企业级权限管理框架

[1. Spring Security 1](#_Toc45036201)

[1.1用户接口带权限 1](#_Toc45036202)

# Spring Security

## 1.1基础配置，用户带权限

Spring Security配置类

|  |
| --- |
| @EnableGlobalMethodSecurity(prePostEnabled=true)//开启用户角色校验  @Configuration  @Slf4j  public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {  @Autowired  private PasswordEncoder encoder;  //设置默认用户权限  @Override  protected void configure(AuthenticationManagerBuilder auth) throws Exception {  String pwd=encoder.encode("123456");  log.info(pwd);  auth.inMemoryAuthentication().withUser("admin").password(pwd).roles("ADMIN");  auth.inMemoryAuthentication().withUser("zs").password(pwd).roles("ADMIN");  auth.inMemoryAuthentication().withUser("user").password(pwd).roles("USER");  auth.inMemoryAuthentication().withUser("ls").password(pwd).roles("TOURIST");  }  //放行某些路径  @Override  protected void configure(HttpSecurity http) throws Exception {  http.authorizeRequests()  .antMatchers("/").permitAll()  .anyRequest().authenticated()  .and()  .logout().permitAll()  .and()  .formLogin();  http.csrf().disable();  }  //配置静态文件  @Override  public void configure(WebSecurity web) throws Exception {  web.ignoring().antMatchers("/js/\*\*", "/css/\*\*", "/images/\*\*");  }  //配置密码  @Bean  public PasswordEncoder passwordEncoder() {  // 使用 BCrypt 加密  return new BCryptPasswordEncoder();  }  } |

## 1.2接口校验

HomeController设置

|  |
| --- |
| @GetMapping("/hello")  public String hello(){  return "Hello Spring Security login";  }  @GetMapping("/role")  //权限用户  @PreAuthorize("hasRole('ROLE\_USER')")  public String roleUser(){  return "Hello Spring Security PreAuthorize hasRole";  } |

role接口只有有user角色的用户才能访问 -user用户

hello 所有用户都能访问

自定义自己的service查询

|  |
| --- |
| @Override  protected void configure(AuthenticationManagerBuilder auth) throws Exception {  //指定数据库查询方法  auth.userDetailsService(myUserService).passwordEncoder(encoder);  } |

## 1.3 权限配置

### 1.3.1 方法权限控制

表达式注解，控制权限,有四个

PreAuthorize 在方法调用前校验

PostAuthorize 在方法调用后校验

PreFilter 过滤参数中集合中的元素

PostFilter 过滤返回值中的元素

### 1.3.2 表达式

上面四个方法中写表达式为String类型

|  |
| --- |
| @PreAuthorize("hasRole('ROLE\_USER')") 判断是否有该角色 |

表达式中可添加and or等关联字

|  |
| --- |
| @PreAuthorize("hasRole('ROLE\_USER') or hasRole('ROLE\_ADMIN')") |

以上是对用户角色的校验

user用户名登录

1. 数的校验，参数中的id进行校验,超过10无权限

|  |
| --- |
| @PreAuthorize("#id<10") @GetMapping("/test") public String test(Integer id){  return "test auth"; } |

访问/test/param?id=10 提示403无权限

/test/param?id=1 返回正常

1. 是否是用户的名称

|  |  |
| --- | --- |
| @PreAuthorize("principal.username.equals(#username)") @GetMapping("/test") public String test( String username) {  return "test auth"; } | |
| 路径 | 接口 |
| /test/principal?username=user | 正常 |
| /test/principal?username=ab | 403 |

1. 对参数中的某个对象进行校验，例如User对象的name进行校验，指定为某一个人

|  |
| --- |
| @PreAuthorize("#user.username.equals('abc')") @GetMapping("/test/object") public String testUser(User user) {  return "test object"; } |

postman发送json请求，username必须为abc才能访问

1. PostAuthorize 在方法运行后进行校验

可以对返回值进行校验

例如返回值为Integer,随机生产一个数，能被2整除就正常

|  |
| --- |
| @PostAuthorize("returnObject%2==0") @GetMapping("/test/PostAu") public Integer testPostAuthorize(){  int i = RandomUtil.randomInt(1, 10);  log.info("随机数:"+i);  return i; } |

如果能被2整除，则返回值，没有则返回登录页面

返回对象时

|  |
| --- |
| @PostAuthorize("returnObject.age%2==0") @GetMapping("/test/postAuObject") public User testPostAuthObj(){  User u=new User();  int i = RandomUtil.randomInt(1, 10);  log.info("随机数:"+i);  u.setAge(i);  return u; } |