# JAVA编程进阶上机报告

****

**学 院 智能与计算学部**

**专 业 软件工程**

**班 级 1**

**学 号 3018216031**

**姓 名 白文杰**

1. **实验要求**

使用注解和反射来完成动态Sql编程

**1. 提供用户表：user**

表中包含字段：

id，用户名，性别，邮箱，电话等信息。

**2. 要求通过注解和反射的方式封装一个小型的sql操作类，可以通过对应的方法生成增、删、改、查等操作的SQL语句。**

**3. 要求实现注解**：

@Column：用来标注每个field对应的表中的字段是什么  
@Table：用来标记表的名字

1. **源代码**

详细源码可见reflection.zip文件

反射和注解详细内容参照注释

1. 自定义注解：@Table

**package** reflection;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

@Target(ElementType.***TYPE***)

@Retention(RetentionPolicy.***RUNTIME***)

**public** **@interface** Table {

String value();

}

2. 自定义注解：@Column

**package** reflection;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

@Target(ElementType.***FIELD***)

@Retention(RetentionPolicy.***RUNTIME***)

**public** **@interface** Column {

String value();

}

3. 创建User类

**package** reflection;

@Table("user")

**public** **class** User {

@Column("id")

**private** **int** id;

@Column("username")

**private** String username;

@Column("age")

**private** **int** age;

@Column("email")

**private** String email;

@Column("telephone")

**private** String telephone;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getTelephone() {

**return** telephone;

}

**public** **void** setTelephone(String telephone) {

**this**.telephone = telephone;

}

}

4. 创建接口SqlUtil

**package** reflection;

**import** java.util.List;

**public** **interface** SqlUtil {

/\*\*

\* 根据传入的参数返回查询语句

\* **@param** user

\* **@return** 返回查询语句

\*/

String query(User user);

/\*\*

\* 根据传入的参数返回插入语句

\* **@param** user

\* **@return** 返回插入语句

\*/

String insert(User user);

/\*\*

\* 根据传入的参数返回插入语句

\* **@param** users

\* **@return** 返回插入语句

\*/

String insert(List<User> users);

/\*\*

\* 根据传入的参数返回删除语句（删除id为user.id的记录）

\* **@param** user

\* **@return** 返回删除语句

\*/

String delete(User user);

/\*\*

\* 根据传入的参数返回更新语句（将id为user.id的记录的其它字段更新成user中的对应值）

\* **@param** user

\* **@return** 返回更新语句

\*/

String update(User user);

}

5. 新建一个类Sql，实现接口SqlUtil

**package** reflection;

**import** java.lang.reflect.Field;

**import** java.lang.reflect.Method;

**import** java.util.List;

**public** **class** Sql **implements** SqlUtil {

@Override

**public** String query(User user) {

**return** null;

}

@Override

**public** String insert(User user) {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** String insert(List<User> users) {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** String delete(User user) {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** String update(User user) {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

由于内容过多，每个函数的代码写在下面：

5.1 query(User user)

@Override

**public** String query(User user) {

StringBuilder sb = **new** StringBuilder();

// 1. 获取Class

Class<? **extends** User> clazz = user.getClass();

// 判断是否包含Table类型的注解

**if** (!clazz.isAnnotationPresent(Table.**class**)) {

**return** **null**;

}

// 2. 获取到Table的名字

Table t = (Table)clazz.getAnnotation(Table.**class**);

String tableName = t.value();

sb.append("SELSCT \* FROM ").append(tableName);

// 获取类属性的所有字段

Field[] fArray = clazz.getDeclaredFields();

// 3. 遍历所有字段

**for** (Field field : fArray) {

// 4. 处理每个字段对应的sql

// 判断是否包含Column类型的注解

**if** (!field.isAnnotationPresent(Column.**class**)) {

**continue**;

}

// 4.1 拿到Column注解上面的值

Column column = field.getAnnotation(Column.**class**);

String columnName = column.value();

// 4.2 拿到字段的名字

String fieldName = field.getName();

// 获取相应字段的get方法

String getMethodName = "get" + fieldName.substring(0, 1).toUpperCase()+ fieldName.substring(1);

// 字段的值，由于值的类型很多，所以设为Object

Object fieldValue = **null**;

**try** {

Method getMethod = clazz.getMethod(getMethodName);

fieldValue = getMethod.invoke(user);

} **catch** (Exception e) {

e.printStackTrace();

}

// 4.3 拼接sql

**if** (fieldValue == **null**||fieldValue **instanceof** Integer&&(Integer)fieldValue == 0) {

**continue**;

}

sb.append(" AND ").append(fieldName);

**if**(fieldValue **instanceof** String){

**if**(((String)fieldValue).contains(",")){

String[] values = ((String)fieldValue).split(",");

sb.append(" IN(");

**for** (String v : values) {

sb.append("'").append(v).append("'").append(",");

}

sb.deleteCharAt(sb.length()-1);

sb.append(")");

}

**else**{

sb.append(" = ").append("'").append(fieldValue).append("'");

}

}

**else** **if**(fieldValue **instanceof** Integer){

sb.append(" = ").append(fieldValue);

}

}

**return** sb.toString();

}

5.2 insert(User user)

@Override

**public** String insert(User user) {

StringBuilder sb = **new** StringBuilder();

// 1. 获取class

Class clazz = user.getClass();

// 判断是否包含Table类型的注解

**if** (!clazz.isAnnotationPresent(Table.**class**)) {

**return** **null**;

}

// 2. 获取Table的名字

Table table = (Table)clazz.getAnnotation(Table.**class**);

String tableName = table.value();

sb.append("INSERT INTO ").append(tableName);

// 3. 获取类属性的所有字段

Field[] fArray = clazz.getDeclaredFields();

// 设置一个属性列表

StringBuilder attribute = **new** StringBuilder(" (");

// 设置一个属性值列表

StringBuilder attributeValue = **new** StringBuilder(" VALUES (");

// 遍历字段

**for** (Field field:fArray) {

// 4. 处理每个字段

// 判断是否包含Column类型的注解

**if** (!field.isAnnotationPresent(Column.**class**)) {

System.***out***.println("无效插入信息");

**continue**;

}

// 4.1 拿到Column注解上面的值

Column column = field.getAnnotation(Column.**class**);

String columnName = column.value();

// 4.2 拿到字段的名字

String fieldName = field.getName();

// 获取相应字段的get方法

String getMethodName = "get" + fieldName.substring(0, 1).toUpperCase()+ fieldName.substring(1);

// 字段的值，由于值的类型很多，所以设为Object

Object fieldValue = **null**;

**try** {

Method getMethod = clazz.getMethod(getMethodName);

fieldValue = getMethod.invoke(user);

} **catch** (Exception e) {

e.printStackTrace();

}

// 4.3 构造插入列表

**if** (fieldValue == **null**||fieldValue **instanceof** Integer&&(Integer)fieldValue == 0) {

**continue**;

}

attribute.append("`").append(fieldName).append("`, ");

attributeValue.append("`").append(fieldValue).append("`, ");

}

attribute.deleteCharAt(attribute.length()-1).deleteCharAt(attribute.length()-1).append(")");

attributeValue.deleteCharAt(attributeValue.length()-1).deleteCharAt(attributeValue.length()-1).append(")");

// 4.4 拼接

sb.append(attribute).append(attributeValue);

**return** sb.toString();

}

5.3 insert(List<User> users)

@Override

**public** String insert(List<User> users) {

**boolean** isTable = **true**;

**for** (User user:users) {

Class clazz = user.getClass();

isTable = isTable && clazz.isAnnotationPresent(Table.**class**);

}

**if**(!isTable) {

**return** **null**;

}

StringBuilder sb = **new** StringBuilder("INSERT INTO ");

// 获取table

Table table = users.get(0).getClass().getAnnotation(Table.**class**);

String tableName = table.value();

sb.append(tableName);

// 设置一个属性列表

StringBuilder attribute = **new** StringBuilder(" (");

// 设置一个属性值列表数组，大小为users的长度

StringBuilder[] attributeValues = **new** StringBuilder[users.size()];

// 3. 循环获取类属性的所有字段

**for** (**int** i = 0; i < attributeValues.length; i++) {

User user = users.get(i);

Class clazz = user.getClass();

Field[] fArray = clazz.getDeclaredFields();

attributeValues[i] = **new** StringBuilder("(");

// 遍历字段

**for** (Field field:fArray) {

// 4. 处理每个字段

// 判断是否包含Column类型的注解

**if** (!field.isAnnotationPresent(Column.**class**)) {

System.***out***.println("无效插入信息");

**continue**;

}

// 4.1 拿到Column注解上面的值

Column column = field.getAnnotation(Column.**class**);

String columnName = column.value();

// 4.2 拿到字段的名字

String fieldName = field.getName();

// 获取相应字段的get方法

String getMethodName = "get" + fieldName.substring(0, 1).toUpperCase()+ fieldName.substring(1);

// 字段的值，由于值的类型很多，所以设为Object

Object fieldValue = **null**;

**try** {

Method getMethod = clazz.getMethod(getMethodName);

fieldValue = getMethod.invoke(user);

} **catch** (Exception e) {

e.printStackTrace();

}

// 4.3 构造插入列表

**if** (fieldValue == **null**||fieldValue **instanceof** Integer&&(Integer)fieldValue == 0) {

**continue**;

}

**if**(i == 0) {

attribute.append("`").append(fieldName).append("`, ");

}

attributeValues[i].append("`").append(fieldValue).append("`, ");

}

}

// 4.4 拼接

attribute.deleteCharAt(attribute.length()-1).deleteCharAt(attribute.length()-1).append(") VALUES ");

sb.append(attribute);

**for**(StringBuilder attributeValue:attributeValues) {

attributeValue.deleteCharAt(attributeValue.length()-1).deleteCharAt(attributeValue.length()-1).append(")");

sb.append(attributeValue).append(", ");

}

sb.deleteCharAt(sb.length()-1).deleteCharAt(sb.length()-1);

**return** sb.toString();

}

5.4 update (Uesr user)

@Override

**public** String update(User user) {

StringBuilder sb = **new** StringBuilder();

// 1. 获取Class

Class<? **extends** User> clazz = user.getClass();

// 判断是否包含Table类型的注解

**if** (!clazz.isAnnotationPresent(Table.**class**)) {

**return** **null**;

}

// 2. 获取到Table的名字

Table t = (Table)clazz.getAnnotation(Table.**class**);

String tableName = t.value();

sb.append("UPDATE ").append(tableName);

// 获取类属性的所有字段

Field[] fArray = clazz.getDeclaredFields();

String id = **null**;

// 3. 遍历所有字段

**for** (Field field : fArray) {

// 4. 处理每个字段对应的sql

// 判断是否包含Column类型的注解

**if** (!field.isAnnotationPresent(Column.**class**)) {

**continue**;

}

// 4.1 拿到Column注解上面的值

Column column = field.getAnnotation(Column.**class**);

String columnName = column.value();

// System.out.println(columnName);

// 4.2 拿到字段的名字

String fieldName = field.getName();

// System.out.println(fieldName);

// 获取相应字段的get方法

String getMethodName = "get" + fieldName.substring(0, 1).toUpperCase()+ fieldName.substring(1);

// 字段的值，由于值的类型很多，所以设为Object

Object fieldValue = **null**;

**try** {

Method getMethod = clazz.getMethod(getMethodName);

fieldValue = getMethod.invoke(user);

} **catch** (Exception e) {

e.printStackTrace();

}

// 4.3 拼接sql

**if** (fieldValue == **null**||fieldValue **instanceof** Integer&&(Integer)fieldValue == 0) {

**continue**;

}

**if**(!fieldName.toLowerCase().equals("id")) {

sb.append(" SET ").append(fieldName);

**if**(fieldValue **instanceof** String){

// if(((String)fieldValue).contains(",")){

// String[] values = ((String)fieldValue).split(",");

// sb.append(" IN(");

// for (String v : values) {

// sb.append("'").append(v).append("'").append(",");

// }

// sb.deleteCharAt(sb.length()-1);

// sb.append(")");

// }

// else{

sb.append(" = ").append("'").append(fieldValue).append("'");

// }

}

**else** **if**(fieldValue **instanceof** Integer){

sb.append(" = ").append(fieldValue);

}

}**else** {

id = **new** String(fieldValue.toString());

}

}

sb.append(" WHERE id = ").append(id);

**return** sb.toString();

}

5.5 delete (User user)

@Override

**public** String delete(User user) {

StringBuilder sb = **new** StringBuilder();

// 1. 获取Class

Class<? **extends** User> clazz = user.getClass();

// 判断是否包含Table类型的注解

**if** (!clazz.isAnnotationPresent(Table.**class**)) {

**return** **null**;

}

// 2. 获取到Table的名字

Table t = (Table)clazz.getAnnotation(Table.**class**);

String tableName = t.value();

sb.append("DELETE FROM ").append(tableName);

// 获取类属性的所有字段

Field[] fArray = clazz.getDeclaredFields();

// 3. 遍历所有字段

**for** (Field field : fArray) {

// 4. 处理每个字段对应的sql

// 判断是否包含Column类型的注解

**if** (!field.isAnnotationPresent(Column.**class**)) {

**continue**;

}

// 4.1 拿到Column注解上面的值

Column column = field.getAnnotation(Column.**class**);

String columnName = column.value();

// System.out.println(columnName);

// 4.2 拿到字段的名字

String fieldName = field.getName();

// System.out.println(fieldName);

// 获取相应字段的get方法

String getMethodName = "get" + fieldName.substring(0, 1).toUpperCase()+ fieldName.substring(1);

// 字段的值，由于值的类型很多，所以设为Object

Object fieldValue = **null**;

**try** {

Method getMethod = clazz.getMethod(getMethodName);

fieldValue = getMethod.invoke(user);

} **catch** (Exception e) {

e.printStackTrace();

}

// 4.3 拼接sql

**if** (fieldValue == **null**||fieldValue **instanceof** Integer&&(Integer)fieldValue == 0) {

**continue**;

}

sb.append(" WHERE ").append(fieldName);

**if**(fieldValue **instanceof** String){

**if**(((String)fieldValue).contains(",")){

String[] values = ((String)fieldValue).split(",");

sb.append(" IN(");

**for** (String v : values) {

sb.append("'").append(v).append("'").append(",");

}

sb.deleteCharAt(sb.length()-1);

sb.append(")");

}

**else**{

sb.append(" = ").append("'").append(fieldValue).append("'");

}

}

**else** **if**(fieldValue **instanceof** Integer){

sb.append(" = ").append(fieldValue);

}

}

**return** sb.toString();

}

1. **实验结果**

输入测试用例：

**package** reflection;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** TestDemo {

**public** **static** **void** main(String[] args) {

// initialize util

SqlUtil util = **new** Sql();

// test query1

User user = **new** User();

user.setId(175);

System.***out***.println(util.query(user));

// print: SELECT \* FROM user WHERE id = 175

// test query2

user = **new** User();

user.setUsername("史荣贞");

System.***out***.println(util.query(user));

// print: SELECT \* FROM `user` WHERE `username` LIKE '史荣贞';

// test insert

user = **new** User();

user.setUsername("user");

user.setTelephone("12345678123");

user.setEmail("user@123.com");

user.setAge(20);

System.***out***.println(util.insert(user));

// print: INSERT INTO `user` (`username`, `telephone`, `email`, `age`) VALUES ('user', '12345678123', 'user@123.com', 20)

// test insert list

User user2 = **new** User();

user2.setUsername("user2");

user2.setTelephone("12345678121");

user2.setEmail("user2@123.com");

user2.setAge(20);

List<User> list = **new** ArrayList<>();

list.add(user);

list.add(user2);

System.***out***.println(util.insert(list));

// print: INSERT INTO `user` (`username`, `telephone`, `email`, `age`) VALUES ('user', '12345678123', 'user@123.com', 20), ('user2', '12345678121', 'user2@123.com', 20)

// test update

user = **new** User();

user.setId(1);

user.setEmail("change@123.com");

System.***out***.println(util.update(user));

// print: UPDATE `user` SET `email` = 'change@123.com' WHERE `id` = 1;

// test delete

user = **new** User();

user.setId(1);

System.***out***.println(util.delete(user));

// print: DELETE FROM `user` WHERE `id` = 1;

}

}

运行结果如下：

SELSCT \* FROM user WHERE id = 175

SELSCT \* FROM user WHERE username LIKE '史荣贞'

INSERT INTO user (`username`, `age`, `email`, `telephone`) VALUES (`user`, `20`, `user@123.com`, `12345678123`)

INSERT INTO user (`username`, `age`, `email`, `telephone`) VALUES (`user`, `20`, `user@123.com`, `12345678123`), (`user2`, `20`, `user2@123.com`, `12345678121`)

UPDATE user SET email = 'change@123.com' WHERE id = 1

DELETE FROM user WHERE id = 1

运行结果截图：

