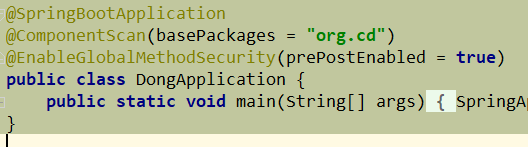
Spring Security 的应用越来越广泛，它支持**页面级别、API级别 以及方法级别**的权限控制，可以说项目中的绝大部分场景都能适用。不管哪种级别的权限控制，都需要进行以下的步骤 （使用Spring Boot项目演示）：

1. 加入Spring Boot Security的依赖 (不需要版本号，因为Spring boot的parent pom里面就已经有版本号了)：compile **"org.springframework.boot:spring-boot-starter-security"**

2. 启动类加注解

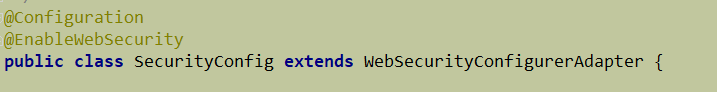
注意这里有个地雷，我不小心踩到过，如果你的Spring Boot版本比较低的话，切记要加上@ComponentScan注解，不然你会遇到绵绵不断的缺少各种依赖，能烦死。

@EnableGlobalMethodSecurity(prePostEnabled = true) 这个注解只是在你需要使用方法级别的权限认证的时候需要加的，否则可以不加。



3. 新加一个继承了WebSecurityConfigurerAdapter的配置类，这可以说是Spring Security的核心了，我们可以在这个类里面配置用户权限，过滤器，认证管理器，拦截路径等。

需要加上@Configuration注解和@EnableWebSecurity注解

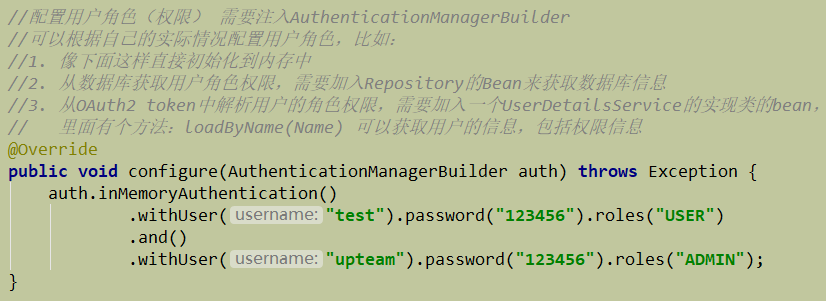


3.1. 在SecurityConfig中配置用户角色权限：

可以直接在启动的时候直接hard-coding方式将用户角色初始化到内存中；

也可以通过DataSource从数据库抓取用户角色信息；

另外还可以通过解析request里面的username & password动态的给用户分配角色权限。



3.2. 在SecurityConfig中配置路径拦截

配置路径和访问权限可以参考Spring Security的官方文档，大致简单介绍一下：

这个配置方式有点类似构建模式，一步步去构建HttpSecurity对象。

authorizeRequests()这个表示获取URL注册对象，得到这个对象之后可以进行一些URL匹配操作

antMatchers() 这个表示使用Ant格式来匹配路径，也就意味着你可以使用Ant里面的通配符

hasRole() 很明显，有参数里面指定的角色才可以访问这个路径

access() 参数里面是一个Spring EL表达式，只有当表达式返回true才允许访问

anyRequest() 表示所有请求，类似AntMathchers(“/\*\*”) 注意这个放到后面，如果放到前面，会覆盖掉前面指定的哪些具体路径

permitAll() 无条件让所有请求通过

denyAll() 无条件让所有请求不通过

@Override  
**public void** configure(HttpSecurity http) **throws** Exception {  
 http.csrf().disable().authorizeRequests()  
 .antMatchers(**"/upteam/test/useraccess"**).hasRole(**"USER"**)  
 .antMatchers(**"/upteam/test/aeminaccess"**).access(**"hasRole('ADMIN')"**)  
 .anyRequest().permitAll();  
 *//.and().exceptionHandling().accessDeniedHandler(myHandleAccessDeniedException);*}

上面这个配置的意思是首先禁用csrf，然后如果访问useraccess API需要有USER权限，如果访问 adminaccess API需要有ADMIN权限，其他的一律让它们通过，不做判断。

4. 对应的页面，API或方法：

@RequestMapping(value = **"/useraccess"**, method = RequestMethod.***POST***)  
**public** String testUserAccess() {  
 System.***out***.println(**"Comming in user access"**);  
 **return " ---- user access"**;  
}  
  
@RequestMapping(value = **"/adminaccess"**, method = RequestMethod.***POST***)  
**public** String testAdminAccess() {  
 System.***out***.println(**"Comming in admin access"**);  
 **return " ---- admin access"**;  
}

上面演示的这种适用于页面或API，因为是在配置里面通过路径去指定的，下面还要介绍一个方法级别的控制。

如果要加入方法级别的权限认证（当然，方法级别的也同样可以控制用户访问API，毕竟在Spring框架中 API是定义在方法上面的），那么：

1. 首先需要在启动类中加入@EnableGlobalMethodSecurity(prePostEnabled = true)这个注解。

2. 同样可以像上面介绍的那样在SecurityConfig里面去配置用户及用户的角色，当然更灵活的我们可以自己去定义适用于每个request的用户角色：

这是用户角色Bean，我自己定义它有一个hasRole方法，你也可以定义别的方法，只要到时候在方法上面的条件表达式能判断即可

**public class** UserAccess {  
  
 **public boolean** hasRole(String role) {  
 **if** (**"Read"**.equalsIgnoreCase(role)) {  
 **return true**;  
 }  
 **return false**;  
 }  
}

在上面那个SecurityConfig里面配置下这个用户角色的Bean，注意我这里将Bean的作用域定义成Request，也就是每个request过来都会创建一个新的Bean，而不是默认的单例Bean。这样我就可以根绝每个request带过来的信息初始化这个Bean 并为之授予相应的权限了。这里为了演示简单，我直接初始化的。

@Bean(**"userAccess"**)  
@Scope(value = **"request"**, proxyMode = ScopedProxyMode.***TARGET\_CLASS***)  
**public** UserAccess userAccessBean() {  
 **return new** UserAccess();  
}

另外，也可以加一个AccessDeniedHandler配置，自己定义当方法禁止访问时返回的消息：

@Component  
**public class** MyHandleAccessDeniedException **implements** AccessDeniedHandler {  
 @Override  
 **public void** handle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, AccessDeniedException e) **throws** IOException, ServletException {  
 httpServletResponse.setContentType(MediaType.***APPLICATION\_JSON\_UTF8\_VALUE***);  
 httpServletResponse.setStatus(HttpServletResponse.***SC\_FORBIDDEN***);  
 httpServletResponse.getWriter().write(**"{\"error\": \"access\_denied\", \"error\_description\": \""** + e.getMessage() + **"\"}"**);  
 }  
}

在SecurityConfig中配置一下这个AccessDeniedHandler才会起作用：

@Autowired  
**private** AccessDeniedHandler **myHandleAccessDeniedException**;

@Override  
**public void** configure(HttpSecurity http) **throws** Exception {  
 http.csrf().disable().authorizeRequests()  
 .antMatchers(**"/upteam/test/useraccess"**).hasRole(**"USER"**)  
 .antMatchers(**"/upteam/test/aeminaccess"**).access(**"hasRole('ADMIN')"**)  
 .anyRequest().permitAll()  
 .and().exceptionHandling().accessDeniedHandler(**myHandleAccessDeniedException**);  
}

3. 好了，接下来可以在方法上加入@PreAuthorize注解，里面定义一个表达式，只要结果返回true就可以访问这个方法，否则会走刚才定义的AccessDeniedHandle返回错误消息。

要先注入这个UserAccess Bean哦。

@Autowired  
**private** UserAccess **userAccess**;  
  
@PreAuthorize(**"@userAccess.hasRole('Read')"**)  
@RequestMapping(value = **"/readaccess"**, method = RequestMethod.***POST***)  
**public** String testRead() {  
 **userAccess**.hasRole(**"Read"**);  
 System.***out***.println(**"Comming in read access"**);  
 **return " ---- read access"**;  
}  
  
@PreAuthorize(**"@userAccess.hasRole('Write')"**)  
@RequestMapping(value = **"/writeaccess"**, method = RequestMethod.***POST***)  
**public** String testWrite() {  
 System.***out***.println(**"Comming in write access"**);  
 **return " ---- write access"**;  
}