\*\*  \* 用户认证  \* **@author** LJH  \*  \*/  **public** **class** TestApp {  @SuppressWarnings("deprecation")  **public** **static** **void** main(String[] args) {  String username="zhangsan1";  String password="123456";  //1,创建securityManager工厂 SecurityManager JDK也有这个类,在java.lang包 注意不要使用jdk里面那个类  Factory<SecurityManager> factory=**new** ~~IniSecurityManagerFactory~~("classpath:shiro.ini");  //2,从工厂里面得到SecurityManager  SecurityManager securityManager = factory.getInstance();  //3,把当前的SecurityManager绑定到当前线程  SecurityUtils.*setSecurityManager*(securityManager);  //4,取出当前的Subject  Subject subject = SecurityUtils.*getSubject*();  // System.out.println(subject);  //5,封装用户名和密码  UsernamePasswordToken token=**new** UsernamePasswordToken(username, password);  **try** {  //6,进行登陆认证  subject.login(token);  System.***out***.println("是否认证成功:"+subject.isAuthenticated());  } **catch** (AuthenticationException e) {  System.***out***.println("用户名或密码不正确");  }  /\*try {  //6,进行登陆认证  subject.login(token);  System.out.println("是否认证成功:"+subject.isAuthenticated());  } catch (IncorrectCredentialsException e) {  System.out.println("密码不正确");  } catch (UnknownAccountException e) {  System.out.println("用户名不存在");  }  //  \*/// a();  //  // b();  //  //  //// ThreadLocal<Subject> xxx=new ThreadLocal<>();  //// xxx.get();  //// xxx.set(value);  } |

### 相关方法总结

1. 注意Subject的或取方式
2. 注意使用异常方式确定登陆是否成功

# 04【掌握】shiro.ini实现授权

## 1，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）。

## 2，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。。。

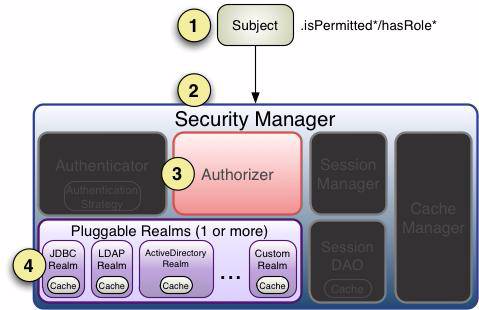
4，角色

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。

## 3，授权流程



## 4，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.

## 5，修改shiro.ini

|  |
| --- |
| #配置用户  [users]  zhangsan=123456,role1  lisi=123456,role2,role4  wangwu=123456,role3  zhaoliu=123456,role4  #配置角色和权限  [roles]  role1=user:query,user:add,user:update,user:delete,user:export  role2=user:query  role3=user:query,user:add  role4=user:query,user:export |

## 6，权限标识符号规则

权限标识符号规则：资源:操作:实例(中间使用半角:分隔)

user:create:01 表示对用户资源的01实例进行create操作。

user:create：表示对用户资源进行create操作，相当于user:create:\*，对所有用户资源实例进行create操作。

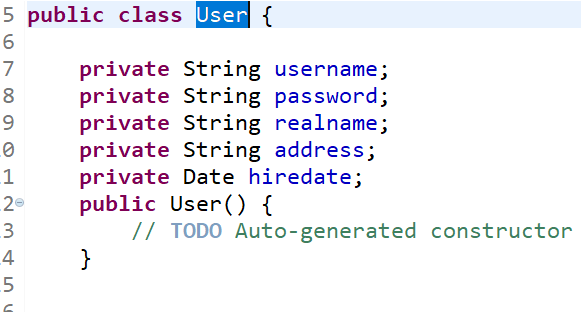
user:\*:01 表示对用户资源实例01进行所有操作。

## 7，测试代码

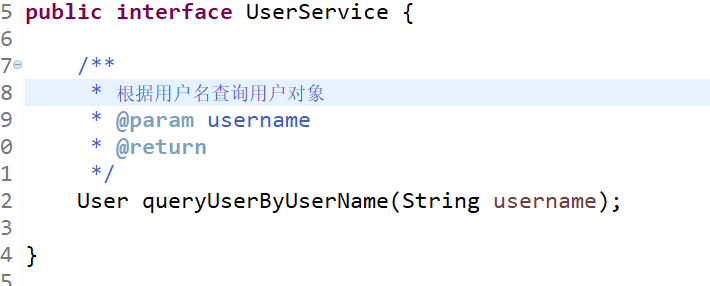
|  |
| --- |
| /\*\*  \* 用户认证授权  \* **@author** LJH  \*  \*/  **public** **class** TestApp {  @SuppressWarnings("deprecation")  **public** **static** **void** main(String[] args) {  String username="zhangsan";  String password="123456";  //1,创建securityManager工厂 SecurityManager JDK也有这个类,在java.lang包 注意不要使用jdk里面那个类  Factory<SecurityManager> factory=**new** ~~IniSecurityManagerFactory~~("classpath:shiro.ini");  //2,从工厂里面得到SecurityManager  SecurityManager securityManager = factory.getInstance();  //3,把当前的SecurityManager绑定到当前线程  SecurityUtils.*setSecurityManager*(securityManager);  //4,取出当前的Subject  Subject subject = SecurityUtils.*getSubject*();  // System.out.println(subject);  //5,封装用户名和密码  UsernamePasswordToken token=**new** UsernamePasswordToken(username, password);  **try** {  //6,进行登陆认证  subject.login(token);  System.***out***.println("是否认证成功:"+subject.isAuthenticated());  } **catch** (AuthenticationException e) {  System.***out***.println("用户名或密码不正确");  }    //1,判断用户是否有某一个角色  **boolean** hasRole1 = subject.hasRole("role1");  System.***out***.println(username+"是否有role1的角色:"+hasRole1);  //2,判断用户是否同时拥有集合里面的所有角色  List<String> roleIdentifiers=**new** ArrayList<>();  roleIdentifiers.add("role1");  roleIdentifiers.add("role2");  roleIdentifiers.add("role3");  **boolean** allRoles = subject.hasAllRoles(roleIdentifiers);  System.***out***.println(username+"是否同时拥有roleIdentifiers集合里面的所有角色:"+allRoles);  //3,分别判断用户是否有集合里面的角色  **boolean**[] hasRoles = subject.hasRoles(roleIdentifiers);  System.***out***.println("分别判断用户是否有集合里面的角色:"+Arrays.*toString*(hasRoles));        //4,判断用户是否有某一个权限  **boolean** permitted = subject.isPermitted("user:query");  System.***out***.println(username+"是否有user:query的权限:"+permitted);  //5,分别判断用户是否有数组里面的权限  String[] permissions= {"user:query","user:add","user:xxxx"};  **boolean**[] permitted2 = subject.isPermitted(permissions);  System.***out***.println("分别判断用户是否有数组里面的权限:"+Arrays.*toString*(permitted2));    //6,判断用户是否有数组里面的所有权限  **boolean** permittedAll = subject.isPermittedAll(permissions);  System.***out***.println("判断用户是否有数组里面的所有权限:"+permittedAll);    //7,  **boolean** permitted3 = subject.isPermitted("user:\*");  System.***out***.println(permitted3);                    }      } |

# 05【掌握】自定义Realm实现认证

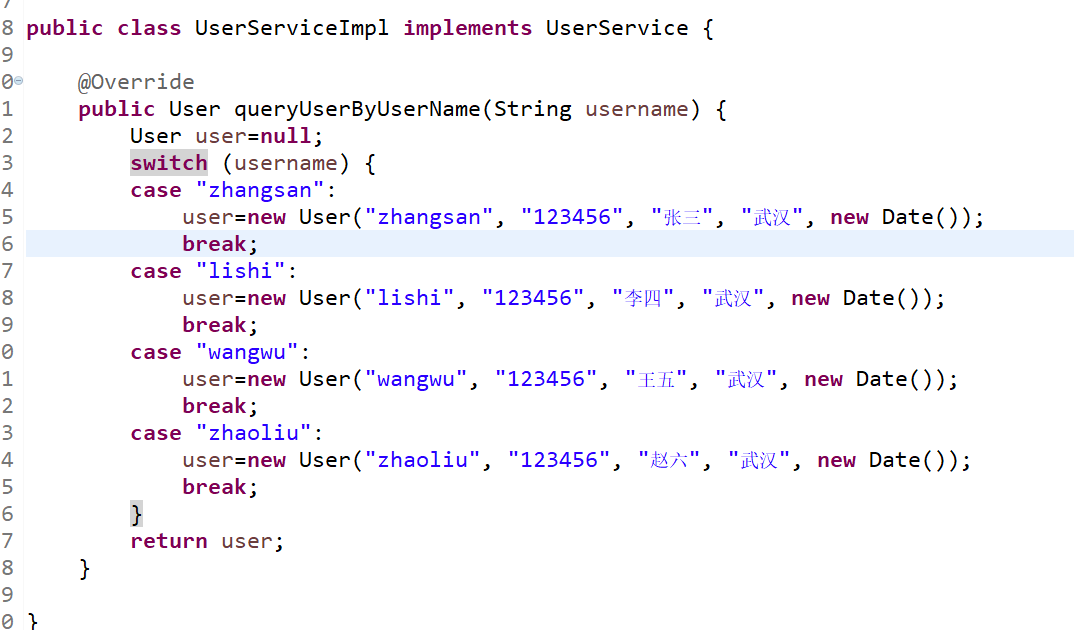
## 创建User



## 创建UserService



## 创建UserServiceImpl



## 创建UserRealm

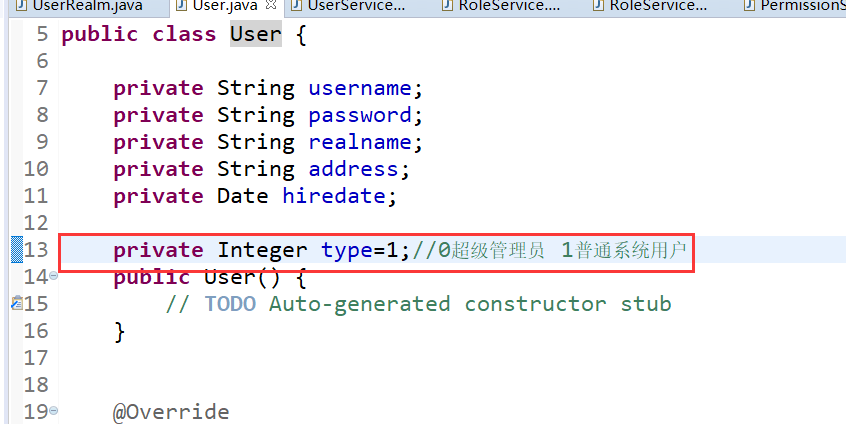
|  |
| --- |
| **public** **class** UserRealm **extends** AuthenticatingRealm{    @Override  **public** String getName() {  **return** **this**.getClass().getSimpleName();  }    **private** UserService userService=**new** UserServiceImpl();      /\*\*  \* 完成认证的方法  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  String username=token.getPrincipal().toString();  Object credentials = token.getCredentials();//用户登陆时传过来的  System.***out***.println(Arrays.*toString*((**char**[])credentials));  //根据用户名查询用户是否存在  User user=**this**.userService.queryUserByUserName(username);  //返回null说明用户不存在  **if**(**null**!=user) {  /\*\*  \* 参数1 用户身份  \* 参数2 用户在数据库里面存放的密码  \* 参数3 当前类名  \*/  SimpleAuthenticationInfo info=**new** SimpleAuthenticationInfo(user, user.getPassword(), **this**.getName());  // SimpleAuthenticationInfo info=new SimpleAuthenticationInfo(principal, hashedCredentials, credentialsSalt, realmName)    **return** info;    }  **return** **null**;  }  } |

## 创建TestApp测试

|  |
| --- |
| /\*\*  \* 用户认证授权  \* **@author** LJH  \*  \*/  **public** **class** TestApp {  @SuppressWarnings("deprecation")  **public** **static** **void** main(String[] args) {  String username="zhangsan";  String password="123456";  //1,创建securityManager工厂 SecurityManager JDK也有这个类,在java.lang包 注意不要使用jdk里面那个类  Factory<SecurityManager> factory=**new** ~~IniSecurityManagerFactory~~("classpath:shiro.ini");  //2,从工厂里面得到SecurityManager  DefaultSecurityManager securityManager = (DefaultSecurityManager) factory.getInstance();    //创建自定义realm 并注入到安全管理器里面  Realm realm=**new** UserRealm();  securityManager.setRealm(realm);    //3,把当前的SecurityManager绑定到当前线程  SecurityUtils.*setSecurityManager*(securityManager);  //4,取出当前的Subject  Subject subject = SecurityUtils.*getSubject*();  // System.out.println(subject);  //5,封装用户名和密码  UsernamePasswordToken token=**new** UsernamePasswordToken(username, password);  **try** {  //6,进行登陆认证  subject.login(token);  System.***out***.println("是否认证成功:"+subject.isAuthenticated());  Object object = subject.getPrincipal(); //得到doGetAuthenticationInfo的返回值SimpleAuthenticationInfo里面的第一个参数  System.***out***.println(object);    } **catch** (AuthenticationException e) {  System.***out***.println("用户名或密码不正确");  }    //1,判断用户是否有某一个角色  **boolean** hasRole1 = subject.hasRole("role1");  **boolean** hasRole3 = subject.hasRole("role1");  System.***out***.println(username+"是否有role1的角色:"+hasRole1);    }  } |

# 06【掌握】自定义Realm实现授权

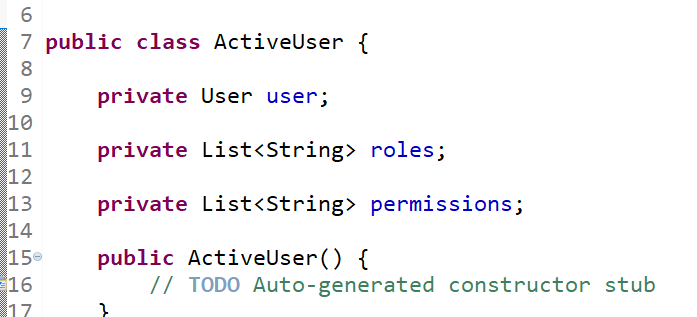
## 修改User



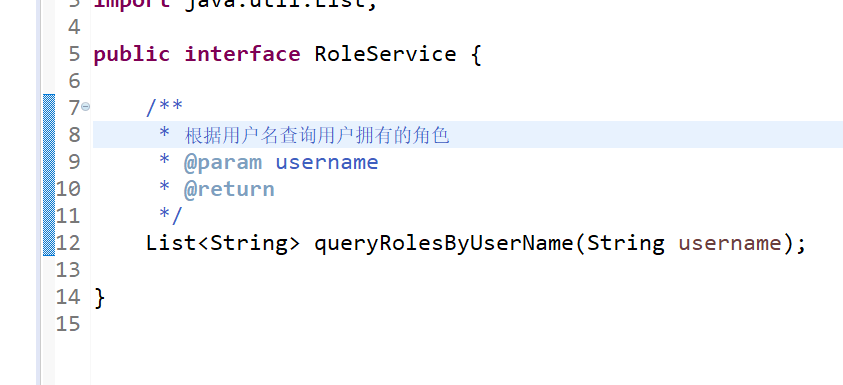
## 修改UserServiceImpl



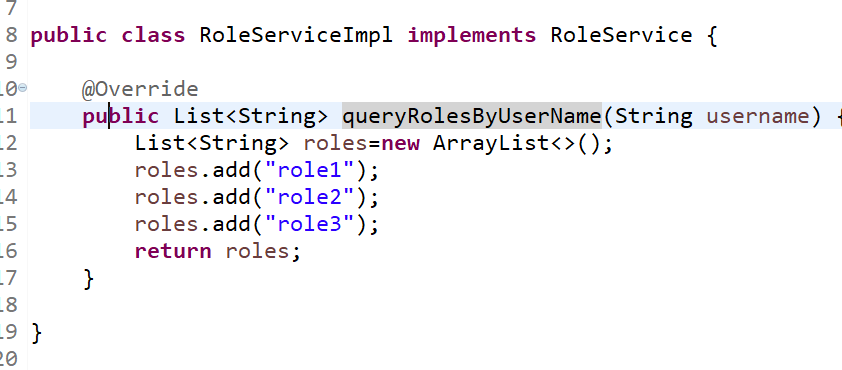
## 创建ActiverUser



## 创建RoleService



## 创建RoleServiceImpl



## 创建PermissionService



## 创建PermissionServiceImpl



## 修改UserRealm

|  |
| --- |
| **public** **class** UserRealm **extends** AuthorizingRealm{    @Override  **public** String getName() {  **return** **this**.getClass().getSimpleName();  }    **private** UserService userService=**new** UserServiceImpl();    **private** RoleService roleService=**new** RoleServiceImpl();    **private** PermissionService permssionService=**new** PermissionServiceImpl();      /\*\*  \* 完成认证的方法  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  String username=token.getPrincipal().toString();  Object credentials = token.getCredentials();//用户登陆时传过来的  System.***out***.println(Arrays.*toString*((**char**[])credentials));  //根据用户名查询用户是否存在  User user=**this**.userService.queryUserByUserName(username);  //返回null说明用户不存在  **if**(**null**!=user) {  //根据用户名去查询用户拥有哪些角色  List<String> roles= roleService.queryRolesByUserName(user.getUsername());  //根据用户名查询用户拥有哪些权限  List<String> permissions=**this**.permssionService.queryPermissionsByUserName(user.getUsername());    ActiveUser activeUser=**new** ActiveUser(user, roles, permissions);  /\*\*  \* 参数1 用户身份  \* 参数2 用户在数据库里面存放的密码  \* 参数3 当前类名  \*/  SimpleAuthenticationInfo info=**new** SimpleAuthenticationInfo(activeUser, user.getPassword(), **this**.getName());  // SimpleAuthenticationInfo info=new SimpleAuthenticationInfo(principal, hashedCredentials, credentialsSalt, realmName)    **return** info;    }  **return** **null**;  }  /\*\*  \* 授权  \*/  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {    ActiveUser activeUser = (ActiveUser) principals.getPrimaryPrincipal();  System.***out***.println("doGetAuthorizationInfo");  SimpleAuthorizationInfo info=**new** SimpleAuthorizationInfo();  **if**(activeUser.getUser().getType()==0) {  info.addStringPermission("\*:\*");  }**else** {  //根据用户名去查询用户拥有哪些角色  List<String> roles= activeUser.getRoles();  **if**(**null**!=roles&&roles.size()>0) {  //添加角色  info.addRoles(roles);  }  //根据用户名查询用户拥有哪些权限  List<String> permissions=activeUser.getPermissions();  //添加权限  **if**(**null**!=permissions&&permissions.size()>0) {  //添加角色  info.addStringPermissions(permissions);  }  }  **return** info;  }  } |

# 07【掌握】散列算法+凭证配置

## 散列算法[就是加密的算法]

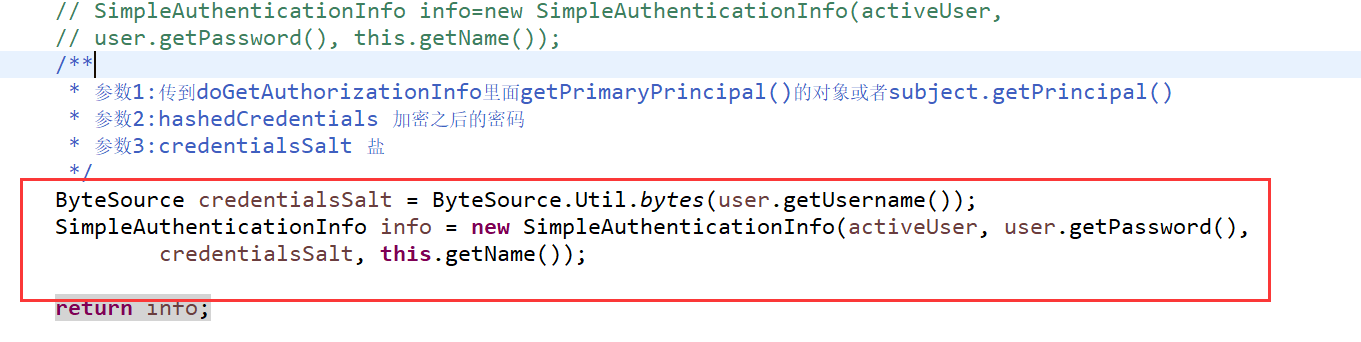
MD5

SHA1

|  |
| --- |
| **public** **class** MD5Utils {    // public static void main(String[] args) {  // String source="123456";//明文  // //1,使用MD5加密一次  // Md5Hash hash1=new Md5Hash(source);  // System.out.println("使用MD5加密一次："+hash1);  // //2,使用MD5加密两次  // Md5Hash hash2=new Md5Hash(hash1.toString());  // System.out.println("使用MD5加密二次："+hash2);  // //3,使用MD5加密一次并加盐  // Md5Hash hash3=new Md5Hash(source, "武汉尚学堂");  // System.out.println("使用MD5加密一次并加盐:"+hash3);  // //4,使用MD5加密二次并加盐  // Md5Hash hash4=new Md5Hash(source, "武汉尚学堂",2);  // System.out.println("使用MD5加密二次并加盐"+hash4);  // }    **public** **static** **void** main(String[] args) {  System.***out***.println(*md5*("123456","zhaoliu",2));  }    /\*\*  \*  \* **@param** source明文  \* **@param** salt 盐  \* **@param** hashIterations 散列次数  \* **@return**  \*/  **public** **static** String md5(Object source, Object salt,Integer hashIterations) {  **return** **new** Md5Hash(source, salt, hashIterations).toString();  }    } |
|  |

## 凭证配置

### 修改UserRealm



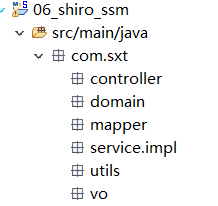
### 修改TestApp

|  |
| --- |
| @SuppressWarnings("deprecation")  **public** **static** **void** main(String[] args) {  String username="zhangsan";  String password="123456";  //1,创建securityManager工厂 SecurityManager JDK也有这个类,在java.lang包 注意不要使用jdk里面那个类  Factory<SecurityManager> factory=**new** ~~IniSecurityManagerFactory~~("classpath:shiro.ini");  //2,从工厂里面得到SecurityManager  DefaultSecurityManager securityManager = (DefaultSecurityManager) factory.getInstance();    //创建自定义realm 并注入到安全管理器里面  UserRealm realm=**new** UserRealm();//<bean  //设置密码学相关的加密方式  HashedCredentialsMatcher credentialsMatcher =**new** HashedCredentialsMatcher();//<bean  //设置加密方式  credentialsMatcher.setHashAlgorithmName("md5"); //<property  //设置散列次数  credentialsMatcher.setHashIterations(2);//<property  //给自定义realm注入凭证匹配器  realm.setCredentialsMatcher(credentialsMatcher);//<property  securityManager.setRealm(realm);//<property    //3,把当前的SecurityManager绑定到当前线程  SecurityUtils.*setSecurityManager*(securityManager);  //4,取出当前的Subject  Subject subject = SecurityUtils.*getSubject*();  // System.out.println(subject);  //5,封装用户名和密码  UsernamePasswordToken token=**new** UsernamePasswordToken(username, password);  **try** {  //6,进行登陆认证  subject.login(token);  System.***out***.println("是否认证成功:"+subject.isAuthenticated());  Object object = subject.getPrincipal(); //得到doGetAuthenticationInfo的返回值SimpleAuthenticationInfo里面的第一个参数  System.***out***.println(object);    } **catch** (AuthenticationException e) {  System.***out***.println("用户名或密码不正确");  }    //1,判断用户是否有某一个角色  **boolean** hasRole1 = subject.hasRole("role1");  System.***out***.println(username+"是否有role1的角色:"+hasRole1);  List<String> roleIdentifiers=**new** ArrayList<>();  roleIdentifiers.add("role1");  roleIdentifiers.add("role2");  roleIdentifiers.add("role3");  **boolean** allRoles = subject.hasAllRoles(roleIdentifiers);  System.***out***.println(username+"是否同时拥有roleIdentifiers集合里面的所有角色:"+allRoles);  //3,分别判断用户是否有集合里面的角色  **boolean**[] hasRoles = subject.hasRoles(roleIdentifiers);  System.***out***.println("分别判断用户是否有集合里面的角色:"+Arrays.*toString*(hasRoles));        //4,判断用户是否有某一个权限  **boolean** permitted = subject.isPermitted("user:query");  System.***out***.println(username+"是否有user:query的权限:"+permitted);  //5,分别判断用户是否有数组里面的权限  String[] permissions= {"user:query","user:add","user:xxxx"};  **boolean**[] permitted2 = subject.isPermitted(permissions);  System.***out***.println("分别判断用户是否有数组里面的权限:"+Arrays.*toString*(permitted2));    //6,判断用户是否有数组里面的所有权限  **boolean** permittedAll = subject.isPermittedAll(permissions);  System.***out***.println("判断用户是否有数组里面的所有权限:"+permittedAll);    //7,  **boolean** permitted3 = subject.isPermitted("user:xxx");  System.***out***.println(permitted3);  }  } |

# 08【掌握】ssm+shiro+Realm实现集成

## 搭建SSM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\_shiro\_ssm</artifactId>  <packaging>war</packaging>  <version>1.0</version>  <name>06\_shiro\_ssm Maven Webapp</name>  <url>http://maven.apache.org</url>  <!-- jar包版本声明 -->  <properties>  <servlet.version>3.1.0</servlet.version>  <jsp.version>2.3.1</jsp.version>  <jstl.version>1.1.2</jstl.version>  <mybatis.version>3.5.2</mybatis.version>  <mybatis-spring.version>1.3.3</mybatis-spring.version>  <spring.version>4.3.24.RELEASE</spring.version>  <druid.version>1.1.20</druid.version>  <mysql.version>5.1.47</mysql.version>  <jackson.version>2.9.9</jackson.version>  <pagehelper.version>5.1.8</pagehelper.version>  <!-- 注意只能使用2.0以下的版本 -->  <log4j.version>1.2.17</log4j.version>  <collection4.version>4.3</collection4.version>  </properties>  <dependencies>  <!-- 导入jsp -->  <dependency>  <groupId>javax.servlet.jsp</groupId>  <artifactId>javax.servlet.jsp-api</artifactId>  <version>${jsp.version}</version>  <scope>provided</scope>  </dependency>  <!-- 导入servlet -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>${servlet.version}</version>  <scope>provided</scope>  </dependency>  <!-- 添加jstl的依赖 -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  <version>${jstl.version}</version>  </dependency>  <dependency>  <groupId>taglibs</groupId>  <artifactId>standard</artifactId>  <version>${jstl.version}</version>  </dependency>  <!-- pagepelper -->  <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>${pagehelper.version}</version>  </dependency>  <!-- mybatis -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <!-- mybatis-spring -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${mybatis-spring.version}</version>  </dependency>  <!-- 导入spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <!-- springmvc -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <!-- mysql数据库驱动 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <!-- druid -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>${druid.version}</version>  </dependency>  <!-- jackson -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>${jackson.version}</version>  </dependency>  <!-- log4j -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>${log4j.version}</version>  </dependency>  <!-- commons-collections4 -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-collections4</artifactId>  <version>${collection4.version}</version>  </dependency>  </dependencies>  <build>  <finalName>06\_shiro\_ssm</finalName>  <!-- 配置插件 -->  <plugins>  <plugin>  <!--maven的tomcat插件 -->  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  <configuration>  <uriEncoding>UTF-8</uriEncoding> <!--解决页面提交数据乱码问题 -->  <port>8080</port><!-- tomcat插件的请求端口 -->  <path>/bjsxt</path><!-- 项目的请求路径 -->  </configuration>  </plugin>  </plugins>  </build>  </project> |

### 创建db.properties

|  |
| --- |
| #  driverClassName=com.mysql.Driver  url=jdbc:mysql://127.0.0.1:3306/shiro?useUnicode=true&characterEncoding=utf8&useSSL=true  username=root  password=123456  #  initialSize=5  maxActive=20  minIdle=5  maxIdle=10  filters=stat,wall,log4j |

### 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-4.3.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controller -->  <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 id=*"viewResolver"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <!-- 配置前缀 -->  <property name=*"prefix"* value=*"/WEB-INF/view/"*></property>  <!-- 配置后缀 -->  <property name=*"suffix"* value=*".jsp"*></property>  </bean>    <!-- 配置文件上传的类 -->    <!-- 配置拦截器 -->    <!-- 配置静态资源放行 -->  <mvc:default-servlet-handler/>    </beans> |

### 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-4.3.xsd"*>  <!-- 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"*>  <!-- 注入相关属性 -->  <property name=*"driverClassName"* value=*"${driverClassName}"*></property>  <property name=*"url"* value=*"${url}"*></property>  <property name=*"username"* value=*"${username}"*></property>  <property name=*"password"* value=*"${password}"*></property>  <property name=*"initialSize"* value=*"${initialSize}"*></property>  <property name=*"maxActive"* value=*"${maxActive}"*></property>  <property name=*"maxIdle"* value=*"${maxIdle}"*></property>  <property name=*"minIdle"* value=*"${minIdle}"*></property>  <property name=*"filters"* value=*"${filters}"*></property>  </bean>  <!-- 声明mybatis的配置类 -->  <bean id=*"configuration"*  class=*"org.apache.ibatis.session.Configuration"*>  <property name=*"logImpl"*  value=*"org.apache.ibatis.logging.stdout.StdOutImpl"*></property>  </bean>  <!-- 配置sqlSessionFacotry -->  <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>  <!-- 配置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>  <!-- 配置扫描mapper接口的对象 -->  <bean 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> -->  <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-4.3.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-4.3.xsd"*>  <!-- 扫描servie -->  <context:component-scan base-package=*"com.sxt.service.impl"*></context:component-scan>      <!-- 声明事务管理器 -->  <bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- 配置事务的传播特性 -->  <tx:advice id=*"myAdvise"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <tx:method name=*"insert\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"add\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"save\*"* 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=*"get\*"* read-only=*"true"*/>  <tx:method name=*"\*"* read-only=*"true"*/>  </tx:attributes>  </tx:advice>    <!-- 配置事务切面 -->  <aop:config>  <aop:pointcut expression=*"execution(\* com.sxt.service.impl.\*.\*(..))"* id=*"pc"*/>  <!-- 织入 -->  <aop:advisor advice-ref=*"myAdvise"* pointcut-ref=*"pc"*/>  </aop:config>    </beans> |

### 创建applicationContext.xml

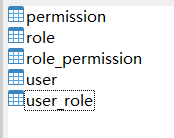
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>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

### 修改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\_shiro\_ssm</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>  <!-- 注入属性 -->  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>EncodingFilter</filter-name>  <!-- <url-pattern>/\*</url-pattern> -->  <servlet-name>springmvc</servlet-name>  </filter-mapping>  <!-- 配置编码过滤器结束 -->  <!-- 配置监听器加载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>  <!-- 配置启动加载 -->  <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.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  </web-app> |

## 集成shiro

### 创建数据库



permission 菜单和权限表

role 角色表

role\_permission 角色和权限的关系表

user 用户表

user\_role用户和角色之间的关系表

密码说明

使用登陆名和地址作为盐 散列2次

zhangsan user:query user:add user:update user:delete

lisi user:query user:add user:update

wangwu user:query user:export

### 修改pom.xml引入shiro包

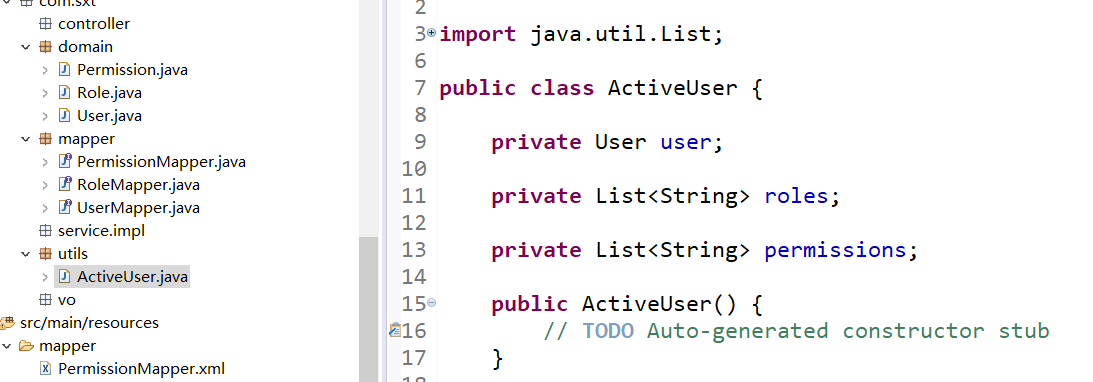
|  |
| --- |
| <shiro.version>1.4.1</shiro.version>  <!-- 依赖shiro -->  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-spring</artifactId>  <version>${shiro.version}</version>  </dependency> |

### 生成User UserMapper UserMapper.xml

### 生成Role RoleMapper RoleMapper.xml

### 生成Permission PermissionMapper PermissionMapper.xml

### 创建ActiverUser



### 创建UserRealm

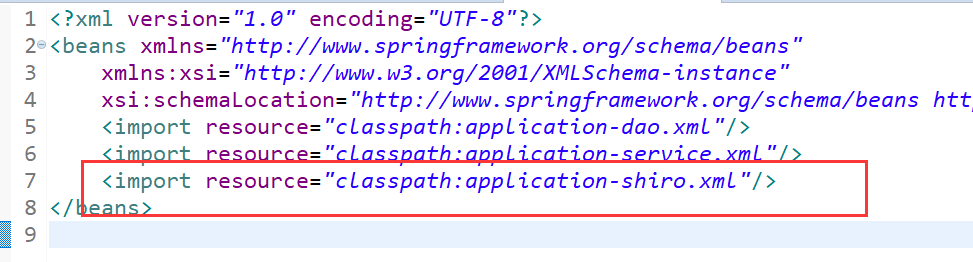
|  |
| --- |
| **public** **class** UserRealm **extends** AuthorizingRealm {  @Override  **public** String getName() {  **return** **this**.getClass().getSimpleName();  }  @Autowired  **private** UserService userService;  @Autowired  **private** RoleService roleService;  @Autowired  **private** PermissionService permssionService;  /\*\*  \* 完成认证的方法  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  String username = token.getPrincipal().toString();  Object credentials = token.getCredentials();// 用户登陆时传过来的  System.***out***.println(Arrays.*toString*((**char**[]) credentials));  // 根据用户名查询用户是否存在  User user = **this**.userService.queryUserByUserName(username);  // 返回null说明用户不存在  **if** (**null** != user) {  // 根据用户名去查询用户拥有哪些角色  List<String> roles = roleService.queryRolesByUserId(user.getUserid());  // 根据用户名查询用户拥有哪些权限  List<String> permissions = **this**.permssionService.queryPermissionsByUserId(user.getUserid());  ActiveUser activeUser = **new** ActiveUser(user, roles, permissions);  /\*\*  \* 参数1 用户身份 参数2 用户在数据库里面存放的密码 参数3 当前类名  \*/  // SimpleAuthenticationInfo info=new SimpleAuthenticationInfo(activeUser,  // user.getPassword(), this.getName());  /\*\*  \* 参数1:传到doGetAuthorizationInfo里面getPrimaryPrincipal()的对象或者subject.getPrincipal()  \* 参数2:hashedCredentials 加密之后的密码 参数3:credentialsSalt 盐  \*/  ByteSource credentialsSalt = ByteSource.Util.*bytes*(user.getUsername()+user.getAddress());  SimpleAuthenticationInfo info = **new** SimpleAuthenticationInfo(activeUser, user.getUserpwd(), credentialsSalt,  **this**.getName());  **return** info;  }  **return** **null**;  }  /\*\*  \* 授权  \*/  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  ActiveUser activeUser = (ActiveUser) principals.getPrimaryPrincipal();  SimpleAuthorizationInfo info = **new** SimpleAuthorizationInfo();  // 根据用户名去查询用户拥有哪些角色  List<String> roles = activeUser.getRoles();  **if** (**null** != roles && roles.size() > 0) {  // 添加角色  info.addRoles(roles);  }  // 根据用户名查询用户拥有哪些权限  List<String> permissions = activeUser.getPermissions();  // 添加权限  **if** (**null** != permissions && permissions.size() > 0) {  // 添加角色  info.addStringPermissions(permissions);  }  **return** info;  }  } |

### 修改web.xml

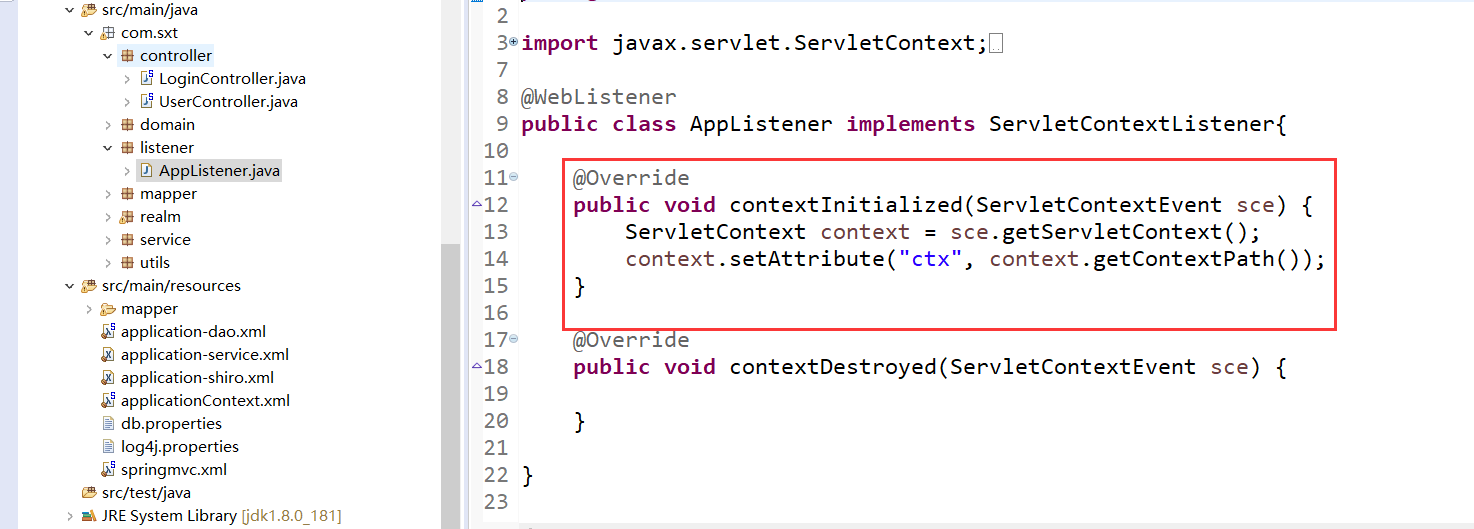
|  |
| --- |
| <!-- 配置shiro的代理过滤器 开始 -->  <filter>  <filter-name>shiroFilter</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  <init-param>  <param-name>targetFilterLifecycle</param-name>  <param-value>true</param-value>  </init-param>  <init-param>  <!-- 这里的shrioFilter必须和application-shrio.xml里面的 过滤器ID一致 -->  <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的代理过滤器 结束 --> |

### 创建application-shiro.xml

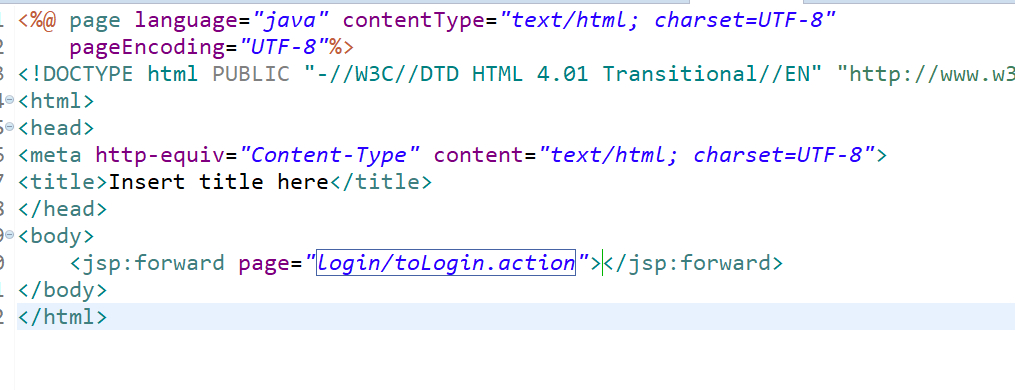
### 创建applicationContext.xml



### 创建AppListener



### 创建index.jsp



### 创建WEB-INF/view/login.jsp



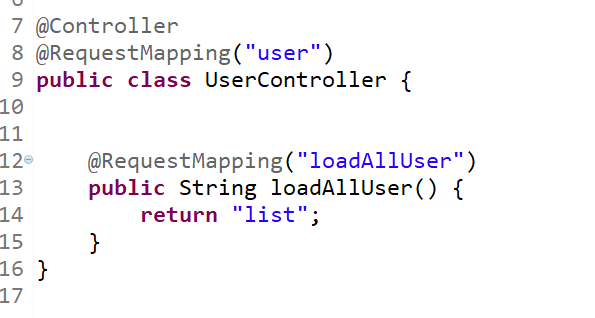
### 创建WEB-INF/view/list.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <%@taglib prefix=*"shiro"* uri=*"http://shiro.apache.org/tags"*%>  <!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"*>  <h2>  <a href=*""*>查询用户</a>  </h2>  </shiro:hasPermission>  <shiro:hasPermission name=*"user:add"*>  <h2>  <a href=*""*>添加用户</a>  </h2>  </shiro:hasPermission>  <shiro:hasPermission name=*"user:update"*>  <h2>  <a href=*""*>修改用户</a>  </h2>  </shiro:hasPermission>  <shiro:hasPermission name=*"user:delete"*>  <h2>  <a href=*""*>删除用户</a>  </h2>  </shiro:hasPermission>  <shiro:hasPermission name=*"user:export"*>  <h2>  <a href=*""*>导出用户</a>  </h2>  </shiro:hasPermission>  </body>  </html> |

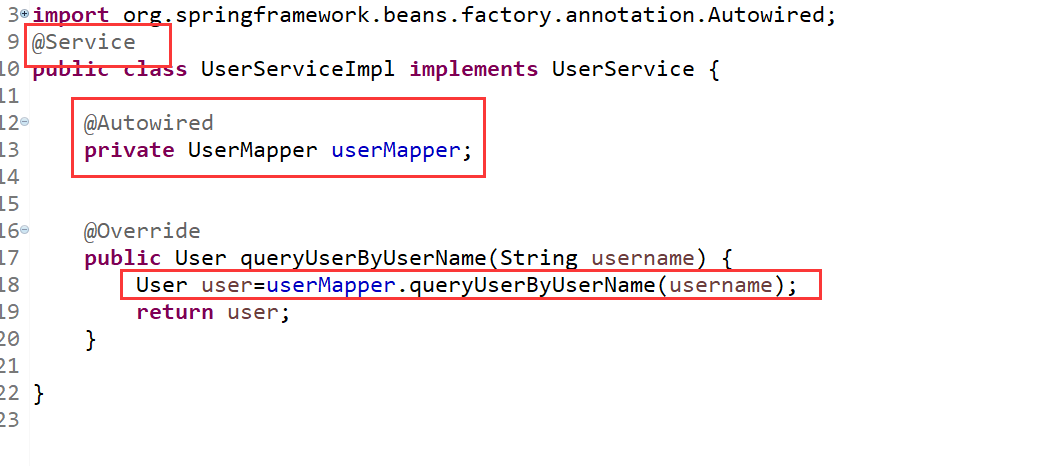
### 创建LoginController

|  |
| --- |
| @Controller  @RequestMapping("login")  **public** **class** LoginController {  /\*\*  \* 跳转到登陆页面  \*/  @RequestMapping("toLogin")  **public** String toLogin() {  **return** "login";  }  /\*\*  \* 完成登陆的方法  \*/  @RequestMapping("login")  **public** String login(String username, String password, HttpSession session) {  // 1，得到主体  Subject subject = SecurityUtils.*getSubject*();  // 2,封装用户名和密码  UsernamePasswordToken token = **new** UsernamePasswordToken(username, password);  **try** {  subject.login(token);  ActiveUser activerUser = (ActiveUser) subject.getPrincipal();  session.setAttribute("user", activerUser.getUser());  **return** "redirect:/user/loadAllUser.action";  } **catch** (AuthenticationException e) {  System.***out***.println("用户名或密码不正确");  }  **return** "redirect:index.jsp";  }  } |

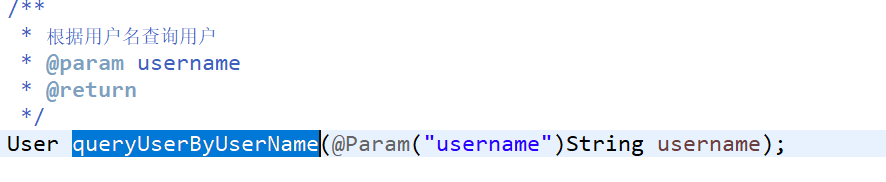
### 创建UserController



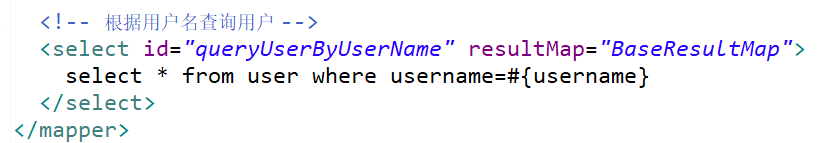
### 修改UserServiceImpl



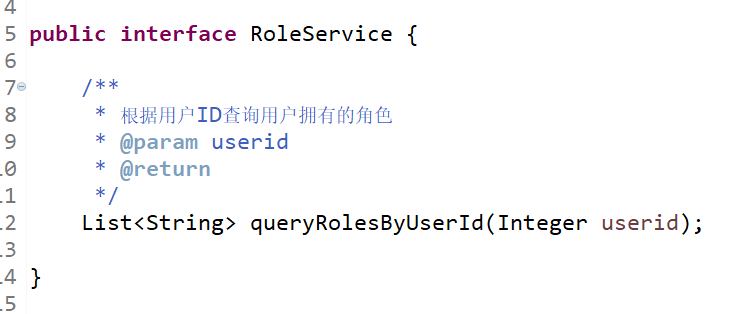
### 修改UserMapper



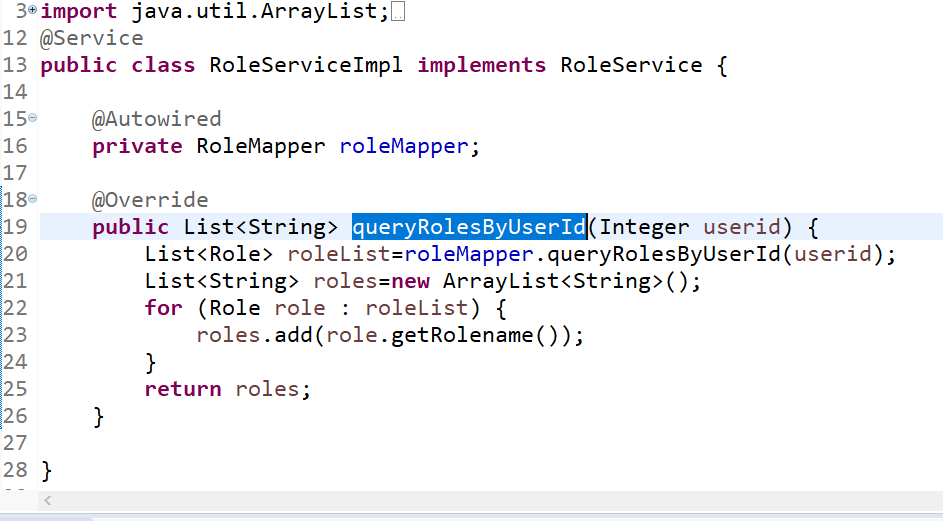
### 修改UserMapper.xml



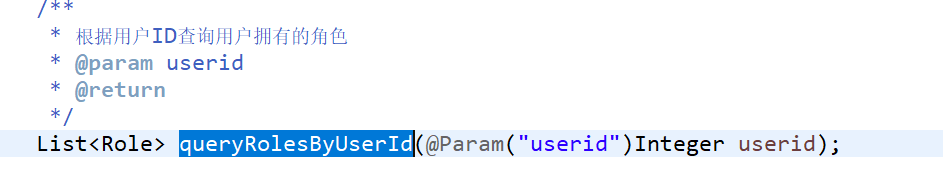
### 修改RoleService



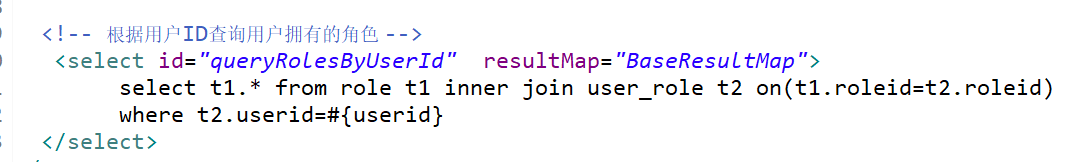
### 修改RoleServiceImpl



### 修改RoleMapper



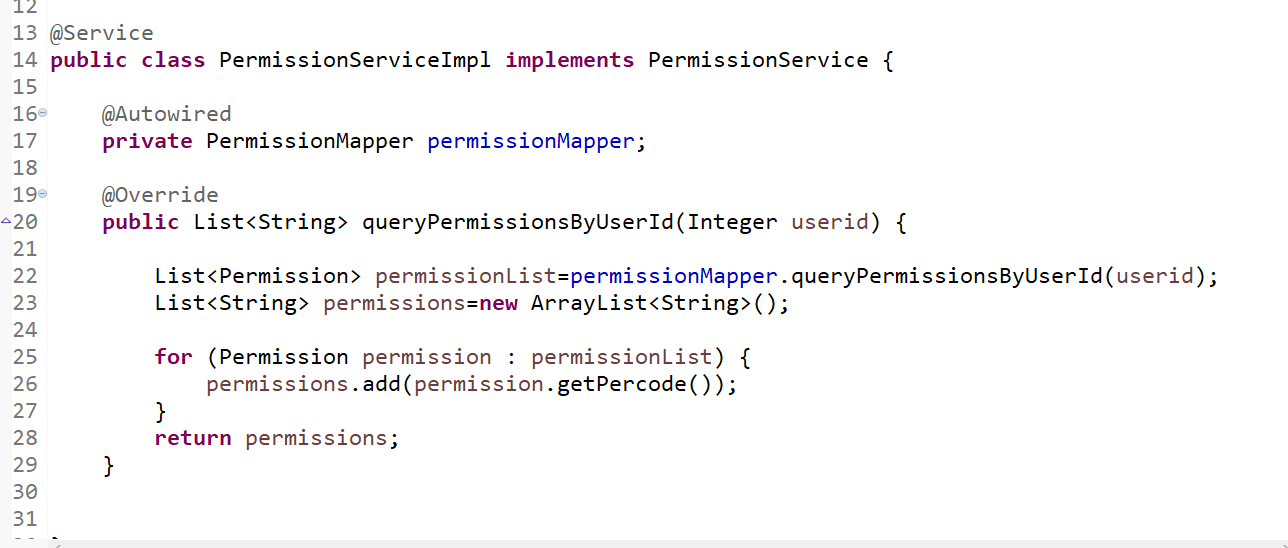
### 修改RoleMapper.xml



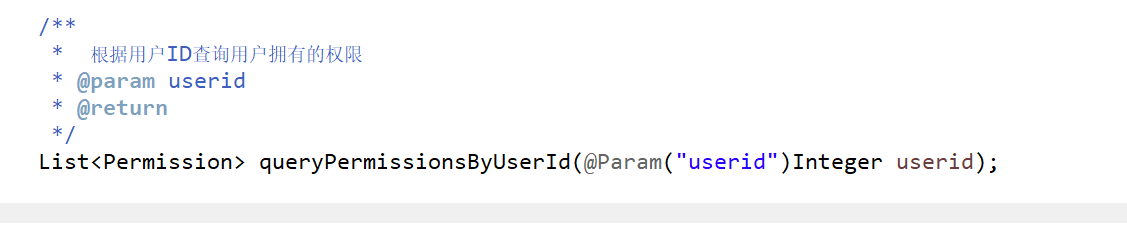
### 修改PermissionService



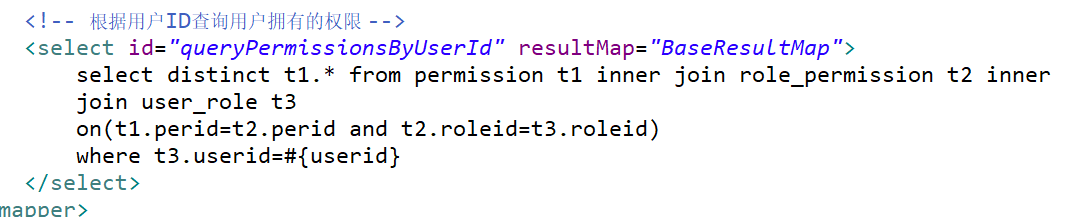
### 修改PermissionServiceImpl



### 修改PermissionMapper



### 修改PermissionMapper.xml



# 09【熟悉】记住我的功能

## 修改login.jsp



## 修改LoginController

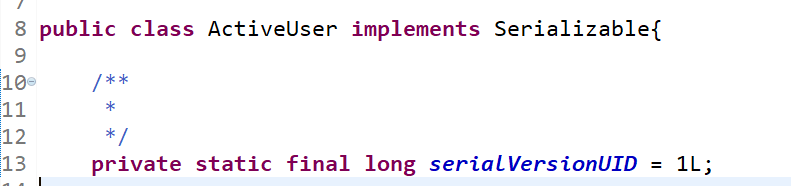


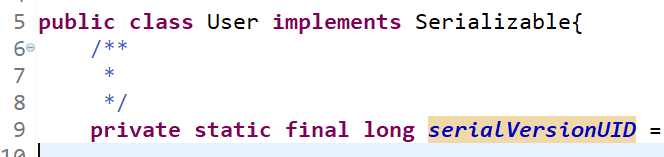
## 修改application-shiro.xml





## 序列化shiro使用到自定义类





## 测试

登陆并选中记住我

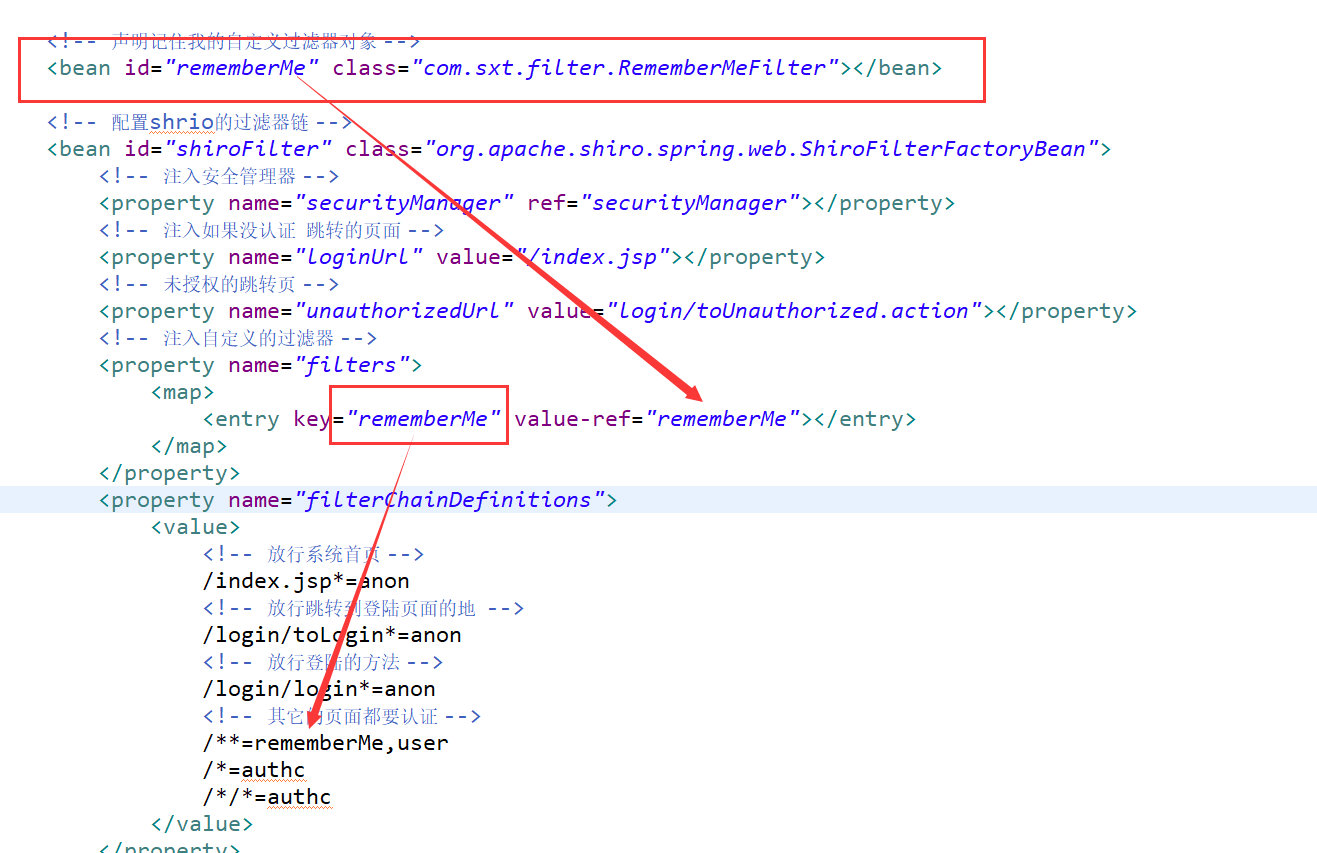
关闭浏览器直接访问要认证的地址

## 解决HttpSession丢失数据的问题

### 创建RmemeberMeFilter



### 修改application-shiro.xml



# 10【掌握】ssm+shrio前后端分离的使用方法

## 概述【什么叫前后端分离】

前台和后台是两个不同的项目，部署在不同的服务器上的

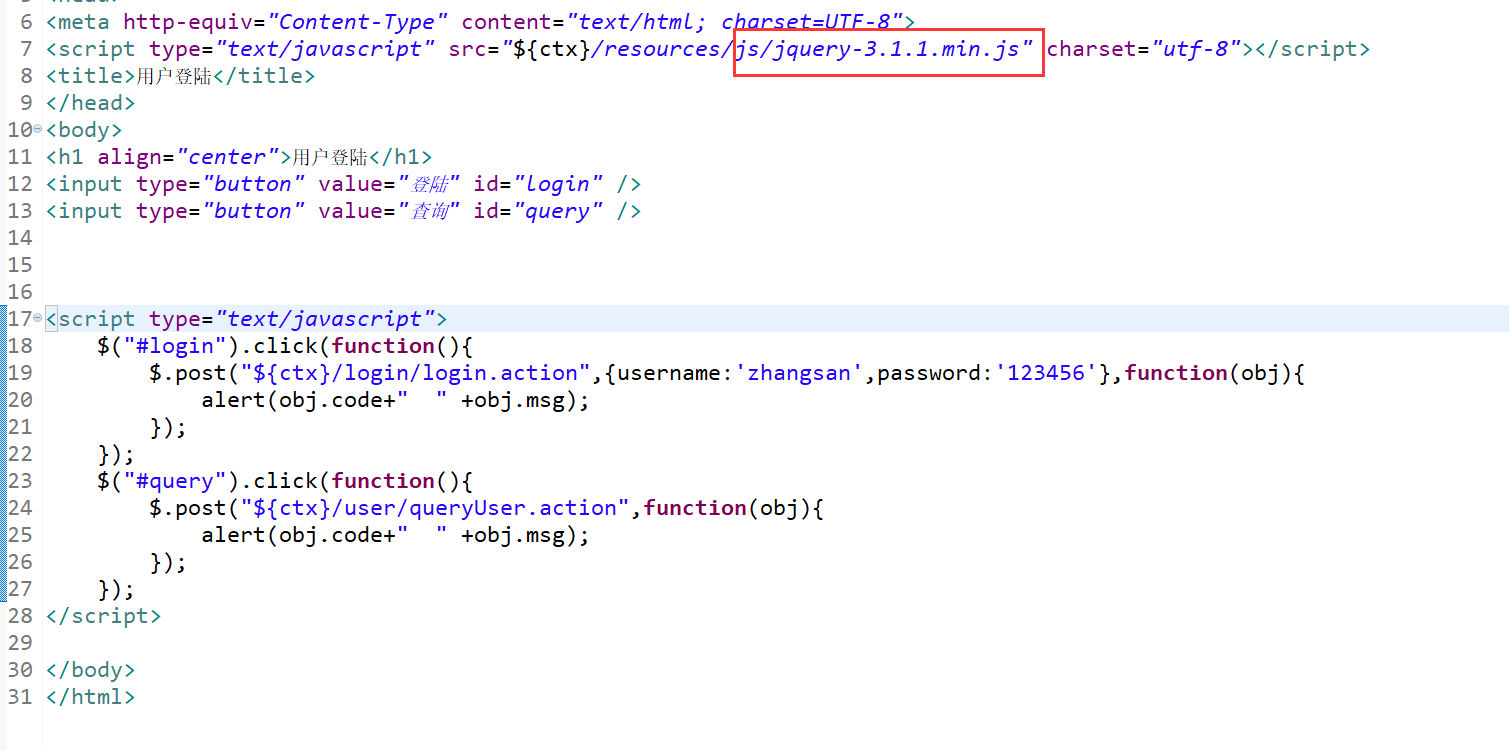
两个项目进行数据交互使用json

## 基于ssm\_shrio的项目修改

### 修改LoginController



### 修改index.jsp



### 修改UserController

|  |
| --- |
| @RestController  @RequestMapping("user")  **public** **class** UserController {  @RequiresPermissions("user:query")  @RequestMapping("queryUser")  **public** Map<String,Object> queryUser(){  Map<String,Object> map=**new** HashMap<>();  map.put("code", 0);  map.put("msg", "查询成功");  **return** map;  }  @RequiresPermissions("user:add")  @RequestMapping("addUser")  **public** Map<String,Object> addUser(){  Map<String,Object> map=**new** HashMap<>();  map.put("code", 0);  map.put("msg", "添加成功");  **return** map;  }  @RequiresPermissions("user:update")  @RequestMapping("updateUser")  **public** Map<String,Object> updateUser(){  Map<String,Object> map=**new** HashMap<>();  map.put("code", 0);  map.put("msg", "更新成功");  **return** map;  }    @RequiresPermissions("user:delete")  @RequestMapping("deleteUser")  **public** Map<String,Object> deleteUser(){  Map<String,Object> map=**new** HashMap<>();  map.put("code", 0);  map.put("msg", "删除成功");  **return** map;  }  @RequiresPermissions("user:export")  @RequestMapping("exportUser")  **public** Map<String,Object> exportUser(){  Map<String,Object> map=**new** HashMap<>();  map.put("code", 0);  map.put("msg", "导出成功");  **return** map;  }    } |

### 创建AppException[解决未授权的全局异常]



### 引入fastjson

### 创建MyFormAuthenticationFilter [解决未认证跳页面返回json?]

|  |
| --- |
| /\*\*  \* 重写shrio authc 的过滤器  \* 解决 当用户使用浏览器访问未认证的页面时跳转到页面  \* 当用户使用ajax访问未认证的数据时返回json串  \* **@author** LJH  \*  \*/  **public** **class** MyFormAuthenticationFilter **extends** FormAuthenticationFilter {  /\*\*  \* 在访问controller前判断是否登录，返回json，不进行重定向。  \*  \* **@param** request  \* **@param** response  \* **@return** true-继续往下执行，false-该filter过滤器已经处理，不继续执行其他过滤器  \* **@throws** Exception  \*/  @Override  **protected** **boolean** onAccessDenied(ServletRequest request, ServletResponse response) **throws** Exception {    //判断是否为ajax请求  **if**(isAjax(request)) {  response.setCharacterEncoding("UTF-8");  response.setContentType("application/json");  Map<String,Object> resultData = **new** HashMap<>();  resultData.put("code", -1);  resultData.put("msg", "登录认证失效，请重新登录!");  PrintWriter out = response.getWriter();  out.write(JSON.*toJSONString*(resultData));  }**else** {  //重定向  response.reset();  HttpServletResponse servletResponse=(HttpServletResponse) response;  servletResponse.sendRedirect("../index.jsp");  }  **return** **false**;    }    /\*\*  \* 判断是否为ajax请求  \* **@param** request  \* **@return**  \*/  **private** **boolean** isAjax(ServletRequest request) {  String header = ((HttpServletRequest) request).getHeader("X-Requested-With");  **if** ("XMLHttpRequest".equalsIgnoreCase(header)) {  **return** Boolean.***TRUE***;  }  **return** Boolean.***FALSE***;  }      } |

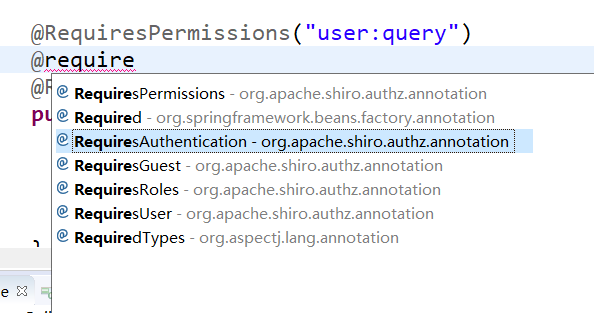
### 修改application-shrio.xml[覆盖atuhc的过滤器]



### 修改springmvc.xml[解决注解不生效问题]

|  |
| --- |
| <!-- 扫描全局处理对象 -->  <context:component-scan base-package=*"com.sxt.exception"*></context:component-scan>    <!-- 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> |

## 基于Shrio的注解权限说明



@RequiresPermissions(value={“”,””})判断当前请求的线程是否有value里面的所有权限

@RequiresAuthenticatin 判断当前线程是否认证

@RequiresRoles(value={}) 判断当前线程是否有value里面的所有角色