# 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 3 班**

**学 号 3018216157**

**姓 名 郑开**

1. **实验目的**

使用注解和反射来完成动态Sql编程

1. **实验要求**
2. **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。

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

1. **源代码**

* 自定义注解

**注解Column**

package annotation;   
   
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();   
}

**注解Table**

package annotation;   
   
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();   
}

* 定义数据库表对应实例

**抽象类BaseDB**

package db;   
   
import annotation.Column;   
   
import java.lang.annotation.Inherited;   
   
public abstract class BaseDB {   
 @Column("id")   
 protected Integer id;   
   
 public Integer getId() {   
 return id;   
 }   
   
 public void setId(int id) {   
 this.id = id;   
 }   
   
}

**表User**

package db;   
   
import annotation.Column;   
import annotation.Table;   
   
import java.lang.annotation.Target;   
   
@Table("user")   
public class User extends BaseDB {   
   
 @Column("username")   
 private String username;   
 @Column("email")   
 private String email;   
 @Column("telephone")   
 private String telephone;   
 @Column("age")   
 private Integer age;   
   
   
 public String getUsername() {   
 return username;   
 }   
   
 public void setUsername(String username) {   
 this.username = username;   
 }   
   
 public String getEmail() {   
 return email;   
 }   
   
 public void setEmail(String email) {   
 this.email = email;   
 }   
   
 public String getTelephone() {   
 return telephone;   
 }   
   
 public void setTelephone(String telephone) {   
 this.telephone = telephone;   
 }   
   
 public int getAge() {   
 return age;   
 }   
   
 public void setAge(int age) {   
 this.age = age;   
 }   
}

* SqlUtil类

package sql;   
   
import annotation.Column;   
import annotation.Table;   
import db.BaseDB;   
   
import java.lang.reflect.Field;   
import java.lang.reflect.Type;   
import java.util.\*;   
   
public class SqlUtil<T extends BaseDB> {   
   
 private String getTableName(Class clazz) throws Exception {   
 if (clazz.isAnnotationPresent(Table.class)) {   
 Table table = (Table) clazz.getAnnotation(Table.class);   
 return table.value();   
 }   
 throw new Exception("Require Annotation Table");   
 }   
   
 private Map<String, Object> getColumns(Class clazz, T t) throws Exception {   
 Map<String, Object> map = new HashMap<>();   
 for (; clazz != Object.class; clazz = clazz.getSuperclass()) {   
 Field[] fields = clazz.getDeclaredFields();   
 for (Field field : fields) {   
 field.setAccessible(true);   
 if (field.isAnnotationPresent(Column.class)) {   
 map.put(field.getAnnotation(Column.class).value(), field.get(t));   
 }   
 }   
 }   
 return map;   
 }   
   
 /\*\*   
 \* 根据传入的参数返回查询语句   
 \*   
 \* @param t   
 \* @return 返回查询语句   
 \*/   
 public String query(T t) throws Exception {   
 StringBuilder builder = new StringBuilder();   
 builder.append("SELECT \* FROM ");   
 Class clazz = t.getClass();   
 builder.append("`" + getTableName(clazz) + "` WHERE ");   
 ArrayList<Map.Entry<String, Object>> arrayList = new ArrayList<>(getColumns(clazz, t).entrySet());   
 boolean first = true;   
 for (Map.Entry<String, Object> entry : arrayList) {   
 Object value = entry.getValue();   
 String name = entry.getKey();   
   
 if (value instanceof Integer) {   
 if (!first) {   
 builder.append(" AND ");   
 }   
 builder.append("`" + name + "` = " + value);   
 first = false;   
 } else if (value instanceof String) {   
 if (!first) {   
 builder.append(" AND ");   
 }   
 builder.append("`" + name + "` LIKE '" + value + "'");   
 first = false;   
 }   
 }   
 builder.append(" ;");   
 return builder.toString();   
 }   
   
 private List<String> getKeys(Map<String, Object> map) {   
 List<String> keys = new ArrayList<>();   
 for (Map.Entry<String, Object> entry : map.entrySet()) {   
 if (entry.getValue() != null) {   
 keys.add(entry.getKey());   
 }   
 }   
 return keys;   
 }   
   
 private String getInsertFields(List<String> keys) {   
 StringBuilder builder = new StringBuilder();   
 builder.append("(");   
 boolean first = true;   
 for (String name : keys) {   
 if (!first) {   
 builder.append(",");   
 }   
 builder.append("`").append(name).append("`");   
 first = false;   
   
 }   
 builder.append(")");   
 return builder.toString();   
 }   
   
 private String getInsertValues(Map<String, Object> map, List<String> keys) {   
 StringBuilder builder = new StringBuilder();   
 builder.append("(");   
 boolean first = true;   
 for (String name : keys) {   
 Object value = map.get(name);   
 if (value instanceof Integer) {   
 if (!first) {   
 builder.append(",");   
 }   
 builder.append(value);   
 first = false;   
 } else if (value instanceof String) {   
 if (!first) {   
 builder.append(",");   
 }   
 builder.append("'" + value + "'");   
 first = false;   
 }   
 if (value == null) {   
 if (!name.equals("id")) {   
 if (!first) {   
 builder.append(",");   
 }   
 builder.append("'null'");   
 first = false;   
 }   
 }   
 }   
 builder.append(")");   
 return builder.toString();   
 }   
   
 /\*\*   
 \* 根据传入的参数返回插入语句   
 \*   
 \* @param t   
 \* @return 返回插入语句   
 \*/   
 public String insert(T t) throws Exception {   
 StringBuilder builder = new StringBuilder();   
 builder.append("INSERT INTO ");   
 Class clazz = t.getClass();   
 builder.append("`").append(getTableName(clazz)).append("` ");   
 Map<String, Object> map = getColumns(clazz, t);   
 List<String> keys = getKeys(map);   
 builder.append(getInsertFields(keys)).append(" VALUES ");   
 builder.append(getInsertValues(map, keys));   
 builder.append(";");   
 return builder.toString();   
 }   
   
 /\*\*   
 \* 根据传入的参数返回插入语句   
 \*   
 \* @param list   
 \* @return 返回插入语句   
 \*/   
 public String insert(List<T> list) throws Exception {   
 StringBuilder builder = new StringBuilder();   
 builder.append("INSERT INTO ");   
 boolean first = true;   
 for (T t : list) {   
 Class clazz = t.getClass();   
 Map<String, Object> map = getColumns(clazz, t);   
 List<String> keys = getKeys(map);   
 if (!first) {   
 builder.append(",");   
 } else {   
 builder.append("`").append(getTableName(clazz)).append("` ");   
 builder.append(getInsertFields(keys)).append(" VALUES ");   
 }   
 first = false;   
 builder.append(getInsertValues(map, keys));   
 }   
   
 builder.append(";");   
 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 t   
 \* @return 返回删除语句   
 \*/   
 public String delete(T t) throws Exception {   
 StringBuilder builder = new StringBuilder();   
 builder.append("DELETE FROM ");   
 Class clazz = t.getClass();   
 builder.append("`" + getTableName(clazz) + "` WHERE ");   
 ArrayList<Map.Entry<String, Object>> arrayList = new ArrayList<>(getColumns(clazz, t).entrySet());   
 boolean first = true;   
 for (Map.Entry<String, Object> entry : arrayList) {   
 Object value = entry.getValue();   
 String name = entry.getKey();   
   
 if (value instanceof Integer) {   
 if (!first) {   
 builder.append(" AND ");   
 }   
 builder.append("`" + name + "` = " + value);   
 first = false;   
 } else if (value instanceof String) {   
 if (!first) {   
 builder.append(" AND ");   
 }   
 builder.append("`" + name + "` LIKE '" + value + "'");   
 first = false;   
 }   
 }   
 builder.append(" ;");   
 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 t   
 \* @return 返回更新语句   
 \*/   
 public String update(T t) throws Exception {   
 StringBuilder builder = new StringBuilder();   
 builder.append("UPDATE ");   
 Class clazz = t.getClass();   
 builder.append("`" + getTableName(clazz) + "` SET ");   
 ArrayList<Map.Entry<String, Object>> arrayList = new ArrayList<>(getColumns(clazz, t).entrySet());   
 Integer id = null;   
 boolean first = true;   
 for (Map.Entry<String, Object> entry : arrayList) {   
 Object value = entry.getValue();   
 String name = entry.getKey();   
 if (name.equals("id")) {   
 id = (Integer) value;   
 if (id == null) {   
 throw new Exception(" id 不能为空");   
 }   
 continue;   
 }   
 if (value instanceof Integer) {   
 if (!first) {   
 builder.append(" , ");   
 }   
 builder.append("`" + name + "` = " + value);   
 first = false;   
 } else if (value instanceof String) {   
 if (!first) {   
 builder.append(" , ");   
 }   
 builder.append("`" + name + "` LIKE '" + value + "'");   
 first = false;   
 }   
 }   
 if (id == null) {   
 throw new Exception(" id 不能为空");   
 }   
 builder.append(" WHERE `id` = " + id.toString() + " ;");   
 return builder.toString();   
 }   
   
}

* 测试类Main

package main;   
   
import annotation.Column;   
import annotation.Table;   
import db.BaseDB;   
import db.User;   
import sql.SqlUtil;   
   
import java.util.ArrayList;   
import java.util.List;   
   
public class Main {   
 public static void main(String[] args) throws Exception {   
 SqlUtil<User> 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 delete   
 user = new User();   
 user.setId(1);   
 System.out.println(util.delete(user));   
 // print: DELETE FROM `user` WHERE `id` = 1;   
   
 // 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 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)   
   
 }   
}

1. **设计思路**
2. 自定义注解

使用元注解规定Column和Table注解的目标，并将注解保留时间定义到运行时，这样才可以通过反射的方式获取注解，从而解析表明与属性名。

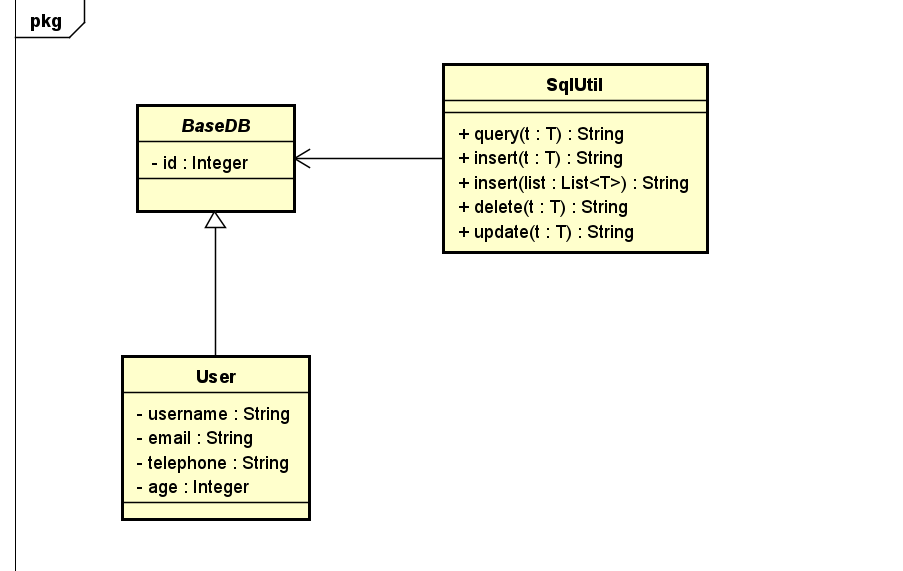
1. 实现表类

为了可以实现自定义表类的功能，我对User做了一些改变。首先定义抽象类BaseDB，所有要转化的表都要继承该类。在BaseDB中定义了id字段，即所有表都默认有一个id字段。

1. Sql语句转化

定义SqlUtil类。该类所有转化sql语句的函数，都需要BaseDB类型的实例，而不是User的实例，这样可以支持用户自定义表。但是因为要求实现的insert函数可以传参List<BaseDB>，BaseDB是基类，可能会出现List中存放的实例并不属于同一个类，可能是User类和其他继承BaseDB的类，这样并不安全。所以在SqlUtil类内定义泛型<T extends BaseDB>，而传参的类型对应改为List<T>。

1. UML图



其中T extends BaseDB