企业级权限管理框架

[1. Spring Security 1](#_Toc46671788)

[1.1基础配置，用户带权限 1](#_Toc46671789)

[1.2接口校验 2](#_Toc46671790)

[1.3 权限配置 3](#_Toc46671791)

[1.3.1 方法权限控制 3](#_Toc46671792)

[1.3.2 表达式 3](#_Toc46671793)

[2. shiro 4](#_Toc46671794)

# 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; } |

# shiro

## 2.1基础配置

创建Spring Boot项目，加入shiro,mybatis,hutool.mybatis-plus等依赖

|  |
| --- |
| dependencies {  implementation 'org.springframework.boot:spring-boot-starter-web'  implementation group: 'cn.hutool', name: 'hutool-all', version: '5.3.9'  implementation group: 'org.mybatis.spring.boot', name: 'mybatis-spring-boot-starter', version: '2.1.3'  implementation group: 'org.apache.shiro', name: 'shiro-core', version: '1.5.3'  implementation group: 'org.apache.shiro', name: 'shiro-spring', version: '1.5.3'  implementation group: 'mysql', name: 'mysql-connector-java', version: '8.0.16'  implementation group: 'com.alibaba', name: 'druid', version: '1.1.23'  compile group: 'com.baomidou', name: 'mybatis-plus-boot-starter', version: '3.3.2'  compileOnly 'org.projectlombok:lombok'  annotationProcessor 'org.projectlombok:lombok'  testImplementation('org.springframework.boot:spring-boot-starter-test') {  exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'  }  } |

创建对应的用户，角色，权限表，以及对应的关联表。见init.sql

创建对应的类和mapper文件。

Yml配置mybatis和plus相关文件。

|  |
| --- |
| # mybatis \u914D\u7F6E  mybatis:  mapper-locations: classpath\*:mapper/\*\*/\*\*/\*Mapper.xml  type-aliases-package: com.swb.\*.model  config-location: classpath:mybatis.xml  mybatis-plus:  global-config:  banner: false  db-config:  update-strategy: not\_null  insert-strategy: not\_empty  logic-delete-value: 1  logic-not-delete-value: 0 |

创建user对应的controller和service，验证方法的合理性。

## 2.2认值授权Realm

新建AuthRealm继承shiro的AuthorizingRealm，实现2个方法。

|  |
| --- |
| public class AuthRealm extends AuthorizingRealm {  @Autowired  private UserService userService;  /\*\*  \* 授权  \* @param principals  \* @return  \*/  @Override  protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  User user= (User) principals.fromRealm(this.getClass().getName()).iterator().next();  List<String> permissions;  Set<Role> roles=user.getRoles();  if(CollUtil.isNotEmpty(roles)){  permissions = roles.stream().map(Role::getPermissions)  .filter(CollUtil::isNotEmpty).flatMap(Set::stream).map(Permission::getName)  .distinct().collect(Collectors.toList());  }else{  permissions=new ArrayList<>();  }  SimpleAuthorizationInfo info=new SimpleAuthorizationInfo();  info.addStringPermissions(permissions);  return info;  }  /\*\*  \* 验证登录  \* @param token  \* @return  \* @throws AuthenticationException  \*/  @Override  protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {  UsernamePasswordToken usernamePasswordToken= (UsernamePasswordToken) token;  String username = usernamePasswordToken.getUsername();  User user=userService.findByUserName(username);  return new SimpleAuthenticationInfo(user,user.getPassword(),this.getClass().getName());  }  } |

第一个方法为鉴权，是获取用户权限的。

第二个访问是登录，通过token获取到Username,查询数据库获取用户信息。校验密码自定一个类CredentialMatcher，继承SimpleCredentialsMatcher类，覆盖doCredentialsMatch方法。

|  |
| --- |
| public class CredentialMatcher extends SimpleCredentialsMatcher {  @Override  public boolean doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) {  UsernamePasswordToken usernamePasswordToken= (UsernamePasswordToken) token;  String password = new String(usernamePasswordToken.getPassword());  String dbPassword= (String) info.getCredentials();  return StrUtil.equals(dbPassword,password);  }  } |

password为页面登录的密码，dbPassword为上一步创建的登录方法里传入的密码。

创建shiro配置类，指定realm，Matcher和SecurityManager, ShiroFilterFactoryBean配置相关信息

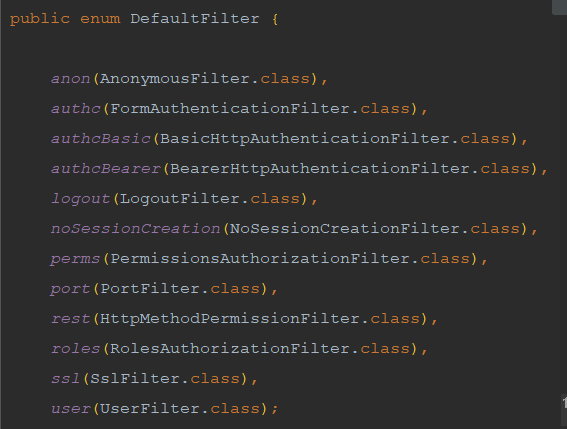
|  |
| --- |
| @Configuration  public class ShiroConfiguration {  @Bean("credentialMatcher")  public CredentialMatcher credentialMatcher(){  return new CredentialMatcher();  }  @Bean("authRealm")  public AuthRealm authRealm(@Qualifier(value = "credentialMatcher") CredentialMatcher credentialMatcher){  AuthRealm realm=new AuthRealm();  realm.setCredentialsMatcher(credentialMatcher);  return realm;  }  @Bean("securityManager")  public SecurityManager securityManager(@Qualifier(value = "authRealm") AuthRealm authRealm){  DefaultWebSecurityManager manager=new DefaultWebSecurityManager();  manager.setRealm(authRealm);  return manager;  }  @Bean  public ShiroFilterFactoryBean shiroFilter(@Qualifier(value = "securityManager") SecurityManager securityManager){  ShiroFilterFactoryBean bean=new ShiroFilterFactoryBean();  bean.setSecurityManager(securityManager);  bean.setLoginUrl("/login");  bean.setSuccessUrl("/index");  bean.setUnauthorizedUrl("/403");  LinkedHashMap<String,String> filterChain=new LinkedHashMap<>();  filterChain.put("/index","authc");  filterChain.put("/login","anon");  bean.setFilterChainDefinitionMap(filterChain);  return bean;  }  } |

依赖关系如下：CredentialMatcher -> AuthRealm -> SecurityManager -> ShiroFilterFactoryBean

之后配置shiro与Spring的关联。

|  |
| --- |
| @Bean AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(  @Qualifier(value = "securityManager") SecurityManager securityManager) {  AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdvisor();  advisor.setSecurityManager(securityManager);  return advisor; } @Bean public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator(){  DefaultAdvisorAutoProxyCreator creator=new DefaultAdvisorAutoProxyCreator();  creator.setProxyTargetClass(true);  return creator; } |

filterChainDefinitionMap添加的antuc和anon含义。DefaultFilter中存在很多的过滤器，，一面制定了很多默认的过滤器



Map中指定的key对应的url会由value对应的过滤器处理。shiroFilter定义自己的验证方式，以及登录操作等。