答案显然是否定的,要操作这些功能必须首先登录到系统才可以。 (用户登录系 ****问题2****: 是不是所有用户,只要登录成功就都可以操作所有功能呢? 答案是否定的,并不是所有的用户都可以操作这些功能。不同的用户可能拥有不 同的权限,这就需要进行授权了。(用户登录之后,对每个用户进行授权,通过 认证和授权概念 授权去访问系统中不同的功能->授权) ****认证****: 系统提供的用于识别用户身份的功能,通常提供用户名和密码进 行登录其实就是在进行认证,认证的目的是让系统知道你是谁。 ****授权****: 用户认证成功后,需要为用户授权,其实就是指定当前用户可以 简介 操作哪些功能。 本章节就是要对后台系统进行权限控制,其本质就是对用户进行认证和授权。 Spring Security是 Spring提供的安全认证服务的框架。 使用Spring Security可 以帮助我们来简化认证和授权的过程。 <!-- spring security安全框架 --> <dependency> <groupId>org.springframework.security</groupId> <artifactId>spring-security-web</artifactId> <version>5.2.7.RELEASE</version> </dependency> 导入maven坐标 <dependency> <groupId>org.springframework.security</groupId> <artifactId>spring-security-config</artifactId> <version>5.2.7.RELEASE</version> </dependency> <!-- SpringSecurity Filter --> <!-- DelegatingFilterProxy用于整合第三方框架(代理过滤器,非真正的过滤器,真正的过滤器需要在spring的配置文件) --> <filter-name>springSecurityFilterChain</filter-name> <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> 配置Spring Security Fillter web.xml </filter> <filter-mapping> <filter-name>springSecurityFilterChain</filter-name> <url-pattern>/*</url-pattern> </filter-mapping> package com.example.config; import org.springframework.context.annotation.Configuration; import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity; import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter; 配置Spring Security java类配置 @Configuration @EnableWebSecurity //@EnableWebSecurity是开启SpringSecurity的默认行为 public class WebSecurityConfig extends WebSecurityConfigurerAdapter { 仅需三个步骤,我们就已经集成好了Spring Security,其他的事情就可以交给Spring Security为我们处理。 启动项目 访问: http://localhost:8000/ 所有资源访问受限 (包括静态资源) url自动跳转到了一个默认的登录页面(框架自带的),我们目前没有定义login页面及login controller方法。 但是当前没有账号啊!下面我们测试一个最简单的内存分配用户名密码。 基本使用 @Override//后续注释到此方法,从数据库中查找用户 protected void configure(AuthenticationManagerBuilder auth) throws Exception { auth.inMemoryAuthentication() WebSecurityConfig 内存分配用户名密码 .withUser("lucy") 重写configure(AuthenticationManagerBuilder auth)方法 .password(new BCryptPasswordEncoder().encode("123456")) 请求: http://localhost:8000/ 报错: `springsecurity There is no PasswordEncoder mapped for the id "null"` 需要设置加密方式 * 必须指定加密方式,上下加密方式要一致 * @return 上下加密方式要一致 设置加密方式 @Bean public PasswordEncoder passwordEncoder(){ return new BCryptPasswordEncoder(); 登录成功,但是iframe部分页面不显示 @Override 创建UserDetails可以注释到此方法 protected void configure(HttpSecurity http) throws Exception { //必须调用父类的方法,否则就不需要认证即可访问 super.configure(http); 设置允许iframe嵌套显示 //允许iframe嵌套显示 默认Spring Security不允许iframe嵌套显示,我们需要设置 //http.headers().frameOptions().disable(); //允许iframe显示 http.headers().frameOptions().sameOrigin(); 到目前为止,我们通过内存分配用户名密码的方式,可以访问后台页面了。 public class MyUserDetailsService implements UserDetailsService { @Reference private AdminService adminService; //登录时,SpringSecurity会自动调用该方法,并将前端用户名传入到方法中 @Override public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException { Admin admin = adminService.getAdminByUserName(username); if(admin == null){ 创建MyUserDetailsService类继承UserDetailsService接口 ----throw new UsernameNotFoundException("用户不存在!"); //这里传入数据库中查出的密码会和前端传入的密码进行校验 springsecurity return new User(username,admin.getPassword(), AuthorityUtils.commaSeparatedStringToAuthorityList("")); @Configuration @EnableWebSecurity//开启spring security的自动配置,会给我们生成一个登录 public class MySpringSecurityConfig extends WebSecurityConfigurerAdapter { //在内存中设置一个认证的用户名和密码 /* @Override//已经被注释调 protected void configure(AuthenticationManagerBuilder auth) throws Exception { auth.inMemoryAuthentication().withUser("admin") .password(new BCryptPasswordEncoder().encode("111111")) .roles(""); //创建一个密码加密器放到IOC容器中 public PasswordEncoder getPasswordEncoder(){ return new BCryptPasswordEncoder(); @Override protected void configure(HttpSecurity http) throws Exception { 修改configure方法后 //必须调用父类的方法,否则认证过程将失效,除非当前方法配置了认证 MySpringSecurityConfig最终配置如下 //super.configure(http);//当前已配置认证可以注释调 //配置允许iframe标签访问 http.headers().frameOptions().sameOrigin(); //配置可以匿名访问的资源 http.authorizeRequests().antMatchers("/static/**","/login") .permitAll().anyRequest().authenticated(); //配置自定义登录页面 http.formLogin().loginPage("/login")//配置去自定义页面访问的路径 .defaultSuccessUrl("/");//配置登录成功之后前往的地址 配置指定登录页面 //配置登出的地址及登出成功之后去往的地址 修改configure方法 http.logout().logoutUrl("/logout").logoutSuccessUrl("/login"); //关闭跨域请求伪造: 防止钓鱼网站,springsecurity自带的登录页面会生成一个 令牌进行效验,如果请求没有令牌,则服务器不处理次请求, //我们写的是静态页面,没有生成令牌,故将其令牌校验关闭 http.csrf().disable(); 配置登录页面后记得重写login.html页面,此时 springsecurity不会使用默认登录页面 //通过Spring Security获取User对象 User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal(); 登录成功后获取用户对象 String username = user.getUsername(); Admin admin = adminService.getAdminByUserName(username); //获取当前用户权限的方法 List<String> permissionCodes = permissionService.getPermissionCodesByAdminId(admin.getId()); //创建一个用于授权的对象 List<GrantedAuthority> grantedAuthorities = new ArrayList<>(); for (String permissionCode : permissionCodes) { if(!StringUtils.isEmpty(permissionCode)){ 加入获取用户权限的代码块 //将查出来的权限封装成SimpleGrantedAuthority对象 SimpleGrantedAuthority simpleGrantedAuthority = new SimpleGrantedAuthority(permissionCode); //SimpleGrantedAuthority对象放到用于授权的对象集合中 grantedAuthorities.add(simpleGrantedAuthority); return new User(username,admin.getPassword(),grantedAuthorities); @Component public class MyUserDetailsService implements UserDetailsService { 获取用户的权限 更新MyUserDetailsService方法 private PermissionService permissionService; @Reference private AdminService adminService; //登录时,SpringSecurity会自动调用该方法,并将前端用户名传入到方法中 @Override public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException { Admin admin = adminService.getAdminByUserName(username); if(admin == null){ throw new UsernameNotFoundException("用户不存在!"); //获取当前用户权限的方法 List<String> permissionCodes = permissionService.getPermissionCodesByAdminId(admin.getId()); 完整代码 //创建一个用于授权的对象 List<GrantedAuthority> grantedAuthorities = new ArrayList<>(); for (String permissionCode : permissionCodes) { if(!StringUtils.isEmpty(permissionCode)){ SimpleGrantedAuthority simpleGrantedAuthority = new SimpleGrantedAuthority(permissionCode); grantedAuthorities.add(simpleGrantedAuthority); 进阶使用 //给用户授权 //这里传入数据库中查出的密码会和前端传入的密码进行校验 /*return new User(username,admin.getPassword(), AuthorityUtils.commaSeparatedStringToAuthorityList(""));*/ return new User(username,admin.getPassword(),grantedAuthorities); @Configuration @EnableWebSecurity//开启spring security的自动配置,会给我们生成一个登录 @EnableGlobalMethodSecurity(prePostEnabled = true)//开启Controller中 方法的权限控制 public class MySpringSecurityConfig extends
WebSecurityConfigurerAdapter { //在内存中设置一个认证的用户名和密码 /* @Override 要禁用用户权限,必须先获取用户权限 protected void configure(AuthenticationManagerBuilder auth) throws Exception { auth.inMemoryAuthentication().withUser("admin") .password(new BCryptPasswordEncoder().encode("111111")) .roles(""); **}*/** //创建一个密码加密器放到IOC容器中 public PasswordEncoder getPasswordEncoder(){ return new BCryptPasswordEncoder(); 在MySpringSecurityConfig方法中加上注解 @Override @EnableGlobalMethodSecurity(prePostEnabled = true)// 完整代码 protected void configure(HttpSecurity http) throws Exception { 开启Controller中方法的权限控制 //必须调用父类的方法,否则认证过程将失效,除非当前方法配置了认证 //super.configure(http); //配置允许iframe标签访问 http.headers().frameOptions().sameOrigin(); //配置可以匿名访问的资源 http.authorizeRequests().antMatchers("/static/**","/login") .permitAll().anyRequest().authenticated(); //配置自定义登录页面 http.formLogin().loginPage("/login")//配置去自定义页面访问的路径 .defaultSuccessUrl("/");//配置登录成功之后前往的地址 //配置登出的地址及登出成功之后去往的地址 http.logout().logoutUrl("/logout").logoutSuccessUrl("/login"); //关闭跨域请求伪造: 防止钓鱼网站, springsecurity 自带的登录页面会生成一个令 牌进行效验,如果请求没有令牌,则服务器不处理次请求, //我们写的是静态页面,没有生成令牌,故将其令牌校验关闭 http.csrf().disable(); 在禁用该角色操作的方法上加上@PreAuthorize("hasAuthority('权限内容')") 禁用用户权限(本质: 禁用该操作的方法) @PreAuthorize("hasAuthority('Delete')") 此时只有Delete(使用该方法的角色在重权限表中查出来的字段要和单引号中的内 容一模一样才能执行该方法)权限的人才能删 @PreAuthorize("hasAuthority('Delete')")//此时只有Delete(使用该方法的角色 在重权限表中查出来的字段要和单引号中的内容一模一样才能执行该方法)权限的 @RequestMapping("/delete/{roleId}") 示例 public String delete(@PathVariable("roleId")Long roleId){ roleService.delete(roleId); //重定向到查询所以有角色的方法 return "redirect:/role"; public class CustomAccessDeniedHandler implements AccessDeniedHandler { public void handle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, AccessDeniedException e) throws IOException, ServletException { //用户使用了他没有的权限操作后,将页面进行重定向 创建CustomAccessDeniedHandler类 httpServletResponse.sendRedirect("/auth"); //去没有权限的提示页面 定义用户使用禁用操作后去往的页面 @RequestMapping("/auth") 同时在IndexController中定义控制器 public String auth(){ 填入的对象 return "frame/auth"; //配置自定义的无权限访问的处理器(所填入的对象就是你创建的CustomAccessDeniedHandler类) 在MySpringSecurityConfig中配置自定义的无权限访问的处理器 http.exceptionHandling().accessDeniedHandler(new CustomAccessDeniedHandler()); <thymeleaf-springsecurity5.version>3.0.4.RELEASE</thymeleaf-springsecurity5.version> <!--用于springsecurity5标签--> <dependency> 添加依赖管理 <groupId>org.thymeleaf.extras</groupId> <artifactId>thymeleaf-extras-springsecurity5</artifactId> <version>\${thymeleaf-springsecurity5.version}</version> </dependency> <!--配置模板引擎--> <bean id="templateEngine" class="org.thymeleaf.spring5.SpringTemplateEngine"> <!--引用视图解析器--> roperty name="templateResolver" ref="templateResolver"> <!-- 添加spring security 标签支持: sec --> 模板引擎配置spring security 标签支持 当时用thymeleaf时对页面功能按钮是否显示的权限控制 property name="additionalDialects"> 修改spring-mvc.xml,在模板引擎配置spring security 标签支持 <set> <bean class="org.thymeleaf.extras.springsecurity5.dialect.SpringSecurityDialect" /> </property> </bean> <html xmlns:th="http://www.thymeleaf.org" ■ 在html文件里面申明使用spring-security标签 xmlns:sec="http://www.thymeleaf.org/extras/spring-security"> 按钮上使用标签 <button type="button" class="btn btn-sm btn-primary create" sec:authorize="hasAuthority('role.create')">新增

****问题1****: 在生产环境下我们如果不登录后台系统就可以完成这些功能操