## 框架所扮演的角色

|  |
| --- |
| 在ssm项目中使用springsecurity框架，其相当于Controller |

## web.xml

|  |
| --- |
| <!-- 配置加载类路径的配置文件 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath\*:applicationContext.xml,classpath\*:spring-security.xml</param-value>  </context-param>  <!-- 配置监听器 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <filter>  <filter-name>springSecurityFilterChain</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  </filter>  <filter-mapping>  <filter-name>springSecurityFilterChain</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

## spring-security.xml

|  |
| --- |
| <security:global-method-security pre-post-annotations="enabled" jsr250-annotations="enabled" secured-annotations="enabled"></security:global-method-security>  <!-- 配置不拦截的资源 -->  <security:http pattern="/login.jsp" security="none"/>  <security:http pattern="/failer.jsp" security="none"/>  <security:http pattern="/css/\*\*" security="none"/>  <security:http pattern="/img/\*\*" security="none"/>  <security:http pattern="/plugins/\*\*" security="none"/>  <!--  配置具体的规则  auto-config="true" 不用自己编写登录的页面，框架提供默认登录页面  use-expressions="false" 是否使用SPEL表达式  -->  <security:http auto-config="true" use-expressions="true">  <!-- 配置具体的拦截的规则 pattern="请求路径的规则" access="访问系统的人，必须有ROLE\_USER的角色" -->  <security:intercept-url pattern="/\*\*" access="hasAnyRole('ROLE\_USER','ROLE\_ADMIN')"/>  <!-- 定义跳转的具体的页面 -->  <security:form-login  login-page="/login.jsp"  login-processing-url="/login.do"  default-target-url="/index.jsp"  authentication-failure-url="/failer.jsp"  authentication-success-forward-url="/pages/main.jsp"  />  <!-- 关闭跨域请求 -->  <security:csrf disabled="true"/>  <!-- 退出 -->  <security:logout invalidate-session="true" logout-url="/logout.do" logout-success-url="/login.jsp"/>  </security:http>  <!-- 切换成数据库中的用户名和密码 -->  <security:authentication-manager>  <security:authentication-provider user-service-ref="userService">  <!-- 配置加密的方式-->  <security:password-encoder ref="passwordEncoder"/>  </security:authentication-provider>  </security:authentication-manager>  <!-- 配置加密类 -->  <bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>  <!-- <bean id="webexpressionHandler" class="org.springframework.security.web.access.expression.DefaultWebSecurityExpressionHandler" />-->  <!-- 提供了入门的方式，在内存中存入用户名和密码  <security:authentication-manager>  <security:authentication-provider>  <security:user-service>  <security:user name="admin" password="{noop}admin" authorities="ROLE\_USER"/>  </security:user-service>  </security:authentication-provider>  </security:authentication-manager>  --> |

## Service

|  |
| --- |
| 需要实现UserDetailsService接口  重写UserDetails loadUserByUsername(String username)方法  由于返回值是UserDetails类型，所以需要将从数据库中查出的用户信息重新封装到框架自带的User类型对象中，该类实现了UserDetails接口 |

## 服务器端方法级权限控制

|  |
| --- |
| JSR-250注解  第一步：配置文件中开启注解  <security:global-method-security jsr250-annotations="enabled"/>  第二步：在指定的方法上使用注解  @RolesAllowed @PermitAll @DenyAll  第三步：导入依赖(jsr250-api)  @Secured注解  第一步：配置文件中开启注解  <security:global-method-security secured-annotations="enabled"/>  第二步：在指定方法上使用注解(角色不能省略前缀ROLE\_)  @Secured  支持表达式的注解  第一步：配置文件中开启注解  <security:global-method-security pre-post-annotations="disabled"/>  第二步：在指定方法上使用注解(使用spel表达式)  @PreAuthorize @PostAuthorize |

## 页面端标签控制权限

|  |
| --- |
| 第一步：导入依赖  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-taglibs</artifactId>  <version>version</version>  </dependency>  第二步：页面导入  <%@taglib uri="http://www.springframework.org/security/tags" prefix="security"%>  第三步：页面使用标签  获取当前使用的用户  <security:authentication property="" htmlEscape="" scope="" var=""/>  控制页面上某些内容是否可以显示  <security:authorize access="" method="" url="" var=""></security:authorize> |

  Spring Security允许我们在定义URL访问或方法访问所应有的权限时使用Spring EL表达式，在定义所需的访问权限时如果对应的表达式返回结果为true则表示拥有对应的权限，反之则无。Spring Security可用表达式对象的基类是SecurityExpressionRoot，其为我们提供了如下在使用Spring EL表达式对URL或方法进行权限控制时通用的内置表达式。

|  |  |
| --- | --- |
| **表达式** | **描述** |
| hasRole([role]) | 当前用户是否拥有指定角色。 |
| hasAnyRole([role1,role2]) | 多个角色是一个以逗号进行分隔的字符串。如果当前用户拥有指定角色中的任意一个则返回true。 |
| hasAuthority([auth]) | 等同于hasRole |
| hasAnyAuthority([auth1,auth2]) | 等同于hasAnyRole |
| Principle | 代表当前用户的principle对象 |
| authentication | 直接从SecurityContext获取的当前Authentication对象 |
| permitAll | 总是返回true，表示允许所有的 |
| denyAll | 总是返回false，表示拒绝所有的 |
| isAnonymous() | 当前用户是否是一个匿名用户 |
| isRememberMe() | 表示当前用户是否是通过Remember-Me自动登录的 |
| isAuthenticated() | 表示当前用户是否已经登录认证成功了。 |
| isFullyAuthenticated() | 如果当前用户既不是一个匿名用户，同时又不是通过Remember-Me自动登录的，则返回true。 |