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　　3018216281

姓　名　　朱明煊

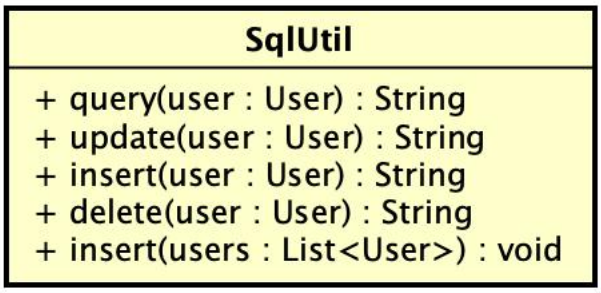
一、实验目的

使用注解和反射来完成动态 Sql 编程

二、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

三、类图



四、总体设计

1、创建 User 类作为实体集，类顶部标注 Table 注解，类中属性作为实体集属性顶部标示 Column 注解

2、对 Table 注解与 Column 注解进行解释，定义了元素种类，保留规则

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface Table {
    public String value()default "";
}
```

```java
@Target({ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Column {
     public String value()default "";
     public int length() default 255;
}
```

3、完善 SQL 类功能

query **查询功能实现**：

首先判断是否是注解

```java
Class c = user.getClass();
//判断表是否为注解
if(!c.isAnnotationPresent(Table.class)) {
    return null;
}
Table t = (Table)c.getAnnotation(Table.class);
```

建立 StringBuffer 添加字符串

```java
//得到表名称
String tName = t.value();
Builder.append("SELECT * FROM `").append(tName).append("` WHERE ");
```

利用映射得到变量，并引入计数器，当查询多个条件时向字符串里添加 and

```java
//利用映射得到类的变量
Field[] Array = c.getDeclaredFields();
int temp = 1;
```

对实体集属性进行注解判断，并用映射得到名称

```
for (Field field : Array) {
     //判断是否包含Column类型的注解
    if(!field.isAnnotationPresent(Column.class)){
        continue;
    }
    //得到类型上面注解的值
    Column column = field.getAnnotation(Column.class);
    String columnName = column.value();
    //拿到字段名
    String filedName = field.getName();
```

将字符类型变量与整形变量打印方式进行区分

```
if(temp!=1) {
 Builder.append(" and ") ;
 if(fieldValue instanceof String) {
    Builder.append("`").append(filedName).append("` ").append("LIKE").append(" `'").append((String)fieldValue).append("` ");
 }
 else {
    Builder.append(filedName).append(" = ").append(fieldValue).append(" ");
 }
}
```

update 更新功能实现：

首先判断是否是注解

```
Class c = user.getClass();
//判断表是否为注解
if(!c.isAnnotationPresent(Table.class)) {
    return null;
}
Table t = (Table)c.getAnnotation(Table.class);
```

建立 StringBuffer 添加字符串

```
//得到表名称
String tName = t.value();
Builder.append("SELECT * FROM `").append(tName).append("` WHERE ");
```

利用映射得到变量并对实体集属性进行注解判断，并用映射得到名称

```
for (Field field : Array) {
     //判断是否包含Column类型的注解
    if(!field.isAnnotationPresent(Column.class)){
        continue;
    }
    //得到类型上面注解的值
    Column column = field.getAnnotation(Column.class);
    String columnName = column.value();
    //拿到字段名
    String filedName = field.getName();
```

对更新过属性进行打印

```
//更新过的属性
  if(filedName != "id") {
     Builder.append("`").append(filedName).append("` ").append("=").append(" `").append((String)fieldValue).append("` ");
  }
```

`

打印当前用户 id

```
//更新的id
Builder.append("WHERE `id` = " ).append(user.getId());
```

delete 删除功能实现

首先判断是否是注解

```
Class c = user.getClass();
//判断表是否为注解
if(!c.isAnnotationPresent(Table.class)) {
    return null;
}
Table t = (Table)c.getAnnotation(Table.class);
```

建立 StringBuffer 添加字符串

```
//得到表名称
String tName = t.value();
Builder.append("SELECT * FROM `").append(tName).append("` WHERE ");
```

利用映射得到变量并对实体集属性进行注解判断，并用映射得到名称

```
for (Field field : Array) {
     //判断是否包含Column类型的注解
    if(!field.isAnnotationPresent(Column.class)){
        continue;
    }
    //得到类型上面注解的值
    Column column = field.getAnnotation(Column.class);
    String columnName = column.value();
    //拿到字段名
    String filedName = field.getName();
```

删除对应 id 记录

```
Builder.append("DELETE FROM `").append(tName).append("` WHERE `id` = ").append(user.getId());
return Builder.toString();
```

Insert 插入 1 个实体功能实现

首先判断是否是注解

```
Class c = user.getClass();
//判断表是否为注解
if(!c.isAnnotationPresent(Table.class)) {
    return null;
}
Table t = (Table)c.getAnnotation(Table.class);
```

建立 StringBuffer 添加字符串

```
//得到表名称
String tName = t.value();
Builder.append("SELECT * FROM `").append(tName).append("` WHERE ");
```

利用映射得到变量并对实体集属性进行注解判断，并用映射得到名称

```
for (Field field : Array) {
     //判断是否包含Column类型的注解
    if(!field.isAnnotationPresent(Column.class)){
        continue;
    }
    //得到类型上面注解的值
    Column column = field.getAnnotation(Column.class);
    String columnName = column.value();
    //拿到字段名
    String filedName = field.getName();
```

首先用一个映射对变量名进行打印，并删除多余的，

```
      //不打印id
      if(filedName != "id") {
         Builder.append("`").append(filedName).append("`,");
       }
  }

  //删除多余的,
  Builder.deleteCharAt(Builder.length()-1);
  Builder.append(")VLAUES(");
```

再对实体集属性的值进行打印，并区分字符类型与常量

```
    }
    if(filedName != "id") {
       if(fieldValue instanceof String) {
           Builder.append("`").append(fieldValue).append("`,");
       }
       else {
           Builder.append(fieldValue).append(",");
       }
     }
}
```

Insert 插入实体列表功能实现

首先判断是否是注解

```
Class c = user.getClass();
//判断表是否为注解
if(!c.isAnnotationPresent(Table.class)) {
    return null;
}
Table t = (Table)c.getAnnotation(Table.class);
```

建立 StringBuffer 添加字符串

```
//得到表名称
String tName = t.value();
Builder.append("SELECT * FROM `").append(tName).append("` WHERE ");
```

利用映射得到变量并对实体集属性进行注解判断，并用映射得到名称

```
for (Field field : Array) {
     //判断是否包含Column类型的注解
    if(!field.isAnnotationPresent(Column.class)){
        continue;
    }
    //得到类型上面注解的值
    Column column = field.getAnnotation(Column.class);
    String columnName = column.value();
    //拿到字段名
    String filedName = field.getName();
```

首先用一个映射对变量名进行打印，并删除多余的，

```
    //不打印id
    if(filedName != "id") {
       Builder.append("`").append(filedName).append("`,");
     }
}

//删除多余的，
Builder.deleteCharAt(Builder.length()-1);
Builder.append(")VLAUES(");
```

对列表每一个 User 进行遍历

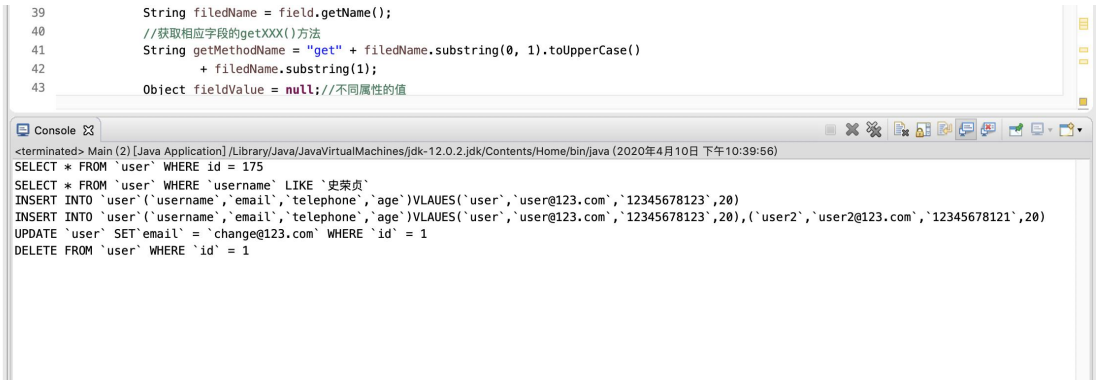
```
Builder.append(")VLAUES(");
//遍历列表
for(int i=0;i<users.size();i++) {
    Field[] Array2 = c.getDeclaredFields();
    for (Field field : Array2) {
        //判断是否包含Column类型的注解
        if(!field.isAnnotationPresent(Column.class)){
            continue;
        }
```

再对实体集属性的值进行打印，并区分字符类型与常量

```
    }
    if(filedName != "id") {
        if(fieldValue instanceof String) {
            Builder.append("`").append(fieldValue).append("`,");
        }
        else {
            Builder.append(fieldValue).append(",");
        }
    }
}
```

五、实验结果

成功实现了题目用例的要求



六、代码

```
package Lab3;
import java.lang.annotation.Annotation;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.util.Objects;
```

```
import Lab3.User;
import java.util.List;
public class 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) {
    StringBuilder Builder = new StringBuilder();
    Class c = user.getClass();
    //判断表是否为注解
    if(!c.isAnnotationPresent(Table.class)) {
    return null;
    }
    Table t = (Table)c.getAnnotation(Table.class);

    //得到表名称
    String tName = t.value();
    Builder.append("SELECT * FROM `").append(tName).append("` WHERE ");

    //利用映射得到类的变量
    Field[] Array = c.getDeclaredFields();
    int temp = 1;
    for (Field field : Array) {
     //判断是否包含 Column 类型的注解
            if(!field.isAnnotationPresent(Column.class)){
                continue;
            }
            //得到类型上面注解的值
            Column column = field.getAnnotation(Column.class);
            String columnName = column.value();
            //拿到字段名
            String filedName = field.getName();
            //获取相应字段的 getXXX()方法
            String getMethodName = "get" + filedName.substring(0, 1).toUpperCase()
                    + filedName.substring(1);
        Object fieldValue = null;//不同属性的值
            try {
                Method getMethod = c.getMethod(getMethodName);
                fieldValue = getMethod.invoke(user);
            } catch (Exception e) {
               e.printStackTrace();
            }
```

```
                if(fieldValue==null || (fieldValue instanceof Integer && (Integer)fieldValue==0)){
                    continue;
                }
                if(temp!=1) {
            Builder.append(" and ") ;
                if(fieldValue instanceof String) {
                Builder.append("`").append(filedName).append("` ").append("LIKE").append("
`").append((String)fieldValue).append("` ");
                }
                else {
                Builder.append(filedName).append(" = ").append(fieldValue).append(" ");
                }
              }
                else {

                    if(fieldValue instanceof String) {
                    Builder.append("`").append(filedName).append("` ").append("LIKE").append("
`").append((String)fieldValue).append("` ");
                    }
                    else {
                    Builder.append(filedName).append(" = ").append(fieldValue).append(" ");
                    }
                }
                temp++;
            }
        return Builder.toString();
        }
//
        /**
         *  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) {
        StringBuilder Builder = new StringBuilder();
        Class c = user.getClass();
        //判断表是否为注解
        if(!c.isAnnotationPresent(Table.class)) {
        return null;
        }
        Table t = (Table)c.getAnnotation(Table.class);

        //得到表名称
        String tName = t.value();
```

```
Builder.append("INSERT INTO `").append(tName).append("`(");
//利用映射得到类的变量
Field[] Array = c.getDeclaredFields();

//打印表的属性字段
for (Field field : Array) {
 //判断是否包含 Column 类型的注解
        if(!field.isAnnotationPresent(Column.class)){
            continue;
        }
        //得到类型上面注解的值
        Column column = field.getAnnotation(Column.class);
        String columnName = column.value();
        //拿到字段名
        String filedName = field.getName();
        //获取相应字段的 getXXX()方法
        String getMethodName = "get" + filedName.substring(0, 1).toUpperCase()
                + filedName.substring(1);
    Object fieldValue = null;//不同属性的值
        try {
            Method getMethod = c.getMethod(getMethodName);
            fieldValue = getMethod.invoke(user);
        } catch (Exception e) {
           e.printStackTrace();
        }
        if(fieldValue==null || (fieldValue instanceof Integer && (Integer)fieldValue==0)){
           continue;
        }
        //不打印 id
        if(filedName != "id") {
        Builder.append("`").append(filedName).append("`,");
         }
    }

//删除多余的,
Builder.deleteCharAt(Builder.length()-1);
Builder.append(")VLAUES(");

//打印表的属性对应的值
Field[] Array2 = c.getDeclaredFields();
for (Field field : Array2) {
 //判断是否包含 Column 类型的注解
        if(!field.isAnnotationPresent(Column.class)){
            continue;
```

```
            }
            //得到类型上面注解的值
            Column column = field.getAnnotation(Column.class);
            String columnName = column.value();
            //拿到字段名
            String filedName = field.getName();
            //获取相应字段的 getXXX()方法
            String getMethodName = "get" + filedName.substring(0, 1).toUpperCase()
                    + filedName.substring(1);
        Object fieldValue = null;//不同属性的值
            try {
                Method getMethod = c.getMethod(getMethodName);
                fieldValue = getMethod.invoke(user);
            } catch (Exception e) {
               e.printStackTrace();
            }
            if(fieldValue==null || (fieldValue instanceof Integer && (Integer)fieldValue==0)){
               continue;
            }
            if(filedName != "id") {
               if(fieldValue instanceof String) {
                 Builder.append("`").append(fieldValue).append("`,");
               }
               else {
               Builder.append(fieldValue).append(",");
               }
             }
        }
    Builder.deleteCharAt(Builder.length()-1);
    Builder.append(")");
    return Builder.toString();
    }
//
    /**
     *  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) {
    StringBuilder Builder = new StringBuilder();
    Class c = users.get(0).getClass();
    //判断表是否为注解
    if(!c.isAnnotationPresent(Table.class)) {
    return null;
```

```
}
Table t = (Table)c.getAnnotation(Table.class);

//得到表名称
String tName = t.value();
Builder.append("INSERT INTO `").append(tName).append("`(");
//利用映射得到类的变量
Field[] Array = c.getDeclaredFields();

//打印表的属性字段
for (Field field : Array) {
 //判断是否包含 Column 类型的注解
        if(!field.isAnnotationPresent(Column.class)){
            continue;
        }
        //得到类型上面注解的值
        Column column = field.getAnnotation(Column.class);
        String columnName = column.value();
        //拿到字段名
        String filedName = field.getName();
        //获取相应字段的 getXXX()方法
        String getMethodName = "get" + filedName.substring(0, 1).toUpperCase()
                + filedName.substring(1);
    Object fieldValue = null;//不同属性的值
        try {
            Method getMethod = c.getMethod(getMethodName);
            fieldValue = getMethod.invoke(users.get(0));
        } catch (Exception e) {
           e.printStackTrace();
        }
        if(fieldValue==null || (fieldValue instanceof Integer && (Integer)fieldValue==0)){
           continue;
        }
        //不打印 id
        if(filedName != "id") {
        Builder.append("`").append(filedName).append("`,");
         }
    }

//删除多余的,
Builder.deleteCharAt(Builder.length()-1);
Builder.append(")VLAUES(");
//遍历列表
for(int i=0;i<users.size();i++) {
```

```
Field[] Array2 = c.getDeclaredFields();
    for (Field field : Array2) {
     //判断是否包含 Column 类型的注解
            if(!field.isAnnotationPresent(Column.class)){
                continue;
            }
            //得到类型上面注解的值
            Column column = field.getAnnotation(Column.class);
            String columnName = column.value();
            //拿到字段名
            String filedName = field.getName();
            //获取相应字段的 getXXX()方法
            String getMethodName = "get" + filedName.substring(0, 1).toUpperCase()
                    + filedName.substring(1);
        Object fieldValue = null;//不同属性的值
            try {
                Method getMethod = c.getMethod(getMethodName);
                fieldValue = getMethod.invoke(users.get(i));
            } catch (Exception e) {
               e.printStackTrace();
            }
            if(fieldValue==null || (fieldValue instanceof Integer && (Integer)fieldValue==0)){
               continue;
            }
            if(filedName != "id") {
               if(fieldValue instanceof String) {
                 Builder.append("`").append(fieldValue).append("`,");
               }
               else {
               Builder.append(fieldValue).append(",");
               }
             }
        }
    Builder.deleteCharAt(Builder.length()-1);
    Builder.append("),(");
}
Builder.deleteCharAt(Builder.length()-1);
Builder.deleteCharAt(Builder.length()-1);
return Builder.toString();

}

/**
 * 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) {
        StringBuilder Builder = new StringBuilder();
    Class c = user.getClass();
    //判断表是否为注解
    if(!c.isAnnotationPresent(Table.class)) {
    return null;
    }
    Table t = (Table)c.getAnnotation(Table.class);
    //得到表名称
    String tName = t.value();
    Builder.append("DELETE FROM `").append(tName).append("` WHERE `id` =
").append(user.getId());
    return Builder.toString();
    };
    /**
     *  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) {
    StringBuilder Builder = new StringBuilder();
    Class c = user.getClass();
    //判断表是否为注解
    if(!c.isAnnotationPresent(Table.class)) {
    return null;
    }
    Table t = (Table)c.getAnnotation(Table.class);

    //得到表名称
    String tName = t.value();
    Builder.append("UPDATE `").append(tName).append("` SET");
    //利用映射得到类的变量
    Field[] Array = c.getDeclaredFields();
    for (Field field : Array) {
     //判断是否包含 Column 类型的注解
            if(!field.isAnnotationPresent(Column.class)){
                continue;
            }
            //得到类型上面注解的值
            Column column = field.getAnnotation(Column.class);
```

```
                String columnName = column.value();
                //拿到字段名
                String filedName = field.getName();
                //获取相应字段的 getXXX()方法
                String getMethodName = "get" + filedName.substring(0, 1).toUpperCase()
                        + filedName.substring(1);
            Object fieldValue = null;//不同属性的值
                try {
                    Method getMethod = c.getMethod(getMethodName);
                    fieldValue = getMethod.invoke(user);
                } catch (Exception e) {
                   e.printStackTrace();
                }
                if(fieldValue==null || (fieldValue instanceof Integer && (Integer)fieldValue==0)){
                   continue;
                }
              //更新过的属性
                if(filedName != "id") {
               Builder.append("`").append(filedName).append("` ").append("=").append("
`").append((String)fieldValue).append("` ");
                }

            }
        //更新的 id
        Builder.append("WHERE `id` = " ).append(user.getId());
        return Builder.toString();

    }
//
//
}
```