# 1 RBAC模型简介：

# 2 数据库权限表结构设计与创建

基于RBAC权限模型，设计权限表相关表：  
1）用户

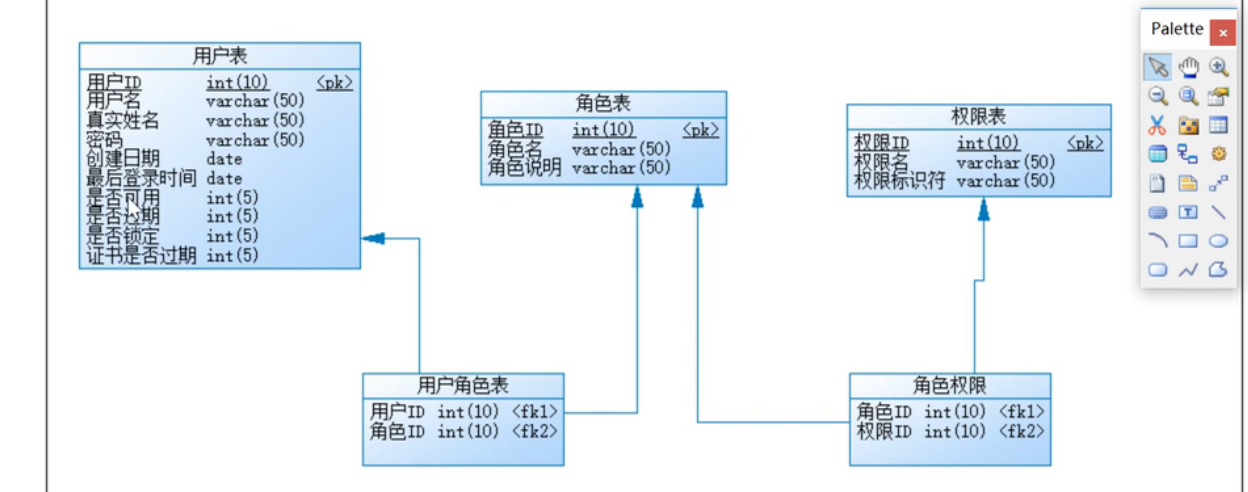
2）角色

3）权限

用户 和 角色多对多关系。

角色 和 权限多对多关系。

使用PowerDsigner设计权限表：

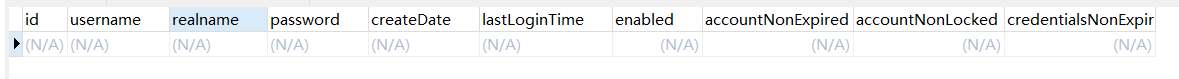


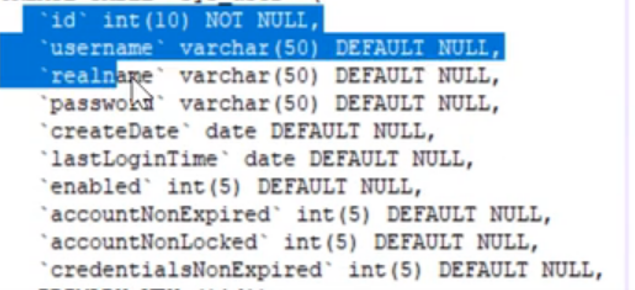
# 3 环境搭建

## 3.1 数据库表建立

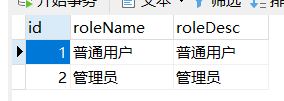
需要建立数据库表：

1. sys\_user (用户)

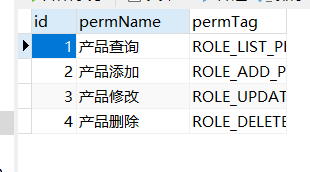




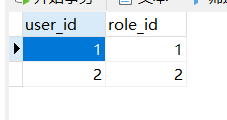
1. sys\_role (角色)



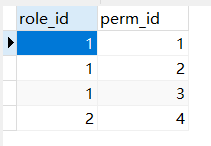
1. sys\_permission (权限)



1. sys\_user\_role(用户角色)



5) sys\_role\_permission(角色权限)



## 3.2 SSM环境搭建

### 3.2.0 依赖

|  |
| --- |
| <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.11</**version**>  <**scope**>test</**scope**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->* <**dependency**>  <**groupId**>com.fasterxml.jackson.core</**groupId**>  <**artifactId**>jackson-databind</**artifactId**>  <**version**>2.9.5</**version**> </**dependency**> <**dependency**>  <**groupId**>com.fasterxml.jackson.core</**groupId**>  <**artifactId**>jackson-databind</**artifactId**>  <**version**>2.9.5</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->* <**dependency**>  <**groupId**>org.mybatis</**groupId**>  <**artifactId**>mybatis-spring</**artifactId**>  <**version**>1.3.2</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-tx -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-tx</**artifactId**>  <**version**>5.1.5.RELEASE</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-jdbc</**artifactId**>  <**version**>5.1.3.RELEASE</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-expression -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-expression</**artifactId**>  <**version**>5.1.5.RELEASE</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-context-support -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-context-support</**artifactId**>  <**version**>5.1.5.RELEASE</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-core</**artifactId**>  <**version**>5.1.5.RELEASE</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-context</**artifactId**>  <**version**>5.1.5.RELEASE</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-beans</**artifactId**>  <**version**>5.1.5.RELEASE</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-aop</**artifactId**>  <**version**>5.1.5.RELEASE</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/commons-logging/commons-logging -->* <**dependency**>  <**groupId**>commons-logging</**groupId**>  <**artifactId**>commons-logging</**artifactId**>  <**version**>1.1.1</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/commons-dbcp/commons-dbcp -->* <**dependency**>  <**groupId**>commons-dbcp</**groupId**>  <**artifactId**>commons-dbcp</**artifactId**>  <**version**>1.4</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.17</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.mybatis</**groupId**>  <**artifactId**>mybatis</**artifactId**>  <**version**>3.5.2</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/log4j/log4j -->* <**dependency**>  <**groupId**>log4j</**groupId**>  <**artifactId**>log4j</**artifactId**>  <**version**>1.2.17</**version**>  <**scope**>compile</**scope**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/commons-pool/commons-pool -->* <**dependency**>  <**groupId**>commons-pool</**groupId**>  <**artifactId**>commons-pool</**artifactId**>  <**version**>1.6</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-web</**artifactId**>  <**version**>5.1.5.RELEASE</**version**> </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-webmvc</**artifactId**>  <**version**>5.1.5.RELEASE</**version**> </**dependency**>   <**dependency**>  <**groupId**>org.springframework.security</**groupId**>  <**artifactId**>spring-security-web</**artifactId**>  <**version**>4.2.3.RELEASE</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework.security</**groupId**>  <**artifactId**>spring-security-config</**artifactId**>  <**version**>4.2.3.RELEASE</**version**> </**dependency**> |

### 3.2.1 web.xml配置

1. 将Spring纳入web.xml中
2. 将SpringMVC纳入web.xml中
3. 将SpringSecurity纳入web.xml中

Spring SpringMVC SpringSecurity

|  |
| --- |
| **<!DOCTYPE web-app PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd" *>*** <**web-app**>  <**display-name**>Archetype Created Web Application</**display-name**>  <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>  classpath:applicationContext.xml  classpath:SpringSecurity.xml  </**param-value**>  </**context-param**>  <**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**>  <**listener**>  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  </**listener**>  <**servlet**>  <**servlet-name**>dispatcherServlet</**servlet-name**>  <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  <**init-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:springmvc.xml</**param-value**>  </**init-param**>  </**servlet**>  <**servlet-mapping**>  <**servlet-name**>dispatcherServlet</**servlet-name**>  <**url-pattern**>/</**url-pattern**>  </**servlet-mapping**> </**web-app**> |

### 3.2.2 db.properties建立

|  |
| --- |
| **driver**=**com.mysql.jdbc.Driver url**=**jdbc:mysql://localhost:3306/security username**=**root password**=**123456** |

### 3.3.3 将MyBatis纳入applicationContext中

配置db.properties 配置datasource 配置sqlSessionFactory 整合MyBatis

|  |
| --- |
| <**bean id="config" class="org.springframework.beans.factory.config.PreferencesPlaceholderConfigurer"** >  <**property name="locations"**>  <**array**>  <**value**>classpath:db.properties</**value**>  </**array**>  </**property**> </**bean**> <**bean id="DataSource" class="org.apache.commons.dbcp.BasicDataSource"**>  <**property name="driverClassName" value="${driver}"**/>  <**property name="url" value="${url}"**/>  <**property name="username" value="${username}"**/>  <**property name="password" value="${password}"**/> </**bean**> <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  <**property name="dataSource" ref="DataSource"**/>  *<!--加载mapper.xml路径-->* <**property name="mapperLocations" value="classpath:org/lzq/Mapper/\*.xml"** /> </**bean**> *<!--Spring 整合MyBatis-->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"**/>  *<!--指定批量产生哪个包的Mapper对象-->* <**property name="basePackage" value="org.lzq.Mapper"** /> </**bean**> |

### 3.3.4 配置SpringMVC和Spring Security文件

SpringMVC：

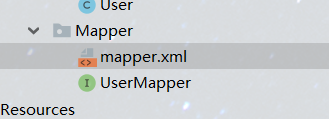
|  |
| --- |
| <**context:component-scan base-package="org.lzq.Controller"**></**context:component-scan**>  <**mvc:annotation-driven**></**mvc:annotation-driven**>  <**bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  <**property name="prefix" value="/security/"** />  <**property name="suffix" value=".jsp"** /> </**bean**> |

Spring Security：

|  |
| --- |
| <**security:http**>  <**security:intercept-url pattern="/\*\*" access="isFullyAuthenticated()"**></**security:intercept-url**>  <**security:form-login**/> </**security:http**>  <**security:authentication-manager**>  <**security:authentication-provider**>  <**security:user-service**>    </**security:user-service**>  </**security:authentication-provider**> </**security:authentication-manager**> |

### 3.3.5 创建mapper.xml和相对于的mapper接口

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>*** *<!--namespace:Mapping文件的包名-->* <**mapper namespace="org.lzq.Mapper.UserMapper"**>  *<!--id:这条语句的唯一标识符  parameterType：输入类型  resultType：返回值类型  -->* </**mapper**> |



# 4 是用户查询与权限查询持久层方法

## 4.1 创建实体类

### 4.1.1 User

|  |
| --- |
| **package** org.lzq.entity;  **import** org.springframework.security.core.GrantedAuthority; **import** org.springframework.security.core.userdetails.UserDetails;  **import** java.util.ArrayList; **import** java.util.Collection; **import** java.util.Date; **import** java.util.List;  **public class** User **implements** UserDetails {  **private** Integer **id**;  **private** String **username**;  **private** String **realname**;  **private** String **password**;  **private** Date **createDate**;  **private** Date **lastLoginTime**;  **private boolean enabled**;  **private boolean accountNonExpired**;  **private boolean accountNonLocked**;  **private boolean credentialsNonExpired**;   *//用户拥有所有权限* **private** List<GrantedAuthority> **authories** = **new** ArrayList<>();   **private** List<GrantedAuthority> getAuthories(){  **return authories**;  }   **+ Get/Set方法** |

### 4.1.2 Permission

|  |
| --- |
| **public class** Permission {  **private** Integer **id**;  **private** String **permName**;  **private** String **permTag**;  **+ Get/Set方法**  } |

### 4.1.3 role

|  |
| --- |
| **public class** Role {  **private** Integer **id**;  **private** String **roleName**;  **private** String **roleDesc**;    **+ Get/Set方法**  } |

## 4.2 创建Mapper接口查询数据

|  |
| --- |
| **public interface** UserMapper {  /\*\*  \* 查询当前用户  \*/  **public** User findUserByName(String username);   /\*\*  \* 查询当前用户拥有权限  \*/  **public** List<Permission> findPermissionByUser(String username); } |

## 4.3 Sql映射文件

|  |
| --- |
| *<!--namespace:Mapping文件的包名-->* <**mapper namespace="org.lzq.Mapper.UserMapper"**>  *<!--id:这条语句的唯一标识符  parameterType：输入类型  resultType：返回值类型  -->   <!-- 查询用户 -->* <**select id="findUserByName" parameterType="string" resultType="user"**>  select *\** from sys\_user where id = #{value}  </**select**>   *<!-- 查询所属权限-->* <**select id="findPermissionByUser" parameterType="string" resultType="permission"**>  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 = #{value}  </**select**> </**mapper**> |

# 5 SSM实现权限自动化查询

步骤：

编写一个基于UserDetailService接口实现类完成自动查询步骤

## 5.1 接口实现类MyUserDetail

|  |
| --- |
| **public class** UserDetails **implements** UserDetailsService {  @Autowired  UserMapper **userMapper**;   @Override  **public** org.springframework.security.core.userdetails.UserDetails loadUserByUsername(String s) **throws** UsernameNotFoundException {  User user = **userMapper**.findUserByName(s);   List<Permission> permissionList = **userMapper**.findPermissionByUser(s);  List<GrantedAuthority> grantedAuthorityList = **new** ArrayList<GrantedAuthority>();  **for**(Permission per:permissionList){  GrantedAuthority grantedAuthority = **new** SimpleGrantedAuthority(per.getPermTag());  grantedAuthorityList.add(grantedAuthority);  }  user.setAuthories(grantedAuthorityList);   **return** user;  } } |

## 5.2 Spring Security配置

用户权限管理使用自动装配

|  |
| --- |
| <**security:http**>  <**security:intercept-url pattern="/security/queryProject.jsp" access="hasAuthority('ROLE\_LIST\_PRODUCT')"**/>  <**security:intercept-url pattern="/security/delProject.jsp" access="hasAuthority('ROLE\_DELETE\_PRODUCT')"**/>  <**security:intercept-url pattern="/security/updataProject.jsp" access="hasAuthority('ROLE\_UPDATE\_PRODUCT')"**/>  <**security:intercept-url pattern="/security/addProject.jsp" access="hasAuthority('ROLE\_ADD\_PRODUCT')"**/>  <**security:intercept-url pattern="index.jsp" access="permitAll()"** />  <**security:form-login**/>   <**security:access-denied-handler error-page="/error.jsp"**/> </**security:http**>  <**security:authentication-manager**>  <**security:authentication-provider user-service-ref="userDetails"**>   </**security:authentication-provider**> </**security:authentication-manager**>  <**bean id="userDetails" class="org.lzq.Sercurity.UserDetails"** /> |

# 6 登陆成功与登陆失败的处理

## 6.1 同步处理

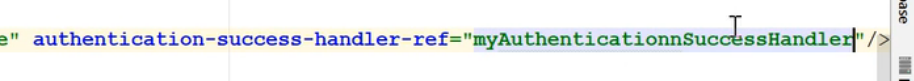
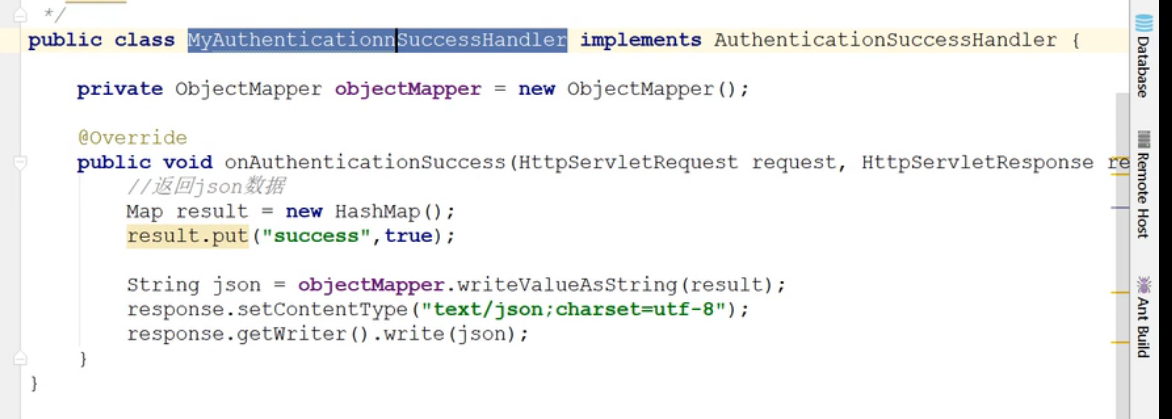
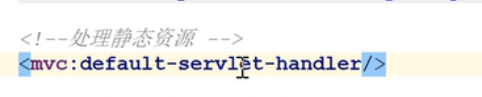
|  |
| --- |
| **<security:form-login authentication-success-forward-url="/index.jsp" authentication-failure-url="/errorUser.jsp"**/> |

## 6.2 异步处理

步骤：  
1.创建关于Json的成功或者失败的handle，在Security配置文件中导入

2.导入Js框架的文件，mvc配置中放行静态文件

3.编写成功或者失败的Js



# 7 PassWordEncoder密码加密

原理：先用**BCryptPasswordEncoder**

将密码加密为Hash格式密码，在配置Spring Security将用户输入的密码和数据库密码对比，相同则通过

步骤：

1. 将原本密码改成Hash+盐
2. 配置SpringSecurity

## 7.1 编写用户更新的Sql功能语句

|  |
| --- |
| *<!-- 更新 -->* <**update id="updateUserPassWord" parameterType="org.lzq.entity.User"**>  update sys\_user set password = #{password} where username = #{username} </**update**> |

## 7.2 Controller层编写修改密码为Hash格式

|  |
| --- |
| *//更新用户* @ResponseBody @RequestMapping(**"/updateUser"**) **private** String updataUser(){  User user = **new** User();  user.setUsername(**"Penis"**);  PasswordEncoder bCryptPasswordEncoder = **new** BCryptPasswordEncoder();  user.setPassword( bCryptPasswordEncoder.encode(**"123456"**));  **mapper**.updateUserPassWord(user);  **return "sucess"**; } |

## 7.3 配置Spring Security

|  |
| --- |
| <**bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"**/> |
| <**security:authentication-manager**>  <**security:authentication-provider user-service-ref="userDetails"**>  <**security:password-encoder ref="passwordEncoder"**/>  </**security:authentication-provider**> </**security:authentication-manager**> |

# 8 图形验证码

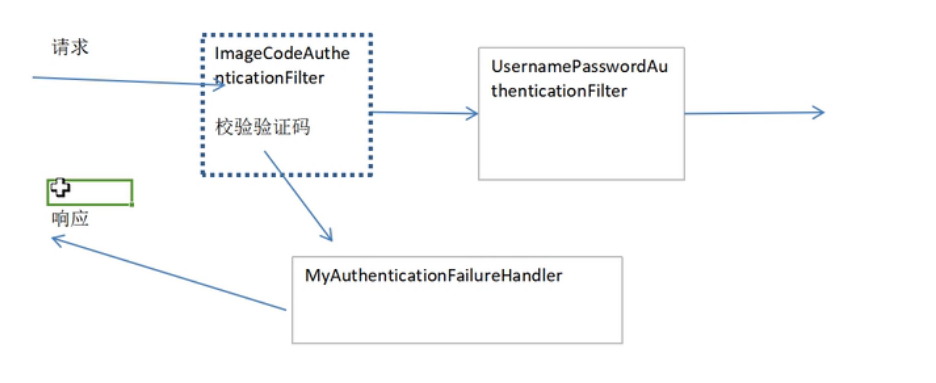
## 8.1 图形验证码的Jsp

|  |
| --- |
| <%@ **page contentType**="**image/jpeg**" **import**="**java.awt.\*,java.awt.image.\*,java.util.\*,javax.imageio.\***" %> <%@ **page trimDirectiveWhitespaces**="**true**" %> **<%!** Color getRandColor(**int** fc,**int** bc){*//给定范围获得随机颜色* Random random = **new** Random();  **if**(fc>255) fc=255;  **if**(bc>255) bc=255;  **int** r=fc+random.nextInt(bc-fc);  **int** g=fc+random.nextInt(bc-fc);  **int** b=fc+random.nextInt(bc-fc);  **return new** Color(r,g,b);  } **%> <%** *//设置页面不缓存* response.setHeader(**"Pragma"**,**"No-cache"**);  response.setHeader(**"Cache-Control"**,**"no-cache"**);  response.setDateHeader(**"Expires"**, 0); *// 在内存中创建图象* **int** width=60, height=20;  BufferedImage image = **new** BufferedImage(width, height, BufferedImage.***TYPE\_INT\_RGB***); *// 获取图形上下文* Graphics g = image.getGraphics(); *//生成随机类* Random random = **new** Random(); *// 设定背景色* g.setColor(getRandColor(200,250));  g.fillRect(0, 0, width, height); *//设定字体* g.setFont(**new** Font(**"Times New Roman"**,Font.***PLAIN***,18)); *//画边框 //g.setColor(new Color()); //g.drawRect(0,0,width-1,height-1); // 随机产生155条干扰线，使图象中的认证码不易被其它程序探测到* g.setColor(getRandColor(160,200));  **for** (**int** i=0;i<155;i++)  {  **int** x = random.nextInt(width);  **int** y = random.nextInt(height);  **int** xl = random.nextInt(12);  **int** yl = random.nextInt(12);  g.drawLine(x,y,x+xl,y+yl);  } *// 取随机产生的认证码(4位数字)* String sRand=**""**;  **for** (**int** i=0;i<4;i++){  String rand=String.*valueOf*(random.nextInt(10));  sRand+=rand;  *// 将认证码显示到图象中* g.setColor(**new** Color(20+random.nextInt(110),20+random.nextInt(110),20+random.nextInt(110)));*//调用函数出来的颜色相同，可能是因为种子太接近，所以只能直接生成* g.drawString(rand,13\*i+6,16);  }  *// System.out.println("freedesign:"+sRand); // 将认证码存入SESSION* session.setAttribute(**"freedesign"**,sRand);  *// 图象生效* g.dispose(); *// 输出图象到页面* ImageIO.*write*(image, **"JPEG"**, response.getOutputStream()); **%>** |

## 8.2 导入到login.jsp中

|  |
| --- |
| 验证码:<**input type="text" name="imageCode"**/><**img src="${**pageContext.request.contextPath**}/imageCode"**/><**br**/> |

## 8.3 实现图形验证码原理



## 8.4 ImageCodeAuthenticationFilter过滤器

|  |
| --- |
| **package** org.lzq.Sercurity;  **import** org.springframework.security.core.AuthenticationException; **import** org.springframework.web.filter.OncePerRequestFilter;  **import** javax.servlet.FilterChain; **import** javax.servlet.ServletException; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  **public class** ImageCodeAuthencaitionFilter **extends** OncePerRequestFilter {  **private** MyFailSecurity **myFailSecurity**;   **public void** setMyFailSecurity(MyFailSecurity myFailSecurity) {  **this**.**myFailSecurity** = myFailSecurity;  }   @Override  **protected void** doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) **throws** ServletException, IOException {  *//判断当前请求是否为login请求* **if**(request.getRequestURI().contains(**"/login"**)){  *//校验验证码* **try** {  *//获取用户输入的验证码* **final** String image = request.getParameter(**"imageCode"**);   *//获取系统生成验证码* String key = (String) request.getSession().getAttribute(**"key"**);   **if** (image == **null**){  **throw new** ImageException(**"验证码必须输入"**);  }   **if** (!image.trim().equals(key.trim())){  **throw new** ImageException(**"验证码输入不正确"**);  }  }**catch** (AuthenticationException e){  **myFailSecurity**.onAuthenticationFailure(request,response, e);  **return**;  }  }  filterChain.doFilter(request,response);  } } |

## 8.5 ImageException定义

|  |
| --- |
| **public class** ImageException **extends** AuthenticationException {   **public** ImageException(String explanation) {  **super**(explanation);  }   } |

### 8.5.1 MyFailHandly的Json数据补充

|  |
| --- |
| **public class** MyFailSecurity **implements** AuthenticationFailureHandler {  ObjectMapper **objectMapper** = **new** ObjectMapper();   @Override  **public void** onAuthenticationFailure(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, AuthenticationException e) **throws** IOException, ServletException {  Map map = **new** HashMap();  map.put(**"sucess"**,**true**);  *//返回错误信息* map.put(**"errorMsg"**,e.getMessage());  String json = **objectMapper**.writeValueAsString(map);  httpServletResponse.getWriter().write(json);  } } |

## 8.6 SpringSecurity配置文件定义

### 8.6.1 定义bean

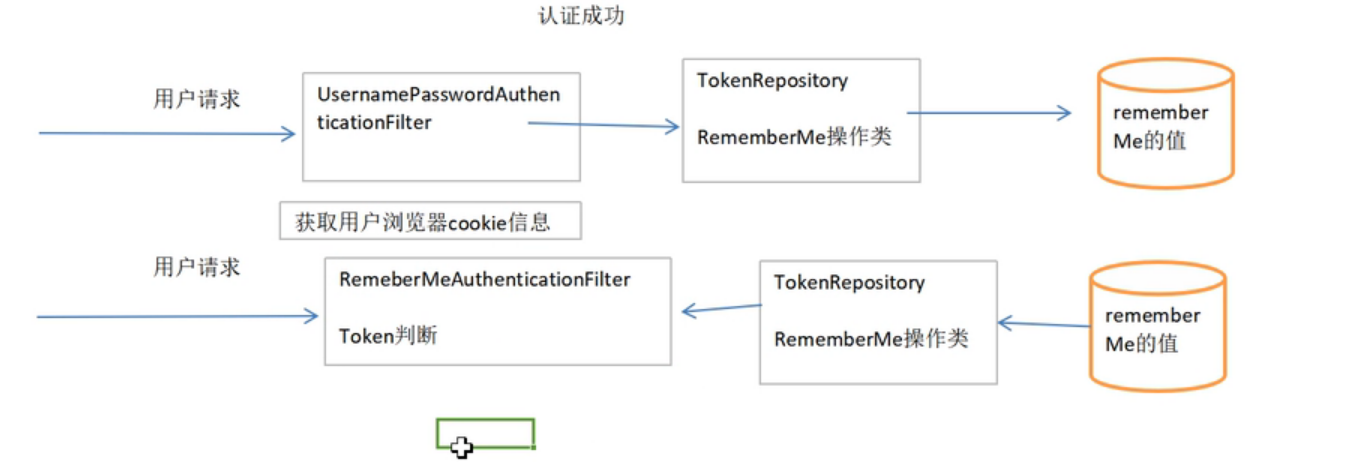
|  |
| --- |
| <**bean id="imageCodeAuthencaitionFilter" class="org.lzq.Sercurity.ImageCodeAuthencaitionFilter"**>  <**property name="myFailSecurity" ref="myFailSecurity"**/> </**bean**> |

### 8.6.2 定义Spring Security自定义过滤器

|  |
| --- |
| <**security:custom-filter ref="imageCodeAuthencaitionFilter" before="FORM\_LOGIN\_FILTER"**/> |

# 9 Remember me记住我

执行流程：



## 9.1 在登陆页面添加remember me复选框

|  |
| --- |
| <**form action="${**pageContext.request.contextPath**}/login" method="POST"**>  用户名:<**input type="text" name="username"**/>  <**br**/>  密码:<**input type="password" name="password"**/>  <**br**/>  验证码:<**input type="text" name="imageCode"**/><**img src="${**pageContext.request.contextPath**}/imageCode"**/><**br**/>  记住我:<**input type="checkbox" name="remeber-me" value="true"**><**br**/>  <**input type="submit" value="登陆"**> </**form**> |

## 9.2 Spring Security配置

|  |
| --- |
| <**bean id="jdbcTokenRepository" class="org.springframework.security.web.authentication.rememberme.JdbcTokenRepositoryImpl"**>  <**property name="dataSource" ref="DataSource"**/>  <**property name="createTableOnStartup" value="true"**/> </**bean**> |

|  |
| --- |
| *<!-- token-validity-seconds:最大记录时间 -->* <**security:remember-me token-repository-ref="jdbcTokenRepository" token-validity-seconds="3600"**/> |

# 10 Spring Security权限标签使用

## 10.1 导入标签库坐标

|  |
| --- |
| <**dependency**>  <**groupId**>org.springframework.security</**groupId**>  <**artifactId**>spring-security-taglibs</**artifactId**>  <**version**>4.2.3.RELEASE</**version**> </**dependency**> |

## 10.2 在Jsp中导入标签库

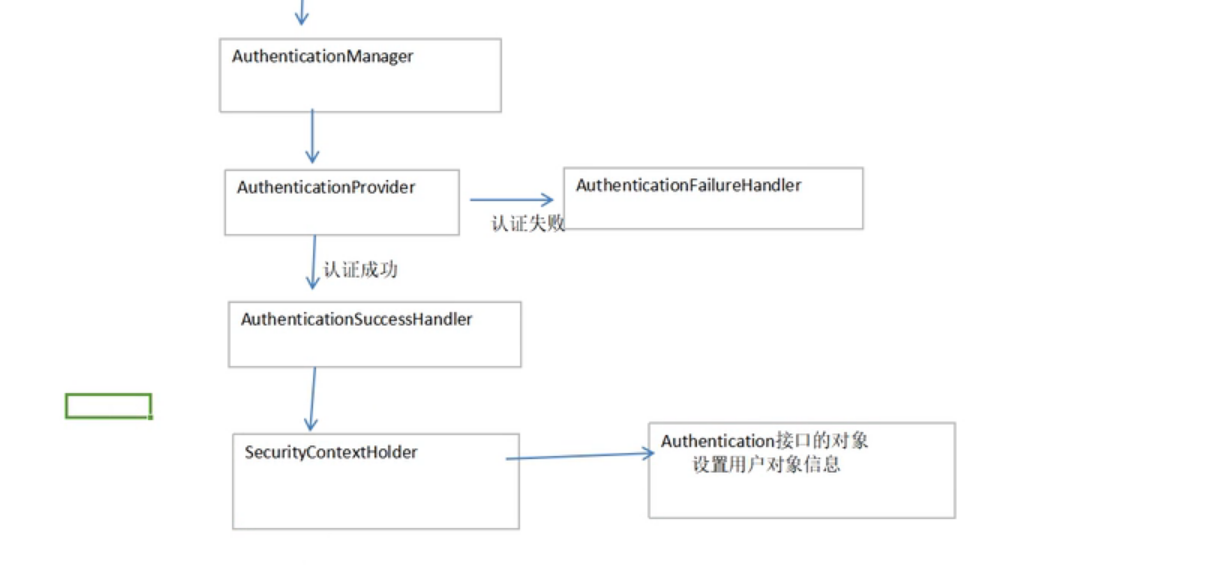
|  |
| --- |
| <%@ **taglib uri**="**http://www.springframework.org/security/tags**" **prefix**="**security**" %> |

## 10.2 Spring Security标签库的使用

|  |
| --- |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <%@ **taglib uri**="**http://www.springframework.org/security/tags**" **prefix**="**security**" %> <**html**> <**head**>  <**title**>Title</**title**> </**head**>  <**body**>  以下是网站功能<**br**/>  <**security:authorize access="hasAuthority('ROLE\_ADD\_PRODUCT')"**>  <**a href="security/addProject.jsp"**>addProject</**a**><**br**/>  </**security:authorize**>  <**security:authorize access="hasAuthority('ROLE\_DELETE\_PRODUCT')"**>  <**a href="security/delProject.jsp"**>delProject</**a**><**br**/>  </**security:authorize**>  <**security:authorize access="hasAuthority('ROLE\_UPDATE\_PRODUCT')"**>  <**a href="security/updataProject.jsp"**>updataProject</**a**><**br**/>  </**security:authorize**>  <**security:authorize access="hasAuthority('ROLE\_LIST\_PRODUCT')"**>  <**a href="security/queryProject.jsp"**>queryProject</**a**><**br**/>  </**security:authorize**>  </**body**> </**html**> |

# 11 如何获取登陆后的用户名

原理：关键使用SecurityContextHoldel接口，用于操作认证信息



## 11.1 Controller逻辑编写

|  |
| --- |
| Object principal = SecurityContextHolder.*getContext*().getAuthentication().getPrincipal(); **if**(principal != **null**){  **if**(principal **instanceof** User){  User userDetails = (User) principal;  String username = userDetails.getUsername();  model.addAttribute(**"username"**,username);  } } |