武汉纺织大学

Java应用开发课程设计

**基于freeMarker的自动代码生成工具**

**学 院： 数学与计算机学院**

**班 级： 软件11802**

**姓 名： 胡徐安**

**学 号： 1804240425**

**指导老师： 聂刚**

**成 绩：**

**完成日期： 2020年6月9日**

目 录

[1需求分析 1](#_Toc42626792)

[1.1配置界面 1](#_Toc42626793)

[1.2配置的各项要求讲解 1](#_Toc42626794)

[1.3各项功能 1](#_Toc42626795)

[1.3.1 vo类实体文件的生成 1](#_Toc42626796)

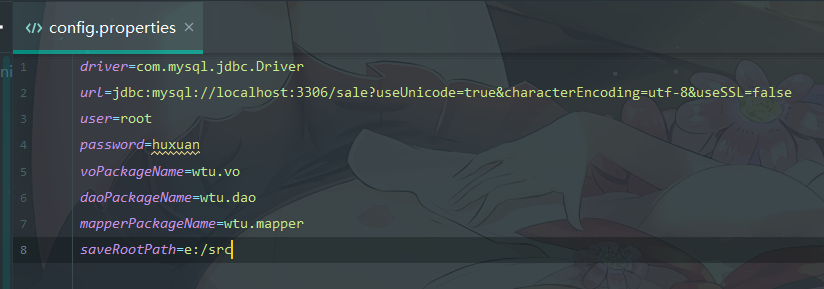
[1.3.2 dao接口文件的生成 2](#_Toc42626797)

[1.3.3 mapper映射文件的生成 2](#_Toc42626798)

## 1需求分析

设计一个代码生成工具，主要目的是生成mybatis框架的对应代码，对数据库中指定的表生成对应的实体类源文件，生成mapper映射文件（包含一些基本的CRUD操作），并生成对应的DAO接口文件。

### 1.1配置界面



### 1.2配置的各项要求讲解

配置文件采用的是properties文件，用户需要在配置文件里配置相关信息，这些基础信息是必须包含的:

* driver：填写数据库驱动的路径
* url:填写数据库连接相关的url信息
* user:数据库连接相关的用户名信息
* password:数据库连接相关的密码信息
* voPackageName:数据库对应vo实体类生成的包路径
* daoPackageName:对应DAO接口类生成的包路径
* mapperPackageName:对应接口的mapper映射文件生成的包路径
* saveRootPath:代码生成存放的根目录

### 1.3各项功能

#### 1.3.1 vo类实体文件的生成

创建vo实体类模板，读取模板，再通过读取用户配置的相关数据，进行对数据库元数据的操作，获得数据模型，最后将模板与对应数据模型组合输出到相应的文件夹中，实现对vo 实体类源文件的生成。

#### 1.3.2 dao接口文件的生成

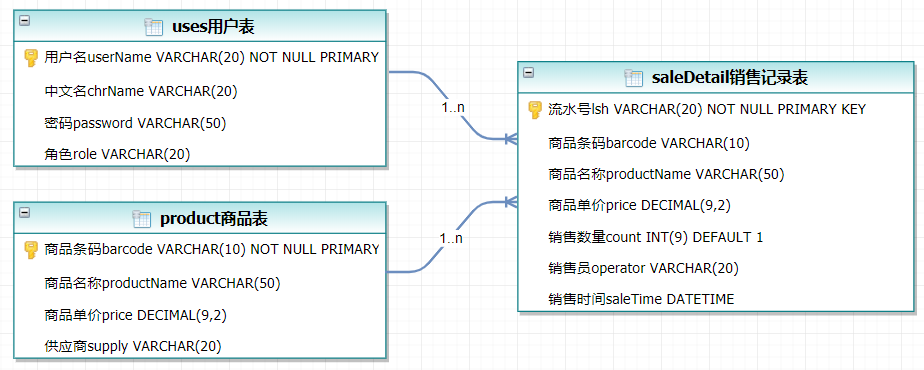
创建dao接口类模板，读取模板，再通过读取用户配置的相关数据，进行对数据库元数据的操作，获得数据模型，最后将模板与对应数据模型组合输出到相应的文件夹中，实现对dao 接口类源文件的生成。

#### 1.3.3 mapper映射文件的生成

创建mapper映射文件模板，读取模板，再通过读取用户配置的相关数据，进行对数据库元数据的操作，获得数据模型，最后将模板与对应数据模型组合输出到相应的文件夹中，实现对mapper映射文件的生成。

## 2.系统设计

### 2.1ER图



数据库ER图

### 2.2UML类图（Class Diagram）

对于代码生成的实现，共设计如下5个类：

* 数据库元数据类dbMeteData:连接数据库，读取数据库元数据并做处理，返回数据库相关元数据信息，数据库的关闭
* 实体类fieldList:与数据库相映射的类，用于封装数据库的字段名称及其对应字段类型
* Vo代码生成模板的数据模型类voTempData:用于获取数据库每张表对应的vo实体类模板的数据模型
* Dao代码生成模板的数据模型类daoTempData:用于获取数据库每张表对应的dao接口类模板的数据模型
* Mapper代码生成模板的数据模型类：mapperTempData:用于获取数据库每张表对应的mapper映射文件的数据模型

各类的结构及类之间的相关关系如图2-2所示：

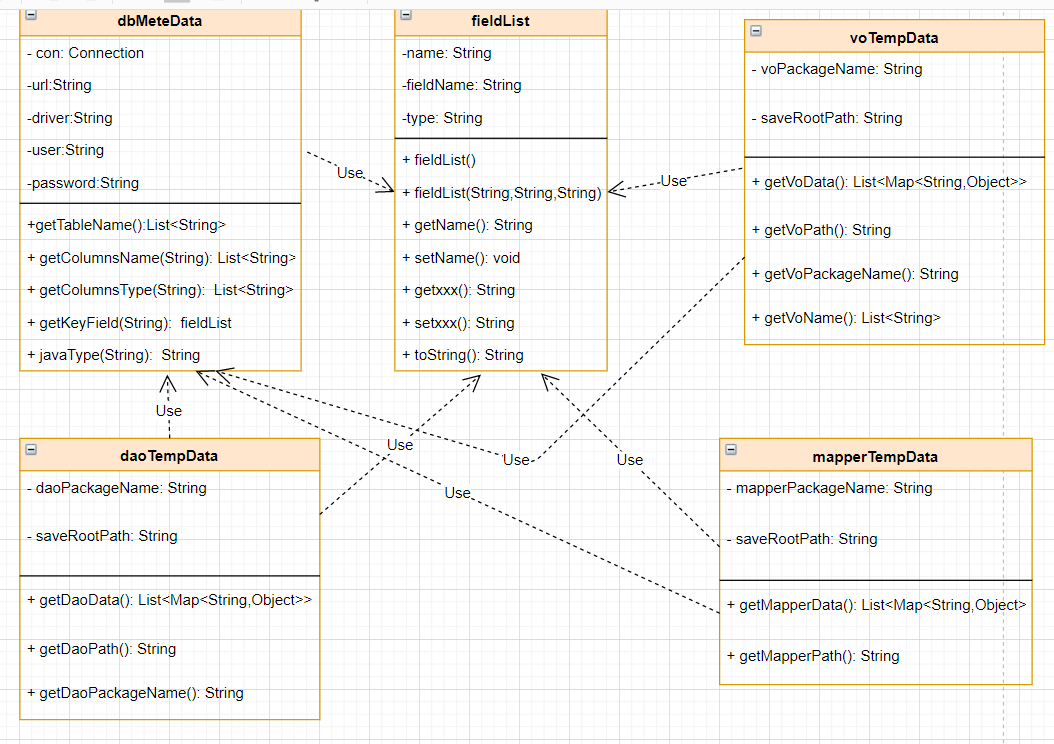


图2-2

### 2.3 UML时序图（Sequence Diagram）

#### 2.3.1vo实体类模板数据获得



#### 2.3.2dao接口类模板数据模型获得



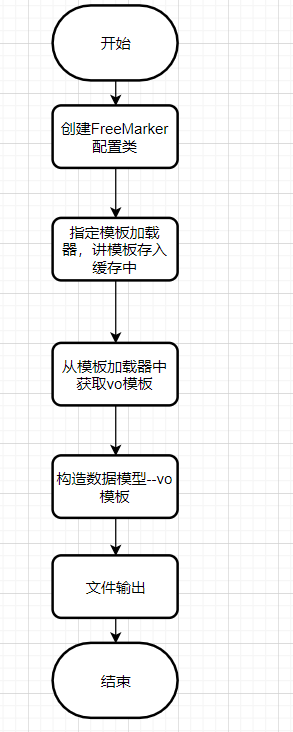
#### 2.3.3mapper映射文件模板数据模型获得



### 2.4UML活动图（Activity Diagram）

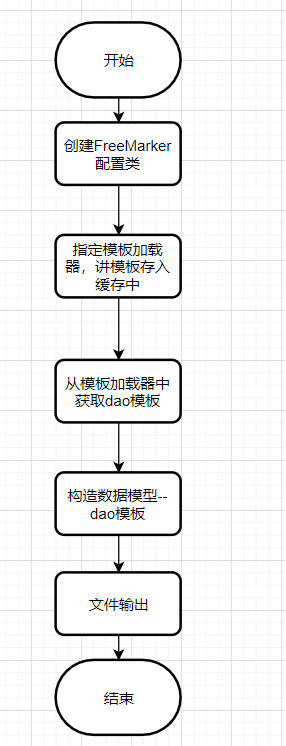
#### 2.4.1vo实体类文件生成

实体类文件生成首先要创建freeMarker配置类，然后加载模板，获取模板，获得对应模板的数据模型，最后输出文件到指定位置。具体流程如下流程图：



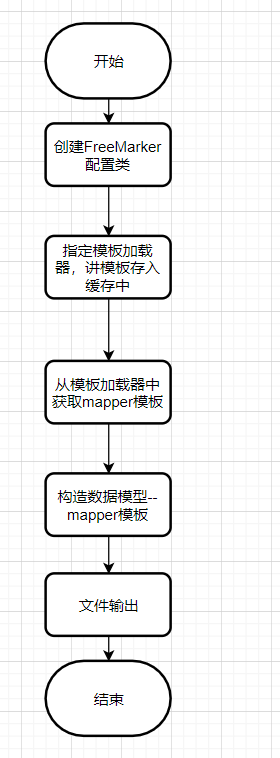
#### 2.4.2dao接口文件的生成

dao接口类文件生成首先要创建freeMarker配置类，然后加载模板，获取模板，获得对应模板的数据模型，最后输出文件到指定位置。具体流程如下流程图：



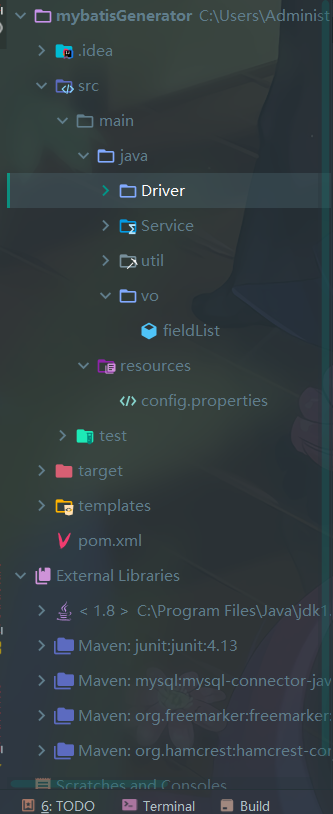
#### 2.4.3mapper映射文件的生成

Mapper映射文件生成首先要创建freeMarker配置类，然后加载模板，获取模板，获得对应模板的数据模型，最后输出文件到指定位置。具体流程如下流程图：



## 3.系统实现

### 3.1项目结构



### 3.2配置文件

新建一个maven项目，在maven项目的pom.xml配置相关依赖

#### 3.2.1pom.xml文件

该文件主要是配置maven项目的相关依赖包

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.study</groupId>  
 <artifactId>mybatisGenerator</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>6</source>  
 <target>6</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
 <dependencies>  
 <dependency>  
 <groupId>org.freemarker</groupId>  
 <artifactId>freemarker</artifactId>  
 <version>2.3.30</version>  
 </dependency>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.49</version>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
</project>

#### 3.2.2config.properties

该文件在创建的maven项目的src/main/resources目录下，用来进行代码生成的相关配置

driver=com.mysql.jdbc.Driver  
url=jdbc:mysql://localhost:3306/sale?useUnicode=true&characterEncoding=utf-8&useSSL=false  
user=root  
password=huxuan  
voPackageName=wtu.vo  
daoPackageName=wtu.dao  
mapperPackageName=wtu.mapper  
saveRootPath=e:/src

注：配置相关要求在第一页

### 3.3vo包里的fieldList.java

该类用来对数据库中的元数据进行封装的数据模型，主要包括数据库字段名，java字段名和对应java类型

package vo;  
  
public class fieldList {  
 private String name;//数据库中的字段名  
 private String fieldName;//对应的实体类中的属性名  
 private String type;//数据库字段类型  
  
 public fieldList() {  
 super();  
 }  
  
 public fieldList(String name, String fieldName, String type) {  
 this.name = name;  
 this.fieldName = fieldName;  
 this.type = type;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getFieldName() {  
 return fieldName;  
 }  
  
 public void setFieldName(String fieldName) {  
 this.fieldName = fieldName;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public void setType(String type) {  
 this.type = type;  
 }  
}

### 3.4util工具包的dbMeteData.java

该类用来连接数据库并处理好元数据，进行封装返回。

package util;  
  
import vo.fieldList;  
  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.*List*;  
import java.util.Properties;  
  
public class dbMeteData {  
 private static *Connection con*;  
 private static String *url*;  
 private static String *driver*;  
 private static String *user*;  
 private static String *password*;  
  
 static {  
 Properties pro = new Properties();  
 InputStream in = dbMeteData.class.getClassLoader().getResourceAsStream("config.properties");  
 try {  
 pro.load(new InputStreamReader(in, "utf-8"));  
 in.close();  
 *driver* = pro.getProperty("driver");  
 *url* = pro.getProperty("url");  
 *user* = pro.getProperty("user");  
 *password* = pro.getProperty("password");  
 Class.*forName*(*driver*);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static *List*<String > getTableName(){  
 *List*<String> tableNames=new ArrayList<>();  
 try {  
 *con* = DriverManager.*getConnection*(*url*, *user*, *password*);  
 *DatabaseMetaData* metaData = *con*.getMetaData();  
 //获取数据库的表信息  
 *ResultSet* rs = metaData.getTables(null, null, null, new String[]{"TABLE"});  
 while (rs.next()) {  
 //获取数据库的表名  
 String tableName = rs.getString("TABLE\_NAME");  
 tableNames.add(tableName);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }finally {  
 try {  
 *con*.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 return tableNames;  
 }  
  
 //获取数据库表中的字段名  
 public static *List*<String> getColumnsName(String tableName) {  
 *List*<String> columnsName = new ArrayList<>();  
 try {  
 *con* = DriverManager.*getConnection*(*url*, *user*, *password*);  
 *DatabaseMetaData* metaData = *con*.getMetaData();  
 //获取表中字段信息  
 *ResultSet* rs = metaData.getColumns(null, null, tableName, null);  
 while (rs.next()) {  
 //获取字段名  
 String name = rs.getString("COLUMN\_NAME");  
 columnsName.add(name);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 *con*.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 return columnsName;  
 }  
  
 //获取数据库表中的属性的类型  
 public static *List*<String> getColumnsType(String tableName){  
 *List*<String> columnsType =new ArrayList<>();  
 try {  
 *con*=DriverManager.*getConnection*(*url*, *user*, *password*);  
 *DatabaseMetaData* metaData = *con*.getMetaData();  
 //获取表中字段信息  
 *ResultSet* rs = metaData.getColumns(null, null, tableName, null);  
 while(rs.next()){  
 String columnType = rs.getString("TYPE\_NAME");  
 columnsType.add(columnType);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }finally {  
 try {  
 *con*.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 return columnsType;  
 }  
  
 //获取数据库表中的主键名称  
 public static fieldList getKeyField(String tableName){  
 fieldList keyField=new fieldList();  
 String name="";  
 String fieldName="";  
 String type="";  
 try {  
 *con*=DriverManager.*getConnection*(*url*, *user*, *password*);  
 *DatabaseMetaData* metaData = *con*.getMetaData();  
 //获取表中字段信息  
 *ResultSet* rs = metaData.getPrimaryKeys(null,null,tableName);  
 while(rs.next()){  
 name=rs.getString("COLUMN\_NAME");  
 keyField.setName(name);  
 fieldName=name.substring(0, 1).toLowerCase() + name.substring(1);  
 keyField.setFieldName(fieldName);  
 *ResultSet* rs1 = metaData.getColumns(null, null, tableName, name);  
 while(rs1.next()){  
 type=rs1.getString("TYPE\_NAME");  
 type=*javaType*(type);  
 }  
 keyField.setType(type);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }finally {  
 try {  
 *con*.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 return keyField;  
 }  
  
 //将数据库中字段类型转化为对应的java类型  
 public static String javaType(String columnType) {  
 String Type = String.*valueOf*(columnType);  
 String type = "";  
 switch (Type) {  
 case "VARCHAR":  
 type = "String";  
 break;  
 case "DECIMAL":  
 type = "double";  
 break;  
 case "INT":  
 type = "int";  
 break;  
 case "DATETIME":  
 type = "String";  
 break;  
 default:  
 type = "";  
 }//后面可以自己添加类型  
 return type;  
 }  
}

### 3.5template包

该包用来存模板

#### 3.5.1vo实体类的模板文件voTemplate.ftl

package ${packageName};  
  
public class ${className}{  
<#list fieldList as field>  
 private ${field.type} ${field.name};  
</#list>  
  
 public ${className}() {  
 super();  
 }  
  
 public ${className}(<#list fieldList as field>${field.type} ${field.name}<#sep>,</#list>) {  
 <#list fieldList as field>  
 this.${field.name}=${field.name};  
 </#list>  
 }  
  
<#list fieldList as field>  
 public void set${field.name?cap\_first}(${field.type} ${field.name}) {  
 this.${field.name}=${field.name};  
 }  
  
 public ${field.type} get${field.name?cap\_first}() {  
 return ${field.name};  
 }  
</#list>  
  
 public String toString() {  
 return "${className}{"+  
 <#list fieldList as field>"${field.name}="+${field.name}<#sep >+","+</#list>;  
 }  
}

#### 3.5.2dao接口类的模板文件daoTemplate.ftl

package ${daoPackageName};  
import java.util.List;  
import ${voPackageName}.${voName};  
  
public interface I${voName}DAO{  
 //根据主键查找记录  
 public ${voName} getBy${keyField.name?cap\_first}(${keyField.type} ${keyField.name});  
 //根据条件查找  
 public List<${voName}> query(${voName} ${voName?uncap\_first});  
 //增加  
 public int insert(${voName} ${voName?uncap\_first});  
 //修改  
 public int update(${voName} ${voName?uncap\_first});  
 //按主键删除  
 public int delete(${keyField.type} ${keyField.name});  
 //按条件删除  
 public int deleteCondition(${voName} ${voName?uncap\_first});  
}

#### 3.5.3mapper映射文件的模板文件mapperTemplate.ftl

<?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="${daoPackageName}.I${voName}DAO">  
 <!--当前数据库字段名与实体类中属性名不一致时，需要显示定义字段映射 -->  
 <resultMap id="${voName?uncap\_first}ResultMap" type="${voPackageName}.${voName}">  
 <id property="${keyField.fieldName}" column="${keyField.name}"/>  
 <#list fieldList as field>  
 <#if field.name !=keyField.name>  
 <result property="${field.fieldName}" column="${field.name}"/>  
 </#if>  
 </#list>  
 </resultMap>  
  
 <sql id="${voName?uncap\_first}Colums">  
 <#list fieldList as field>${field.name}<#sep >,</#list>  
 </sql>  
  
 <select id="getBy${keyField.name?cap\_first}" parameterType="${keyField.type}" resultMap="${voName?uncap\_first}ResultMap">  
 select  
 <include refid="${voName?uncap\_first}Colums"></include>  
 from ${voName?uncap\_first}  
 where ${keyField.name}=<#noparse>#{</#noparse>${keyField.fieldName}}  
 </select>  
  
 <select id="query" parameterType="${voPackageName}.${voName}" resultMap="${voName?uncap\_first}ResultMap">  
 select  
 <include refid="${voName?uncap\_first}Colums"></include>  
 from ${voName?uncap\_first}  
 <where>  
 <#list fieldList as field>  
 <if test="${field.name}!=null">  
 and ${field.name} = <#noparse>#{</#noparse>${field.fieldName}}  
 </if>  
 </#list>  
 </where>  
 </select>  
  
 <insert id="insert" parameterType="${voPackageName}.${voName}">  
 insert into ${voName?uncap\_first}(<include refid="${voName?uncap\_first}Colums"></include>)  
 values(<#list fieldList as field><#noparse>#{</#noparse>${field.fieldName}}<#sep>,</#list>)  
 </insert>  
  
 <update id="update" parameterType="${voPackageName}.${voName}">  
 update ${voName?uncap\_first}  
 <set>  
 <#list fieldList as field>  
 <if test="${field.name}!=null">  
 ${field.name} = <#noparse>#{</#noparse>${field.fieldName}}<#sep>,</#sep>  
 </if>  
 </#list>  
 </set>  
 where ${keyField.name}=<#noparse>#{</#noparse>${keyField.fieldName}}  
 </update>  
  
 <delete id="delete" parameterType="${keyField.type}">  
 delete from  
 ${voName?uncap\_first}  
 where ${keyField.name}=<#noparse>#{</#noparse>${keyField.fieldName}}  
 </delete>  
  
 <delete id="deleteCondition" parameterType="${voPackageName}.${voName}">  
 delete from  
 ${voName?uncap\_first}  
 <where>  
 <#list fieldList as field>  
 <if test="${field.name}!=null">  
 and ${field.name} = <#noparse>#{</#noparse>${field.fieldName}}  
 </if>  
 </#list>  
 </where>  
 </delete>  
   
</mapper>

### 3.6Service包

该包主要用于为主程序提供每个模板对应的数据模型集合和相关路径名

#### 3.6.1vo模板的数据模型相关voTempData.java

package Service;  
  
import util.dbMeteData;  
import vo.fieldList;  
  
import java.io.File;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.util.\*;  
  
public class voTempData {  
 private static String *voPackageName*;  
 private static String *saveRootPath*;  
  
 static {  
 Properties pro = new Properties();  
 InputStream in = voTempData.class.getClassLoader().getResourceAsStream("config.properties");  
 try {  
 pro.load(new InputStreamReader(in, "utf-8"));  
 in.close();  
 *voPackageName* = pro.getProperty("voPackageName");  
 *saveRootPath*=pro.getProperty("saveRootPath");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 //vo模板对应的数据模型的集合，包含每张表的数据模型  
 public static *List*<*Map*<String,Object>> getVoData(){  
 *List*<*Map*<String,Object>> voList=new ArrayList<>();  
 String []strings=*voPackageName*.split("\\.");//获取vo 类的路径和包名  
 int length=strings.length;  
 String packageName=strings[length-1];//获取vo的包名  
 *List*<String> tableNames=new ArrayList<>();  
 tableNames= dbMeteData.*getTableName*();//获得数据库中的所有表名  
 for (String tableName:tableNames) {  
 *Map*<String,Object> VoData=new HashMap<>();  
 String className = tableName.substring(0, 1).toUpperCase() + tableName.substring(1);  
 *List*<fieldList> fieldList=new ArrayList<>();  
 *List*<String> columnsName=dbMeteData.*getColumnsName*(tableName);//获取对应表中的所有字段名  
 *List*<String> columnsType=dbMeteData.*getColumnsType*(tableName);//获取对应表中的所有字段的数据库类型  
 for (int i = 0; i <columnsName.size() ; i++) {  
 fieldList field=new fieldList();  
 String name=columnsName.get(i);  
 field.setName(name);  
 String fieldName = name.substring(0, 1).toLowerCase() + name.substring(1);  
 field.setFieldName(fieldName);  
 String columnType=columnsType.get(i);  
 String type = dbMeteData.*javaType*(columnType);  
 field.setType(type);  
 fieldList.add(field);  
 }  
 VoData.put("packageName",packageName);  
 VoData.put("className",className);  
 VoData.put("fieldList",fieldList);  
 voList.add(VoData);  
 }  
 return voList;  
 }  
  
 //返回vo文件生成的路径  
 public static String getVoPath(){  
 String voPath=*saveRootPath*;  
 String []strings=*voPackageName*.split("\\.");//获取vo类的路径和包名  
 int length=strings.length;  
 for (int i = 0; i <length; i++) {  
 voPath=voPath+"/"+strings[i];  
 }  
 File file=new File(voPath);  
 if(!file.exists()){  
 file.mkdirs();  
 }  
 return voPath;  
 }  
  
 //返回vo文件所有表的对应vo实体类名集合  
 public static *List*<String> getVoName(){  
 *List*<String> voNameList=new ArrayList<>();  
 *List*<String> tableNames=new ArrayList<>();  
 tableNames= dbMeteData.*getTableName*();  
 for (String tableName:tableNames){  
 String className = tableName.substring(0, 1).toUpperCase() + tableName.substring(1);  
 voNameList.add(className);  
 }  
 return voNameList;  
 }  
  
 //返回vo 类的包名  
 public static String getVoPackageName(){  
 String []strings=*voPackageName*.split("\\.");//获取vo 类的路径和包名  
 int length=strings.length;  
 String packageName=strings[length-1];  
 return packageName;  
 }  
}

#### 3.6.2dao模板的数据模型相关daoTempData.java

package Service;  
  
import util.dbMeteData;  
import vo.fieldList;  
  
import java.io.File;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.util.\*;  
  
public class daoTempData {  
 private static String *daoPackageName*;  
 private static String *saveRootPath*;  
  
 static {  
 Properties pro = new Properties();  
 InputStream in = voTempData.class.getClassLoader().getResourceAsStream("config.properties");  
 try {  
 pro.load(new InputStreamReader(in, "utf-8"));  
 in.close();  
 *daoPackageName* = pro.getProperty("daoPackageName");  
 *saveRootPath*=pro.getProperty("saveRootPath");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 //dao模板对应的数据模型的集合，包含每张表的数据模型  
 public static *List*<*Map*<String,Object>> getDaoData(){  
 *List*<*Map*<String,Object>> daoList=new ArrayList<>();  
 String []strings=*daoPackageName*.split("\\.");//获取dao类的路径和包名  
 int length=strings.length;  
 String daoPackageName=strings[length-1];  
 *List*<String> tableNames=new ArrayList<>();//获取表名  
 tableNames= dbMeteData.*getTableName*();  
 *List*<String> voNames=voTempData.*getVoName*();//获取vo名  
 for (int i = 0; i <tableNames.size() ; i++) {  
 fieldList keyField=new fieldList();  
 String voName=voNames.get(i);  
 String tableName=tableNames.get(i);  
 String voPackageName=voTempData.*getVoPackageName*();  
 keyField=dbMeteData.*getKeyField*(tableName);  
 *Map*<String,Object> daoData=new HashMap<>();  
 daoData.put("keyField",keyField);  
 daoData.put("voName",voName);  
 daoData.put("voPackageName",voPackageName);  
 daoData.put("daoPackageName",daoPackageName);  
 daoList.add(daoData);  
 }  
 return daoList;  
 }  
  
 //返回dao文件生成的路径  
 public static String getDaoPath(){  
 String daoPath=*saveRootPath*;  
 String []strings=*daoPackageName*.split("\\.");//获取dao类的路径和包名  
 int length=strings.length;  
 for (int i = 0; i <length; i++) {  
 daoPath=daoPath+"/"+strings[i];  
 }  
 File file=new File(daoPath);  
 if(!file.exists()){  
 file.mkdirs();  
 }  
 return daoPath;  
 }  
  
 //返回dao 类的包名  
 public static String getDaoPackageName(){  
 String []strings=*daoPackageName*.split("\\.");//获取dao类的路径和包名  
 int length=strings.length;  
 String daoPackageName=strings[length-1];  
 return daoPackageName;  
 }  
}

#### 3.6.3mapper模板的数据模型相关mapperTempData.java

package Service;  
  
import util.dbMeteData;  
import vo.fieldList;  
  
import java.io.File;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.util.\*;  
  
public class mapperTempData {  
 private static String *mapperPackageName*;  
 private static String *saveRootPath*;  
  
 static {  
 Properties pro = new Properties();  
 InputStream in = voTempData.class.getClassLoader().getResourceAsStream("config.properties");  
 try {  
 pro.load(new InputStreamReader(in, "utf-8"));  
 in.close();  
 *mapperPackageName* = pro.getProperty("mapperPackageName");  
 *saveRootPath*=pro.getProperty("saveRootPath");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 //mapper模板对应的数据模型的集合，包含每张表的数据模型  
 public static *List*<*Map*<String,Object>> getMapperData(){  
 *List*<*Map*<String,Object>> mapperList=new ArrayList<>();  
 *List*<String> tableNames=new ArrayList<>();//获取表名  
 tableNames= dbMeteData.*getTableName*();  
 *List*<String> voNames=voTempData.*getVoName*();//获取vo名  
 for (int i = 0; i <tableNames.size() ; i++){  
 *List*<fieldList> fieldList=new ArrayList<>();  
 fieldList keyField=new fieldList();  
 String voName=voNames.get(i);  
 String tableName=tableNames.get(i);  
 String voPackageName=voTempData.*getVoPackageName*();//  
 String daoPackageName=daoTempData.*getDaoPackageName*();  
 keyField=dbMeteData.*getKeyField*(tableName);  
 *List*<String> columnsName=dbMeteData.*getColumnsName*(tableName);  
 *List*<String> columnsType=dbMeteData.*getColumnsType*(tableName);  
 for (int j = 0; j <columnsName.size() ; j++) {  
 fieldList field=new fieldList();  
 String name=columnsName.get(j);  
 field.setName(name);  
 String fieldName = name.substring(0, 1).toLowerCase() + name.substring(1);  
 field.setFieldName(fieldName);  
 String columnType=columnsType.get(j);  
 String type = dbMeteData.*javaType*(columnType);  
 field.setType(type);  
 fieldList.add(field);  
 }  
 *Map*<String,Object> mapperData=new HashMap<>();  
 mapperData.put("daoPackageName",daoPackageName);  
 mapperData.put("voName",voName);  
 mapperData.put("voPackageName",voPackageName);  
 mapperData.put("keyField",keyField);  
 mapperData.put("fieldList",fieldList);  
 mapperList.add(mapperData);  
 }  
 return mapperList;  
 }  
  
 //获取mapper映射文件生成的路径  
 public static String getMapperPath(){  
 String mapperPath=*saveRootPath*;  
 String []strings=*mapperPackageName*.split("\\.");//获取mapper类的路径和包名  
 int length=strings.length;  
 for (int i = 0; i <length; i++) {  
 mapperPath=mapperPath+"/"+strings[i];  
 }  
 File file=new File(mapperPath);  
 if(!file.exists()){  
 file.mkdirs();  
 }  
 return mapperPath;  
 }  
}

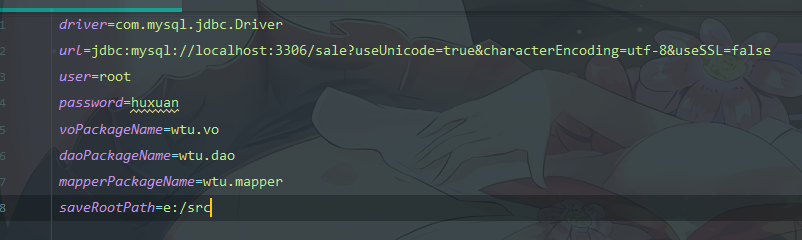
### 3.7Driver.java

主函数，控制代码生成流程

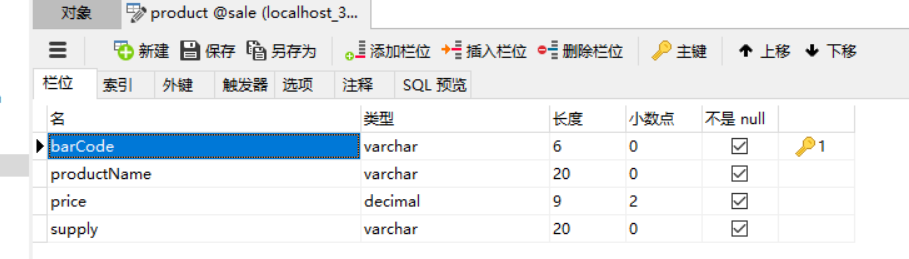
package Driver;  
  
import Service.daoTempData;  
import Service.mapperTempData;  
import Service.voTempData;  
import freemarker.template.Configuration;  
import freemarker.template.Template;  
import freemarker.template.*TemplateExceptionHandler*;  
import util.dbMeteData;  
  
import java.io.File;  
import java.io.FileWriter;  
import java.io.PrintWriter;  
import java.util.*List*;  
import java.util.*Map*;  
  
public class Driver {  
 public static void main(String[] args) throws Exception {  
 *List*<String> tableNames = dbMeteData.*getTableName*();  
  
  
 //1.创建FreeMarker配置类  
 Configuration cfg = new Configuration(Configuration.*VERSION\_2\_3\_22*);  
 cfg.setDefaultEncoding("UTF-8");  
 cfg.setTemplateExceptionHandler(*TemplateExceptionHandler*.*RETHROW\_HANDLER*);  
 //2.指定模板加载器，将模板存入缓存中  
 cfg.setDirectoryForTemplateLoading(new File("templates"));  
 //3.获取模板  
 Template temp = cfg.getTemplate("voTemplate.ftl");  
 Template temp1=cfg.getTemplate("daoTemplate.ftl");  
 Template temp2=cfg.getTemplate("mapperTemplate.ftl");  
 //4.构造数据模型---vo模板  
 *List*<*Map*<String, Object>> voList = voTempData.*getVoData*();  
 for (*Map*<String, Object> voData : voList) {  
 String voPath = voTempData.*getVoPath*();  
 String className= (String) voData.get("className");  
 //System.out.println(className);  
 //System.out.println(voList.size());  
 //5.文件输出  
 File file = new File(voPath + "/" + className + ".java");  
 FileWriter fw = new FileWriter(file);  
 PrintWriter pw = new PrintWriter(fw);  
 temp.process(voData, pw);  
 System.*out*.println("成功生成vo." + className + ".java文件");  
 }  
  
 //4.构造数据模型---dao模板  
 *List*<*Map*<String, Object>> daoList= daoTempData.*getDaoData*();  
 for (*Map*<String, Object> daoData:daoList) {  
 String daoPath=daoTempData.*getDaoPath*();  
 String className="I"+daoData.get("voName")+"DAO";  
 //5.文件输出  
 File file = new File(daoPath + "/" + className + ".java");  
 FileWriter fw = new FileWriter(file);  
 PrintWriter pw = new PrintWriter(fw);  
 temp1.process(daoData, pw);  
 System.*out*.println("成功生成dao." + className + ".java文件");  
 }  
  
 //4.构造数据模型---mapper模板  
 *List*<*Map*<String, Object>> mapperList= mapperTempData.*getMapperData*();  
 for (*Map*<String, Object> mapperData:mapperList) {  
 String mapperPath=mapperTempData.*getMapperPath*();  
 String className=mapperData.get("voName")+"Mapper";  
 //5.文件输出  
 File file = new File(mapperPath + "/" + className + ".xml");  
 FileWriter fw = new FileWriter(file);  
 PrintWriter pw = new PrintWriter(fw);  
 temp2.process(mapperData, pw);  
 //System.out.println(mapperPath);  
 System.*out*.println("成功生成mapper." + className + ".xml文件");  
 }  
 }  
}

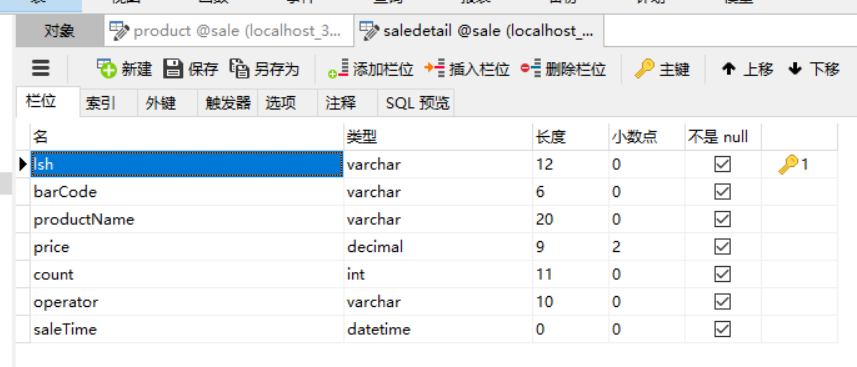
## 4 系统测试

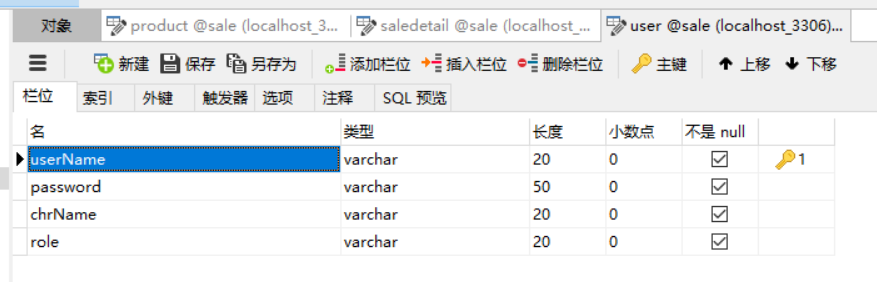
### 4.1配置相关文件：config.properties



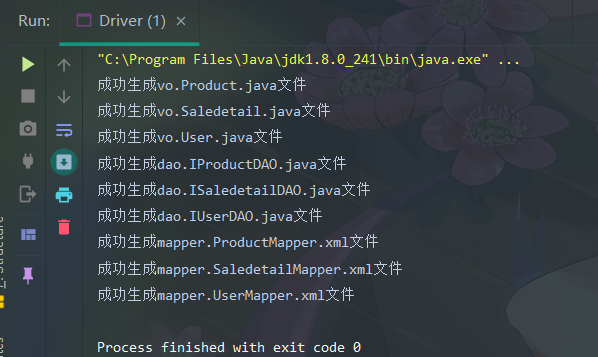
### 4.2数据库测试用表文件



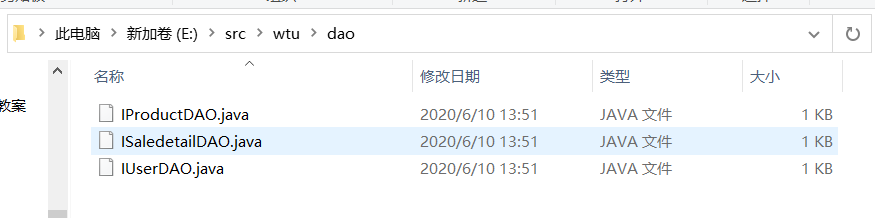


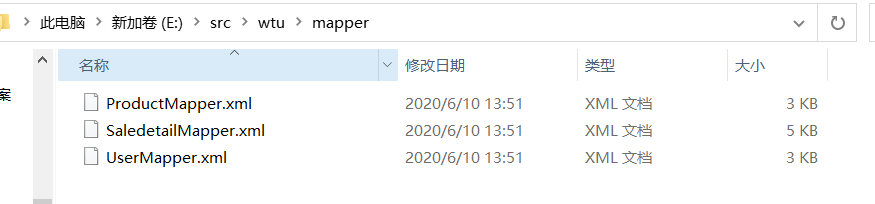


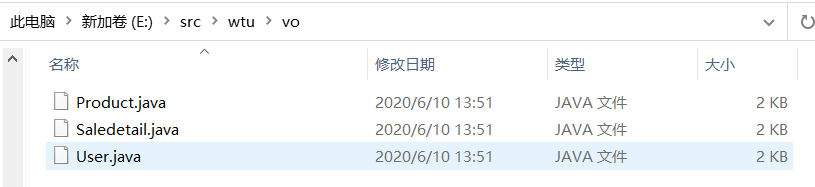
### 4.3代码运行控制台结果

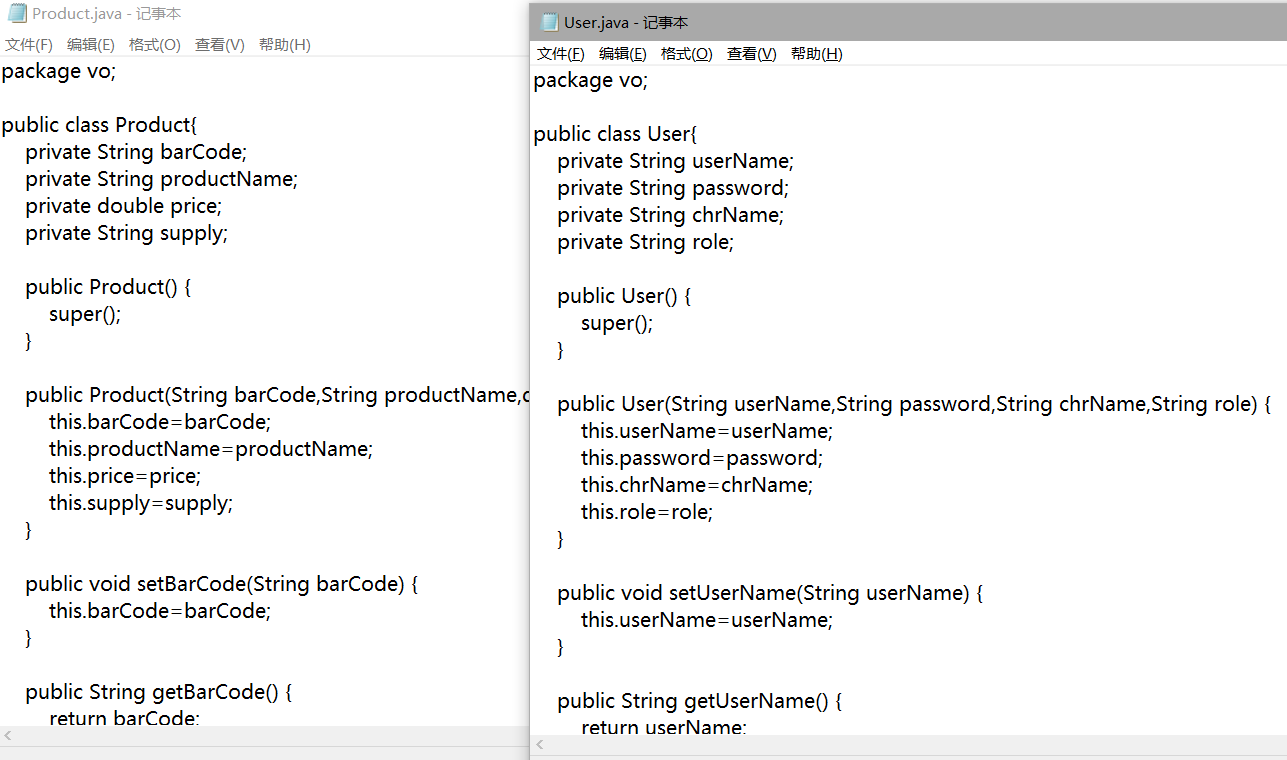


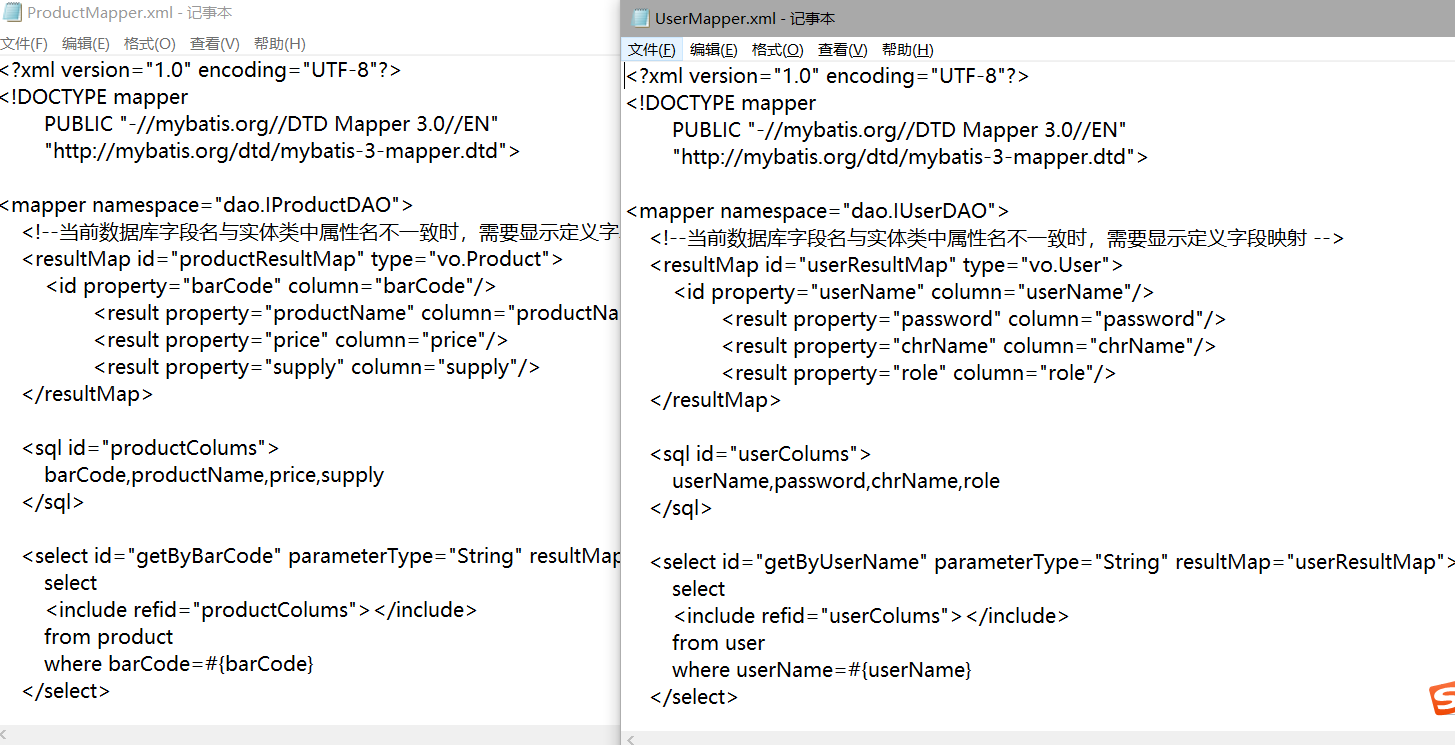
### 4.4运行后文件结果

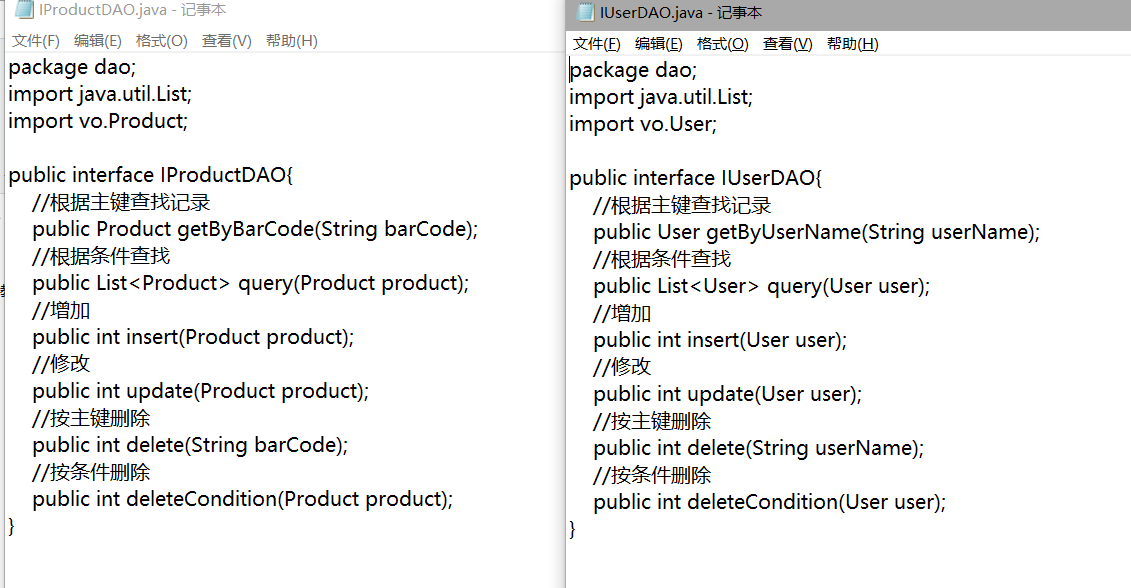




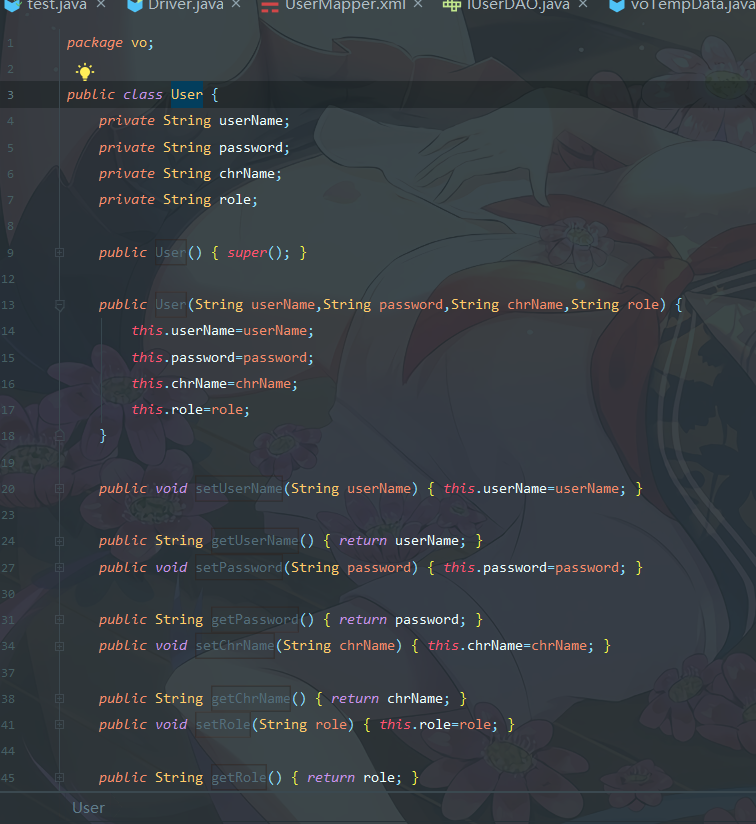




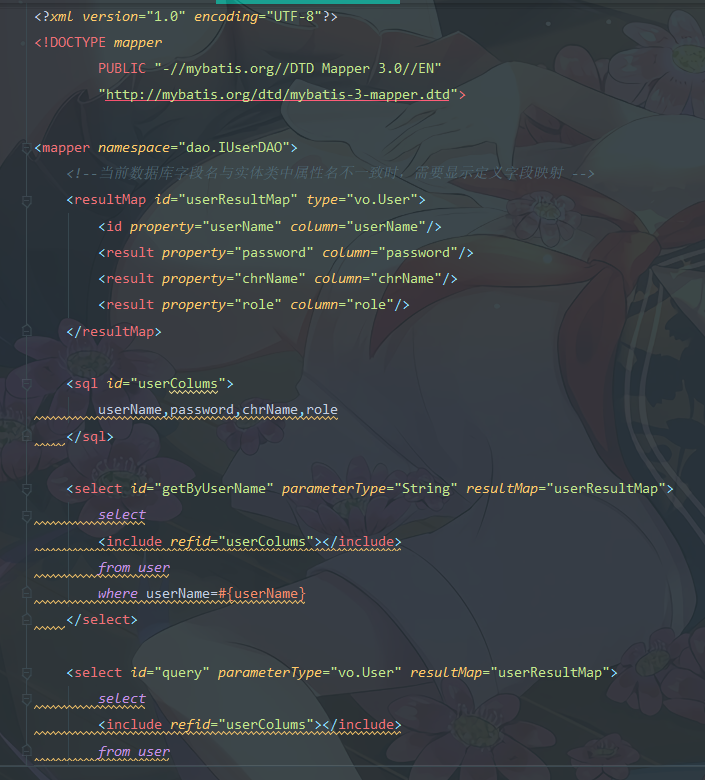




把代码放到编译器中也没问题：







## 5.系统总结

这个系统设计主要的问题时掌握对freeMarker模板引擎的应用，难点在于代码生成模板的撰写和获得数据模型的处理工作，解决是就是要很细心，一一进行对照，系统存在的不足是目前只能生成mybatis框架的代码，而且好多代码都是重复性的，可以进行适当的优化处理，

本门课程的收获是学到了一个新的知识，了解到了freeMarker及其应用，同时课内收获也巨大，基本知识全部都掌握了，也能自己编写一些简单的小项目，今后的发展考虑一直主学java吧，掌握更多java知识与框架，然后考虑考研之类的。

自评成绩：90