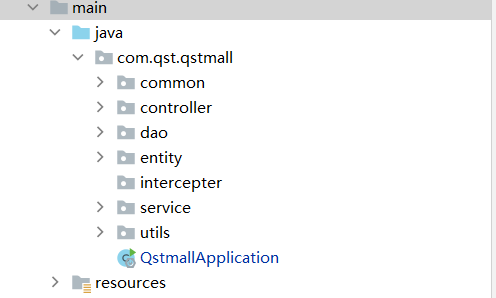
### 1、项目新建

IDEA中新建Spring Initializr（Spring boot）项目qstmall

### 2、项目包结构

在项目Java目录下创建如下包结构：



### 3、静态资源导入

将所有静态资源复制到resources-》static目录，静态资源请从资料共享中下载

### 4、依赖

在pop.xml添加依赖，本任务用到的所有依赖如下：

<dependencies>  
    *<!--thymeleaf模板引擎-->*  
<dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-thymeleaf</artifactId>  
    </dependency>  
    *<!--Springboot核心依赖-->*  
<dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-web</artifactId>  
    </dependency>

    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-test</artifactId>  
        <scope>test</scope>  
    </dependency>  
  
    *<!--mysqljdbc-->*  
<dependency>  
        <groupId>mysql</groupId>  
        <artifactId>mysql-connector-java</artifactId>  
    </dependency>  
  
    *<!--MyBatis-->*  
<dependency>  
        <groupId>org.mybatis.spring.boot</groupId>  
        <artifactId>mybatis-spring-boot-starter</artifactId>  
        <version>1.3.2</version>  
    </dependency>  
  
    *<!-- 验证码 -->*  
<dependency>  
        <groupId>com.github.penggle</groupId>  
        <artifactId>kaptcha</artifactId>  
        <version>2.3.2</version>  
    </dependency>  
</dependencies>

### 5、mapper目录

在resources目录中新建目录mapper，用来存放mapper映射文件

### 6、application.properties配置

在application.properties配置文件添加如下配置，配置数据源和数据库连接池（数据库用户名密码应换成自己的）

*#数据库数据源*  
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/qst\_mall\_db?useUnicode=true&characterEncoding=utf8&autoReconnect=true&useSSL=false&allowMultiQueries=true  
spring.datasource.username=root  
spring.datasource.password=123456  
*#数据库连接池设置*  
spring.datasource.hikari.minimum-idle=5  
spring.datasource.hikari.maximum-pool-size=15  
spring.datasource.hikari.auto-commit=true  
spring.datasource.hikari.idle-timeout=20000  
spring.datasource.hikari.pool-name=hikariCP  
spring.datasource.hikari.max-lifetime=30000  
spring.datasource.hikari.connection-timeout=30000  
spring.datasource.hikari.connection-test-query=SELECT 1  
  
*#配置mapper存放地址*  
mybatis.mapper-locations=classpath:mapper/\*Mapper.xml

#### 7、templates模板引擎文件目录

在templates目录下新建admin和mall目录，分别用来存放后台管理员页面和前台用户页面

### 8、结果类和工具类创建

（1）在com.qst.qstmall.utils包中，新建Result.java 该类为请求结果类，比如为ajax返回请求数据，就是一个Result对象

public class Result<T> implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 private int resultCode; *//请求码* private String message; *//请求信息* private T data; *//请求返回的数据* public Result() {  
 }  
  
 public Result(int resultCode, String message) {  
 this.resultCode = resultCode;  
 this.message = message;  
 }  
  
 public Result(int resultCode, String message,T data) {  
 this.resultCode = resultCode;  
 this.message = message;  
 this.data=data;  
 }  
  
 public int getResultCode() {  
 return resultCode;  
 }  
  
 public void setResultCode(int resultCode) {  
 this.resultCode = resultCode;  
 }  
  
 public String getMessage() {  
 return message;  
 }  
  
 public void setMessage(String message) {  
 this.message = message;  
 }  
  
 public T getData() {  
 return data;  
 }  
  
 public void setData(T data) {  
 this.data = data;  
 }  
  
 @Override  
 public String toString() {  
 return "Result{" +  
 "resultCode=" + resultCode +  
 ", message='" + message + '\'' +  
 ", data=" + data +  
 '}';  
 }  
}

（2）在com.qst.qstmall.utils包中新建ResultGenerator类，该类用于为请求构建一个Result对象结果提供工具方法

public class ResultGenerator {  
 private static final String *DEFAULT\_SUCCESS\_MESSAGE* = "SUCCESS";  
 private static final String *DEFAULT\_FAIL\_MESSAGE* = "FAIL";  
 private static final int *RESULT\_CODE\_SUCCESS* = 200;  
 private static final int *RESULT\_CODE\_SERVER\_ERROR* = 500;  
  
 public static Result genSuccessResult() {  
 Result result = new Result();  
 result.setResultCode(*RESULT\_CODE\_SUCCESS*);  
 result.setMessage(*DEFAULT\_SUCCESS\_MESSAGE*);  
 return result;  
 }  
  
 public static Result genSuccessResult(String message) {  
 Result result = new Result();  
 result.setResultCode(*RESULT\_CODE\_SUCCESS*);  
 result.setMessage(message);  
 return result;  
 }  
  
 public static Result genSuccessResult(Object data) {  
 Result result = new Result();  
 result.setResultCode(*RESULT\_CODE\_SUCCESS*);  
 result.setMessage(*DEFAULT\_SUCCESS\_MESSAGE*);  
 result.setData(data);  
 return result;  
 }  
  
 public static Result genFailResult(String message) {  
 Result result = new Result();  
 result.setResultCode(*RESULT\_CODE\_SERVER\_ERROR*);  
 if (StringUtils.*isEmpty*(message)) {  
 result.setMessage(*DEFAULT\_FAIL\_MESSAGE*);  
 } else {  
 result.setMessage(message);  
 }  
 return result;  
 }  
  
 public static Result genErrorResult(int code, String message) {  
 Result result = new Result();  
 result.setResultCode(code);  
 result.setMessage(message);  
 return result;  
 }  
}

（3）在com.qst.qstmall.utils新建MD5加密工具类，提供为登录密码进行MD5加密的工具方法

public class MD5Util {  
  
 public static String MD5Encode(String origin) {  
 String resultString = null;  
 try {  
 MessageDigest md5 = MessageDigest.getInstance("MD5");  
 BASE64Encoder base64Encoder=new BASE64Encoder();  
 //加密字符串  
 resultString=base64Encoder.encode(md5.digest(origin.getBytes("utf-8")));  
 } catch (Exception exception) {  
 exception.printStackTrace();  
 }  
 return resultString;  
 }  
}

### 9、实体类创建

在com.qst.qstmall.entity包下新建用户实体类MallUser.java

public class MallUser {  
 private Long userId;  
 private String nickName;  
 private String loginName;  
 private String passwordMd5;  
 private String introduceSign;  
 private String address;  
 private Byte isDeleted;  
 private Byte lockedFlag;  
 @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")  
 private Date createTime;

…………get和set方法省略，自行添加

### 10、常量类创建

在com.qst.qstmall.common创建常量类，Constants.java

如下（包含本项目用到的所有常量）

public class Constants {  
 public final static String *FILE\_UPLOAD\_DIC* = "D:\\WXD\\upload\\";*//上传文件的默认url前缀，根据部署设置自行修改* public final static int *INDEX\_CAROUSEL\_NUMBER* = 5;*//首页轮播图数量(可根据自身需求修改)* public final static int *INDEX\_CATEGORY\_NUMBER* = 10;*//首页一级分类的最大数量* public final static int *SEARCH\_CATEGORY\_NUMBER* = 8;*//搜索页一级分类的最大数量* public final static int *INDEX\_GOODS\_HOT\_NUMBER* = 4;*//首页热卖商品数量* public final static int *INDEX\_GOODS\_NEW\_NUMBER* = 5;*//首页新品数量* public final static int *INDEX\_GOODS\_RECOMMOND\_NUMBER* = 10;*//首页推荐商品数量* public final static int *PROMOTION\_GOODS\_NUMBER* = 20;*//首页推荐商品数量* public final static int *SHOPPING\_CART\_ITEM\_TOTAL\_NUMBER* = 20;*//购物车中商品的最大数量(可根据自身需求修改)* public final static int *SHOPPING\_CART\_ITEM\_LIMIT\_NUMBER* = 200;*//购物车中单个商品的最大购买数量(可根据自身需求修改)* public final static String *MALL\_VERIFY\_CODE\_KEY* = "mallVerifyCode";*//验证码key* public final static String *MALL\_USER\_SESSION\_KEY* = "qstMallUser";*//session中user的key* public final static int *GOODS\_SEARCH\_PAGE\_LIMIT* = 10;*//搜索分页的默认条数(每页10条)* public final static int *ORDER\_SEARCH\_PAGE\_LIMIT* = 5;*//我的订单列表分页的默认条数(每页5条)* public final static int *SELL\_STATUS\_UP* = 0;*//商品上架状态* public final static int *SELL\_STATUS\_DOWN* = 1;*//商品下架状态* public final static int *PROMOTION\_STATUS\_NONE* = 0;*//无促销状态* public final static int *PROMOTION\_STATUS\_NOT\_STARTED* = 1;*//促销还未开始状态* public final static int *PROMOTION\_STATUS\_STARTED* = 2;*//促销开始状态* public final static int *PROMOTION\_STATUS\_END* = 3;*//促销结束状态* public final static int *ADMIN\_USER\_LOCKED* = 1;*//管理员账号锁定*}

### 11、枚举常量类创建

在com.qst.qstmall.common包下创建枚举类ServiceResultEnum.java,存放服务层处理的所有结果如下：

public enum ServiceResultEnum {  
 *ERROR*("error"),  
 *SUCCESS*("success"),  
 *DATA\_NOT\_EXIST*("未查询到记录！"),  
 *SAME\_CATEGORY\_EXIST*("有同级同名的分类！"),  
 *SAME\_LOGIN\_NAME\_EXIST*("用户名已存在！"),  
 *LOGIN\_NAME\_NULL*("请输入登录名！"),  
 *LOGIN\_PASSWORD\_NULL*("请输入密码！"),  
 *LOGIN\_VERIFY\_CODE\_NULL*("请输入验证码！"),  
 *LOGIN\_VERIFY\_CODE\_ERROR*("验证码错误！"),  
 *GOODS\_NOT\_EXIST*("商品不存在！"),  
 *GOODS\_DOWN*("商品已存在！"),  
 *SHOPPING\_CART\_ITEM\_LIMIT\_NUMBER\_ERROR*("超出单个商品的最大购买数量！"),  
 *SHOPPING\_CART\_ITEM\_EXIST\_AND\_LIMIT\_NUMBER\_ERROR*("购物车中已有此商品且购买数量达到上限"),  
 *SHOPPING\_CART\_ITEM\_TOTAL\_NUMBER\_ERROR*("超出用户购物车上限！"),  
 *LOGIN\_ERROR*("登录失败！"),  
 *LOGIN\_USER\_LOCKED*("用户已被禁止登录！"),  
 *ORDER\_NOT\_EXIST\_ERROR*("订单不存在！"),  
 *NULL\_ADDRESS\_ERROR*("地址不能为空！"),  
 *ORDER\_PRICE\_ERROR*("订单价格异常！"),  
 *ORDER\_GENERATE\_ERROR*("生成订单异常！"),  
 *SHOPPING\_ITEM\_ERROR*("购物车数据异常！"),  
 *SHOPPING\_ITEM\_COUNT\_ERROR*("库存不足！"),  
 *ORDER\_STATUS\_ERROR*("订单状态异常！"),  
 *OPERATE\_ERROR*("操作失败！"),  
 *DB\_ERROR*("database error");  
 private String result;  
 ServiceResultEnum(String result) {  
 this.result = result;  
 }  
 public String getResult() {  
 return result;  
 }  
 public void setResult(String result) {  
 this.result = result;  
 }  
}

### 12、dao层实现

（1）在com.qst.qstmall.dao包下新建dao接口MallUserMapper.java

其中@Mapper注解声明该接口实现类是mapper映射文件，并自动注入该接口对象

import com.qst.qstmall.entity.MallUser;  
import org.apache.ibatis.annotations.Mapper;  
  
@Mapper  
public interface MallUserMapper {  
 */\*注册，插入一个用户\*/* int insertSelective(MallUser record);  
 */\*根据用户名返回一个用户\*/* MallUser selectByLoginName(String loginName);  
}

（2）在resources-》mapper目录下新建Mapper映射文件MallUserMapper.xml

*<?*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"*>*<mapper namespace="com.qst.qstmall.dao.MallUserMapper">  
  
 *<!--定义查询结果集合，查询到的是AdminUser的一个集合，并定义了数据库字段和实体类属性的对应关系-->* <resultMap id="BaseResultMap" type="com.qst.qstmall.entity.MallUser">  
 <id column="user\_id" jdbcType="BIGINT" property="userId"/>  
 <result column="nick\_name" jdbcType="VARCHAR" property="nickName"/>  
 <result column="login\_name" jdbcType="VARCHAR" property="loginName"/>  
 <result column="password\_md5" jdbcType="VARCHAR" property="passwordMd5"/>  
 <result column="introduce\_sign" jdbcType="VARCHAR" property="introduceSign"/>  
 <result column="address" jdbcType="VARCHAR" property="address"/>  
 <result column="is\_deleted" jdbcType="TINYINT" property="isDeleted"/>  
 <result column="locked\_flag" jdbcType="TINYINT" property="lockedFlag"/>  
 <result column="create\_time" jdbcType="TIMESTAMP" property="createTime"/>  
 </resultMap>  
  
 *<!--定义查询表中的哪些列（为了实现复用）-->* <sql id="Base\_Column\_List">  
 user\_id, nick\_name, login\_name, password\_md5, introduce\_sign, address, is\_deleted,  
 locked\_flag, create\_time  
 </sql>  
*<!--根据用户名查询用户-->* <select id="selectByLoginName" parameterType="java.lang.String" resultMap="BaseResultMap">  
 select  
 <include refid="Base\_Column\_List"/>  
 from tb\_qst\_mall\_user  
 where login\_name = #{loginName} and is\_deleted = 0  
 </select>  
  
 *<!--插入一个用户（注册）-->* <insert id="insertSelective" parameterType="com.qst.qstmall.entity.MallUser">  
 insert into tb\_qst\_mall\_user  
 <trim prefix="(" suffix=")" suffixOverrides=",">  
 <if test="userId != null">  
 user\_id,  
 </if>  
 <if test="nickName != null">  
 nick\_name,  
 </if>  
 <if test="loginName != null">  
 login\_name,  
 </if>  
 <if test="passwordMd5 != null">  
 password\_md5,  
 </if>  
 <if test="introduceSign != null">  
 introduce\_sign,  
 </if>  
 <if test="address != null">  
 address,  
 </if>  
 <if test="isDeleted != null">  
 is\_deleted,  
 </if>  
 <if test="lockedFlag != null">  
 locked\_flag,  
 </if>  
 <if test="createTime != null">  
 create\_time,  
 </if>  
 </trim>  
 <trim prefix="values (" suffix=")" suffixOverrides=",">  
 <if test="userId != null">  
 #{userId,jdbcType=BIGINT},  
 </if>  
 <if test="nickName != null">  
 #{nickName,jdbcType=VARCHAR},  
 </if>  
 <if test="loginName != null">  
 #{loginName,jdbcType=VARCHAR},  
 </if>  
 <if test="passwordMd5 != null">  
 #{passwordMd5,jdbcType=VARCHAR},  
 </if>  
 <if test="introduceSign != null">  
 #{introduceSign,jdbcType=VARCHAR},  
 </if>  
 <if test="address != null">  
 #{address,jdbcType=VARCHAR},  
 </if>  
 <if test="isDeleted != null">  
 #{isDeleted,jdbcType=TINYINT},  
 </if>  
 <if test="lockedFlag != null">  
 #{lockedFlag,jdbcType=TINYINT},  
 </if>  
 <if test="createTime != null">  
 #{createTime,jdbcType=TIMESTAMP},  
 </if>  
 </trim>  
 </insert>  
</mapper>

### 13、Service层实现

（1）在com.qst.qstmall.service包中新建接口QSTMallUserService.java

public interface QstMallUserService {  
  
 */\*\*  
 \* 用户注册  
 \*/* String register(String loginName, String password);  
}

(2)在com.qst.qstmall.service包中新建子包impl，该包用来存放service实现类，并在com.qst.qstmall.service.impl包中新建QSTMallUserService接口实现类QstMallUserServiceImpl

@Service  
public class QstMallUserServiceImpl implements QstMallUserService {  
  
 @Autowired  
 private MallUserMapper mallUserMapper;  
  
 @Override  
 public String register(String loginName, String password) {  
 if (mallUserMapper.selectByLoginName(loginName) != null) {  
 return ServiceResultEnum.*SAME\_LOGIN\_NAME\_EXIST*.getResult();  
 }  
 MallUser registerUser = new MallUser();  
 registerUser.setLoginName(loginName);  
 registerUser.setNickName(loginName);  
 String passwordMD5 = MD5Util.*MD5Encode*(password);  
 registerUser.setPasswordMd5(passwordMD5);  
 if (mallUserMapper.insertSelective(registerUser) > 0) {  
 return ServiceResultEnum.*SUCCESS*.getResult();  
 }  
 return ServiceResultEnum.*DB\_ERROR*.getResult();  
 }  
}

### 14、Controller层实现

（1）在com.qst.qstmall.controller包中新建CommonController类，该类用于获取验证码的请求

@Controller  
public class CommonController {  
  
 @GetMapping("/common/kaptcha")  
 public void mallKaptcha(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse) throws Exception {  
 DefaultKaptcha qstMallLoginKaptcha = new DefaultKaptcha();  
 Properties properties = new Properties();  
 properties.put("kaptcha.border", "no");  
 properties.put("kaptcha.textproducer.font.color", "27,174,171");  
 properties.put("kaptcha.noise.color", "20,33,42");  
 properties.put("kaptcha.textproducer.font.size", "30");  
 properties.put("kaptcha.image.width", "110");  
 properties.put("kaptcha.image.height", "40");  
 properties.put("kaptcha.session.key", Constants.*MALL\_VERIFY\_CODE\_KEY*);  
 properties.put("kaptcha.textproducer.char.space", "2");  
 properties.put("kaptcha.textproducer.char.length", "6");  
 Config config = new Config(properties);  
 qstMallLoginKaptcha.setConfig(config);  
 byte[] captchaOutputStream = null;  
 ByteArrayOutputStream imgOutputStream = new ByteArrayOutputStream();  
 try {  
 *//生产验证码字符串并保存到session中* String verifyCode = qstMallLoginKaptcha.createText();  
 httpServletRequest.getSession().setAttribute(Constants.*MALL\_VERIFY\_CODE\_KEY*, verifyCode);  
 BufferedImage challenge = qstMallLoginKaptcha.createImage(verifyCode);  
 ImageIO.*write*(challenge, "jpg", imgOutputStream);  
 } catch (IllegalArgumentException e) {  
 httpServletResponse.sendError(HttpServletResponse.*SC\_NOT\_FOUND*);  
 return;  
 }  
 captchaOutputStream = imgOutputStream.toByteArray();  
 httpServletResponse.setHeader("Cache-Control", "no-store");  
 httpServletResponse.setHeader("Pragma", "no-cache");  
 httpServletResponse.setDateHeader("Expires", 0);  
 httpServletResponse.setContentType("image/jpeg");  
 ServletOutputStream responseOutputStream = httpServletResponse.getOutputStream();  
 responseOutputStream.write(captchaOutputStream);  
 responseOutputStream.flush();  
 responseOutputStream.close();  
 }  
}

（2）在com.qst.qstmall.controller包中新建PersonalController类，处理用户注册请求

@Controller  
public class PersonalController {  
  
 @Resource  
 private QstMallUserService qstMallUserService;  
  
 @GetMapping({"/register","/register.html"})  
 public String register(){  
 return "mall/register";  
 }  
  
 @PostMapping("/register")  
 @ResponseBody  
 public Result register(String loginName, String verifyCode, String password, HttpSession httpSession) {  
 if (StringUtils.*isEmpty*(loginName)) {  
 return ResultGenerator.*genFailResult*(ServiceResultEnum.*LOGIN\_NAME\_NULL*.getResult());  
 }  
 if (StringUtils.*isEmpty*(password)) {  
 return ResultGenerator.*genFailResult*(ServiceResultEnum.*LOGIN\_PASSWORD\_NULL*.getResult());  
 }  
 if (StringUtils.*isEmpty*(verifyCode)) {  
 return ResultGenerator.*genFailResult*(ServiceResultEnum.*LOGIN\_VERIFY\_CODE\_NULL*.getResult());  
 }  
 String kaptchaCode = httpSession.getAttribute(Constants.*MALL\_VERIFY\_CODE\_KEY*) + "";  
 if (StringUtils.*isEmpty*(kaptchaCode) || !verifyCode.equals(kaptchaCode)) {  
 return ResultGenerator.*genFailResult*(ServiceResultEnum.*LOGIN\_VERIFY\_CODE\_ERROR*.getResult());  
 }  
 *//todo 清verifyCode* String registerResult = qstMallUserService.register(loginName, password);  
 *//注册成功* if (ServiceResultEnum.*SUCCESS*.getResult().equals(registerResult)) {  
 return ResultGenerator.*genSuccessResult*();  
 }  
 *//注册失败* return ResultGenerator.*genFailResult*(registerResult);  
 }  
}

### 15、View视图层实现

在templates-》mall目录下新建普通用户注册页面register.html，参考代码见华为云代码仓库