目录

**[1、依赖](#_Toc1128199777_WPSOffice_Level1)** **[1](#_Toc1128199777_WPSOffice_Level1)**

**[2、配置注解](#_Toc1955254043_WPSOffice_Level1)** **[2](#_Toc1955254043_WPSOffice_Level1)**

**[3、使用](#_Toc1159934307_WPSOffice_Level1)** **[2](#_Toc1159934307_WPSOffice_Level1)**

[3.1、用户配置](#_Toc1955254043_WPSOffice_Level2) [2](#_Toc1955254043_WPSOffice_Level2)

[3.1、静态的用户名密码进行配置](#_Toc1159934307_WPSOffice_Level2) [2](#_Toc1159934307_WPSOffice_Level2)

[3.2、登陆页面](#_Toc159350283_WPSOffice_Level2) [4](#_Toc159350283_WPSOffice_Level2)

[3.3、不拦截系统资源](#_Toc288098572_WPSOffice_Level2) [5](#_Toc288098572_WPSOffice_Level2)

[3.4、跳转登陆/登出成功界面](#_Toc1644559266_WPSOffice_Level2) [6](#_Toc1644559266_WPSOffice_Level2)

[3.5、自定义登陆成功/登陆失败/登出的操作](#_Toc1993046772_WPSOffice_Level2) [6](#_Toc1993046772_WPSOffice_Level2)

[3.6、跨域请求](#_Toc687171098_WPSOffice_Level2) [11](#_Toc687171098_WPSOffice_Level2)

[3.7、url区分不同的登录失败场景](#_Toc117590520_WPSOffice_Level2) [11](#_Toc117590520_WPSOffice_Level2)

[3.8、访问权限投票器](#_Toc658914400_WPSOffice_Level2) [11](#_Toc658914400_WPSOffice_Level2)

[3.9、访问决策管理器](#_Toc1948636868_WPSOffice_Level2) [12](#_Toc1948636868_WPSOffice_Level2)

[3.10、自定义用户名密码参数名及用户名密码验证路径](#_Toc1614223726_WPSOffice_Level2) [12](#_Toc1614223726_WPSOffice_Level2)

[3.11、添加验证码](#_Toc1097250331_WPSOffice_Level2) [13](#_Toc1097250331_WPSOffice_Level2)

[3.12、RememberMe（记住我）](#_Toc1044236328_WPSOffice_Level2) [13](#_Toc1044236328_WPSOffice_Level2)

**1、依赖**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

**2、配置注解**

需要添加 **@EnableWebSecurity、@Configuration**两个注解

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

}

**3、使用**

**3.1、用户配置**

**3.1、静态的用户名密码进行配置**

spring:

security:

user:

name: zhangsan

password: 123456

在配置文件中配置的用户信息，包括用户名、密码、角色等，而这些，会被初始化到**InMemoryUserDetailsManager**中，即存储在内存中。

**3.1.2、动态用户**

但是，在实际的业务场景中，不可能只有一个用户，更不可能用户信息都是固定的，而是动态的，需要从存储的地方获取，如数据库。

自定义 **UserDetailsService**，密码加密方式使用是 **BCryptPasswordEncoder**。

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Autowired

private UserDao userDao;

@Override

public void configure(WebSecurity web) throws Exception {

...

}

@Override

protected void configure(HttpSecurity http) throws Exception {

...

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(customJdbcUserDetailsService()).passwordEncoder(new BCryptPasswordEncoder());

}

private UserDetailsService customJdbcUserDetailsService() {

CustomJdbcUserDetailsService userDetailsService = new CustomJdbcUserDetailsService();

userDetailsService.setUserDao(userDao);

return userDetailsService;

}

}

//自定义 UserDetailsService

public class CustomJdbcUserDetailsService implements UserDetailsService {

private UserDao userDao;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

SysUser sysUser = this.userDao.getByUsername(username);

UserDetails user = User.builder().username(sysUser.getUsername()).password(sysUser.getPassword()).roles("User").build();

return user;

}

public UserDao getUserDao() {

return userDao;

}

public void setUserDao(UserDao userDao) {

this.userDao = userDao;

}

}

**3.1.3、动态角色**

public class CustomJdbcUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

SysUser sysUser = this.userDao.getByUsername(username);

User.UserBuilder builder = User.builder()

.username(sysUser.getUsername())

.password(sysUser.getPassword());

List<String> roles = this.roleDao.listRoleCodeByUserId(sysUser.getId());

if (!CollectionUtils.isEmpty(roles)) {

builder.roles(roles.toArray(new String[] {}));

}

return builder.build();

}

}

**3.1.4、动态资源**

自定义拦截器

自定义 **FilterSecurityInterceptor**。注意，该类**没有任何实质性内容，空类，但不可省略**。

public class CustomFilterSecurityInterceptor extends FilterSecurityInterceptor {

}

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

...

http.addFilterAfter(customFilterSecurityInterceptor(), FilterSecurityInterceptor.class);

}

private FilterSecurityInterceptor customFilterSecurityInterceptor() throws Exception {

CustomFilterSecurityInterceptor filterSecurityInterceptor = new CustomFilterSecurityInterceptor();

filterSecurityInterceptor.setSecurityMetadataSource(new DefaultFilterInvocationSecurityMetadataSource(obtainRequestMap()));

filterSecurityInterceptor.setAccessDecisionManager(accessDecisionManager());

filterSecurityInterceptor.setAuthenticationManager(authenticationManager());

return filterSecurityInterceptor;

}

}

注：如果是动态资源配置需要把 Controller 中所有 @PreAuthorize 注解 和 Spring Security 配置类中的 @EnableGlobalMethodSecurity 注解删除

**3.2、登陆页面**

Spring Security框架默认的登录页但是界面元素太少，况且不易去调整，也不满足日常的场景。所以，大多数情况下，我们需要自定义登录页。

自定义登录页的实现只需要一个登录页面、一个Controller、Spring Security配置稍微调整

@Controller

public class LoginController {

@RequestMapping("/login")

public String login() {

return "login";

}

}

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin().loginPage("/login").permitAll()//指定登录页的路径

.and()

.authorizeRequests()//返回一个授权对象

//尚未匹配的任何URL都要求用户进行身份验证

.anyRequest().authenticated();

}

}

**3.3、不拦截系统资源**

方式1、重写configure(WebSecurity web)方法

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

public void configure(WebSecurity web) throws Exception {

web.ignoring().antMatchers("/css/\*\*", "/js/\*\*", "/plugins/\*\*", "/images/\*\*", "/fonts/\*\*");

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin().loginPage("/login").permitAll()

.and()

.authorizeRequests()

.anyRequest()

.authenticated();

}

}

方式2、重写configure(HttpSecurity http)方法

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin().loginPage("/login").permitAll()

.and()

.authorizeRequests()

.antMatchers(HttpMethod.GET, // 允许对于网站静态资源的无授权访问

"/",

"/\*.html",

"/favicon.ico",

"/\*\*/\*.html",

"/\*\*/\*.css",

"/\*\*/\*.js",

"/swagger-resources/\*\*",

"/v2/api-docs/\*\*").permitAll()

.anyRequest().authenticated();

}

}

**3.4、跳转登陆/登出成功界面**

访问一个系统时，一般会重定向到我们收藏的某个页面，或者首页，又或者是访问根路径。如果并没有明确的目的指向，或者说我们想要让系统在登录成功后，一直指向某个页面的操作。

添加一项 **defaultSuccessUrl** 配置即可

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

//指定登录页的路径，登陆成功跳转页面和登陆失败跳转页面

http.formLogin()

.loginPage("/login")

.defaultSuccessUrl("/index")

.failureUrl("/login?error").permitAll()//自定义用户登录失败页面

.and()

.authorizeRequests()//返回一个授权对象

//尚未匹配的任何URL都要求用户进行身份验证

.anyRequest().authenticated()

//指定退出url和退出成功跳转页面

.and().logout()

.logoutUrl("/logout")

.logoutSuccessUrl("/logout\_success").permitAll();

}

}

**3.5、自定义登陆成功/登陆失败/登出的操作**

**3.5.1、登陆成功**

在登录某个网站之后，微信、短信、邮箱可能会接受到一条这样的信息/邮件，或者记录每一次的登录信息到数据库、到日志文件等等，方便后续做审计、分析。

自定义实现类的方法：继承**SavedRequestAwareAuthenticationSuccessHandler类重写**onAuthenticationSuccess方法。

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin().loginPage("/login").successHandler(customAuthenticationSuccessHandler()).permitAll()

.and()

.authorizeRequests()//返回一个授权对象

//尚未匹配的任何URL都要求用户进行身份验证

.anyRequest().authenticated()

.and().logout().logoutSuccessUrl("/logout\_success").permitAll();

}

}

@Bean

public AuthenticationSuccessHandler customAuthenticationSuccessHandler() {

CustomSavedRequestAwareAuthenticationSuccessHandler customAuthenticationSuccessHandler = new CustomSavedRequestAwareAuthenticationSuccessHandler();

//指定登录页的路径

customAuthenticationSuccessHandler.setDefaultTargetUrl("/index");

customAuthenticationSuccessHandler.setEmailService(emailService);

customAuthenticationSuccessHandler.setSmsService(smsService);

customAuthenticationSuccessHandler.setWeChatService(wechatService);

return customAuthenticationSuccessHandler;

}

public class CustomSavedRequestAwareAuthenticationSuccessHandler extends SavedRequestAwareAuthenticationSuccessHandler {

private EmailService emailService;

private SmsService smsService;

private WeChatService weChatService;

@Override

public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws ServletException, IOException {

super.onAuthenticationSuccess(request, response, authentication);

this.logger.info(String.format("IP %s，用户 %s， 于 %s 成功登录系统。", request.getRemoteHost(), authentication.getName(), LocalDateTime.now()));

try {

this.emailService.send();// 发邮件

this.smsService.send();// 发短信

this.weChatService.send();// 发微信

} catch (Exception ex) {

this.logger.error(ex.getMessage(), ex);

}

}

...

}

**3.5.2、登出失败**

记录登录失败时的IP、时间、SessionId；发送登录失败提醒到微信、邮箱、短信，提醒用户当前登录失败事件；同时记录到日志中，或者发送到远端日志监控平台，分析是否是攻击行为等等。

自定义实现类的方法：继承**SavedRequestAwareAuthenticationSuccessHandler类重写**onAuthenticationSuccess方法。

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin()

.loginPage("/login")

.successHandler(customAuthenticationSuccessHandler())

.failureHandler(customSimpleUrlAuthenticationFailureHandler())

.permitAll()

.and()

.authorizeRequests()//返回一个授权对象

//尚未匹配的任何URL都要求用户进行身份验证

.anyRequest().authenticated()

.and()

.logout().logoutSuccessHandler(customLogoutSuccessHandler()).permitAll();

}

}

@Bean

public LogoutSuccessHandler customLogoutSuccessHandler() {

CustomLogoutSuccessHandler customLogoutSuccessHandler = new CustomLogoutSuccessHandler();

customLogoutSuccessHandler.setDefaultTargetUrl("/logout\_success");

customLogoutSuccessHandler.setEmailService(emailService);

customLogoutSuccessHandler.setSmsService(smsService);

customLogoutSuccessHandler.setWeChatService(weChatService);

return customLogoutSuccessHandler;

}

@Bean

public AuthenticationFailureHandler customSimpleUrlAuthenticationFailureHandler() {

CustomSimpleUrlAuthenticationFailureHandler customSimpleUrlAuthenticationFailureHandler = new CustomSimpleUrlAuthenticationFailureHandler();

customSimpleUrlAuthenticationFailureHandler.setDefaultFailureUrl("/login\_fail");

customSimpleUrlAuthenticationFailureHandler.setEmailService(emailService);

customSimpleUrlAuthenticationFailureHandler.setSmsService(smsService);

customSimpleUrlAuthenticationFailureHandler.setWeChatService(weChatService);

return customSimpleUrlAuthenticationFailureHandler;

}

public class CustomLogoutSuccessHandler extends SimpleUrlLogoutSuccessHandler {

private EmailService emailService;

private SmsService smsService;

private WeChatService weChatService;

@Override

public void onLogoutSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException, ServletException {

super.onLogoutSuccess(request, response, authentication);

this.logger.info(String.format("IP %s，用户 %s， 于 %s 退出系统。", request.getRemoteHost(), authentication.getName(), LocalDateTime.now()));

try {

this.emailService.send();// 发邮件

this.smsService.send();// 发短信

this.weChatService.send();// 发微信

} catch (Exception ex) {

this.logger.error(ex.getMessage(), ex);

}

}

...

}

public class CustomSimpleUrlAuthenticationFailureHandler extends SimpleUrlAuthenticationFailureHandler {

private static final String DEFAULT\_FAILURE\_URL = "/login\_fail";

@Override

public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response, AuthenticationException exception) throws IOException, ServletException {

super.onAuthenticationFailure(request, response, exception);

this.logger.info(String.format("IP %s 于 %s 尝试登录系统失败，失败原因：%s", request.getRemoteHost(), LocalDateTime.now(), exception.getMessage()));

try {

// 发邮件

this.emailService.send();

// 发短信

this.smsService.send();

// 发微信

this.weChatService.send();

} catch (Exception ex) {

this.logger.error(ex.getMessage(), ex);

}

}

...

}

**3.5.3、登出**

在用户登出成功后，我们也要通过邮件、短信、微信，来通知用户，在什么时间，什么地点，退出了系统。更甚至，可以通知用户本次登录都操作了那些功能，做了哪些操作等等。

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin().loginPage("/login").successHandler(customAuthenticationSuccessHandler()).permitAll()

.and()

.authorizeRequests()//返回一个授权对象

//尚未匹配的任何URL都要求用户进行身份验证

.anyRequest().authenticated()

.and()

.logout().logoutSuccessHandler(customLogoutSuccessHandler()).permitAll();

}

}

@Bean

public LogoutSuccessHandler customLogoutSuccessHandler() {

CustomLogoutSuccessHandler customLogoutSuccessHandler = new CustomLogoutSuccessHandler();

customLogoutSuccessHandler.setDefaultTargetUrl("/logout\_success");

customLogoutSuccessHandler.setEmailService(emailService);

customLogoutSuccessHandler.setSmsService(smsService);

customLogoutSuccessHandler.setWeChatService(weChatService);

return customLogoutSuccessHandler;

}

public class CustomLogoutSuccessHandler extends SimpleUrlLogoutSuccessHandler {

private EmailService emailService;

private SmsService smsService;

private WeChatService weChatService;

@Override

public void onLogoutSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException, ServletException {

super.onLogoutSuccess(request, response, authentication);

this.logger.info(String.format("IP %s，用户 %s， 于 %s 退出系统。", request.getRemoteHost(), authentication.getName(), LocalDateTime.now()));

try {

this.emailService.send();// 发邮件

this.smsService.send();// 发短信

this.weChatService.send();// 发微信

} catch (Exception ex) {

this.logger.error(ex.getMessage(), ex);

}

}

...

}

**3.6、跨域请求**

在 **Spring Security** 用户登出配置中，**CSRF** 默认是开启的，并没有关闭

关闭跨域请求

@EnableWebSecurity

@Configuration

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable();

}

}

**3.7、url区分不同的登录失败场景**

FAILURE(0, "登录失败！"),​

BADCREDENTIALS(1, "用户名密码错误！"),​

LOCKED(2, "用户已被锁定，无法登录！"),​

ACCOUNTEXPIRED(3, "用户已过时，无法登录！"),​

USERNAMENOTFOUND(4, "用户不存在！");

**3.8、访问权限投票器**

**AccessDecisionVoter**是一个投票器，负责对授权决策进行表决。然后，最终由唱票者**AccessDecisionManager**统计所有的投票器表决后，来做最终的授权决策。

Web[Express](https://so.csdn.net/so/search?from=pc_blog_highlight&q=Express)ionVoter（默认）：基于 Spring-EL进行控制权限

AuthenticatedVoter：认证投票器

PreInvocationAuthorizationAdviceVoter：注解投票器

RoleVoter：角色投票器。

RoleHierarchyVoter：基于 RoleVoter，唯一的不同就是该投票器中的角色是附带上下级关系的。也就是说，角色A包含角色B，角色B包含 角色C，此时，如果用户拥有角色A，那么理论上可以同时拥有角色B、角色C的全部资源访问权限。

**自定义访问权限投票器**

**3.9、访问决策管理器**

**AccessDecisionManager** 顾名思义，访问决策管理器。

常用的 **AccessDecisionManager** 有三个：乐观、共识、一致

AffirmativeBased（默认）基于乐观：只要任一 AccessDecisionVoter 返回肯定的结果，便授予访问权限。

ConsensusBased  基于共识：少数服从多数授权访问决策方案。

如果授予权限和拒绝权限相等时的逻辑。提供了 allowIfEqualGrantedDeniedDecisions 参数，用于给用户提供自定义的机会，其默认值为 true，即代表允许授予权限和拒绝权限相等，且同时也代表授予访问权限。

UnanimousBased 基于一致： 最严格的的授权决策器。要求所有 AccessDecisionVoter 均返回肯定的结果时，才代表授予权限。

也可以按照自身业务场景进行自定义访问决策管理器

**3.10、自定义用户名密码参数名及用户名密码验证路径**

默认的登陆路径为/login，且用户名密码参数为username, password

可以自定义这些默认配置

protected void configure(HttpSecurity http) throws Exception {

http.formLogin()

.loginProcessingUrl("/j\_spring\_security\_check") //修改默认登陆地址

.usernameParameter("j\_username")

.passwordParameter("j\_password")

.loginPage("/login")

.defaultSuccessUrl("/index")

.failureUrl("/login\_fail")

.permitAll()

......

}

**3.11、添加验证码**

//引入第三方包，里面有验证码算法类

<dependency>

<groupId>cn.hutool</groupId>

<artifactId>hutool-all</artifactId>

</dependency>

//验证码生成

@GetMapping("/captcha/generate")

public void captchaGenerate(HttpSession session, HttpServletResponse response) {}

//自定义Filter,Spring Security 框架默认的 UsernamePasswordAuthenticationFilter

//中并没有针对验证码的处理，只有用户名和密码。因此，我们需要自定义一个包含验证码验证的Filter。

//改造WebSecurityConfigurerAdapter

http.addFilterAt(usernamePasswordAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

**3.12、RememberMe（记住我）**

RememberMeServices rememberMeServices = new NullRememberMeServices();

//认证成功后会执行

rememberMeServices.loginSuccess(request, response, authResult);

//RememberMeServices 接口的 loginSuccess 便会判断登录页的 记住我 复选框是否选中，选中的情况下，才会执行后续逻辑

public final void loginSuccess(HttpServletRequest request,

HttpServletResponse response, Authentication successfulAuthentication) {​

if (!rememberMeRequested(request, parameter)) {

logger.debug("Remember-me login not requested.");

return;

}​

onLoginSuccess(request, response, successfulAuthentication);

}

调整配置

protected void configure(HttpSecurity http) throws Exception {

http

......

.rememberMe()

.userDetailsService(userDetailsService())

.tokenValiditySeconds(14 \* 24 \* 60 \* 60)

......

​

}

基于 **RememberMeServices** 接口的实现 **TokenBasedRememberMeServices**。此实现是将 **username**、**过期时间**、**password**、**key**等按照一定的规则组合之后取MD5值，之后将此值存在浏览器的Cookie中。大部分网站的RememberMe 实现方式都是基于此。但是，将用户的 password 存储在浏览器的 Cookie 中.

Spring Security 框架提供了另外一种实现方式，基于 **数据库**和 **Cookie** 的实现方式：**PersistentTokenBasedRememberMeServices。**