Lab3实验报告

实验目的

1. 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。

2. 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

设计思路

首先要创建@Column和@Table这两个注解。@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，所以他们两个都只有一个配置参数，分别表示字段名字和表的名字，并且不需要默认值。

@Table这个注解时修饰User类的，所以设置@Target元注解，并且设置修饰类型为`ElementType.TYPE`，并且这个注解应当在运行时一直有效，因为我们要通过这个注解得到表的名字。所以设置@Retention元注解，设置生命周期为`RetentionPolicy.RUNTIME`。

```
package SqlUtil;

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();
}
```

@Column注解修饰属性，所以设置@Target元注解，并且设置修饰类型为`ElementType.FIELD`，因为要通过这个注解获得当前字段的名称，所以生命周期也是`RetentionPolicy.RUNTIME`。

```
package SqlUtil;

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();
}
```

接下来创建user类，在类的开头添加注解@Table("User")，表明表名。

```
@Table("User")
public class User {
```

在每一个属性前都加上注解@Column("xx")，表明字段名称。

```
@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;
```

同时还要有各种属性的get，set方法，便于接下来使用。

```
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}
.....
```

实体类User创建好之后，创建SqlUtil这个接口，写出我们要实现的几个生成SQL语言的函数头。

```
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);
}
```

接着创建`AnnotationsSqlUtil`类，这个类实现了`SqlUtil`接口，对于里面的五个函数，先来说说select的实现过程。

既然是要动态生成相应的SQL语句，我使用了StringBuffer。从传入的user对象得到他的类之后，调用`isAnnotationPresent()`方法看看它是否有注解@Table，如果没有的话可以直接返回null了，因为根本没法知道表的名字。

```
StringBuffer sqlBuffer = new StringBuffer();
        //获取class
        Class c = user.getClass();
        //判断是否包含Table类型的注解
        if(!c.isAnnotationPresent(Table.class)) {
            return null;
        }
```

接下来就可以先写上SQL语句的第一部分，select * from tablename，因为有可能出现没有条件，直接搜索全表的情况，所以这里先不加where，等到后面遍历的时候确定有传入字段值之后再加。

```
//获取table的名字
        Table t = (Table)c.getAnnotation(Table.class);
        String tablename = t.value();

//开始构造SQL语句，为了防止没有条件的情况，这一步不能加where
    sqlBuffer.append("select * from").append(tablename);
```

接下来就要开始使用反射来得到所有属性的字段了。同时要注意到，对于每一个属性，只有使用get方法得到的不是null的时候，才能确定传入了条件。所以设置一个计数器counts=0，当检测到有值的时候，counts++，当counts=1的时候，stringBuffer加上where 和第一个字段，当counts>1的时候，要加and之后，再加上字段和相应的值。

```
//添加一个计数器，这样方便应对需要加where，and的情况
        int counts = 0;
        //获取所有属性的字段
        Field[] fArray = c.getDeclaredFields();
```

遍历所有的字段：注意有的字段没有注解直接跳过，因为不知道他的字段名是什么。

```
//遍历所有的字段
        for (Field field : fArray) {
            //如果这个字段没有注解，直接跳过
            if(!field.isAnnotationPresent(Column.class)) {
                continue;
            }

//拿到字段上面注解的值，即Column的值
        Column column = field.getAnnotation(Column.class);
        String columnName = column.value();
        //拿到字段名
        String fieldName = field.getName();
```

然后就要使用get方法，得到该字段对应的值。因为get方法的形式都是get+第一个字母大写的属性名称，所以可以拼接得到这个方法。然后利用反射中的`getMethod()`方法，调用get函数，注意处理异常

```
String getMethodName = "get" + fieldName.substring(0, 1).toUpperCase()
                    + fieldName.substring(1);
            //属性有int、String等，所以定义为Object类
            Object fieldValue = null;
            try {
                Method getMethod = c.getMethod(getMethodName);
                fieldValue = getMethod.invoke(user);
            } catch (Exception e) {
                e.printStackTrace();
            }
```

然后检测字段的值是否为空，如果为空说明没有设置这个条件，跳过。

```
if(fieldValue==null || (fieldValue instanceof Integer && (Integer)fieldValue==0)){
                continue;
            }
```

这样才算真正确定了有限定的条件，计数器加一，并且根据计数器的值设置Where，and。

```
counts++;
            //直到这里才能确定有限定的条件，如果计数器为一，加一个where
            if(counts==1) {
                sqlBuffer.append(" Where ").append(fieldName);
            }else {
                //如果计数器不为一，说明要加and
                sqlBuffer.append(" and ").append(fieldName);
            }
```

因为字段对应的值有int也有String，String要加引号，int不需要，所以先判断一下类型。

```
if(fieldValue instanceof String) {
                sqlBuffer.append("=").append("'").append(fieldValue).append("'");
            }else {
                //fieldValue instanceof Integer
                sqlBuffer.append("=").append(fieldValue);
            }
```

SELECT的SQL语句就拼接出来了，接下来返回`stringBuffer.toString()`就好了！（终于好了。。。）

其他的四个方法和select大同小异。insert因为SQL语法的缘故，字段名称和对应的值是分开的，`INSERT INTO 'User'(username,age,email,telephone)VLAUES(user,20,user@123.com,12345678123)`，所以要遍历两遍field，一遍得到字段名，StringBuffer拼接上VLAUES之后再遍历得到字段值，一一加上。

如果insert传入的是user的list，那么可以先用`list.get()`方法，得到其中的随便一个user对象，再进行相同的操作。

```
Class c = users.get(0).getClass();
```

第一遍遍历，利用注解反射得到字段名，因为id是自增的，所以当字段名为id的时候不显示。

```
//打印表的属性字段
        for(Field field : fArray) {
            //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,因为id自增的
            if(filedName != "id") {
               sqlBuffer.append("`").append(filedName).append("`,");
             }
        }
```

因为每一个字段的名字后面都加了一个逗号，所以先回退一个逗号，再加VALUES。

```
//删除多余的逗号
        sqlBuffer.deleteCharAt(sqlBuffer.length()-1);
        sqlBuffer.append(")VLAUES(");
```

再次遍历，这次加上字段对应的值，也要注意回退一些多余的字符。

```
//遍历list
        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) {
                        sqlBuffer.append("`").append(fieldValue).append("`,");
                    }
                    else {
                        sqlBuffer.append(fieldValue).append(",");
                    }
                }
            }
            sqlBuffer.deleteCharAt(sqlBuffer.length()-1);
            sqlBuffer.append("),(");
        }

        //删除掉最后list元素多的， (
        sqlBuffer.deleteCharAt(sqlBuffer.length()-1);
        sqlBuffer.deleteCharAt(sqlBuffer.length()-1);
```

update，delete方法都是传入id，对id确定的对象操作。所以得到id可以直接调用getID()方法，就不需要经过注解反射了。

delete比较简单，只需要通过注解得到表格名称就好了。

```
public String delete(User user) {
        StringBuffer sqlBuffer = new StringBuffer();
        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 tableName = t.value();
        sqlBuffer.append("DELETE FROM `").append(tableName).append("` WHERE `id` =
").append(user.getId());
        return sqlBuffer.toString();
    }
```

update会有多个需要update的值，但是他的结构和select是一样的，所以只用遍历一遍，依次加上字段名和值就可以。然后调用getID()方法得到id，拼接就可以了。

```
public String update(User user) {
        StringBuffer sqlBuffer = new StringBuffer();
        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 tableName = t.value();
        sqlBuffer.append("UPDATE `").append(tableName).append("` SET");
        //利用映射得到类的变量
        Field[] fArray = c.getDeclaredFields();
        for (Field field : fArray) {
             //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") {
                sqlBuffer.append("`").append(filedName).append("` ").append("=").append("
`").append((String)fieldValue).append("` ");
            }

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

测试结果

写一个test类测试一下：

```
package SqlUtil;

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

public class test {
    public static void main(String[] args) {
        // initialize util
```

```
        AnnotationsSqlUtil util = new AnnotationsSqlUtil();

        // 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;
    }
}
```

测试结果如图：


Problems
Javadoc
Declaration
Console
<terminated> test [Java Application] C:\Program Files\Java\bin\javaw.exe (2020年4月11日 上午11:58:51)
select * fromUser Where id=175
select * fromUser Where username='史荣贞'
INSERT INTO 'User'(`username`,`age`,`email`,`telephone`)VLAUES(`user`,20,`user@123.com`,`123
INSERT INTO 'User'(`username`,`age`,`email`,`telephone`)VLAUES(`user`,20,`user@123.com`,`123
UPDATE `User` SET`email` = `change@123.com` WHERE `id` = 1
DELETE FROM `User` WHERE `id` = 1