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　6 班

学　号　3018216298

姓　名　米思成

一、实验要求

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

实现对应的接口并进行测试：**注意这里只是需要大家把 sql 语句打印出来即可，不需要进行真正的数据库交互**

二、源代码

由于实验要求已给出其他类，这里只附上通过注解和反射的方式封装一个的 sql 操作类。

1. 代码解析（注解和反射的实现细节）

本 sql 操作类每个方法都是先通过同样的方式获得字段名和字段值，然后拼接出相应 sql 字符串，故选择其中的 query()方法为例解析。

```
public String query(User user) {
        String str = null;

        1.  通过传入的user获得user类的类信息
        Class c = user.getClass();
        2. 判断是否为Table注解标记的表
        if(!c.isAnnotationPresent(Table.class))
            return null;

        3. 通过注解获取表名
        Table table = (Table) c.getAnnotation(Table.class);
        String tableName = table.value();

        4. 通过反射获取表中所有字段
        Field[] fieldArray = c.getDeclaredFields();
```

```
        //遍历表中字段
        for (Field f: fieldArray) {
            5.判断这个字段是否带有Column注解
            if(!f.isAnnotationPresent(Column.class))
                continue;
            6.通过反射获取字段名
            String fieldName = f.getName();
            7.通过反射获取字段值
            String getMethodName = "get" + fieldName.substring(0,
1).toUpperCase() + fieldName.substring(1);
            Object value = null;
            try {
                Method getMethod = c.getMethod(getMethodName);
                value = getMethod.invoke(user);
            } catch (Exception e) {
                e.printStackTrace();
            }
            //拼接Sql语句
            if(value==null || (value instanceof Integer &&
(Integer)value==0)){
                continue;
            }
            if (value instanceof Integer) {
                str = "SELECT * FROM '" + tableName + "' WHERE '" + fieldName
+ "' = " + value;
            }
            else if (value instanceof Object) {
                str = "SELECT * FROM '" + tableName + "' WHERE '" + fieldName
+ "' LIKE '" + value + "'";
            }

        }
        return str + ";";
    }
```

2.源代码

```
public class ReturnSql implements SqlUtil{
```

```
    @Override
    public String query(User user) {
        String str = null;

        //获取Class
        Class c = user.getClass();
        if(!c.isAnnotationPresent(Table.class))
            return null;

        //获取表名
        Table table = (Table) c.getAnnotation(Table.class);
        String tableName = table.value();

        //获取表中所有字段
        Field[] fieldArray = c.getDeclaredFields();

        //遍历表中字段
        for (Field f: fieldArray) {
            //判断这个字段是否带有Column注解
            if(!f.isAnnotationPresent(Column.class))
                continue;
            //获取字段名
            String fieldName = f.getName();
            //获取字段值
            String getMethodName = "get" + fieldName.substring(0,
1).toUpperCase() + fieldName.substring(1);
            Object value = null;
            try {
                Method getMethod = c.getMethod(getMethodName);
                value = getMethod.invoke(user);
            } catch (Exception e) {
                e.printStackTrace();
            }
            //拼接Sql语句
            if(value==null || (value instanceof Integer &&
```

```
(Integer)value==0)){
                continue;
            }
            if (value instanceof Integer) {
                str = "SELECT * FROM '" + tableName + "' WHERE '" + fieldName
+ "' = " + value;
            }
            else if (value instanceof Object) {
                str = "SELECT * FROM '" + tableName + "' WHERE '" + fieldName
+ "' LIKE '" + value + "'";
            }

        }
        return str + ";";
    }


    @Override
    public String insert(User user) {
        String str1 = "INSERT INTO ";
        String str2 = "VALUES (";


        //获取Class

        Class c = user.getClass();
        if(!c.isAnnotationPresent(Table.class))
            return null;


        //获取表名

        Table table = (Table) c.getAnnotation(Table.class);
        String tableName = table.value();
        str1 += "'" + tableName + "' (";


        //获取表中所有字段

        Field[] fieldArray = c.getDeclaredFields();


        //遍历表中字段

        for (int i = 0; i<fieldArray.length; i++) {
            Field f = fieldArray[i];

            //判断这个字段是否带有Column注解
```

```
            if(!f.isAnnotationPresent(Column.class))
                continue;

            //获取字段名

            String fieldName = f.getName();

            //获取字段值

            String getMethodName = "get" + fieldName.substring(0,
1).toUpperCase() + fieldName.substring(1);
            Object value = null;
            try {
                Method getMethod = c.getMethod(getMethodName);
                value = getMethod.invoke(user);
            } catch (Exception e) {
                e.printStackTrace();
            }

            //拼接Sql语句

            if(value==null || (value instanceof Integer &&
(Integer)value==0)){
                continue;
            }
            if (i==fieldArray.length-1) {
                str1 += "'" + fieldName + "') ";
                if (value instanceof Integer) {
                    str2 += value + ") ";
                }
                else if (value instanceof Object) {
                    str2 += "'" + value + "') ";
                }
            }
            else{
                str1 += "'" + fieldName + "', ";
                if (value instanceof Integer) {
                    str2 += value + ", ";
                }
                else if (value instanceof Object) {
                    str2 += "'" + value + "', ";
                }
            }

        }
        return str1 + str2 + ";";
    }
```

```
String tableName;
@Override
public String insert(List<User> users) {
    tableName = User.class.getAnnotation(Table.class).value();
    if (!User.class.isAnnotationPresent(Table.class))
        return null;

    Field[] fields = User.class.getDeclaredFields();
    for (Field field : fields)
    {
        field.setAccessible(true);
    }
    StringBuffer sql = new StringBuffer("INSERT INTO ");
    sql.append("`").append(tableName).append("`").append(" (");

    for (Field field : fields)
    {
        Column column = field.getAnnotation(Column.class);
        sql.append("`").append(column.value()).append("`, ");
    }
    sql.delete(sql.length() - 2, sql.length()).append(") ");
    sql.append(" VALUES (");
    for (User user : users)
    {
        if (!User.class.isAnnotationPresent(Table.class))
        {
            return null;
        }
        for (Field field : fields)
        {
            Object value = null;
            try
            {
                value = field.get(user);
            }
            catch (IllegalAccessException e)
            {
                e.printStackTrace();
            }
            if (value == null)
            {
                return null;
            }
```

```
            if (field.getType().equals(String.class))
            {
                sql.append("'").append(value.toString()).append("'");
            }
            else
            {
                sql.append(value.toString());
            }
            sql.append(", ");
        }
        sql.delete(sql.length() - 2, sql.length()).append("), (");
    }
    sql.delete(sql.length() - 3, sql.length()).append(";");
    return sql.toString();

}

@Override
public String delete(User user) {
    String str = null;


    //获取Class

    Class c = user.getClass();
    if(!c.isAnnotationPresent(Table.class))
        return null;


    //获取表名

    Table table = (Table) c.getAnnotation(Table.class);
    String tableName = table.value();


    //获取表中所有字段

    Field[] fieldArray = c.getDeclaredFields();


    //遍历表中字段

    for (Field f: fieldArray) {

        //判断这个字段是否带有Column注解

        if(!f.isAnnotationPresent(Column.class))
            continue;

        //获取字段名
```

```
            String fieldName = f.getName();

            //获取字段值

            String getMethodName = "get" + fieldName.substring(0,
1).toUpperCase() + fieldName.substring(1);
            Object value = null;
            try {
                Method getMethod = c.getMethod(getMethodName);
                value = getMethod.invoke(user);
            } catch (Exception e) {
                e.printStackTrace();
            }

            //拼接Sql语句

            if(value==null || (value instanceof Integer &&
(Integer)value==0)){
                continue;
            }
            if (value instanceof Integer) {
                str = "DELETE FROM '" + tableName + "' WHERE '" + fieldName
+ "' = " + value;
            }
            else if (value instanceof Object) {
                str = "DELETE FROM '" + tableName + "' WHERE '" + fieldName
+ "' = '" + value + "'";
            }
        }
        return str + ";";
    }

    @Override
    public String update(User user) {
        String str1 = "UPDATE ";
        String str2 = "WHERE ";

        //获取Class

        Class c = user.getClass();
        if(!c.isAnnotationPresent(Table.class))
            return null;


        //获取表名

        Table table = (Table) c.getAnnotation(Table.class);
        String tableName = table.value();
```

```
        str1 += "'" + tableName + "' SET ";

        //获取表中所有字段
        Field[] fieldArray = c.getDeclaredFields();

        //遍历表中字段
        for (int i = 0; i<fieldArray.length; i++) {
            Field f = fieldArray[i];
            //判断这个字段是否带有Column注解
            if(!f.isAnnotationPresent(Column.class))
                continue;
            //获取字段名
            String fieldName = f.getName();
            //获取字段值
            String getMethodName = "get" + fieldName.substring(0,
1).toUpperCase() + fieldName.substring(1);
            Object value = null;
            try {
                Method getMethod = c.getMethod(getMethodName);
                value = getMethod.invoke(user);
            } catch (Exception e) {
                e.printStackTrace();
            }
            //拼接Sql语句
            if(value==null || (value instanceof Integer &&
(Integer)value==0)){
                continue;
            }
            if(fieldName.equals("id")) {
                str2 += "'" + fieldName + "' = " + value;
            }
            else{
                str1 += "'" + fieldName + "' = ";
                if (value instanceof Integer) {
                    str1 += value + " ";
                }
                else if (value instanceof Object) {
                    str1 += "'" + value + "' ";
```

```
                }
            }

        }
        return str1 + str2 + ";";
    }

}
```

三、实验结果