# 什么是Security

# Security应用场景

# SpringBoot整合Security

## Maven依赖信息

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.1.RELEASE</version>  </parent>  <!-- 管理依赖 -->  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Finchley.M7</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <dependencies>  <!-- SpringBoot整合Web组件 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </dependency>  <!-- springboot整合freemarker -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-freemarker</artifactId>  </dependency>  </dependencies>  <!-- 注意： 这里必须要添加， 否者各种依赖有问题 -->  <repositories>  <repository>  <id>spring-milestones</id>  <name>Spring Milestones</name>  <url>https://repo.spring.io/libs-milestone</url>  <snapshots>  <enabled>false</enabled>  </snapshots>  </repository>  </repositories> |

## application.yml

|  |
| --- |
| # 配置freemarker  spring:  freemarker:  # 设置模板后缀名  suffix: .ftl  # 设置文档类型  content-type: text/html  # 设置页面编码格式  charset: UTF-8  # 设置页面缓存  cache: **false**  # 设置ftl文件路径  template-loader-path:  - classpath:/templates  # 设置静态文件路径，js,css等  mvc:  static-path-pattern: /static/\*\* |

## HttpBasic模式

### 什么是Basic认证

在HTTP协议进行通信的过程中，HTTP协议定义了基本认证过程以允许HTTP服务器对WEB浏览器进行用户身份证的方法，当一个客户端向HTTP服务 器进行数据请求时，如果客户端未被认证，则HTTP服务器将通过基本认证过程对客户端的用户名及密码进行验证，以决定用户是否合法。客户端在接收到HTTP服务器的身份认证要求后，会提示用户输入用户名及密码，然后将用户名及密码以BASE64加密，加密后的密文将附加于请求信息中， 如当用户名为mayikt，密码为：123456时，客户端将用户名和密码用“：”合并，并将合并后的字符串用BASE64加密为密文，并于每次请求数据 时，将密文附加于请求头（Request Header）中。HTTP服务器在每次收到请求包后，根据协议取得客户端附加的用户信息（BASE64加密的用户名和密码），解开请求包，对用户名及密码进行验证，如果用 户名及密码正确，则根据客户端请求，返回客户端所需要的数据;否则，返回错误代码或重新要求客户端提供用户名及密码。

### Maven依赖

|  |
| --- |
| <!-->spring-boot 整合security -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  </dependency> |

### SecurityConfig

|  |
| --- |
| @Component  @EnableWebSecurity  **public** **class** SecurityConfig **extends** WebSecurityConfigurerAdapter {  // 用户认证信息  @Override  **protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  // 设置用户账号信息和权限  auth.inMemoryAuthentication().withUser("admin").password("123456").authorities("addOrder");  }  // 配置HttpSecurity 拦截资源  **protected** **void** configure(HttpSecurity http) **throws** Exception {  http.authorizeRequests().antMatchers("/\*\*").fullyAuthenticated().and().httpBasic();  }  } |

### 启动项目

There is no PasswordEncoder mapped for the id "null"

原因:升级为Security5.0以上密码支持多中加密方式，回复以前模式

|  |
| --- |
| @Bean  **public** **static** ~~NoOpPasswordEncoder~~ passwordEncoder() {  **return** (~~NoOpPasswordEncoder~~) ~~NoOpPasswordEncoder~~.~~getInstance~~();  } |

## FromLogin

FromLogin以表单形式进行认证

|  |
| --- |
| // 配置HttpSecurity 拦截资源  **protected** **void** configure(HttpSecurity http) **throws** Exception {  http.authorizeRequests().antMatchers("/\*\*").fullyAuthenticated().and().formLogin();  } |

# Security权限控制

|  |
| --- |
| @Component  @EnableWebSecurity  **public** **class** SecurityConfig **extends** WebSecurityConfigurerAdapter {  // 用户认证信息  @Override  **protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  // 设置用户账号信息和权限  auth.inMemoryAuthentication().withUser("admin").password("123456").authorities("showOrder","addOrder","updateOrder","deleteOrder");  // 添加 useradd账号 只有添加查询和添加订单权限  auth.inMemoryAuthentication().withUser("userAdd").password("123456")  .authorities("showOrder","addOrder");  }  // 配置HttpSecurity 拦截资源  **protected** **void** configure(HttpSecurity http) **throws** Exception {  // 拦截请求, 权限名称  http.authorizeRequests()  .antMatchers("/showOrder").hasAnyAuthority("showOrder")  .antMatchers("/addOrder").hasAnyAuthority("addOrder")  .antMatchers("/updateOrder").hasAnyAuthority("updateOrder")  .antMatchers("/deleteOrder").hasAnyAuthority("deleteOrder")  .antMatchers("/\*\*").fullyAuthenticated().and().formLogin();  }  // SpringBoot2.0抛弃了原来的NoOpPasswordEncoder，要求用户保存的密码必须要使用加密算法后存储，在登录验证的时候Security会将获得的密码在进行编码后再和数据库中加密后的密码进行对比  @Bean  **public** **static** ~~NoOpPasswordEncoder~~ passwordEncoder() {  **return** (~~NoOpPasswordEncoder~~) ~~NoOpPasswordEncoder~~.~~getInstance~~();  }  } |

# 修改403错误页面

403报错权限不足

## 控制器页面请求跳转

|  |
| --- |
| @Controller  **public** **class** ErrorController {  @RequestMapping("/error/403")  **public** String error() {  **return** "/error/403";  }  } |

## 自定义WEB 服务器参数

|  |
| --- |
| /\*\*  \* 自定义 WEB 服务器参数 可以配置默认错误页面  \*  \* **@author** 余胜军  \* **@version** 2018年11月12日  \*/  @Configuration  **public** **class** WebServerAutoConfiguration {  @Bean  **public** ConfigurableServletWebServerFactory webServerFactory() {  TomcatServletWebServerFactory factory = **new** TomcatServletWebServerFactory();  ErrorPage errorPage400 = **new** ErrorPage(HttpStatus.***BAD\_REQUEST***, "/error/400");  ErrorPage errorPage401 = **new** ErrorPage(HttpStatus.***UNAUTHORIZED***, "/error/401");  ErrorPage errorPage403 = **new** ErrorPage(HttpStatus.***FORBIDDEN***, "/error/403");  ErrorPage errorPage404 = **new** ErrorPage(HttpStatus.***NOT\_FOUND***, "/error/404");  ErrorPage errorPage415 = **new** ErrorPage(HttpStatus.***UNSUPPORTED\_MEDIA\_TYPE***, "/error/415");  ErrorPage errorPage500 = **new** ErrorPage(HttpStatus.***INTERNAL\_SERVER\_ERROR***, "/error/500");  factory.addErrorPages(errorPage400, errorPage401, errorPage403, errorPage404, errorPage415, errorPage500);  **return** factory;  }  } |

# 修改fromLogin登陆页面

关闭csdrf、配置loginpage即可

|  |
| --- |
| @Component  @EnableWebSecurity  **public** **class** SecurityConfig **extends** WebSecurityConfigurerAdapter {  // 用户认证信息  @Override  **protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  // 设置用户账号信息和权限  auth.inMemoryAuthentication().withUser("admin").password("123456").authorities("showOrder","addOrder","updateOrder","deleteOrder");  // 添加 useradd账号 只有添加查询和添加订单权限  auth.inMemoryAuthentication().withUser("userAdd").password("123456")  .authorities("showOrder","addOrder");  }  // 配置HttpSecurity 拦截资源  **protected** **void** configure(HttpSecurity http) **throws** Exception {  // // 拦截请求, 权限名称  http.authorizeRequests()  .antMatchers("/showOrder").hasAnyAuthority("showOrder")  .antMatchers("/addOrder").hasAnyAuthority("addOrder")  .antMatchers("/login").permitAll()  .antMatchers("/updateOrder").hasAnyAuthority("updateOrder")  .antMatchers("/deleteOrder").hasAnyAuthority("deleteOrder")  //并且关闭csrf  .antMatchers("/\*\*").fullyAuthenticated().and().formLogin().loginPage("/login").and().csrf().disable();      }  // SpringBoot2.0抛弃了原来的NoOpPasswordEncoder，要求用户保存的密码必须要使用加密算法后存储，在登录验证的时候Security会将获得的密码在进行编码后再和数据库中加密后的密码进行对比  @Bean  **public** **static** ~~NoOpPasswordEncoder~~ passwordEncoder() {  **return** (~~NoOpPasswordEncoder~~) ~~NoOpPasswordEncoder~~.~~getInstance~~();  }  } |

|  |
| --- |
| <#if RequestParameters['error']??>  用户名称或者密码错误  </#if> |

# 认证成功或者失败处理

AuthenticationFailureHandler 认证失败接口

AuthenticationSuccessHandler 认证成功接口

|  |
| --- |
| @Component  **public** **class** MyAuthenticationFailureHandler **implements** AuthenticationFailureHandler {  **public** **void** onAuthenticationFailure(HttpServletRequest req, HttpServletResponse res, AuthenticationException auth)  **throws** IOException, ServletException {  System.***out***.println("用户认证失败");  res.sendRedirect("http://www.mayikt.com");  }  } |

|  |
| --- |
| @Component  **public** **class** MyAuthenticationSuccessHandler **implements** AuthenticationSuccessHandler {  // 用户认证成功  **public** **void** onAuthenticationSuccess(HttpServletRequest req, HttpServletResponse res, Authentication auth)  **throws** IOException, ServletException {  System.***out***.println("用户登陆成功");  res.sendRedirect("/");  }  } |

|  |
| --- |
| @Component  @EnableWebSecurity  **public** **class** SecurityConfig **extends** WebSecurityConfigurerAdapter {  @Autowired  **private** MyAuthenticationSuccessHandler successHandler;  @Autowired  **private** MyAuthenticationFailureHandler failHandler;  // 用户认证信息  @Override  **protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  // 设置用户账号信息和权限  auth.inMemoryAuthentication().withUser("admin").password("123456").authorities("showOrder","addOrder","updateOrder","deleteOrder");  // 添加 useradd账号 只有添加查询和添加订单权限  auth.inMemoryAuthentication().withUser("userAdd").password("123456")  .authorities("showOrder","addOrder");  }  // 配置HttpSecurity 拦截资源  **protected** **void** configure(HttpSecurity http) **throws** Exception {  // // 拦截请求, 权限名称  http.authorizeRequests()  .antMatchers("/showOrder").hasAnyAuthority("showOrder")  .antMatchers("/addOrder").hasAnyAuthority("addOrder")  .antMatchers("/login").permitAll()  .antMatchers("/updateOrder").hasAnyAuthority("updateOrder")  .antMatchers("/deleteOrder").hasAnyAuthority("deleteOrder")  //并且关闭csrf  .antMatchers("/\*\*").fullyAuthenticated().and().formLogin().loginPage("/login").successHandler(successHandler).failureHandler(failHandler).and().csrf().disable();      }  // SpringBoot2.0抛弃了原来的NoOpPasswordEncoder，要求用户保存的密码必须要使用加密算法后存储，在登录验证的时候Security会将获得的密码在进行编码后再和数据库中加密后的密码进行对比  @Bean  **public** **static** ~~NoOpPasswordEncoder~~ passwordEncoder() {  **return** (~~NoOpPasswordEncoder~~) ~~NoOpPasswordEncoder~~.~~getInstance~~();  }  } |

# 整合Mybatis框架

## Maven依赖

|  |
| --- |
| <!-->spring-boot 整合security -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  </dependency>  <!-- springboot 整合mybatis框架 -->  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.2</version>  </dependency>  <!-- alibaba的druid数据库连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  <version>1.1.9</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency> |

## application.yml

|  |
| --- |
| # 配置freemarker  spring:  freemarker:  # 设置模板后缀名  suffix: .ftl  # 设置文档类型  content-type: text/html  # 设置页面编码格式  charset: UTF-8  # 设置页面缓存  cache: **false**  # 设置ftl文件路径  template-loader-path:  - classpath:/templates  # 设置静态文件路径，js,css等  mvc:  static-path-pattern: /static/\*\*  ####整合数据库层  datasource:  name: test  url: jdbc:mysql://127.0.0.1:3306/rbac\_db  username: root  password: root  # druid 连接池  type: com.alibaba.druid.pool.DruidDataSource  driver-class-name: com.mysql.jdbc.Driver |

## 导入mapper

|  |
| --- |
| **public** **interface** UserMapper {  // 查询用户信息  @Select(" select \* from sys\_user where username = #{userName}")  User findByUsername(@Param("userName") String userName);  // 查询用户的权限  @Select(" select permission.\* from sys\_user user" + " inner join sys\_user\_role user\_role"  + " on user.id = user\_role.user\_id inner join "  + "sys\_role\_permission role\_permission on user\_role.role\_id = role\_permission.role\_id "  + " inner join sys\_permission permission on role\_permission.perm\_id = permission.id where user.username = #{userName};")  List<Permission> findPermissionByUsername(@Param("userName") String userName);  } |

|  |
| --- |
| **public** **interface** PermissionMapper {  // 查询苏所有权限  @Select(" select \* from sys\_permission ")  List<Permission> findAllPermission();  } |

## 动态查询账号

使用UserDetailsService实现动态查询数据库验证账号

|  |
| --- |
| @Component  **public** **class** MyUserDetailsService **implements** UserDetailsService {  @Autowired  **private** UserMapper userMapper;  @Override  **public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  // 1.根据数据库查询，用户是否登陆  User user = userMapper.findByUsername(username);  // 2.查询该用户信息权限  **if** (user != **null**) {  // 设置用户权限  List<Permission> listPermission = userMapper.findPermissionByUsername(username);  System.***out***.println("用户信息权限:" + user.getUsername() + ",权限:" + listPermission.toString());  **if** (listPermission != **null** && listPermission.size() > 0) {  List<GrantedAuthority> authorities = **new** ArrayList<GrantedAuthority>();  **for** (Permission permission : listPermission) {  // 添加用户权限  authorities.add(**new** SimpleGrantedAuthority(permission.getPermTag()));  }  // 设置用户权限  user.setAuthorities(authorities);  }  }  **return** user;  }  } |

|  |
| --- |
| **protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  // // 添加admin账号  // auth.inMemoryAuthentication().withUser("admin").password("123456").  // authorities("showOrder","addOrder","updateOrder","deleteOrder");  // // 添加userAdd账号  // auth.inMemoryAuthentication().withUser("userAdd").password("123456").authorities("showOrder","addOrder");  // // 如果想实现动态账号与数据库关联 在该地方改为查询数据库  auth.userDetailsService(myUserDetailsService);  } |

### 密码使用MD5加密

|  |
| --- |
| **public** **class** MD5Util {  **private** **static** **final** String ***SALT*** = "mayikt";  **public** **static** String encode(String password) {  password = password + ***SALT***;  MessageDigest md5 = **null**;  **try** {  md5 = MessageDigest.*getInstance*("MD5");  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  }  **char**[] charArray = password.toCharArray();  **byte**[] byteArray = **new** **byte**[charArray.length];  **for** (**int** i = 0; i < charArray.length; i++)  byteArray[i] = (**byte**) charArray[i];  **byte**[] md5Bytes = md5.digest(byteArray);  StringBuffer hexValue = **new** StringBuffer();  **for** (**int** i = 0; i < md5Bytes.length; i++) {  **int** val = ((**int**) md5Bytes[i]) & 0xff;  **if** (val < 16) {  hexValue.append("0");  }  hexValue.append(Integer.*toHexString*(val));  }  **return** hexValue.toString();  }  **public** **static** **void** main(String[] args) {  System.***out***.println(MD5Util.*encode*("123456"));  }  } |

|  |
| --- |
| auth.userDetailsService(myUserDetailsService).passwordEncoder(**new** PasswordEncoder() {    //验证密码  **public** **boolean** matches(CharSequence rawPassword, String encodedPassword) {  String rawPass= MD5Util.*encode*((String)rawPassword);  **boolean** result = rawPass.equals(encodedPassword);  **return** result;  }    // 加密密码  **public** String encode(CharSequence rawPassword) {  **return** MD5Util.*encode*((String)rawPassword);  }  }); |

## 动态请求资源

|  |
| --- |
| // 配置拦截请求资源  **protected** **void** configure(HttpSecurity http) **throws** Exception {  ExpressionUrlAuthorizationConfigurer<HttpSecurity>.ExpressionInterceptUrlRegistry authorizeRequests = http  .authorizeRequests();  // 查询数据库获取权限列表  List<Permission> listPermission = permissionMapper.findAllPermission();  **for** (Permission permission : listPermission) {  authorizeRequests.antMatchers(permission.getUrl()).hasAuthority(permission.getPermTag());  }  authorizeRequests.antMatchers("/login").permitAll().antMatchers("/\*\*").fullyAuthenticated().and().formLogin()  .loginPage("/login").successHandler(successHandler).failureHandler(failureHandler).and().csrf()  .disable();  } |