# 简介

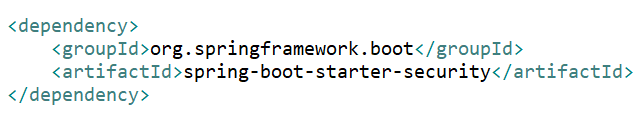
安全是公司的红线，一般项目都会有严格的认证和授权操作，在Java开发领域常见的安全框架有Shiro和Spring Security。

1. Shiro是一个轻量级的安全管理框架，提供了认证、授权、会话管理、密码管理、缓存管理等功能；
2. Spring Secuity是一个相对复杂的安全管理框架，功能比Shiro更强大，权限控制细粒度更高，对OAuth2的支持也更友好，又因为Spring Security源自Spring家族，因此可以和Spring框架无缝整合，在Spring Boot中提供自动化配置方案，可以让Spring Security的使用更加便捷。

# Spring Security

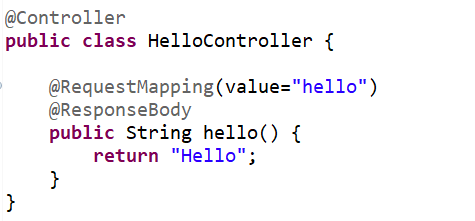
## 基本用法

1. 添加spring-boot-starter-security依赖，如下：

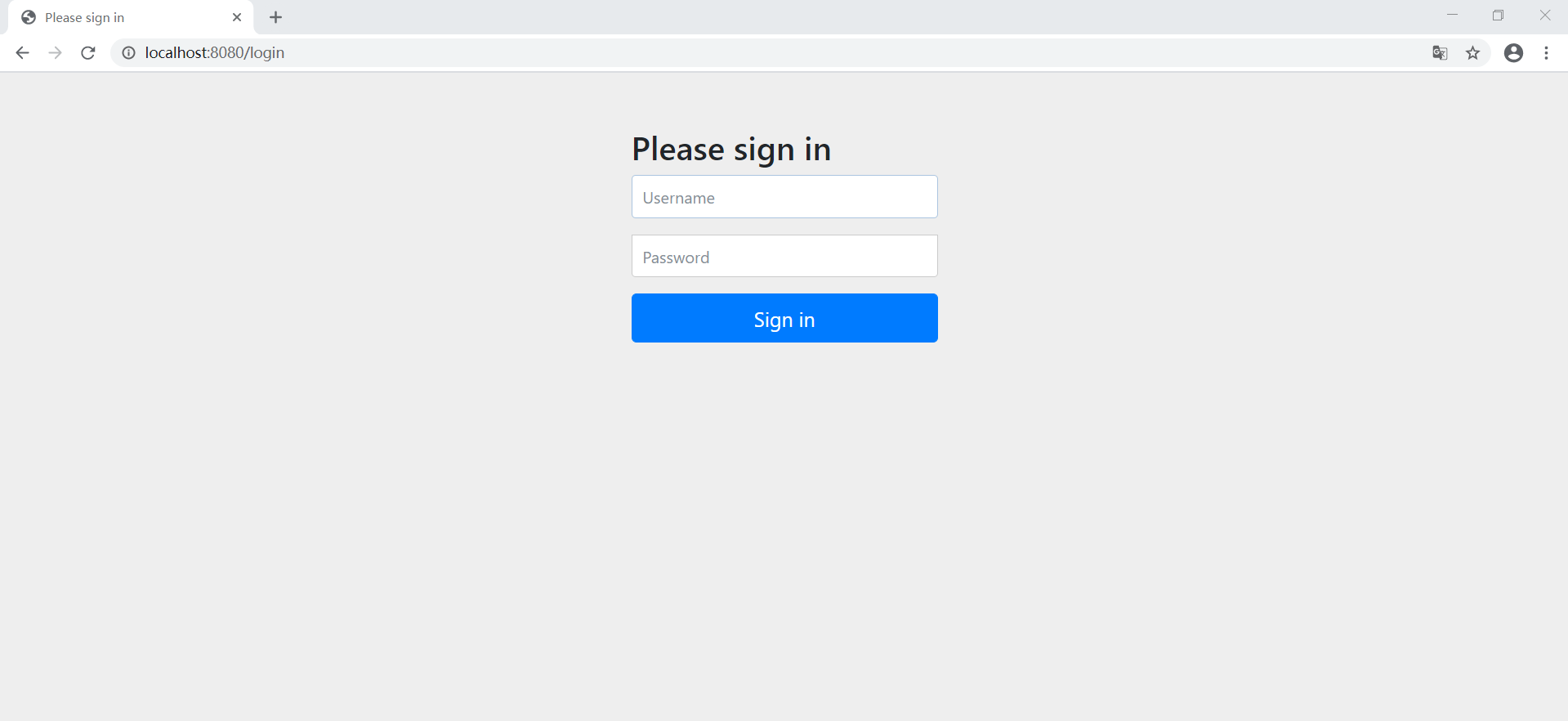


添加spring-boot-starter-security依赖后，项目中所有的资源都会被保护起来；

1. 创建控制器访问项目，如下：

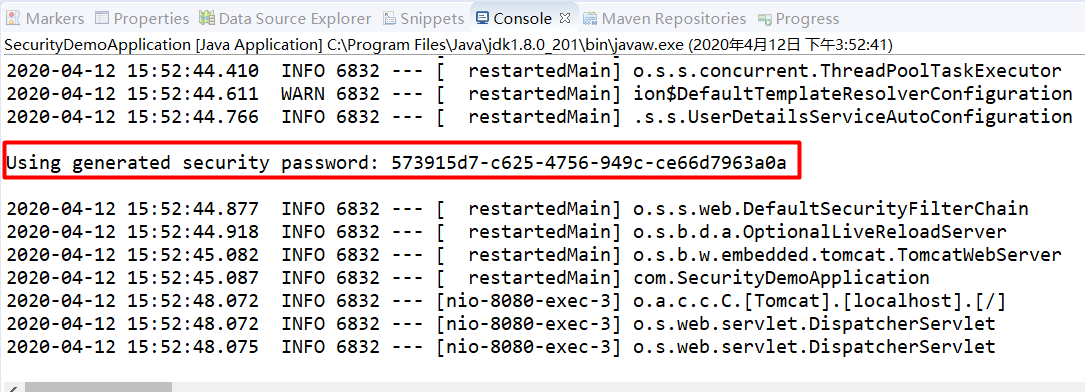


1. 启动项目，访问/hello路径，如下：

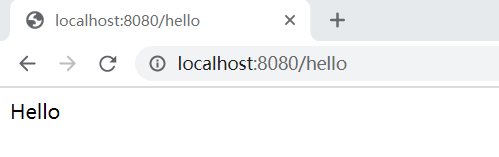


可以发现，并没有访问成功，而是跳转到一个登录页，这是因为添加了spring-boot-starter-security依赖；

1. 默认的用户名是user，默认的登录密码则在每次启动项目时在控制台打印，如下：



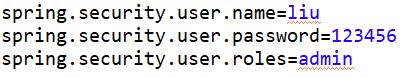
1. 此时，使用用户名和密码进行登录，即可访问的/hello路径提供的内容，如下：



登录成功后，本次会话即可可以访问项目资源；

## 配置用户名密码

如果开发者对默认的用户名和密码不满意，可以在application.properties中配置默认的用户名和密码，以及用户角色，如下：

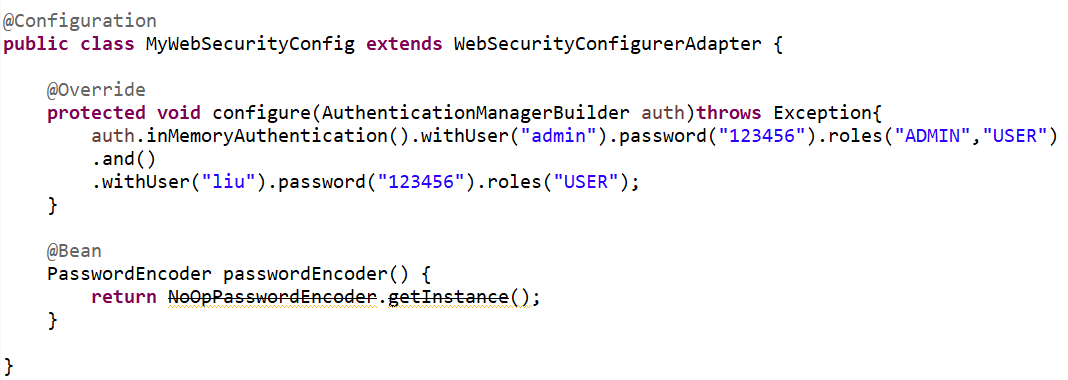


配置了默认的用户名和密码后，访问资源时使用配置的用户名和密码进行登录，控制台将不再打印随机密码，

在这里还配置了用户角色，所以该用户还具有一个角色---admin；

## 基于内存的认证

开发者也可以自定义类继承WebSecurityConfigurerAdapter类，进而实现对Spring Security更多的自定义配置。例如基于内存的认证，如下：



在自定义配置类中，覆写了WebSecurityConfgurerAdapter类中的configure方法，在该方法中配置了两个用户：一个用户名为admin，密码为123456，具备两个角色ADMIN和USER；另一个用户名为liu，密码为123456，具备一个角色USER；

本例中使用的Spring Security版本为5.0之后，在Spring Security5.0后引入了多种密码加密方式，开发者必须指定一种，在这里使用NoOpPasswordEncoding，表示不对密码进行加密；

配置好后，就可以使用这两个用户进行登录了。

## HttpSecurity

虽然实现了认证功能，但是受保护的资源都是默认的，所有的资源都被保护了，而且也不能根据实际情况进行角色管理；如果要实现这些功能，就需要重写WebSecurityConfigurerAdapter中的另一个方法，如下：

1）

在第一个configure方法中，配置了三个用户，root用户具有ADMIN和DBA的角色；admin用户具有ADMIN和USER的角色；liu用户具有USER的角色；

第二个方法passwordEncoding在前面已经讲过，在这里表示不对密码进行加密；

第三个方法configure方法中，通过HttpSecurity对象调用authorizeRequest方法开启HttpSecurity配置；之后设置了“/admin/\*\*”模式的URL能够被ADMIN角色所访问“/user/\*\*”模式的URL能够被ADMIN和USER角色所访问；“/db/\*\*”模式的URL能够被ADMIN和DBA角色所访问；

调用anyRequest方法和authenticated方法表示用户访问除了以上定义URL路径外，都必须经过认证后访问，即必须登录访问；

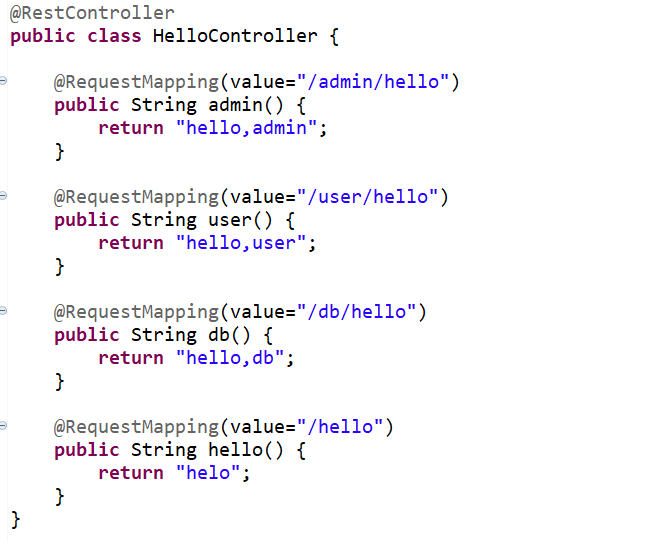
调用and方法、formLogin方法表示开启表单登录，即一开始看到的登录页；

调用loginProcessingUrl方法用于配置登录页的URL路径、发起一个post请求进行登录，登录参数中用户名必须命名为username，密码必须命名为password，通过loginProcessingUrl方法主要是方便Ajax或移动端调用登录接口

调用permitAll方法表示和登录相关的接口都不需要认证即可访问

调用and方法和csrf方法表示关闭csrf；

2）编写控制器，用于测试，如下：

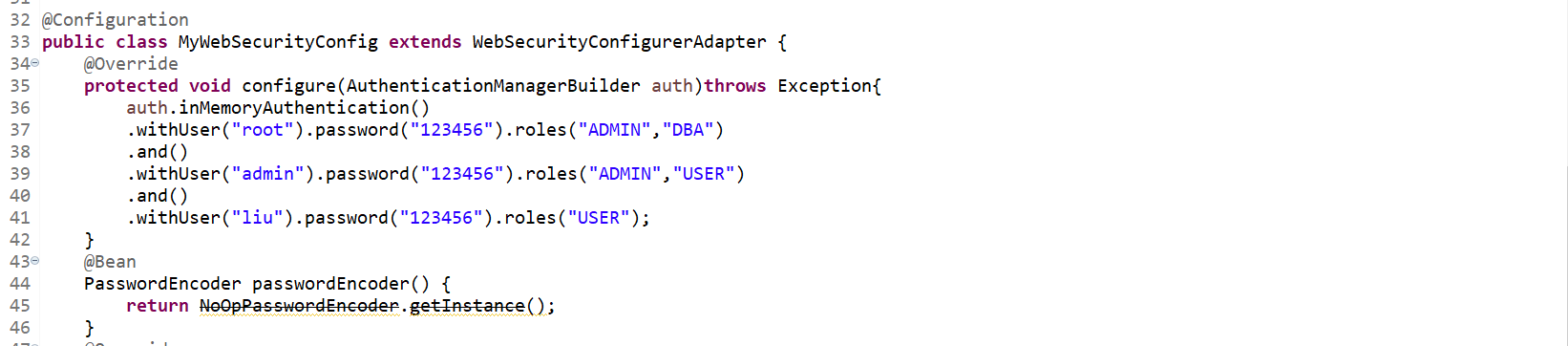


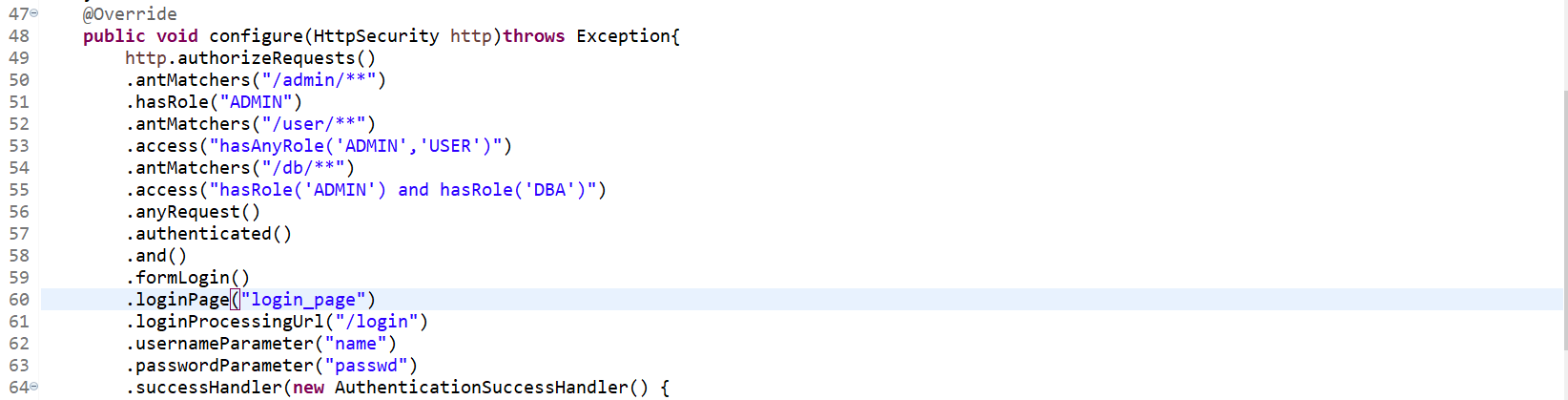
1. 启动项目，进行测试，这样一来，就规定了每个角色所能访问的路径（资源）。

## 登录表单详细配置

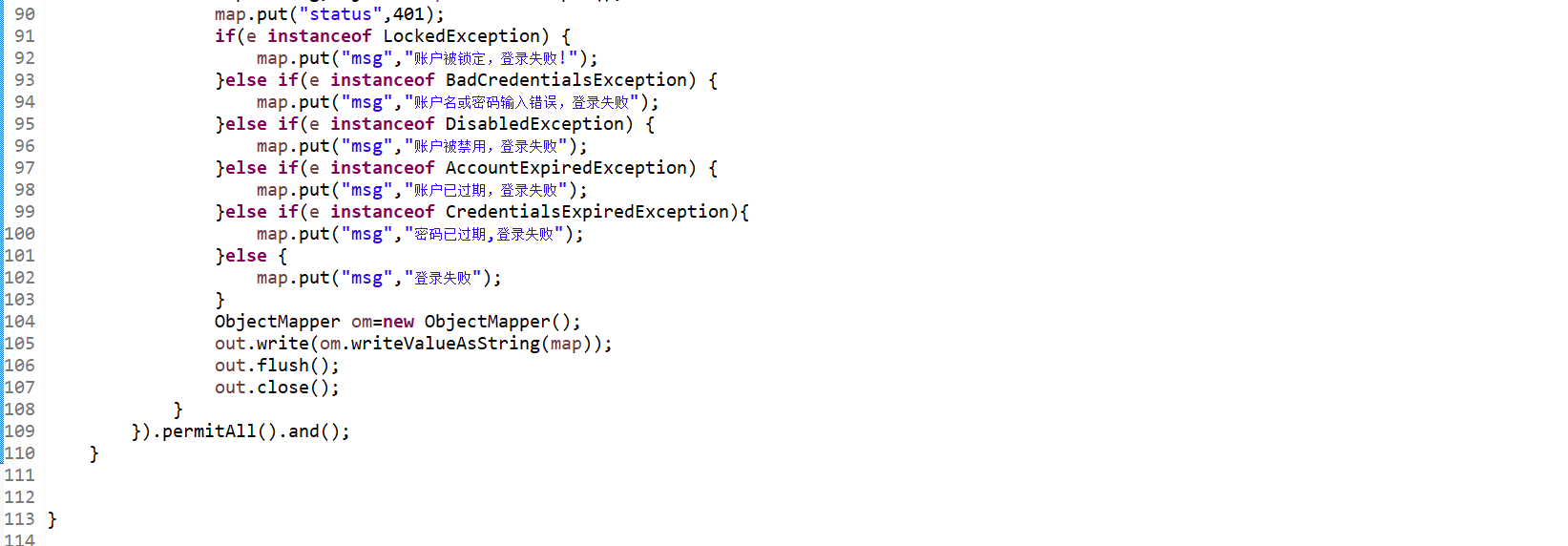
在登录验证的时候使用的是Spring Security提供的登录页面，登录成功后也是默认的页面跳转。但是，前后端正在成为企业级应用开发的主流，在前后端分离的开发模式中，前后端的数据交互通过Ajax进行，这时，登录成功后就不是页面的跳转了，而是一段JSON提示，告知前端是否登录验证成功。

Spring Security要实现这样的功能，需要继续完成上文的配置，代码如下：









第一个方法configure用于增加用户，设置用户的用户名和密码，以及角色；

第二个方法passwordEncoder表示对密码不进行加密；

第三个方法confiure设置不同角色所拥有的访问路径权限；

在第60行配置了loginPage，即登录页面，配置了loginPage后，如果用户未授权就访问一个需要授权才能访问的接口，就会自动跳转到login\_page页面，这个login\_page页面时开发者自已进行定义的登录页面，而不再使用Spring Security提供的默认登录页；

第61行配置了loginProcessingUrl，表示登录请求处理接口，无论是自定义登录页还是移动端登录，都需要使用该接口；

第62、63行定义了认证所需的用户名和密码的参数名，默认用户名参数是username，密码参数是password，在这里进行了自定义，设置成了name和passwd；

第64行到81行定义了登录成功的处理逻辑，用户登录成功后可以跳转到某一个页面，也可以返回一段JSON，这个要看具体的业务逻辑，在本例中假设的是返回一段JSON，用户登录成功后，返回一段登录成功的JSON，onAuthenticationSuccess方法的第三个参数一般用来获取当前登录用户的信息，在登录成功后，可以获取当前登录用户的信息一起返回给客户端；

第81行到108行定义了登录失败的处理逻辑，和登录成功类似，不同的是，登录失败的回调方法里有一个AuthenticationException参数，通过这个异常类对象可以获取登录失败的原因，进而给用户一个明确的提示；

配置完成后进行测试（登录请求的参数用户名已在62行自定义为name，密码已在63行自定为passwd）。

## 注销登录配置

如果想要注销登录，也只需要提供简单的配置即可。如下：



第48行表示开启注销登录的配置；

第49行表示注销登录请求URL为“/logout”，默认也是“/logout”；

第50行表示是否清除身份认证，默认为true，表示清除；

第51行配置了一个LogoutHandler，开发者可以在LogoutHandler中完成一些数据清除工作，如Cookie的清除，Spring Security提供了一些常见的实现，请自行查阅；

第58行配置了一个logoutSuccessHandler，开发者可以在这里处理注销成功后的业务逻辑，如返回一段JSON提示或者跳转到登录页面等；

## 多个HttpSecurity

如果业务比较复杂，开发者也可以配置多个HttpSecurity，实现对WebSecurityConfigurerAdapter的多次扩展，如下：



配置多个HttpSecurity时，MultiHttpSecurityConfig不需要继承WebSecurityConfigurerAdapter，在MultiHttpSecurityConfig中创建静态内部类继承WebSecurityConfigurerAdapter即可，静态内部类上添加@Configuration注解和@Order注解，@Order注解表示该配置的优先级，数字越小优先级越大，未加@Order注解的配置优先级最小。

第25行到32行表示该静态类用于处理“/admin/\*\*”模式的URL能够被何种角色访问；

其他的URL在第34行到48行配置的HttpSecurity中进行处理；

## 密码加密

### 为什么加密

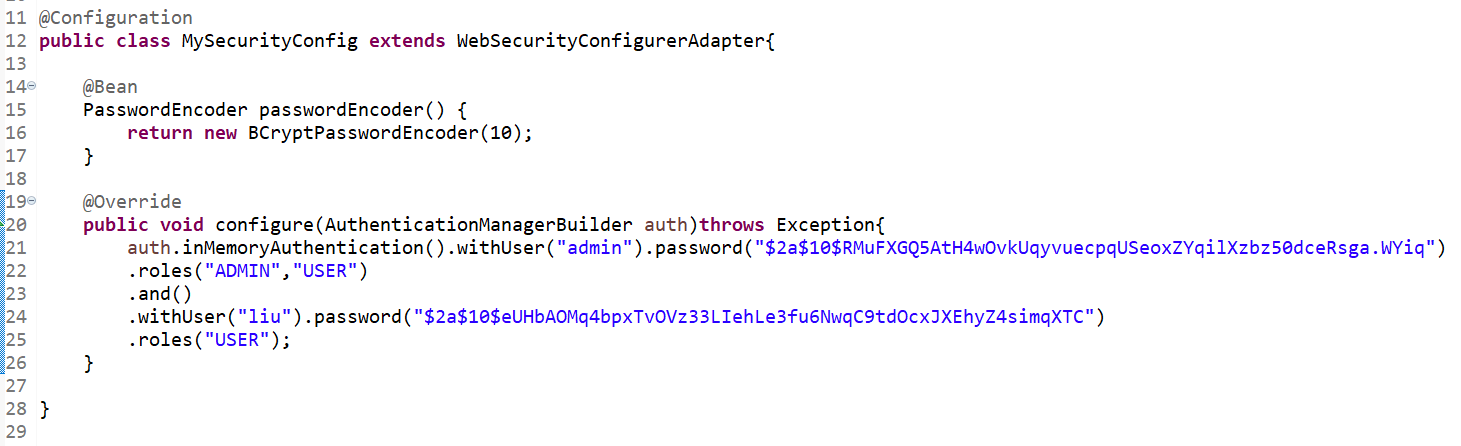
如果使用明文存储用户密码，一旦泄露，后果将不堪设想，因为很多用户为了方便记住密码，通常是多个账户公用一个密码，如QQ密码和微信密码相等。所以，在做项目时，密码需要进行加密处理。

### 加密方案

密码加密一般会用到散列函数，又称散列算法、哈希函数，这是一种从任何数据中创建数字“指纹的方法”。散列函数把消息或数据压缩成摘要，使得数据量变小，将数据的格式固定下来，然后将数据打乱混合，重新创建一个散列值。散列值通常用一个短的随机字母和数字组成的字符串来表示。好的散列函数在输入域中很少出现散列冲突。在散列表和数据处理中，不抑制冲突来区别数据会使得数据库记录更难找到。我们常用的散列函数由MD5消息摘要算法、安全散列算法（SecureHash Algorithm）。

### 实践

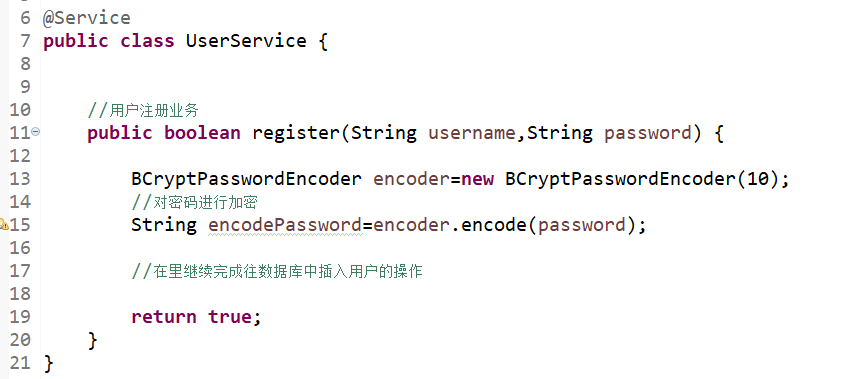
在Spring Boot中配置密码加密非常容易，只需要修改上文配置的PasswordEncoding这个Bean即可实现。如下：



第一方法passwordEncoder用于密码加密，传入的10表示密匙的迭代次数，也可以不配合，默认为10，同时在第二个方法中内存用户的密码也不再是明文了。

这里的密码是使用BCryptPasswordEncoder加密后的密码，但是明文都是123，配置完成后，admin/123与liu/123就可以实现登录，

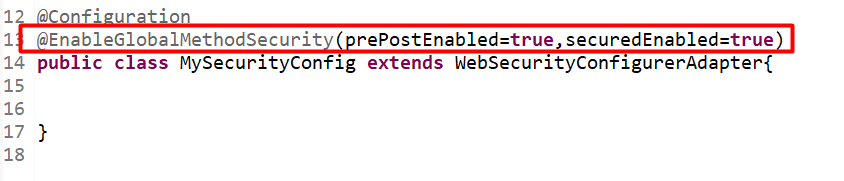
本案例使用了配置在内存中的用户，在实际情况下，用户信息都是存储在数据库中的，因此需要在用户注册时对密码进行加密处理。如下：



## 方法安全

上面介绍的认证与授权都是基于URL的，开发者也可以通过注解来灵活配置方法安全。

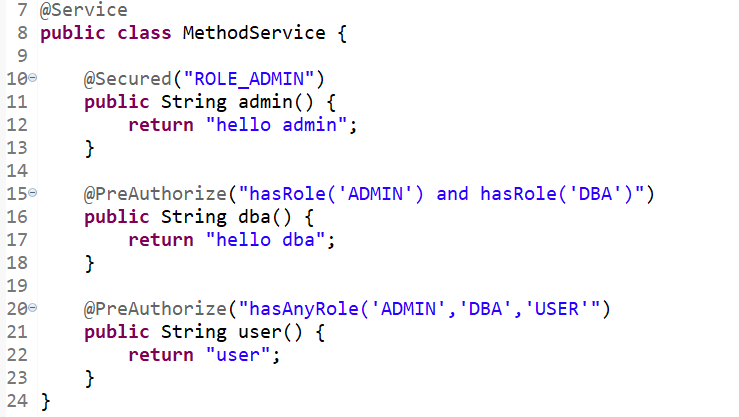
1. 因为要使用到相关注解，所以首先需要通过@EnableGlobaMethodSecurity开启基于注解的安全配置，如下：



prePostEnabled=true表示解锁@PreAuthorize和@PostAuthorize两个注解，@PreAuthorize注解会在方法执行前执行，@PostAuthorize注解会在方法执行后进行验证。

secureEnabled=true表示解锁@Secured注解；

1. 开启注解安全配置后，创建MethodService进行测试，如下：



@Secured(“ROLE\_ADMIN”)表示ADMIN角色才能够访问该方法，注意：角色前面加了前缀“ROLE\_”；

@PreAuthorize(“hasRole(‘ADMIN’) and hasRole(‘DBC’)”)表示既是ADMIN角色又是DBA角色的用户才能够访问该方法；

@PreAuthorize(“hasAnyRole(‘ADMIN’,’DBA’,’USE’)表示ADMIN角色、或DBA角色、或USER角色才能够访问该方法；

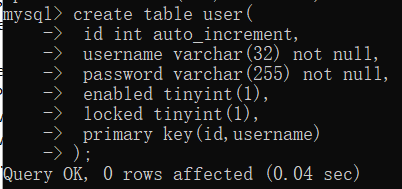
在这里没有用到@PostAuthorize，它与@PreAuthorize注解的使用方法相同，只不过@PreAuthorize是在方法执行前进行验证、而@PostAuthorize是在方法执行后进行验证。

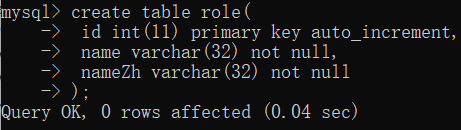
## 基于数据库的认证

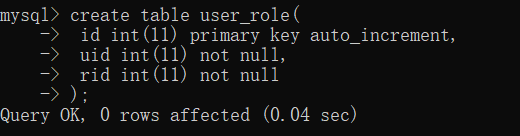
上面介绍的认证数据都是定义在内存中的，在真实项目中，用户的基本信息以及角色等都存储在数据库中，因此需要从数据库中获取数据进行认证。

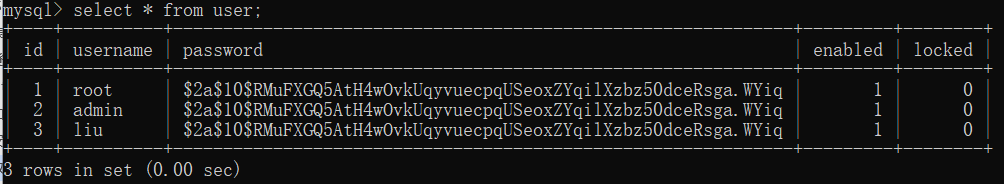
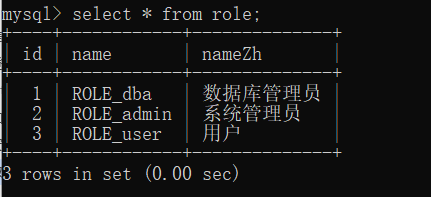
1. 设计数据表，如下：

设计三个表，user、role、user\_role表，user表示用户表用于存放用户信息，role表表示角色表，user\_role表表示用户角色关联表。

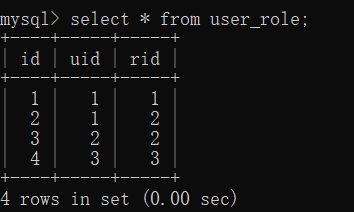








角色名有一个默认的前缀ROLE\_。

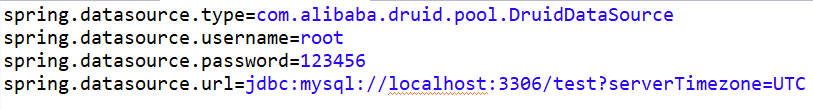


1. 因为要从数据库中取出用户进行验证，所以这里选择的持久层技术是Mybatis，整合MyBatis如下：

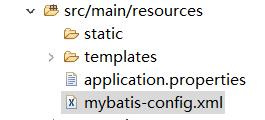
1导入依赖



2配置数据库连接信息

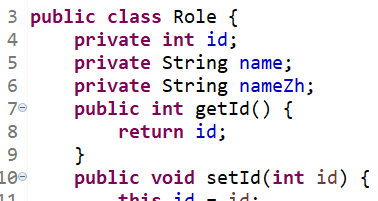


3创建Mybatis配置文件并在指明其文件位置

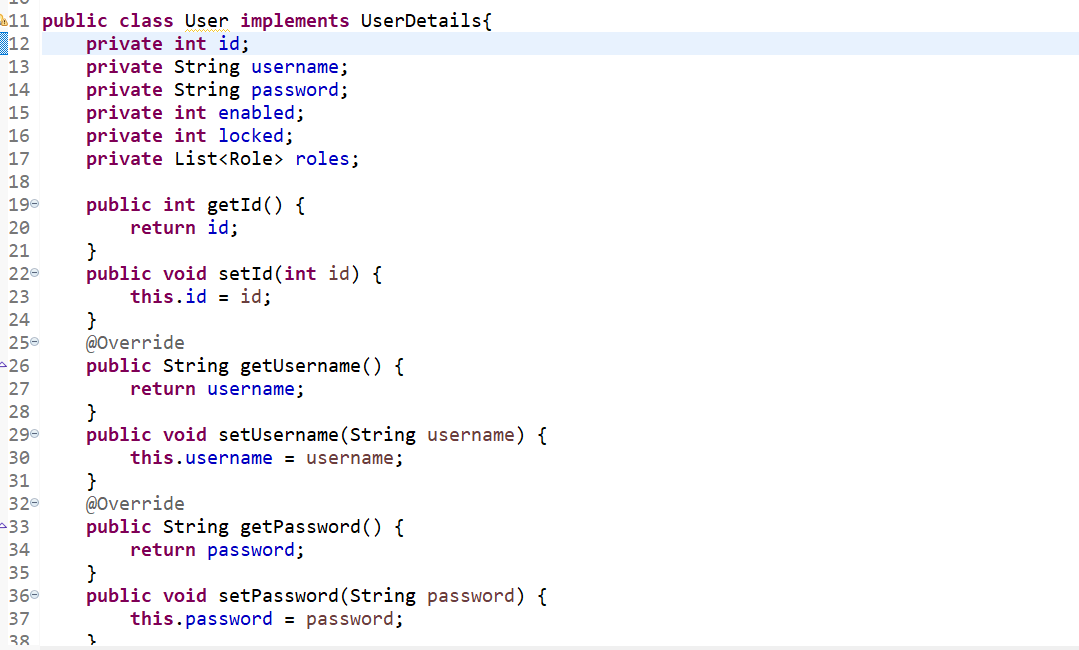


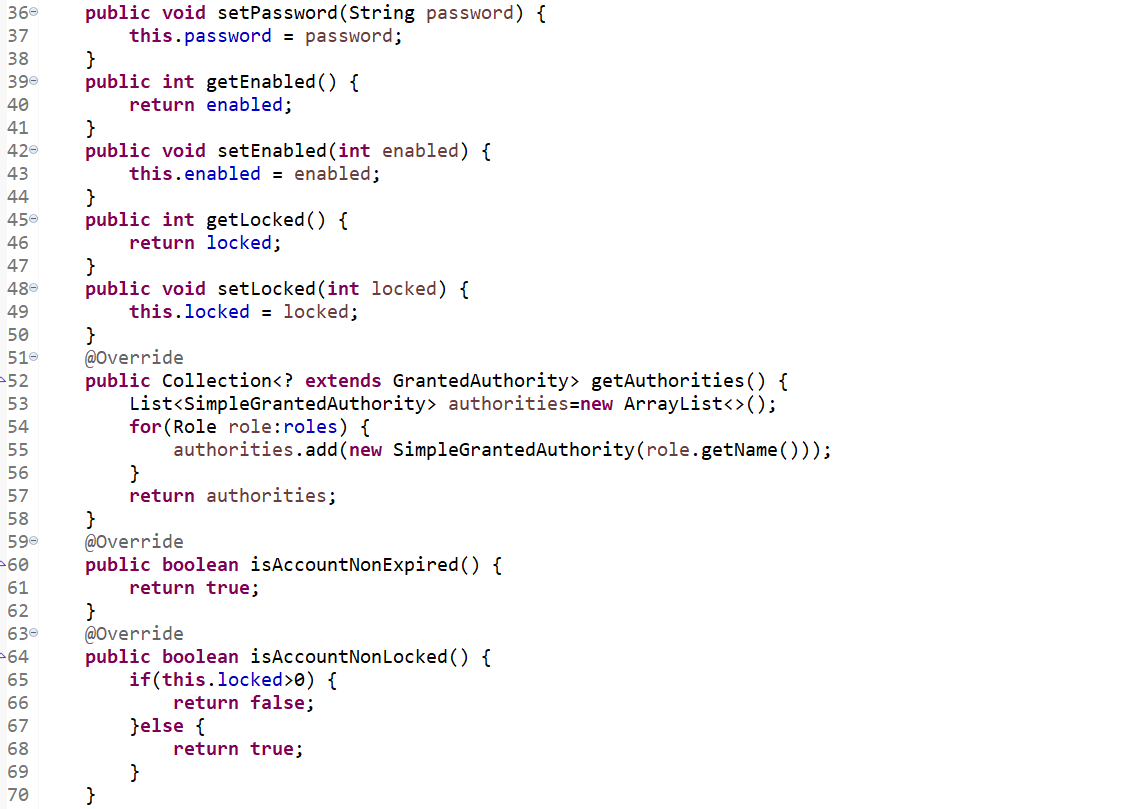


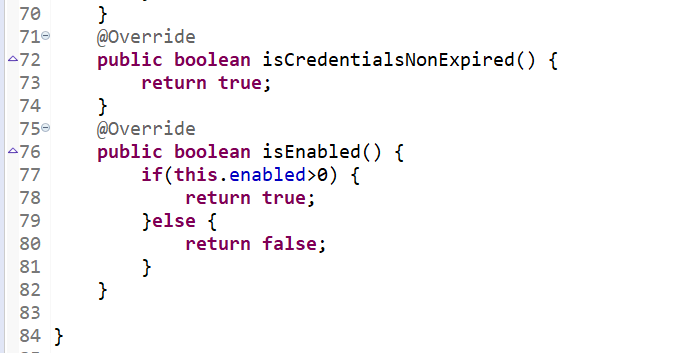
1. 创建user表与role表对应的实体类



User表需要实现UserDetail接口，然后覆写多个方法，具体如下：







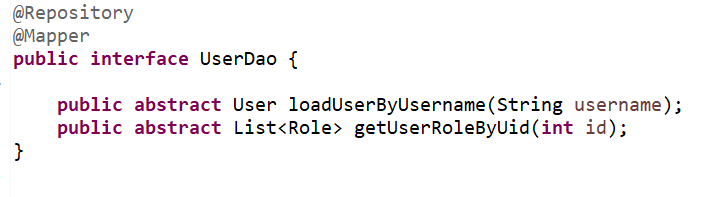
在User实体类中，实现了UsrDetail接口，并实现了该接口的7个方法，如下：

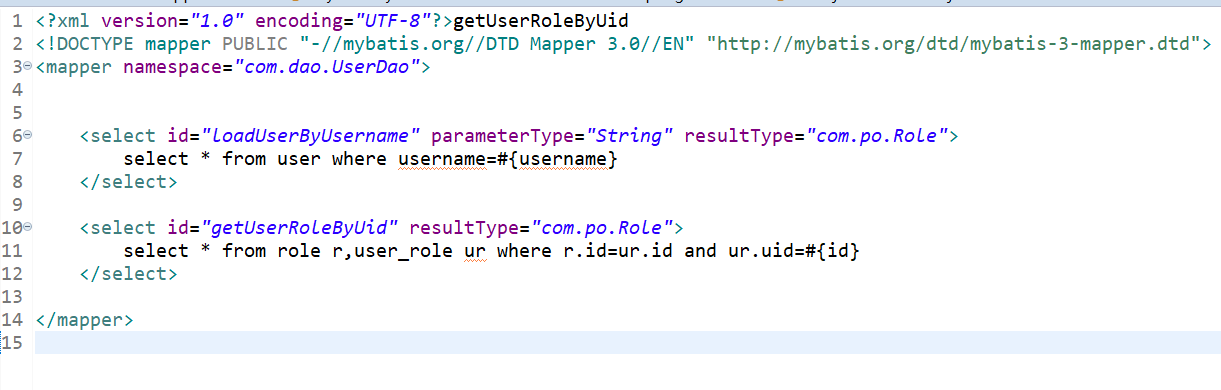
|  |  |
| --- | --- |
| 方法名 | 解释 |
| getAuthorities() | 获取当前用户对象所具有的角色信息 |
| getPassword() | 获取当前对象的密码 |
| getUsername() | 获取当前对象的用户名 |
| isAccountNonExpired() | 当前账户是否未过期 |
| isAccountNonLocked() | 当前账户是否未锁定 |
| isCredentialsNonExpired() | 当前账户密码时否未过期 |
| isEnabled() | 当前账户是否可用 |

开发者应根据实际情况设置这7个方法的返回值。因为默认情况下不需要开发者自己进行密码角色等信息的比对，开发者只需要提供相关的信息即可，例如getPassword（）方法返回的密码和用户输入的登录密码不匹配，会自动抛出BadCredentialsException异常，isAccountNonExpired()方法返回了false，会自动抛出AccountExpiredException异常，因此对于开发者而言，只需要按照数据库中的数据在这里返回相应的配置即可。本案例因为数据库中只有enabled和locked字段，故账户过期和密码未过期两个方法都返回tue；

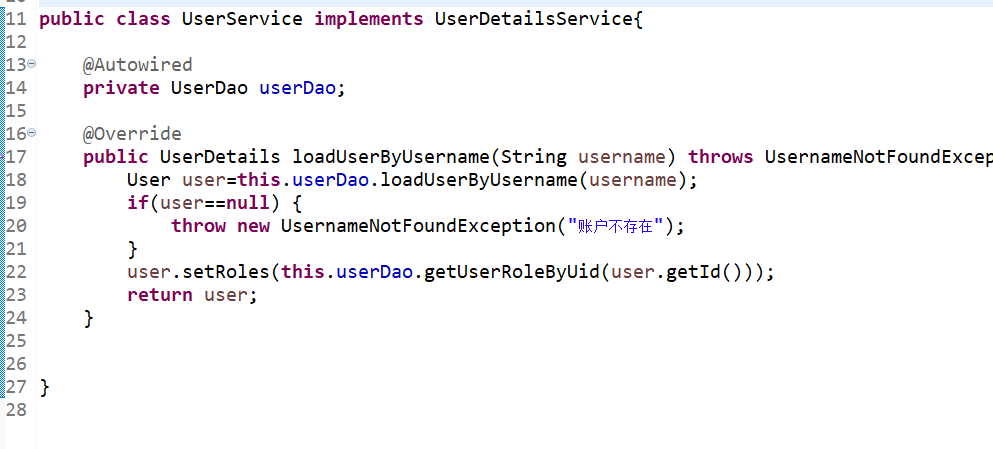
getAuthorities()方法用来获取当前用户所具有的角色信息，本案例中，用户所具有的角色存储在roles属性中，因此该方法直接遍历roles属性，然后构造SimpleGrantedAuthority集合并返回；

1. 创建UserDao以及UserMapper.xml，如下：





1. 创建UserService，如下：



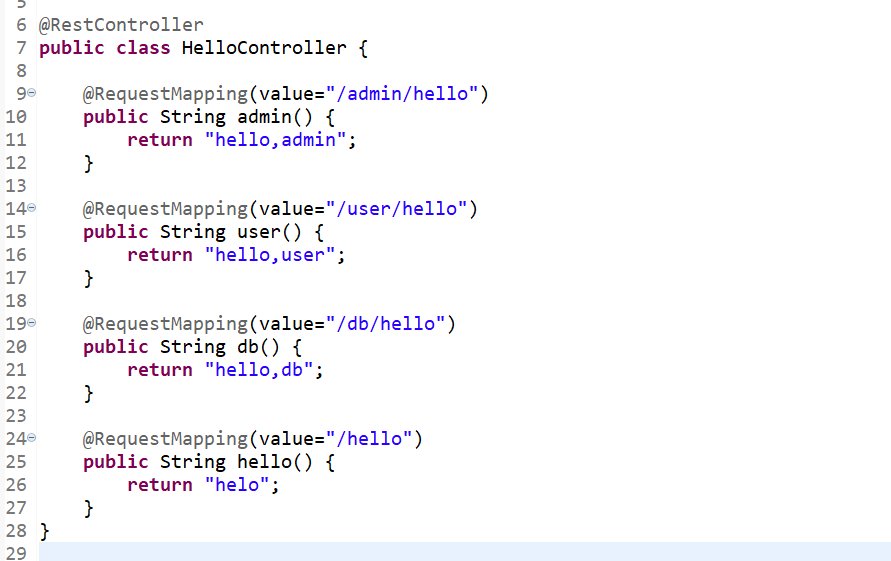
该UserService实现了UserDetailsService接口，并实现该接口中的loadUserByUsername方法，该方法的参数就是用户登录时输入的用户名，通过用户名取数据库中查找用户，如果没有查找到用户，就抛出一个用户不存在的异常，如果查找到了用户，就继续查找该用户所具有的角色信息，并将获取到的user对象返回，再由系统提供的DaoAuthenticationProvider类去比对密码是否正确；

loadUserByUsername方法在用户登录时自动调用；

1. 配置Spring Security，如下：



1. 编写控制器进行测试，如下：

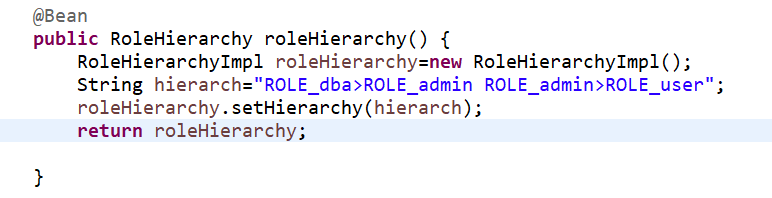


启动项目，运行，访问路径，然后使用数据库中的用户名和密码进行登录。

## 角色继承

在上一章节中，定义了三种角色，但是这三个角色之间不具备任何关系，一般来说角色是有关系的，例如ROLE\_admin一般既具有admin的权限，又具有user的权限。那么如何配置这种角色继承关系呢？在Spring Security中只需要开发者提供一个RoleHierarchy即可。

以上一章基于数据库的认证章节的案例为例，假设ROLE\_dba是终极Boss，具有所有的权限，ROLE\_admin具有ROLE\_user的权限、ROLE\_user则是一个公共角色，据ROLE\_admin继承ROLE\_user、ROLE\_dba继承ROLE\_admin，要描述这种关系，只需要开发者在Spring Security的配置类中提供一个RoleHierarchy即可。如下：



配置完成后，dba角色不仅可以访问自己可以访问的资源，也能够访问admin角色访问的资源，还可以访问user角色可以访问的资源；

admin角色不仅可以访问自己可以访问的资源，也能够访问user角色可以访问的资源；

user角色只能访问自己可以访问的资源；

# OAuth2

## OAuth2简介

OAuth是一个开放标准，该标准允许用户让第三方应用访问该用户在某一网站上存储的私密资源，如头像、照片、视频等，而在这个过程中无须将用户名和密码提供给第三方应用。

实现这个功能是通过提供一个令牌(token)，而不是用户名和密码来访问他们存放在特定服务器提供者的数据。

每一个令牌授权一个特定的网站在特定的时段内访问特定的资源。这样OAuth让用户可以授权第三方网站灵活地访问存储在另外一些资源服务器的特定信息，而非所有内容。例如：用户通过QQ登录知乎，这时知乎就是一个第三方应用，知乎要访问用户的一些基本信息就需要得到用户的授权，如果用户把自己的QQ用户名和密码告诉知乎，那么知乎就可以访问用户的所有数据，但只有用户修改密码才能收回授权，这种授权方式安全隐患很大，如果使用OAuth，就能很好的解决这一问题；

采用令牌（token）的方式可以让用户灵活地对第三方应用授权或者收回权限。OAuth2是OAuth协议的下一个版本，但不向下兼容OAuth1.0。OAuth2关注客户端开发者的简易性，同时为Web应用、桌面应用、移动设备、起居室设备提供专门的认证流程。

传统的Web开发登录认证一般都是基于session的，但是在前后端分离的架构中继续使用session会存在很多不便，因为移动端要么不支持Cookie，要么使用非常不方便，对于这些问题，使用OAuth2.0认证都能解决。

## OAuth2角色

要了解OAuth2，需要先了解OAuth2中几个基本的角色。

1. 资源所有者：资源所有者即用户，具有头像、照片、视频等资源；
2. 客户端：客户端即第三方应用，授权微信登录拼多多，此时拼多多就是第三方应用；
3. 授权服务器：授权服务器用来验证用户提供的信息是否正切，并返回一个令牌给第三方应用；
4. 资源服务器：资源服务器是提供给用户资源的服务器，例如头像、照片、视频等。

一般情况，授权服务器与资源服务器是同一台服务器；

## OAuth 2授权流程

OAuth 2的授权流程如下：

1. 用户通过第三方应用发起授权；
2. 跳入授权页，用户同意授权后，服务端返回一个授权许可凭证给第三方应用；
3. 第三方应用拿着授权许可证取授权服务器申请令牌；
4. 授权服务器验证信息无误后，发送令牌给第三方应用；
5. 资源验证器验证令牌无误后开放资源；

1、请求授权

第三方应用

用户

2、同意授权

3、申请令牌

授权服务器

4、发放令牌

5、申请资源

资源服务器

6、开放资源

## OAuth 2授权模式

OAuth授权模式共分为4种，如下：

1. 授权码模式

授权码模式是功能最完整、流程最严谨的授权模式。它的特点是通过第三方应用的服务器与授权服务器进行交互，国内常用的第三方平台登录功能基本都是使用这种模式；

1. 简化模式

简化模式不需要第三方应用的服务器参与，直接在浏览器中向授权服务器申请令牌，一般若网站是静态页面，则可以采用此种方式；

1. 密码模式

密码模式是用户把用户名和密码直接告诉第三方应用，第三方应用使用这些信息向授权服务器申请令牌。这需要用户对第三方应用的高度信任，例如第三方应用和服务提供商是同一家公司；

1. 客户端模式

客户端模式是指第三方应用使用自己的名义而不是用户的名义向服务提供者申请授权。严格来说，客户单模式并不能算作OAuth协议要解决的问题的一种解决方案，但是，对于开发者而言，在一些前后端分离应用或者移动端提供的认证授权服务器上使用这种模式还是非常方便的；

这4种授权模式各有千秋，应根据实际开发场景进行选择。

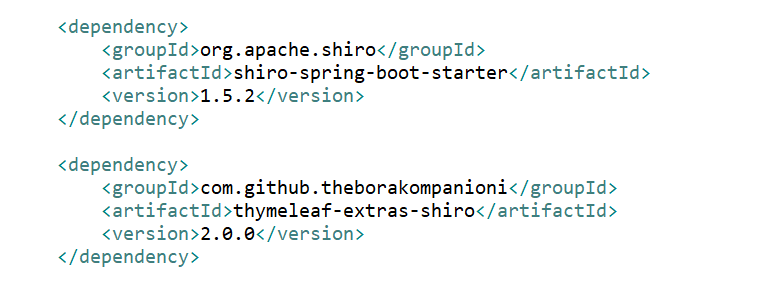
# Shiro

## Shiro简介

1. Apache Shiro是一个开源的轻量级的Java安全框架，它提供身份验证、授权、密码管理以及会话管理等功能；
2. 相对于Spring Security，Shiro框架更加直观、易用、同时也能提供健壮的安全性；

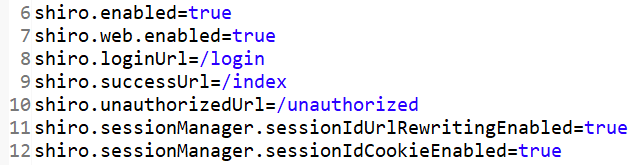
## 整合Shiro

1. 添加Shiro依赖，如下：



添加thymeleaf-extras-shiro依赖的目的是能够在Thymeleaf视图中使用shiro标签，如果spring-boot-starter-thymeleaf依赖不存在，则该依赖添加了也没有意义；

1. 配置Shrio的基本信息，如下：



第6行表示开启Shiro配置，默认为true；

第7行表示开启Shiro Web配置，默认为true；

第8行表示登录地址，默认为“/login.jsp”；

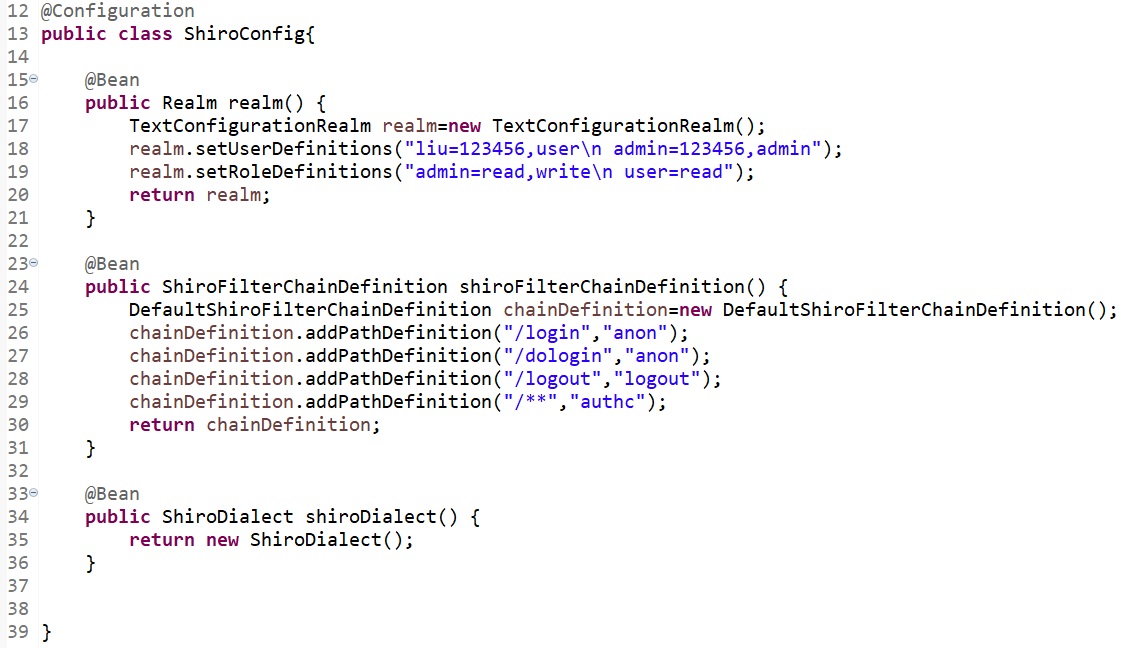
第9行表示登录成功的地址，默认为“/”

第10行表示未获得授权的默认跳转地址；

第11行表示是否允许通过URL参数实现会话跟踪，如果网站支持Cookie，可以关闭此选项，默认为true；

第12行表示是否允许通过Cookie实现会话跟踪，默认为true；

1. 定义一个Shiro配置类，如下：



在这个配置类中，向Spring容器中注册了三个Bean；

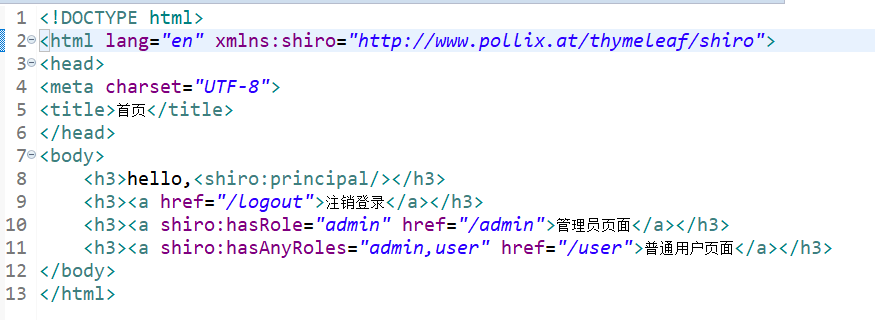
第一个Bean即realm，这个Realm可以是自定义Realm，也可以是Shiro提供的Realm，在这里，配置了两个用户和对应密码，用户liu角色是user，用户admin角色是admin；admin角色具有读写权限，user角色只具有读权限；

第二个Bean即shiroFilterChainDefinition，在这里，配置了基本的过滤规则，“/login”和“/dologin”路径可以匿名访问，“/logout”是一个注销登录请求，其余请求则都需要进行认证才能访问；

第三个Bean即shiroDialect，它是为了能够在thymeleaf视图中使用Shiro标签而配置的，如果不在thymeleaf视图中使用Shiro标签，那么可以不提供它；

1. 在resources/templates下创建5个页面：

index.html



该页面时登录成功后的首页，首先展示当前登录用户的用户名（第8行），然后展示一个“注销登录”的链接（第9行），若当前登录的用户具有admin角色，则展示 “管理员页面”的链接（第10行），若当前登录用户具有admin或user角色，则展示“普通用户页面”的链接；

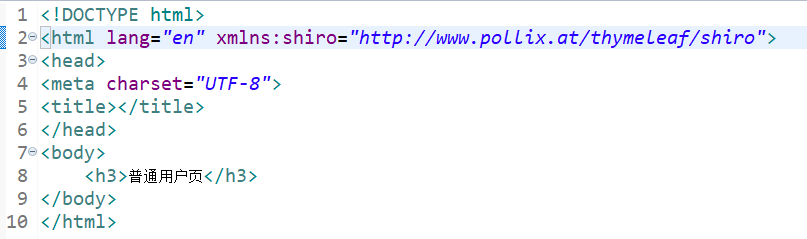
另外，这些页面中导入的名称空间时<http://www.pollix.at/thymeleaf/shiro>，和以前不一样；

login.html



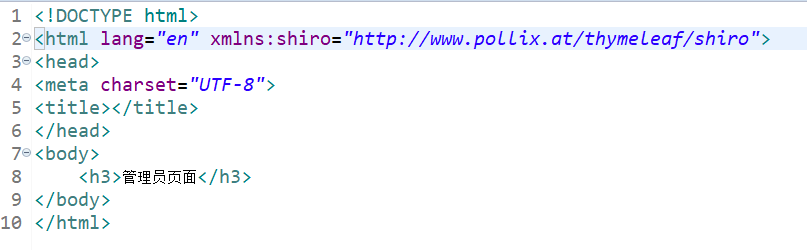
这是登录页面，登录失败时通过一个<div>显示登录失败的提示信息；

user.html



这是普通用户页面；

admin.html



这是管理员页面；

unauthorized.html



此页面用于展示授权失败，也会展示授权出错的信息；

1. 编写控制器，如下：

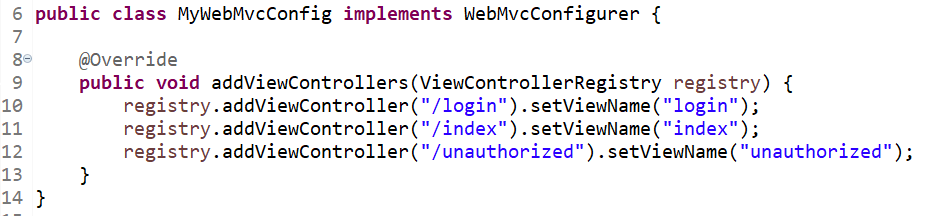


在doLogin方法中，首先构造一个UsernamePasswordToken实例，然后获取一个Subject对象并调用该对象中的login方法执行登录操作，在登录操作的执行过程中，当有异常发生，说明登录失败，携带错误信息返回登录视图；如果登录成功，则重定向到index.html页面，即首页；

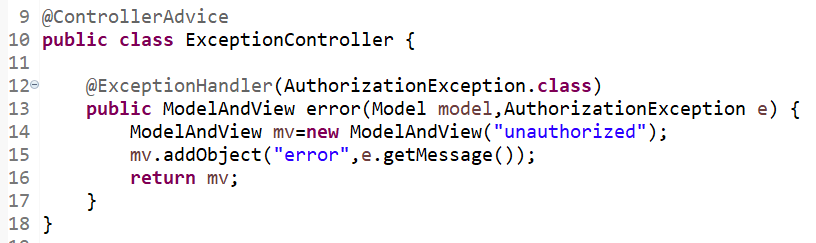
@ReqiresRoles(“admin”)注解表示该请求处理方法只能是admin角色访问；

@RequiresRoles(value={“admin,user”}表示该请求处理方法可以被admin角色或user角色所访问；

对于其他不需要角色就能访问的接口，直接在WebMvc中配置即可。如下：



1. 创建全局异常处理器，进行全局异常处理，主要是处理授权异常，如下：



7）启动项目，访问路径进行测试。