# 权限管理概述

**主体**(**principal**)

使用系统的用户或设备或从其他系统远程登录的用户等等。简单说就是谁使用系统谁就是主体。

**认证**(**authentication**)

权限管理系统确认一个主体的身份，允许主体进入系统。简单说就是“主体”证明自己是谁。笼统的认为就是以前所做的登录操作。

**授权**(**authorization**)

将操作系统的“权力”“授予”“主体”，这样主体就具备了操作系统中特定功能的能力。所以简单来说，授权就是给用户分配权限。

**SpringSecurity特点**

 ⚪和Spring无缝整合。

 ⚪全面的权限控制。

 ⚪专门为Web开发而设计。

 ⚪旧版本不能脱离Web环境使用。新版本对整个框架进行了分层抽取，分成了核心模块和Web模块。单独引入核心模块就可以脱离Web环境。

 ⚪重量级。

**shiro特点**

 ⚪轻量级。Shiro主张的理念是把复杂的事情变简单。针对对性能有更高要求的互联网应用有更好表现。

 ⚪通用性。

 ⚪好处：不局限于Web环境，可以脱离Web环境使用。

 ⚪缺陷：在Web环境下一些特定的需求需要手动编写代码定制。

## 使用配置类代替XML配置文件

@Configuration

类标记了这个注解就可以使用这个类代替Spring的XML配置文件。由于该注解带有@Component注解，所以标记当前注解的类可以享受包的自动扫描。

|  |
| --- |
| @Target(ElementType.*TYPE*) @Retention(RetentionPolicy.*RUNTIME*) @Documented @Component public @interface Configuration {String value() default ""; } |

@Bean

注解配置类

|  |
| --- |
| @Configuration public class AnnotaionConfig{  @Bean  public EmpHandler getEmpHandler(){  return new EmpHandler();  } } |

spring的xml配置文件

|  |
| --- |
| <bean id="empHandler" class="com.maven.component.EmpHandler">  <property /> </bean> |

用来代替XML配置文件中的bean标签。Spring通过调用标记了@Bean注解的方法将对象放入IOC容器行为不会重复调用方法。原因是Spring想要获取bean对应的实例对象时会查看IOC容器中是否已经有了这个对象，如果有则不会执行这个方法，从而保证这个bean是单一实例的。

如果希望对应的bean是多实例的，则可以配合@Scope注解。

# 入门示例

**创建maven工程并加入依赖**

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>4.3.20.RELEASE</version> </dependency> <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>provided</scope> </dependency> <dependency>  <groupId>javax.servlet.jsp</groupId>  <artifactId>jsp-api</artifactId>  <version>2.1.3-b06</version> </dependency> |

**配置spring-mvc配置文件**

|  |
| --- |
| <context:component-scan base-package="com.maven.security"/> <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="prefix" value="/WEB-INF/views/"/>  <property name="suffix" value=".jsp"/> </bean> <mvc:annotation-driven/> <mvc:default-servlet-handler/> |

**配置web.xml配置文件**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  version="4.0">  <servlet>  <servlet-name>springDispatcherServlet</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring-mvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>springDispatcherServlet</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  <!--SpringSecurity的filter-name必须设置为springSecurityFilterChain-->  <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> </web-app> |

**加入SpringSecurity依赖**

|  |
| --- |
| <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-web</artifactId>  <version>4.2.10.RELEASE</version> </dependency> <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-config</artifactId>  <version>4.2.10.RELEASE</version> </dependency> <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-taglibs</artifactId>  <version>4.2.10.RELEASE</version> </dependency> |

加入SpringSecurity控制权限Filter

SpringSecurity使用的是过滤器Filter而不是拦截器Interceptor，意味着SpringSecurity能够管理的不仅仅是SpringMVC中的handler请求，还包含Web应用中所有请求。比如：项目中的静态资源也会被拦截，从而进行权限控制。

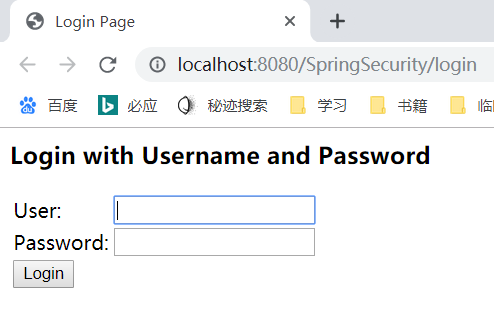
|  |
| --- |
| <!--SpringSecurity的filter-name必须设置为springSecurityFilterChain--> <filter>  <filter-name>springSecurityFilterChain</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter> <filter-mapping>  <filter-name>springSecurityFilterChain</filter-name>  <url-pattern>/\*</url-pattern> </filter-mapping> |

<filter-name>springSecurityFilterChain</filter-name>标签中必须是springSecurityFilterChain。因为springSecurityFilterChain在IOC容器中对应真正执行权限控制的二十几个Filter，只有叫这个名字才能够加载到这些Filter。

**创建配置类**

|  |
| --- |
| @Configuration //@EnableWebSecurity注解表示启用Web安全功能。 @EnableWebSecurity public class WebAppSecurityConfig extends WebSecurityConfigurerAdapter { } |

部署到服务器并访问项目中的资源，出现如下内容则说明SpringSecurity配置成功。



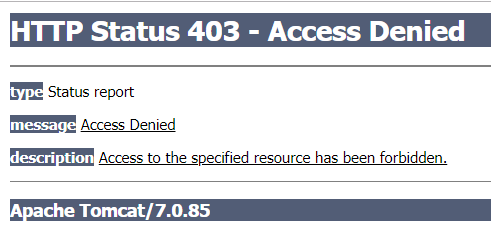
# 功能实现

## 放行首页和静态资源

重写父类WebSecurityConfigurerAdapter的configure(HttpSecurity security)方法。

|  |
| --- |
| @Override protected void configure(HttpSecurity httpSecurity) throws Exception {  // 取消父类方法中的默认规则  //super.configure(httpSecurity);  httpSecurity.authorizeRequests()  // 基于ANT风格设置具体请求访问规则:首页  .antMatchers("/index.jsp")  // 全部允许方式  .permitAll()  // 基于ANT风格设置具体请求访问规则:静态资源  .antMatchers("/layui/\*\*")  .permitAll()  // 其他未设置的请求，需要登录（认证）才可以访问  .anyRequest().authenticated(); } |

此时访问index.jsp和静态资源以外的资源都会跳转到403错误页面。



## 未登录跳转到登陆页面

|  |
| --- |
| @Override protected void configure(HttpSecurity httpSecurity) throws Exception {  ……  .anyRequest().authenticated()  .and()  // 指定使用表单进行登录操作  .formLogin()  // 登录页地址  .loginPage("/index.jsp").permitAll() } |

指定登录页前后SpringSecurity登录地址变化

|  |  |
| --- | --- |
| 指定前 | /login GET - the login form  /login POST - process the credentials and if valid authenticate the user  /login?error GET - redirect here for failed authentication attempts  /login?logout GET - redirect here after successfully logging out |
| 指定后 | /index.jsp GET - the login form  /index.jsp POST - process the credentials and if valid authenticate the user  /index.jsp?error GET - redirect here for failed authentication attempts  /index.jsp?logout GET - redirect here after successfully logging out |

## 设置登陆系统账号密码

在WebAppSecurityConfig中重写父类WebSecurityConfigurerAdapter的另外一个方法protected void configure(AuthenticationManagerBuilder builder),并调用AuthenticationManagerBuilder的inMemoryAuthentication的方法在内存中存入用户名、密码、角色和权限等信息。

|  |
| --- |
| @Override protected void configure(AuthenticationManagerBuilder builder) throws Exception {  //在内存中进行认证（通常不会用于实际开发）  builder.inMemoryAuthentication()  .withUser("root")// 指定用户名  .password("root")// 指定密码  .roles("r\_lv2")//指定当前账号具备的角色  .authorities("r\_lv1")// 指定当前账户具备的权限  .and()  .withUser("xmm")  .password("123")  .roles("r\_lv3"); } |

如果没有设置roles()或authorities()方法，则会显示java.lang.IllegalArgumentException: Cannot pass a null GrantedAuthority collection异常。

roles和authorities方法可以同时设置，但是后面的会覆盖前面的，导致先设置的无法生效。

**提交表单设置**

|  |
| --- |
| <%pageContext.setAttribute("PATH", request.getContextPath());%>  <form action="${PATH}/login" method="post">  <input type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}"/>  ......  </form> |

此时需要指定登陆操作本身的地址，否则无法进行登陆操作，其方式是调用HttpSecurity的loginProcessingUrl()的方法，然后调用permitAll()方法。

SpringSecurity中默认的账号和密码的请求参数名分别为username和password，此时必须在如下操作中选择一项：

将页面中表单项的name属性值修改为username和password：

|  |
| --- |
| <div class="layui-form-item">  <label class="layadmin-user-login-icon layui-icon layui-icon-username"  for="LAY-user-login-username"></label>  <!-- input的name属性值必须符合SpringSecurity规则，除非专门进行了定制，否则用户名必须使用username，密码必须使用password -->  <input type="text" name="loginacct" id="LAY-user-login-username" lay-verify="required"  placeholder="用户名" class="layui-input"> </div> <div class="layui-form-item">  <label class="layadmin-user-login-icon layui-icon layui-icon-password"  for="LAY-user-login-password"></label>  <input type="text" name="credential" id="LAY-user-login-password"  lay-verify="required" placeholder="密码" class="layui-input"> </div> |

调用usernameParameter()和passwordParameter()指定用户名和密码：

|  |
| --- |
| // 允许具体指定登录（检查用户名、密码是否正确）地址不被拦截 .loginProcessingUrl("/login").permitAll() //修改默认账号的请求参数名 .usernameParameter("loginacct") //修改默认密码的请求参数名 .passwordParameter("credential") // 指定登录成功后前往的地址 .defaultSuccessUrl("/main.html") |

### 跨站请求伪造

在单点登录系统中，认证中心根据浏览器的Cookie识别用户身份。那如果用户的Cookie被劫持仿冒用户身份登录系统怎么办？除了Cookie之外，还使用\_csrf生成的token防止跨站请求伪造。

使用\_csrf(**Cross-site request forgery**)来防止跨站请求伪造，即在表单项中设置input隐藏域，并将其name属性值设置为${\_csrf.parameterName}，其value属性值设置为${\_csrf.token}。

配置完成后发送登陆请求时如果没有携带\_csrf值则会返回403(Invalid CSRF Token)错误，而从钓鱼网站的页面提交的请求无法携带正确的、被承认的令牌。

## 用户注销

通过调用HttpSecurity对象的如下方法设置注销功能：

logout()：开启注销功能

logoutUrl()：自定义注销功能的URL地址

logoutSuccessUrl()：退出成功后跳转的URL地址

addLogoutHandler()：添加退出处理器

logoutSuccessHandler()：设置退出成功处理器

|  |
| --- |
| .logout() .logoutUrl("/my/app/logout") .logoutSuccessUrl("/index.jsp") |

|  |
| --- |
| <form id="logoutForm" action="${pageContext.request.contextPath }/my/app/logout" method="post">  <input type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}"/> </form> <a id="logoutAnchor" href="">退出</a> <script type="text/javascript">  window.onload = function() {  document.getElementById("logoutAnchor").onclick = function() {  document.getElementById("logoutForm").submit();  return false;  };  }; </script> |

## 基于角色或权限访问控制

在SpringSecurity中基于用户配置的角色或权限进行访问控制，当访问不满足条件的资源时就会显示403(Access is denied)状态码，表示权限或角色不满足要求。

|  |
| --- |
| builder.inMemoryAuthentication()  .withUser("root")// 指定用户名  .password("root")// 指定密码  .roles("r\_lv2")//指定当前账号具备的角色  .authorities("r\_lv1")// 指定当前账户具备的权限 |

### 基于角色

hasAnyRole方法表示只要符合任一角色即可访问该角色下匹配的资源，其参数可以有多个；hasRole方法参数只能有一个，表示只要符合该角色即可访问该角色下匹配的资源。

|  |
| --- |
| // 设置访问具体资源访问权限 .antMatchers("/level1/\*\*") .hasAnyRole("r\_lv1","r\_lv2","r\_lv3") .antMatchers("/level2/\*\*") .hasAnyRole("r\_lv2","r\_lv3") .antMatchers("/level3/\*\*") .hasAnyRole("r\_lv3")  .anyRequest().authenticated() |

### 基于权限

hasAnyAuthority方法表示只要符合任一权限即可访问该权限下匹配的资源，其参数可以有多个；hasAuthority方法参数只能有一个，表示只要符合该权限即可访问该权限下匹配的资源。

|  |
| --- |
| // 设置访问具体资源访问权限 .antMatchers("/level1/\*\*") .hasAnyAuthority("r\_lv1","r\_lv2","r\_lv3") .antMatchers("/level2/\*\*") .hasAnyAuthority("r\_lv2","r\_lv3") .antMatchers("/level3/\*\*") .hasAnyAuthority("r\_lv3") .anyRequest().authenticated() |

### 访问控制顺序

在进行配置访问控制时，范围较小的访问控制应该位于范围较大的访问控制的前面，防止发生覆盖，导致范围较小的控制无效。即先配置具体小范围设置，再配置大范围模糊设置。

## 自定义403错误页面

### accessDeniedPage

调用AccessDeniedPage()方法指定需要请求地址，并经由Controller控制器进行跳转

|  |
| --- |
| // 开启异常处理功能 .exceptionHandling() // 指定发生访问拒绝情况时前往的页面 .accessDeniedPage("/to/no/auth/page.html") |

|  |
| --- |
| @RequestMapping("/to/no/auth/page.html") public String toNoAuthPage(){  return "no\_auth"; } |

### accessDeniedHandler

使用accessDeniedHandler()方法通过匿名内部类的方式进行跳转

|  |
| --- |
| // 开启异常处理功能 .exceptionHandling() // 指定发生访问拒绝情况时的处理器类（更精细控制） .accessDeniedHandler(new AccessDeniedHandler() {  @Override  public void handle( HttpServletRequest request,  HttpServletResponse response,  AccessDeniedException accessDeniedException)  throws IOException, ServletException {  request.setAttribute("message", accessDeniedException.getMessage());  request.getRequestDispatcher("/WEB-INF/views/no\_auth.jsp")  .forward(request, response);  } }) |

## 记住用户名密码

### 内存获取

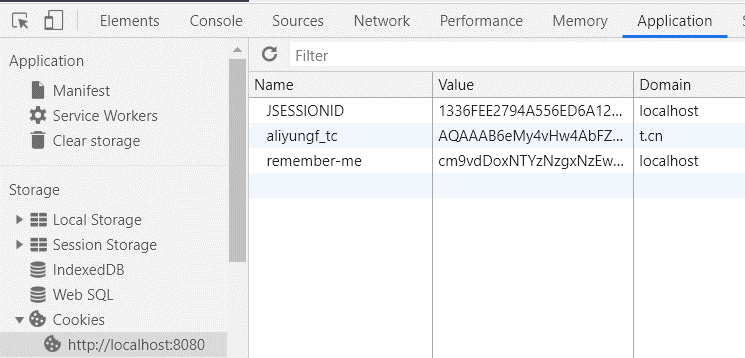
首先使用HttpSecurity对象调用rememberMe()方法开启记住用户名和密码的功能。

|  |
| --- |
| //开启记住用户名密码功能 .rememberMe() |

将请求提交的表单中设置一个携带名为remember-me的请求参数，一般是将checkbox的name属性设置为remember-me。如果不想使用remember-me作为name属性的值，则可以使用HttpSecurity对象调用remembermeParameter()方法自定义name属性值。

|  |
| --- |
| <input type="checkbox" name="remember-me" lay-skin="primary" title="记住我"> |

通过上述设置后再登陆时就会生成名为remember-me的cookie文件存储在浏览器中，且该cookie文件默认的有效事件为两周。



### 数据库获取

为了让服务器重启也不影响记住登录状态，将用户登录状态信息存入数据库。

在进行数据库获取之前需要在maven配置文件中添加依赖

|  |
| --- |
| <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.12</version> </dependency> <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.47</version> </dependency> <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-orm</artifactId>  <version>4.3.20.RELEASE</version> </dependency> |

然后在spring-mvc.xml配置文件中配置数据源和jdbcTemplate

|  |
| --- |
| <!--配置数据源--> <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">  <property name="username" value="root"/>  <property name="password" value="root"/>  <property name="url" value="jdbc:mysql://localhost:3306/security?useSSL=false"/>  <property name="driverClassName" value="com.mysql.jdbc.Driver"/> </bean> <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">  <property name="dataSource" ref="dataSource"/> </bean> |

创建数据库security

|  |
| --- |
| create database `security` character set utf8; create table persistent\_logins (  username varchar(64) not null,  series varchar(64) primary key,  token varchar(64) not null,  last\_used timestamp not null ); |

|  |
| --- |
| @Configuration //@EnableWebSecurity注解表示启用Web安全功能。 @EnableWebSecurity public class WebAppSecurityConfig extends WebSecurityConfigurerAdapter {  @Autowired  private DataSource dataSource;  @Override  protected void configure(HttpSecurity httpSecurity) throws Exception {  // 创建令牌仓库对象  JdbcTokenRepositoryImpl repository = new JdbcTokenRepositoryImpl();  // 给令牌仓库装配数据源  repository.setDataSource(dataSource);  httpSecurity.authorizeRequests()  ...  // 装配令牌仓库  .tokenRepository(repository);  } } |

## 查询数据库完成认证

### SpringSecurity默认实现

SpringSecurity中有默认实现数据库完成认证的方式，使用AuthenticationManagerBuilder对象进行如下操作：

builder.jdbcAuthentication().usersByUsernameQuery(“tom”);

此时usersByUsernameQuery方法最终调用的是JdbcDaoImpl类的方法查询数据库，而SpringSecurity的默认实现将SQL语句嵌入了JdbcDaoImpl类中，这种情况下一般不会使用jdbcAuthentication()方法，而是使用自定义的数据库查询方式。

### 自定义数据库查询方式

自定义数据库查询方式时需要使用AuthenticationManagerBuilder对象的userDetailService方法，因此需要自定义实现UserDetailService接口的类并自动装配。

|  |
| --- |
| @Service public class AppUserDetailService implements UserDetailsService {  @Autowired  private JdbcTemplate jdbcTemplate;  @Override  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  // 使用SQL语句根据用户名查询用户对象  String sql = "SELECT id,loginacct,userpswd,username,email,createtime FROM t\_admin WHERE loginacct = ?";  // 获取查询结果  Map<String, Object> resultMap = jdbcTemplate.queryForMap(sql,username);  // 获取用户名、密码数据  String loginacct = resultMap.get("loginacct").toString();  String userpswd = resultMap.get("userpswd").toString();  // 创建权限列表  // 如果这里添加的是角色，那么一定要加“ROLE\_”前缀，因为SpringSecurity不给自动加了。  List<GrantedAuthority> list = AuthorityUtils.*createAuthorityList*("ROLE\_r\_lv3","USER");  // 创建用户对象  User user = new User(loginacct, userpswd, list);  return user;  } } |

|  |
| --- |
| @Override protected void configure(AuthenticationManagerBuilder builder) throws Exception {  builder.userDetailsService(appUserDetailService); } |

### ROLE\_前缀

在自定义的UserDetailsService中，使用org.springframework.security.core.authority.

AuthorityUtils.createAuthorityList(String...)工具方法获取创建SimpleGrantedAuthority对象添加角色时需要手动在角色名称前加“ROLE\_”前缀。

而在hasRole()方法中添加角色时则不需要手动在角色名称前加ROLE\_前缀。

## 自定义密码加密

使用SpringSecurity中的自定义密码加密功能时需要自定义类，然后实现org.springframework.security.crypto.password.PasswordEncoder（使用没有过时的）接口并重写encode()和mathces()方法。encode()方法表示对明文进行加密，matches()方法表示对明文加密后和密文进行比较。

|  |
| --- |
| @Service public class PasswordEncoderService implements PasswordEncoder {  @Override  public String encode(CharSequence rawPassword) {  // 对输入数据进行校验  if(rawPassword == null || rawPassword.length() == 0){  throw new RuntimeException("密码不能为空!!!");  }  String encodePassword = Utils.*md5*(rawPassword.toString());  return encodePassword;  }  @Override  public boolean matches(CharSequence rawPassword, String encodedPassword) {  if(rawPassword == null || rawPassword.length() == 0) {  throw new RuntimeException("明文密码不能为空！");  }  String formPassword = Utils.*md5*(rawPassword.toString());  return Objects.*equals*(formPassword, encodedPassword);  } } |

|  |
| --- |
| @Autowired private AppUserDetailService appUserDetailService; @Autowired private PasswordEncoderService passwordEncoderService; @Override protected void configure(AuthenticationManagerBuilder builder) throws Exception {  // 借助UserDetailsService类型的对象实现数据库认证 builder.userDetailsService(appUserDetailService).passwordEncoder(passwordEncoderService); } |

### BCryptPasswordEncoder加密

BCryptPasswordEncoder创建对象后代替自定义passwordEncoder对象即可。BCryptPasswordEncoder在加密时通过加入随机盐值让每一次的加密结果都不同。能够避免密码的明文被猜到。

而在对明文和密文进行比较时，BCryptPasswordEncoder会在密文的固定位置取出盐值，重新进行加密。

|  |
| --- |
| BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder(); CharSequence rawPassword = "1234"; String encodedPassword = passwordEncoder.encode(rawPassword); |