# JAVA编程进阶上机报告


学 院 智能与计算

专 业 软件工程

班 级 一班

学 号 3018216063

姓 名 陈宇涛

## 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

## 二、源代码

```
package lab3;

@Table("user")
//表的名字为user
public class User {
    //id表示表中的编号
    @Column(value="编号")
    private int id = -1;
    //username表示表中的用户姓名
    @Column(value="用户姓名")
    private String username = null;
```

```
    //telephone表示表中的电话号码
    @Column(value="电话号码")
    private String telephone = null;
    //email表示表中的邮件地址
    @Column(value="邮件地址")
    private String email = null;
    //age表示表中的年龄
    @Column(value="年龄")
    private int age = -1;


    public void setId(int i) {
        // 改变ID的值
        id = i;
    }

    public void setUsername(String name) {
        // 改变Username的值
        username = name;
    }

    public void setTelephone(String number) {
        // 改变Telephone的值
        telephone = number;
    }

    public void setEmail(String email) {
        // 改变Email的值
        this.email = email;
    }

    public void setAge(int age) {
        //改变Age的值
        this.age = age;
    }

}
```

column 自定义注解，用String类型标注字段含义，并用Target表示注解类型

```
package lab3;
import java.lang.annotation.Target;
import java.lang.annotation.ElementType;
@Target(ElementType.FIELD)
public @interface Column {
    String value() default " ";
    //设置类型，默认值为空
}
```

table 自定义注解，并用String类型标注表的名字

```
package lab3;

public @interface Table {

    String value() default " ";
    //设置类型，默认值为空
}
```

提供sql操作类的接口，与例子中相同

```
package lab3;
import lab3.User;
import java.util.List;
public interface util {

    /**
     * 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);
}
```

实现util接口

query函数先通过Field数组获取User全部属性，进行遍历，再通过setAccessible允许访问，用getType().getCanonicalName()判断属性类型，通过getInt()和get()获得要查看的属性，并将其输出

第一个insert函数先通过Field获取User全部属性，再用数组存储所有要插入的属性，并将其输出

第二个insert函数遍历list，在每遍遍历中与insert操作大致相同

delete函数先获取User所有属性，遍历时判断属性类型，若符合要删除条件，将其输出。所有属性用method获取对应set函数，再用invoke将int类型值置为0，String类型值置为null

update函数先获取User所有属性，遍历时判断属性类型，若符合要修改条件，则将其输出，若该属性要修改，则获取对应set函数，再将其改为对应值

```
package lab3;

import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.List;
import lab3.User;
public class SqlUtil implements util{
    /**
     * 根据传入的参数返回查询语句
     */
    public String query(User user){

        Class c = user.getClass();
        Field[] fields = c.getDeclaredFields();
        //获得user类全部属性
        for (Field field : fields) {
            field.setAccessible(true);
            //设置允许访问
            try {
                if(field.getType().getCanonicalName()==
                   "int"&&field.getInt(user)!=-1) {
                    //属性为int类型，且该属性要被查询
                    return "SELECT * FROM 'user' WHERE '" + field.getName()
                           + "' = " + field.get(user);
                }else if(field.getType().getCanonicalName()
                         =="java.lang.String"&&field.get(user)!=null){
                    //属性为String类型，且该属性要被查询
                    return "SELECT * FROM 'user' WHERE '" + field.getName()
                            + "' LIKE '" + field.get(user)+"'";
                }
            } catch (IllegalArgumentException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        return null;
    }
    /**
     * @return 返回插入语句
     */
    public String insert(User user) {
        Class c = user.getClass();
        Field[] fields = c.getDeclaredFields();
```

```
        //获得user类全部属性
        String[] value = new String[5];
        //存储被插入属性对应的值
        String[] name = {"id","username","telephone","email","age"};
        //存储属性名以便输出
        fields[1].setAccessible(true);
        //设置允许访问
        String result = new String();
        try {
            result = "INSERT INTO `user' (";
        } catch (IllegalArgumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        int cnt = 0;
        //记录已查询个数以便存储属性
        int last = -1;
        //记录最后一个被插入的属性以便及时输出')'
        for (Field field : fields) {
        //记录属性是否应被插入并记录值
            field.setAccessible(true);
            //设置允许访问
            try {
                if(field.getType().getCanonicalName()=="int"
                    &&field.getInt(user)!=-1) {
                    //属性为int类型，且该属性要被插入
                    value[cnt] = field.get(user).toString();
                    last = cnt;
                    //存储并记该属性为最后一个待插入属性

                }else if(field.getType().getCanonicalName()=="java.lang.String"
                         &&field.get(user)!=null){
                    //属性为String类型，且该属性要被插入
                    value[cnt] = "'"+field.get(user).toString()+"'";
                    last = cnt;
                    //存储并记该属性为最后一个待插入属性
                }
            } catch (IllegalArgumentException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            cnt++;

        }
        for(int i=0;i<5;i++) {
            //输出属性名
            if(value[i]!=null&&i<last) {
                //是待插入属性且不为最后一个
                result += "'"+name[i]+"', ";
            }else if(value[i] != null && i == last) {
                //是待插入属性且为最后一个
                result += "'"+name[i]+"') VALUES (";
            }
        }
```

```
        for(int i=0;i<5;i++) {
            //输出待插入属性
            if(value[i]!=null&&i<last) {
                //是待插入属性且不为最后一个
                result += value[i]+", ";
            }else if(value[i] != null && i == last) {
                //是待插入属性且为最后一个
                result += value[i]+")";
            }
        }
        return result;

    }

//    /**
//     * 根据传入的参数返回插入语句
//     */
    public String insert(List<User> users) {
        String result = "INSERT INTO `user' ('username', 'telephone', 'email',
'age') VALUES ";
        for(int i=0;i<users.size();i++) {
            //遍历list中所有User类
            Class c = users.get(i).getClass();
            Field[] fields = c.getDeclaredFields();
            //获得user类全部属性
            String[] value = new String[5];
            //存储被插入属性对应的值
            for (int j=0;j<fields.length;j++) {
            //记录属性是否被插入并记录值
                fields[j].setAccessible(true);

                //设置允许访问
                try {
                    if(fields[j].getType().getCanonicalName()=="int"
                       &&fields[j].getInt(users.get(1))!=-1) {
                        //属性为int类型，且该属性要被插入
                        value[j] = fields[j].get(users.get(i)).toString();

                    }else if(fields[j].getType().getCanonicalName()==

"java.lang.String"&&fields[j].get(users.get(1))!=null){
                        //属性为String类型，且该属性要被插入
                        value[j] = "'"+(String)fields[j].get(users.get(i))+"'";
                    }
                } catch (IllegalArgumentException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IllegalAccessException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

            }
            result += "(";
            for(int k=1;k<value.length;k++) {
                //输出待插入属性
                if(value[k]!=null&&k<fields.length-1) {
                    //是待插入属性且不为最后一个
```

```
                    result += value[k]+", ";
                }else if(value[k] != null && k == fields.length-1) {
                    //是待插入属性且为最后一个
                    result += value[k]+")";
                }
            }
            if(i<users.size()-1) {
                //最后一个User类
                result += ", ";
            }

        }
        return result;
    }
//
//    /**
//     * 根据传入的参数返回删除语句（删除id为user.id的记录）
//     * @return 返回删除语句
//     */
    public String delete(User user) {
        Class c = user.getClass();
        Object o = User.class;
        String[] name =
{"setId","setUsername","setTelephone","setEmail","setAge"};
        //存储属性名
        Field[] fields = c.getDeclaredFields();
        //获得user类全部属性
        for (int i=0;i<fields.length;i++) {
        //遍历找出删除根据的参数
            fields[i].setAccessible(true);
            //设置允许访问
            if(fields[i].getType().getCanonicalName()=="int") {
                //属性种类为int
                try {
                    if(fields[i].getInt(user)!=-1) {
                        //属性为删除根据的参数
                        return "DELETE FROM `user` WHERE
'"+fields[i].getName()+"' = "+fields[i].getInt(user);
                    }
                    Method m = c.getMethod(name[i], new Class[]{int.class});
                    //得到对应set方法
                    try {
                        Object result = m.invoke(User.class.newInstance(), new
Object[]{0});
                        //实现set方法，将值置为0
                    } catch (InstantiationException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    }

                } catch (IllegalArgumentException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IllegalAccessException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (NoSuchMethodException e) {
                    // TODO Auto-generated catch block
```

```
                    e.printStackTrace();
                } catch (SecurityException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (InvocationTargetException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }


            }else if(fields[i].getType().getCanonicalName()=="java.lang.String")
{
                 //属性种类为String
                try {
                    if(fields[i].getInt(user)!=-1) {
                     //属性为删除根据的参数
                        return "DELETE FROM `user` WHERE
'"+fields[i].getName()+"' = '"+fields[i].getInt(user)+"'";
                    }
                    Method m = c.getMethod(name[i], new Class[]{String.class});
                    //得到对应set函数
                    try {
                        Object result = m.invoke(User.class.newInstance(), new
Object[]{null});
                        //调用set函数，将值置为空
                        } catch (InstantiationException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                    }
                } catch (IllegalArgumentException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IllegalAccessException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (NoSuchMethodException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (SecurityException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (InvocationTargetException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }

        return null;

    }
///**
//     * 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）
//     * @return 返回更新语句
//     */
    public String update(User user){
        String result = "UPDATE `user` SET `";
```

```
        Class c = user.getClass();
        Object o = User.class;
        String[] name =
{"setId","setUsername","setTelephone","setEmail","setAge"};
        Field[] fields = c.getDeclaredFields();
        //获得user类全部属性

        for (int i=1;i<fields.length;i++) {
        //记录属性是否被插入并记录值
            fields[i].setAccessible(true);
            //设置允许访问
            if(fields[i].getType().getCanonicalName()=="int") {
                //判断属性类型
                try {
                    //修改值
                    if(fields[i].getInt(user)!=-1) {
                        Method m = c.getMethod(name[i], new Class[]{int.class});
                        try {
                            Object update = m.invoke(User.class.newInstance(),
new Object[]{fields[i].getInt(user)});
                        } catch (InstantiationException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                        result += fields[i].getName()+"' =
'"+fields[i].get(user)+"' ";
                    }
                } catch (IllegalArgumentException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IllegalAccessException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (NoSuchMethodException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (SecurityException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (InvocationTargetException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }else if(fields[i].getType().getCanonicalName()=="java.lang.String")
{
                try {
                    if(fields[i].get(user) != null) {
                        Method m = c.getMethod(name[i], new Class[]
{String.class});
                        try {
                            try {
                                Object update =
m.invoke(User.class.newInstance(), new Object[]{fields[i].get(user)});
                            } catch (InstantiationException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                            }
                        } catch (InvocationTargetException e) {
```

```
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                        result += fields[i].getName()+"' =
'"+fields[i].get(user)+"' ";
                    }
                } catch (IllegalArgumentException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IllegalAccessException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (NoSuchMethodException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (SecurityException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
        fields[0].setAccessible(true);
        try {
            result += "WHERE `id` = " + fields[0].getInt(user);
        } catch (IllegalArgumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return result;
    }
}
```

测试，与例子相同

```
package lab3;

import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) throws Exception {
        // initialize util

        SqlUtil util = new SqlUtil();
        // 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;
    }


}
```

# 三、实验结果