Lab3 Annotation and Reflection.

实验目的

使用注解和反射来完成动态Sql编程

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

包含main的Main

```java
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 MyUtil();

        // 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");
        user.setAge(52);
        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;

    }
}
```

```
/**
     * 根据传入的参数返回查询语句
     * @param user
     * @return 返回查询语句
     */
    @Override
    public String query(User user) throws Exception {
        // TODO Auto-generated method stub
        String gdbc = null;
        Field field = null;
        Method method =null;
        //获得Class类
        Class<?> u = user.getClass();
        //获得所有的成员变量
        Field[] fields = u.getDeclaredFields();
        //为了get修改权限
        for(Field f :fields) {
              f.setAccessible(true);
        }
        //获得所有变量的get方法
        ArrayList<Method> getmethods= getGetMethods(u);
        //检测user的那个成员变量被用于查询
        for(int i =0;i < getmethods.size();i++) {
            //获得
            method =getmethods.get(i);
            if(method.invoke(user)!=null) {
                //不为空说明是需要查询的内容，去掉get
                String searchparam = getmethods.get(i).getName().substring(3);
                //获得对应的成员变量
                field = u.getDeclaredField(initCap_l(searchparam));
                break;
                }
        }
        //获得注释
        Table  table =u.getAnnotation(Table.class);
        Column column = field.getAnnotation(Column.class);

        //还要判断是数字还是字符
        if(column.ColumnName().equals("id")||column.ColumnName().equals("age"))
{
        gdbc = "SELECT * FROM " + table.value()+ " WHERE " + column.ColumnName()
+ " = "+method.invoke(user);
        }else {
        //System.out.println(column.ColumnName());
        gdbc = "SELECT * FROM " + table.value()+ " WHERE " + column.ColumnName()
+ " LIKE "+method.invoke(user);
        }
        return gdbc;
    }
```

```
 /**
     * 根据传入的参数返回插入语句
     * @param user
     * @return 返回插入语句
     */
    // print: INSERT INTO `user` (`username`, `telephone`, `email`, `age`)
VALUES ('user', '12345678123', 'user@123.com', 20)
    @Override
    public String insert(User user) throws Exception{
        // TODO Auto-generated method stub
        String gdbc = null;
        //获得Class类
        Class<?> u = user.getClass();
        //获得所有的成员变量
        Field[] fields = u.getDeclaredFields();
        //为了get修改权限
        for(Field f :fields) {
              f.setAccessible(true);
        }
        //获得所有变量的get方法
        ArrayList<Method> getmethods= getGetMethods(u);
        //需要判断要插入的user初始化了什么成员变量
        ArrayList<Field> validfields = new ArrayList<Field>();
        for(Method m:getmethods) {
            if(m.invoke(user)!=null) {
                Field f =
u.getDeclaredField(initCap_l(m.getName().substring(3)));
                validfields.add(f);
            }
        }
        Table  table =u.getAnnotation(Table.class);
        gdbc = "INSERT INTO "+ table.value();
        String Atributelist = "(";
        String Valueslist = "(";
        for(Field field:validfields) {
            Column column = field.getAnnotation(Column.class);
            Atributelist = Atributelist+column.ColumnName()+",";
            Method m = u.getDeclaredMethod("get"+initCap(field.getName()));
            Valueslist = Valueslist +m.invoke(user)+",";
        }
        Atributelist = Atributelist.substring(0,Atributelist.length()-1)+")";
        Valueslist  =Valueslist.substring(0,Valueslist.length()-1)+")";
        gdbc = gdbc+Atributelist+" VALUES " +Valueslist;
        return gdbc;
    }
```

```
/**
     * 根据传入的参数返回插入语句
     * @param users
     * @return 返回插入语句
     */
    /** 插入一个列表应该需要判断插入的对象初始化的变量是否相同，若不相同抛出异常***/
    @Override
    public String insert(List<User> users)throws Exception {
        // TODO Auto-generated method stub
        if(!checkvalid(users)) { throw new Exception("插入属性不明确");
        }else {
        String gdbc="" ;
        for(User u:users) {
            gdbc = gdbc+insert(u);
        }
        //设置一个不可能出现的string
        String impossible_str = String.valueOf(Double.MAX_VALUE);
        //替换掉第一个valuse做保护
        gdbc = gdbc.replaceFirst("VALUES",impossible_str);
        //把其他的valuse换成，
        gdbc = gdbc.replace("VALUES", ",");
        //还原第一个valuse;
        gdbc = gdbc.replaceFirst(impossible_str,"VALUES");
        //处理“Insert into 语句”
        String useful_str = gdbc.substring(0, gdbc.indexOf(" VALUES"));
        gdbc = gdbc.replace(useful_str, "");
        gdbc = useful_str +gdbc;
        return gdbc;
        }
    }
```

```
 /**
     * 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
     */
    @Override
    public String delete(User user) throws Exception{
        // TODO Auto-generated method stub
        String gdbc = null;
        Field field = null;
        Method method =null;
        //获得Class类
        Class<?> u = user.getClass();
        //获得所有的成员变量
        Field[] fields = u.getDeclaredFields();
        //为了get修改权限
        for(Field f :fields) {
              f.setAccessible(true);
        }
        //获得所有变量的get方法
        ArrayList<Method> getmethods= getGetMethods(u);
        //检测user的那个成员变量被用于查询
        for(int i =0;i < getmethods.size();i++) {
            //获得
            method =getmethods.get(i);
            if(method.invoke(user)!=null) {
                //不为空说明是需要查询的内容，去掉get
                String searchparam = getmethods.get(i).getName().substring(3);
                //获得对应的成员变量
                field = u.getDeclaredField(initCap_l(searchparam));
                break;
                }
        }
        //获得注释
        Table  table =u.getAnnotation(Table.class);
        Column column = field.getAnnotation(Column.class);
        gdbc = "DELETE FROM " +table.value() +" WHERE "+column.ColumnName()+" =
"+method.invoke(user);
        return gdbc;
    }
```

```
/**
     * 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
     *更新时以Id为主键，可以更新多个除Id意外的属性
     */
    @Override
    public String update(User user) throws Exception {
        // TODO Auto-generated method stub
        String gdbc = null;
        String updatecontent ="";
        //获得Class类
        Class<?> u = user.getClass();
        Method getIdMethod  = u.getDeclaredMethod("getId");;
        //获得所有的成员变量
        Field[] fields = u.getDeclaredFields();
        //为了get修改权限
        for(Field f :fields) {
              f.setAccessible(true);
        }
        //获得所有变量的get方法
        ArrayList<Method> getmethods= getGetMethods(u);
        //获得GETid方法
        //检测user的那个成员变量被用于查询
        for(int i =0;i < getmethods.size();i++) {
        //获得
        Method method =getmethods.get(i);
            if(method.invoke(user)!=null&&method.getName()!="getId") {
                updatecontent = updatecontent+" , ";
            //不为空说明是需要更新的内容，去掉get
            String searchparam = getmethods.get(i).getName().substring(3);
            //获得对应的成员变量
            Field field = u.getDeclaredField(initCap_l(searchparam));
            //生成形如 SET `email` = 'change@123.com'这样的updatecontent语句
            updatecontent = updatecontent+field.getName()+" =
"+method.invoke(user);
            }
        }
        updatecontent = updatecontent.replaceFirst(" , ", "");

        //获得注释
        Table  table =u.getAnnotation(Table.class);
        // print: UPDATE `user` SET `email` = 'change@123.com' WHERE `id` = 1;
        gdbc = "UPDATE "+table.value()+" SET "+ updatecontent +" WHERE "+" id =
" + getIdMethod.invoke(user);
        return gdbc;
    }
```

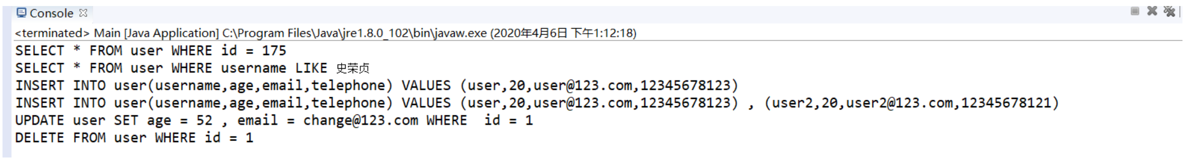
```
//一些辅助的方法
/***首字母小写变大写***/
    public static String initCap(String str){
        return str.substring(0,1).toUpperCase() + str.substring(1);
    }
    /***首字母大写变小写***/
    public static String initCap_l(String str) {
        return str.substring(0,1).toLowerCase() + str.substring(1);
    }

    /***获得GETmethod方法列表***/
    private ArrayList<Method> getGetMethods(Class<?> u) {
        ArrayList<Method> getmethods = new ArrayList<Method>();
        Field []fields = u.getDeclaredFields();
        for(Field field:fields) {
              field.setAccessible(true);
              String getmethod_name = "get"+initCap(field.getName());
              try {
                    //获得方法getmethod会抛出异常，鉴于接口不抛出异常，所以现抛现处理
                    getmethods.add(u.getMethod(getmethod_name));
               } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();}
        }
        return getmethods;
    }

    /***判段传入list的对象时，插入对象的属性是否相同***/
    public boolean checkvalid(List<User>users) throws Exception{
        //获得Class类
        Class<?> u = users.get(0).getClass();
```

```java
        ArrayList<Method> getmethods = getGetMethods(u);
        HashMap<Method,Integer> map = new HashMap<Method,Integer>();
        for(User user:users) {
            for(Method method: getmethods) {
                if(method.invoke(user)!=null) {
                    Integer temp = map.put(method, 1);
                    if(temp!=null) {
                        map.put(method, temp+1);
                    }
                }
            }
        }
        //判断是否map中的key值相同
        int temp = -1;
        for(Entry<Method,Integer> mapper: map.entrySet()){
            int test = temp;
            temp= mapper.getValue();
            if(test!=-1&&temp!=test) {
                return false;
            }
        }
        return true;
    }
```

实验截图


Console ☒

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_102\bin\javaw.exe (2020年4月6日 下午1:12:18)
SELECT * FROM user WHERE id = 175
SELECT * 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 age = 52 , email = change@123.com WHERE  id = 1
DELETE FROM user WHERE id = 1
```