JAVA 编程进阶上机报告

Lab 3 Annotation and Reflection

学　　院	智能与计算学部
专　　业	软件工程
姓　　名	张瑞安
学　　号	3018218063
年　　级	2018
班　　级	4

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

- 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。
- 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。
- 要求实现注解：
  - @Column：用户来标注每一个 `field` 表中的字段是什么
  - @Table：用来标记名字

三、设计思路及源码

分析 `jdbc` 特性，可知 `java` 中的 `SQL` 操作使用 `PreparedStatement` 类实现可防止 `SQL` 注入，所以本实验就基于 `PreparedStatement` 中的 `SQL` 操作实现，其核心就是带占位符 `?` 的 `SQL` 语句与占位符对应的值。

- 为了方便实验项目构建和测试，使用 `maven` 工具。
- 新建 `maven` 项目，在 `pom.xml` 中引入 `junit` 依赖。

  ```
  <dependency>
              <groupId>junit</groupId>
              <artifactId>junit</artifactId>
              <version>4.13</version>
              <scope>test</scope>
          </dependency>
  ```

- 新建包 `com.kaixindeken.Annotation` ，创建 `Column.java`、`Id.java`、`Table.java` 三个注解定义文件。

  `Column.java`

  ```java
  package com.kaixindeken.Annotation;

  import java.lang.annotation.ElementType;
  import java.lang.annotation.Retention;
  import java.lang.annotation.RetentionPolicy;
  import java.lang.annotation.Target;

  @Retention(RetentionPolicy.RUNTIME)
  @Target(ElementType.FIELD)
  public @interface Column {
  ```

```
    //字段名
    String value();
    //字段类型
    Class<?> type() default String.class;
    //字段长度
    int length() default 0;
}
```

`Id.java`

```
package com.kaixindeken.Annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Id {
    //表示数据库字段id名
    String value();
}
```

`Table.java`

```
package com.kaixindeken.Annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

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

- 创建 `com.kaixindeken.TinyOrm.DAO` 包，在包中新建 `Insert.java` , `Update.java` , `Delete.java` , `Get.java` 四个接口文件

  `Insert.java`

```
package com.kaixindeken.TinyOrm.DAO;

public interface Insert<T> {
    public void insert(T t) throws Exception;
}
```

`Update.java`

```java
package com.kaixindeken.TinyOrm.DAO;

public interface Update<T> {
    public void update(T t) throws Exception;
}
```

`Get.java`

```java
package com.kaixindeken.TinyOrm.DAO;

import java.util.List;
import java.util.Map;

public interface Get<T> {
    //根据条件批量查找
    public List<T> get(Map<String, Object> sqlWhereMap, Class<T> tClass)
throws Exception;
}
```

`Delete.java`

```java
package com.kaixindeken.TinyOrm.DAO;

public interface Delete<T> {
    public void delete(Object id, Class<T> tClass) throws Exception;
}
```

- 新建 `com.kaixindeken.TinyOrm.DAOImpl` 包，新建核心类 `Core.java` ，该类用于获得表名以及根据条件返回 `SQL` 条件和条件中占位符的值。（其实就是测试时在终端打印出来）

`Core.java`

```java
package com.kaixindeken.TinyOrm.DAOImpl;

import com.kaixindeken.Annotation.*;

import java.util.*;

public class Core<T> {

    //获得表名
    public String getTableName(Class<?> clazz) throws Exception{
        if(clazz.isAnnotationPresent(Table.class)){
            Table table = clazz.getAnnotation(Table.class);
            return table.value();
        }else{
            throw new Exception(clazz.getName()+" is not Table
Annotation.");
        }
    }

    //根据条件返回sql条件和条件中占位符的值
    public List<Object> getSQLConditionWithVal(Map<String, Object> sqlMap){
        if (sqlMap.size() < 1){
            return null;
        }
        List<Object> list = new ArrayList<Object>();
        List<Object> field_values = new ArrayList<Object>();
        StringBuffer sql = new StringBuffer(" where ");
        Set<Map.Entry<String,Object>> entry_sets = sqlMap.entrySet();
        Iterator<Map.Entry<String,Object>> iterator = entry_sets.iterator();
        while(iterator.hasNext()){
            Map.Entry<String,Object> entry_set = iterator.next();
            field_values.add(entry_set.getValue());
            Object value = entry_set.getValue();
            if (value.getClass() == String.class){
                sql.append(entry_set.getKey()).append(" like
").append("?").append(" and ");
            }else{

 sql.append(entry_set.getKey()).append("=").append("?").append(" and ");
            }
        }

        //删除多余and
        sql.delete(sql.lastIndexOf("and"),sql.length());

        list.add(sql.toString());
        list.add(field_values);
        return list;
    }

}
```

- 分别创建对上面四个接口的实现文件， `InsertImpl.java` , `UpdateImpl.java` , `DeleteImpl.java` , `GetImpl.java`

`InsertImpl.java`

```
package com.kaixindeken.TinyOrm.DAOImpl;

import com.kaixindeken.Annotation.*;
import com.kaixindeken.TinyOrm.DAO.*;

import java.beans.PropertyDescriptor;
import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.List;

public class InsertImpl<T> implements Insert<T> {
    private static final String Table_Alias="t";
    @Override
    public void insert(T t) throws Exception {
        Class<?> clazz = t.getClass();

        //获取表名
        String table_name = new Core<T>().getTableName(clazz);
        //字段名
        StringBuilder field_names = new StringBuilder();
        //字段值
        List<Object> field_values = new ArrayList<Object>();
        //占位符
        StringBuilder place_holders = new StringBuilder();

        //获取字段名和值
        Field[] fields = clazz.getDeclaredFields();
        for (Field field : fields) {
            PropertyDescriptor pd = new
PropertyDescriptor(field.getName(),t.getClass());
            if (field.isAnnotationPresent(Id.class)){

 field_names.append(field.getAnnotation(Id.class).value()).append(",");
                field_values.add(pd.getReadMethod().invoke(t));
            }else if (field.isAnnotationPresent(Column.class)){

 field_names.append(field.getAnnotation(Column.class).value()).append(",");
                field_values.add(pd.getReadMethod().invoke(t));
            }
            place_holders.append("?").append(",");
        }

        //删除最后一个逗号
        field_names.deleteCharAt(field_names.length()-1);
        place_holders.deleteCharAt(place_holders.length()-1);

        //拼接sql
        StringBuilder sql = new StringBuilder("");
        sql.append("insert into ").append(table_name)
                .append(" (")
                .append(field_names.toString())
                .append(")")
                .append(" values ")
                .append("(")
                .append(place_holders)
                .append(")");

        //打印sql以及占位符的值
        System.out.println("\nSQL语句: "+sql+"\n占位符的值:
"+field_values+"\n");
    }
}
```

`UpdateImpl.java` （目前仅能拼出通过 `id` 进行更新的 `SQL` 语句）

```
package com.kaixindeken.TinyOrm.DAOImpl;

import com.kaixindeken.TinyOrm.DAO.Update;

import java.beans.PropertyDescriptor;
import java.lang.reflect.Field;
import java.util.*;
import com.kaixindeken.Annotation.*;

public class UpdateImpl<T> implements Update<T> {

    @Override
    public void update(T t) throws Exception {
        Class<?> clazz = t.getClass();
        //获取表名
        String table_name = new Core<T>().getTableName(clazz);

        List<String> field_names = new ArrayList<String>();
        List<Object> field_values = new ArrayList<Object>();
        List<String> place_holders = new ArrayList<String>();

        String id_field_name="";
        Object id_field_value="";
        //获取字段名和值
        Field[] fields = clazz.getDeclaredFields();
        for (Field field:fields){
            PropertyDescriptor pd = new
PropertyDescriptor(field.getName(),t.getClass());
            if (field.isAnnotationPresent(Id.class)){
                id_field_name = field.getAnnotation(Id.class).value();
                id_field_value = pd.getReadMethod().invoke(t);
            }else if (field.isAnnotationPresent(Column.class)){
                field_names.add(field.getAnnotation(Column.class).value());
                field_values.add(pd.getReadMethod().invoke(t));
                place_holders.add("?");
            }
        }
        field_names.add(id_field_name);
        field_values.add(id_field_value);
        place_holders.add("?");

        StringBuilder sql = new StringBuilder("");
        sql.append("update ").append(table_name).append(" set ");
        int index = field_names.size()-1;
        for (int i=0; i<index; i++){

 sql.append(field_names.get(i)).append("=").append(place_holders.get(i)).app
end(",");
        }
        sql.deleteCharAt(sql.length()-1).append(" where
").append(id_field_name).append("=").append("?");

        //打印sql以及占位符的值
        System.out.println("\nSQL语句: "+sql+"\n占位符的值:
"+field_values+"\n");
    }
}
```

`DeleteImpl.java` （目前仅能拼出通过 `id` 进行删除的 `SQL` 语句）

```
package com.kaixindeken.TinyOrm.DAOImpl;

import com.kaixindeken.TinyOrm.DAO.Delete;

public class DeleteImpl<T> implements Delete<T> {
    @Override
    public void delete(Object id, Class<T> tClass) throws Exception {
        //获取表名
        String table_name = new Core<T>().getTableName(tClass);

        StringBuilder sql = new StringBuilder("");
        sql.append("delete from ").append(table_name).append(" where id =
").append(id);

        //打印sql
        System.out.println("\nSQL语句: "+sql+"\n");
    }
}
```

`GetImpl.java`

```
package com.kaixindeken.TinyOrm.DAOImpl;

import com.kaixindeken.TinyOrm.DAO.Get;

import java.lang.reflect.Field;
import java.util.*;
import com.kaixindeken.Annotation.*;

public class GetImpl<T> implements Get<T> {

    private static final String TABLE_ALIAS = "t";
    @Override
    public List<T> get(Map<String, Object> sql_where_map, Class<T> tClass)
throws Exception {
        List<T> list = new ArrayList<T>();
        String table_name = new Core<T>().getTableName(tClass);
        String id_field_name = "";

        StringBuffer field_names = new StringBuffer();
        Field[] fields = tClass.getDeclaredFields();
        for (Field field:fields){
            String property_name = field.getName();
            if (field.isAnnotationPresent(Id.class)){
                id_field_name = field.getAnnotation(Id.class).value();
                field_names.append(TABLE_ALIAS+"."+id_field_name)
                        .append(" as ")
                        .append(property_name)
                        .append(",");
            }else if (field.isAnnotationPresent(Column.class)){

 field_names.append(TABLE_ALIAS+"."+field.getAnnotation(Column.class).value(
))
                        .append(" as ")
                        .append(property_name)
                        .append(",");
            }
        }
        //删除逗号
        field_names.deleteCharAt(field_names.length()-1);

        //拼装sql
        String sql = "select "+field_names+" from "+table_name+"
"+TABLE_ALIAS;

        List<Object> values = null;
        if (sql_where_map != null){
            List<Object> sql_condition_values = new Core<T>
().getSQLConditionWithVal(sql_where_map);
            if (sql_condition_values != null){
                //拼接sql条件
                String sql_where = (String) sql_condition_values.get(0);
                sql += sql_where;
                //得到sql条件占位符的值
                values = (List<Object>) sql_condition_values.get(1);
            }
        }

        //打印sql以及占位符的值
        System.out.println("\nSQL语句: "+sql+"\n占位符的值: "+values+"\n");

        return list;
    }
}
```

- 关键代码完成，接下来编写单元测试代码，在 `src/test/java` 目录下创建 `com.kaixindeken.TinyOrm` 包。
- 按照实验要求新建 `User` 类，定义 `User` 属性并生成 `Getter` 和 `Setter` 。

  `User.java`

```
package com.kaixindeken.TinyOrm;

import com.kaixindeken.Annotation.Column;
import com.kaixindeken.Annotation.Id;
import com.kaixindeken.Annotation.Table;

//用户表
@Table("users")
public class User {

    @Id("id")
    private int id;

    //姓名
    @Column("name")
    private String name;

    //年龄
    @Column("age")
    private int age;

    //电子邮箱
    @Column("email")
    private String email;

    //电话号码
    @Column("tel")
    private String tel;

    public int getId() { return id; }

    public String getName() { return name; }
```

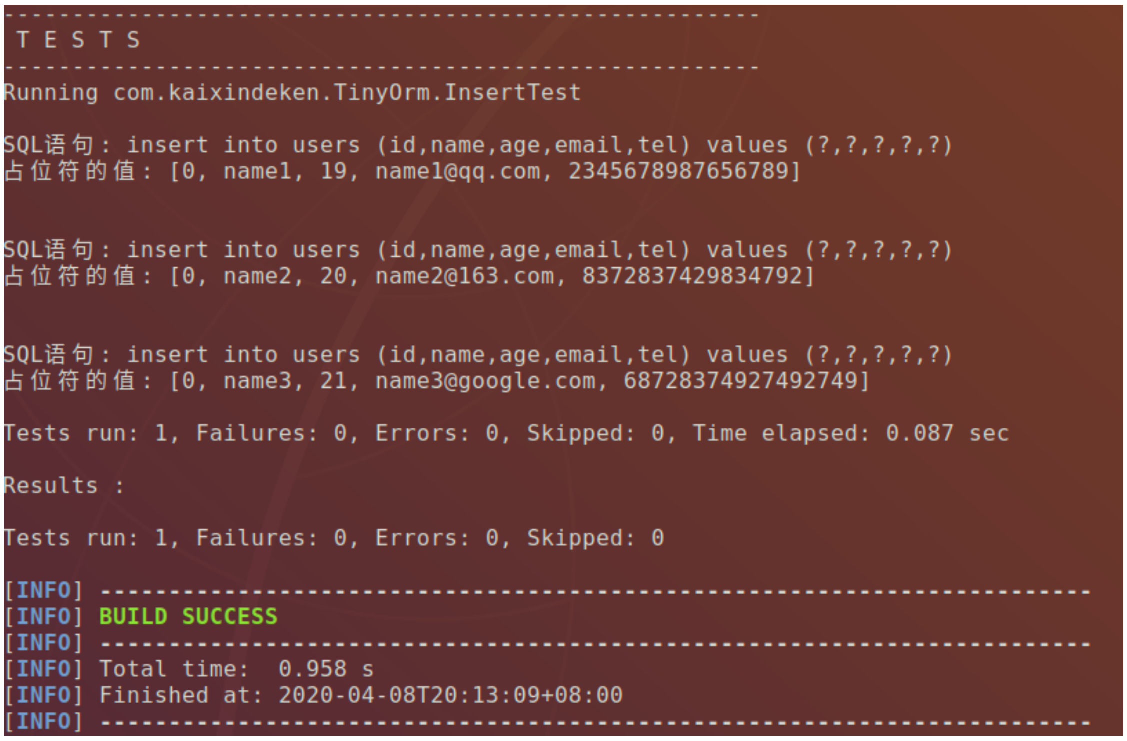
```java
    public int getAge() { return age; }

    public String getEmail() { return email; }

    public String getTel() { return tel; }

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

    public void setName(String name) { this.name = name; }

    public void setAge(int age) { this.age = age; }

    public void setEmail(String email) { this.email = email; }

    public void setTel(String tel) { this.tel = tel; }

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", email='" + email + '\'' +
                ", tel='" + tel + '\'' +
                '}';
    }

}
```

- 新建插入测试类

`InsertTest.java`

```java
package com.kaixindeken.TinyOrm;

import com.kaixindeken.TinyOrm.DAO.Insert;
import com.kaixindeken.TinyOrm.DAOImpl.InsertImpl;
import org.junit.Test;

public class InsertTest {

    private Insert<User> userInsert = new InsertImpl<User>();
    @SuppressWarnings("deprecation")
    @Test
    public void TestInsert() throws Exception{
        //构造用户
        User user1 = new User();
        User user2 = new User();
        User user3 = new User();

        user1.setName("name1");
        user1.setAge(19);
        user1.setEmail("name1@qq.com");
        user1.setTel("2345678987656789");

        user2.setName("name2");
        user2.setAge(20);
        user2.setEmail("name2@163.com");
        user2.setTel("8372837429834792");

        user3.setName("name3");
        user3.setAge(21);
        user3.setEmail("name3@google.com");
        user3.setTel("68728374927492749");

        userInsert.insert(user1);
        userInsert.insert(user2);
        userInsert.insert(user3);
    }

}
```

- 输入 `mvn test -Dtest=InsertTest` 得到测试结果



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.kaixindeken.TinyOrm.InsertTest

SQL语句：insert into users (id,name,age,email,tel) values (?,?,?,?,?)
占位符的值：[0, name1, 19, name1@qq.com, 2345678987656789]


SQL语句：insert into users (id,name,age,email,tel) values (?,?,?,?,?)
占位符的值：[0, name2, 20, name2@163.com, 8372837429834792]


SQL语句：insert into users (id,name,age,email,tel) values (?,?,?,?,?)
占位符的值：[0, name3, 21, name3@google.com, 68728374927492749]

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.087 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.958 s
[INFO] Finished at: 2020-04-08T20:13:09+08:00
[INFO] ------------------------------------------------------------------------
```

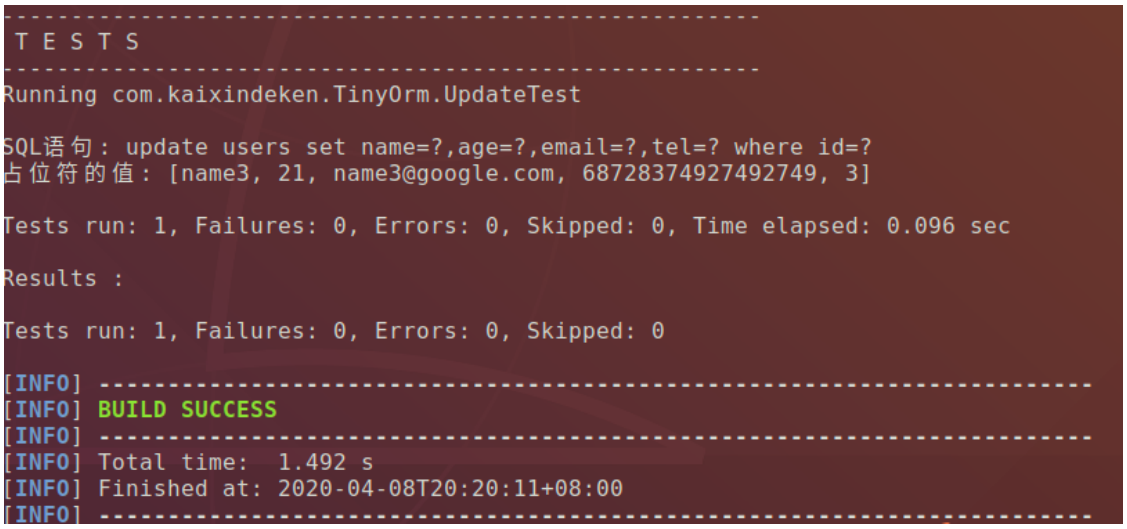
- 新建更新测试类

  `UpdateTest.java`

```
package com.kaixindeken.TinyOrm;

import com.kaixindeken.TinyOrm.DAO.Update;
import com.kaixindeken.TinyOrm.DAOImpl.UpdateImpl;
import org.junit.Test;

public class UpdateTest {

    private Update<User> userUpdate = new UpdateImpl<User>();
    @SuppressWarnings("deprecation")
    @Test
    public void TestUpdate() throws Exception{
        User user = new User();
        user.setId(3);
        user.setName("name3");
        user.setAge(21);
        user.setEmail("name3@google.com");
```

```java
        user.setTel("68728374927492749");
        userUpdate.update(user);
    }
}
```

- 输入 `mvn test -Dtest=UpdateTest` 得到测试结果



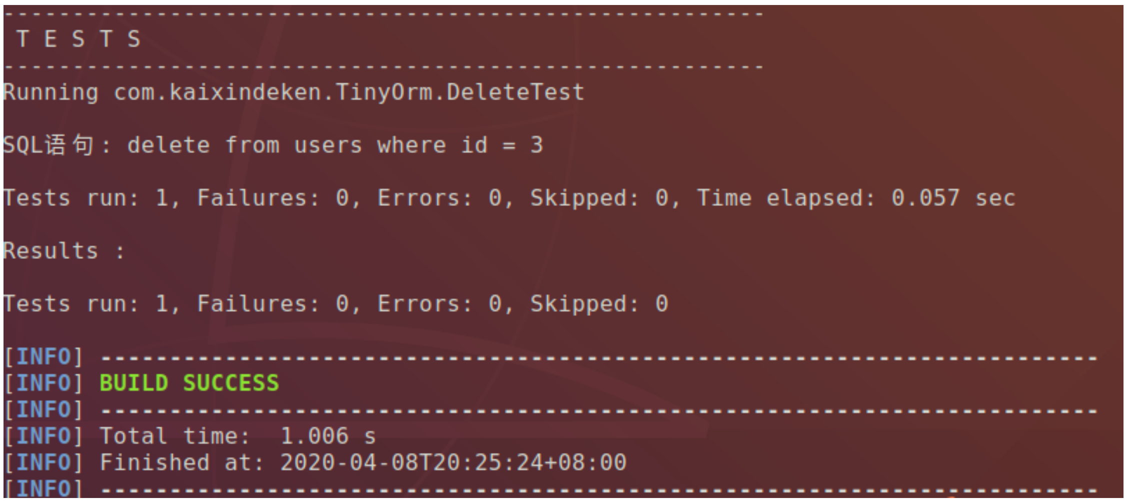
```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.kaixindeken.TinyOrm.UpdateTest

SQL语句: update users set name=?,age=?,email=?,tel=? where id=?
占位符的值: [name3, 21, name3@google.com, 68728374927492749, 3]

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.096 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.492 s
[INFO] Finished at: 2020-04-08T20:20:11+08:00
[INFO] ------------------------------------------------------------------------
```

- 新建删除测试类

  `DeleteTest.java`

```java
package com.kaixindeken.TinyOrm;

import com.kaixindeken.TinyOrm.DAO.Delete;
import com.kaixindeken.TinyOrm.DAOImpl.DeleteImpl;
import org.junit.Test;

public class DeleteTest {

    private Delete<User> userDelete = new DeleteImpl<User>();
    @Test
    public void TestDelete() throws Exception{
        userDelete.delete(3,User.class);
    }
}
```

- 输入 `mvn test -Dtest=DeleteTest` 得到测试结果



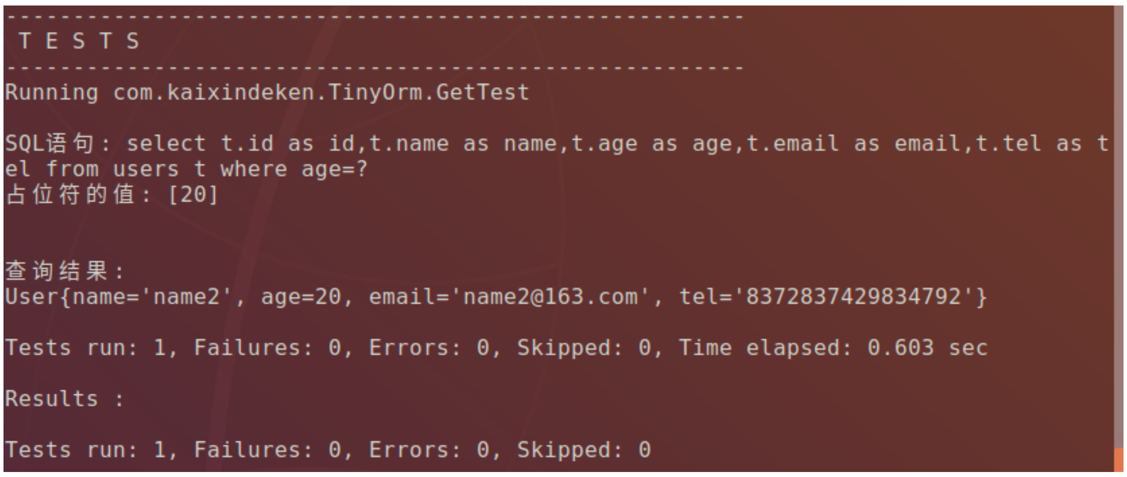
```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.kaixindeken.TinyOrm.DeleteTest

SQL语句: delete from users where id = 3

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.057 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.006 s
[INFO] Finished at: 2020-04-08T20:25:24+08:00
[INFO] ------------------------------------------------------------------------
```

- 新建获取测试类

`GetTest.java`

```java
package com.kaixindeken.TinyOrm;

import com.kaixindeken.TinyOrm.DAO.Get;
import com.kaixindeken.TinyOrm.DAOImpl.GetImpl;
import org.junit.Test;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class GetTest {

    private Get<User> userGet = new GetImpl<User>();

    @Test
    public void TestGet() throws Exception{
        Map<String,Object> sqlMap = new HashMap<String, Object>();
        sqlMap.put("age",20);
        List<User> users = userGet.get(sqlMap,User.class);
        for (User user:users){
            System.out.println("\n查询结果:\n"+user+"\n");
        }
    }

}
```

- 输入 `mvn test -Dtest=GetTest` 得到输出结果



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.kaixindeken.TinyOrm.GetTest
SQL语句: select t.id as id,t.name as name,t.age as age,t.email as email,t.tel as t
el from users t where age=?
占位符的值: [20]


查询结果:
User{name='name2', age=20, email='name2@163.com', tel='8372837429834792'}

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.603 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

四、（进阶）实现 `jdbc` 数据库操作

- 既然我都这么写了，就再花点时间实现一下吧。
- 数据库准备（参考 `User.java` ）

```
create table users
(
    id    bigint auto_increment
        primary key,
    name  varchar(100) null,
    age   int          null,
    email varchar(100) null,
    tel   varchar(100) null
);
```

- 在 `pom.xml` 文件中添加 `mysql-connector-java` 依赖

```
<dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.19</version>
</dependency>
```

- 在两个 `recourses` 文件夹下新建 `mysql.properties` 文件，便于引用 `Properties` 类建立数据库连接（数据库地址，用户名，密码自行填写）

`mysql.properties`

```
DBDriver=com.mysql.cj.jdbc.Driver
URL=
USERNAME=
PASSWORD=
```

- 在包 `com.kaixindeken.TinyOrm` 新建驱动加载类 `JdbcHelper.java`

`JdbcHelper.java`

```
package com.kaixindeken.TinyOrm;

import java.io.*;
import java.sql.*;
import java.util.Properties;

public class JdbcDAOHelper {

    private static Connection con;
    private static Properties properties;

    static {
        properties = new Properties();
        InputStream in =
JdbcDAOHelper.class.getResourceAsStream("/mysql.properties");

        try{
            properties.load(in);
        }catch(IOException ex){
            System.out.println(ex.getMessage());
            ex.printStackTrace();
        }
    }
    //加载驱动建立连接
    public static Connection getCon(){
        try{
            if (con == null){
                Class.forName(properties.getProperty("DBDriver"));
                try{
                    con = DriverManager.getConnection(
                            properties.getProperty("URL"),
                            properties.getProperty("USERNAME"),
                            properties.getProperty("PASSWORD")
                    );
                }catch(SQLException e){
                    e.printStackTrace();
                }
            }
        }catch (ClassNotFoundException e){
            e.printStackTrace();;
        }
        return con;
    }

    //释放资源
    public static void release(PreparedStatement ps, ResultSet rs){
        //关闭连接
        if (con != null){
            try{
                con.close();
            }catch(SQLException e){
                e.printStackTrace();
            }
            con = null;
        }
        //清除PreparedStatement占用的资源
        if (ps != null){
            try{
                ps.close();
            }catch (SQLException e){
                e.printStackTrace();
            }
        }
        //清除ResultSet中的资源
        if (rs != null){
            try{
                rs.close();
            }catch(SQLException e){
                e.printStackTrace();
            }
        }
    }
}
```

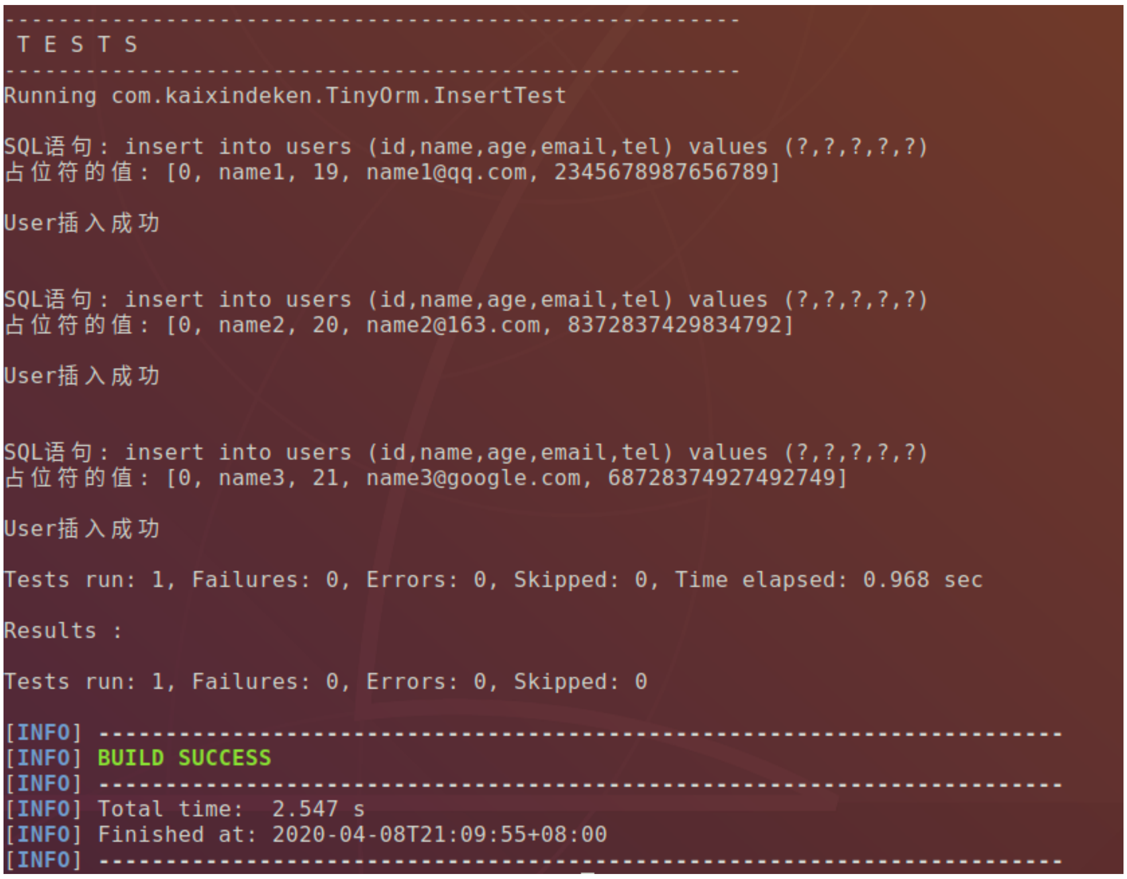
- 在本实验的设计中，占位符较多的为 `insert` ， `update` ， `get` 的实现，所以导入占位符的值的类型区分就有点烦，在已存在的 `Core.java` 中增加设置占位符值的函数

```
//设置sql参数占位符的值
    public void setParam(List<Object> values, PreparedStatement ps, boolean
isSearch) throws SQLException {
        for(int i=1;i<=values.size();i++){
            Object field_value = values.get(i-1);
            Class<?> clazz_value = field_value.getClass();
            if (clazz_value == String.class){
                if (isSearch){
                    ps.setString(i,"%"+field_value+"%");
                }else{
                    ps.setString(i,(String)field_value);
                }
            }else if (clazz_value == boolean.class || clazz_value ==
Boolean.class){
                ps.setBoolean(i,(Boolean) field_value);
            }else if (clazz_value == byte.class || clazz_value ==
Byte.class){
                ps.setByte(i,(Byte)field_value);
            }else if (clazz_value.isArray()){
                Object[] array_value = (Object[]) field_value;
                StringBuffer sb = new StringBuffer();
                for (int j=0;j<array_value.length;j++){
                    sb.append(array_value[j]).append(",");
                }
                ps.setString(i,sb.deleteCharAt(sb.length()-1).toString());
            } else {
                ps.setObject(i,field_value, Types.NUMERIC);
            }
        }
    }
```

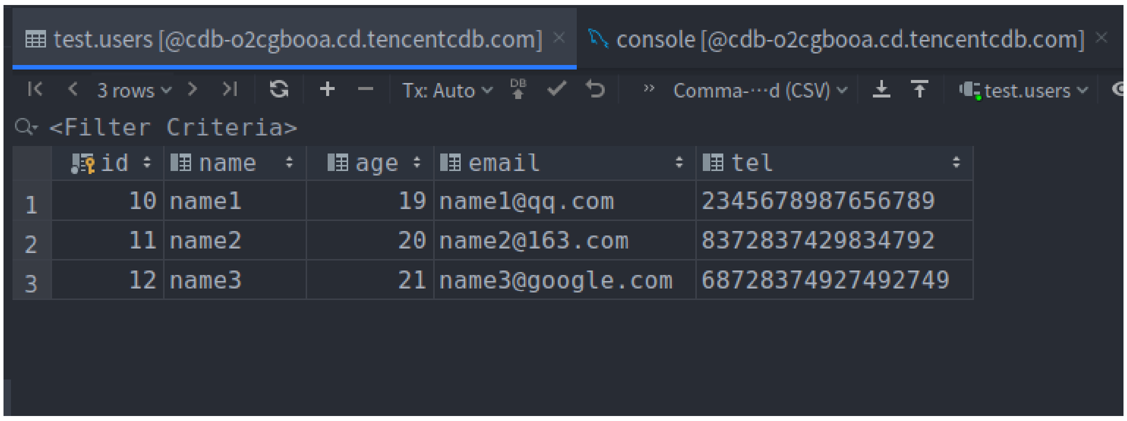
- 好的还差一点点设置就可以开始执行拼写好的 `insert` 语句了，在 `InsertImpl.java` 的末尾打印的指令下载加几行

```
        PreparedStatement ps =
JdbcDAOHelper.getCon().prepareStatement(sql.toString());
        //设置sql参数占位符的值
        new Core<T>().setParam(field_values,ps,false);

        //执行sql
        ps.execute();
        JdbcDAOHelper.release(ps,null);

        System.out.println(clazz.getSimpleName()+"插入成功"+"\n");
```

- 好的我们执行 `mvn test -Dtest=InsertTest` ，查看一下结果



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.kaixindeken.TinyOrm.InsertTest

SQL语句： insert into users (id,name,age,email,tel) values (?,?,?,?,?)
占位符的值： [0, name1, 19, name1@qq.com, 2345678987656789]

User插入成功


SQL语句： insert into users (id,name,age,email,tel) values (?,?,?,?,?)
占位符的值： [0, name2, 20, name2@163.com, 8372837429834792]

User插入成功


SQL语句： insert into users (id,name,age,email,tel) values (?,?,?,?,?)
占位符的值： [0, name3, 21, name3@google.com, 68728374927492749]

User插入成功

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.968 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.547 s
[INFO] Finished at: 2020-04-08T21:09:55+08:00
[INFO] ------------------------------------------------------------------------
```

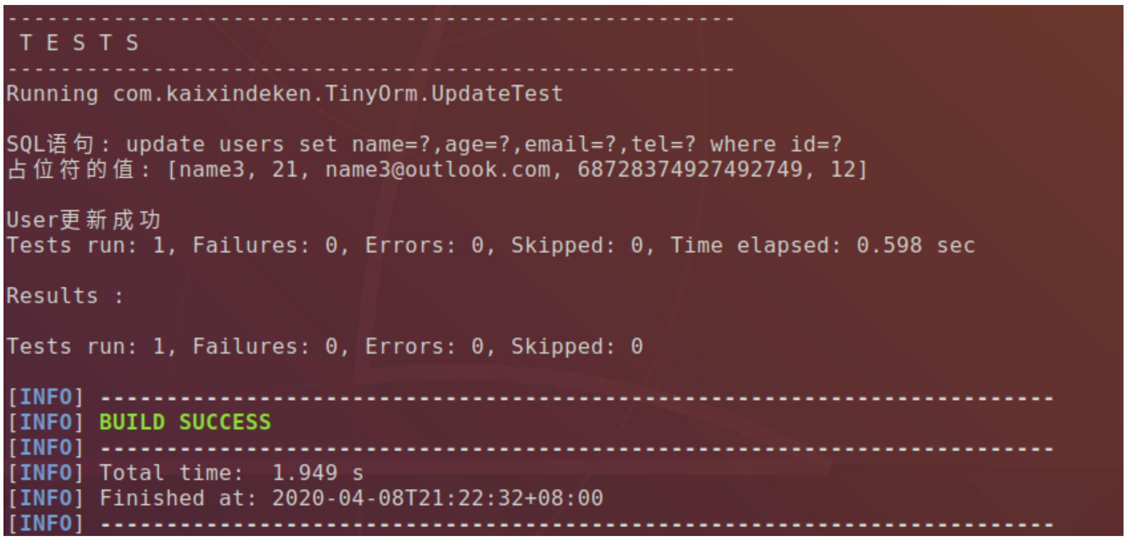

test.users [@cdb-o2cgbooa.cd.tencentcdb.com]
console [@cdb-o2cgbooa.cd.tencentcdb.com]
3 rows
Tx: Auto
Comma-…d (CSV)
test.users
<Filter Criteria>

	id	name	age	email	tel
1	10	name1	19	name1@qq.com	2345678987656789
2	11	name2	20	name2@163.com	8372837429834792
3	12	name3	21	name3@google.com	68728374927492749

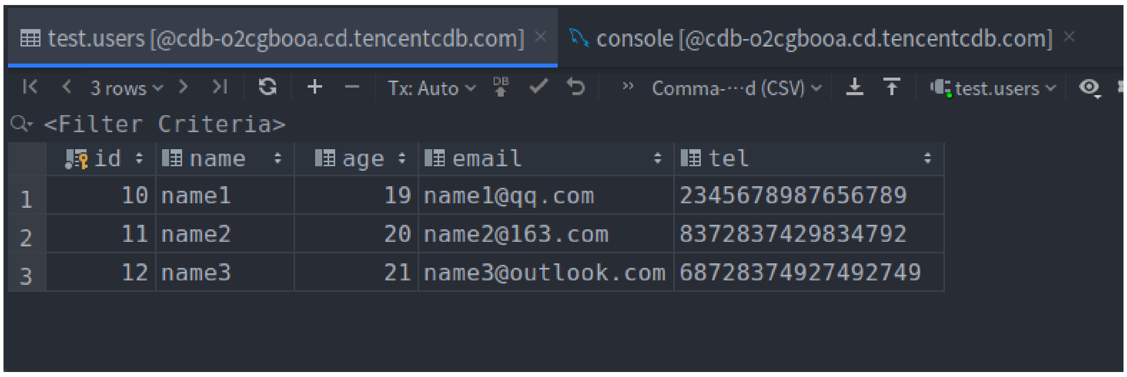
- nice，接下来我们来实现 `update` 语句，在 `UpdateImpl.java` 打印指令下添加

```
        PreparedStatement ps =
JdbcDAOHelper.getCon().prepareStatement(sql.toString());
        //设置SQL参数占位符的值
        new Core<T>().setParam(field_values,ps,false);
        //执行sql
        ps.execute();
        JdbcDAOHelper.release(ps,null);
        System.out.println(clazz.getSimpleName()+"更新成功");
```

- 然后改一下 `UpdateTest.java` 的 `id` 为 12，`setEmail` 为 `name3@outlook.com`，执行 `mvn test -Dtest=UpdateTest`，查看一下结果。



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.kaixindeken.TinyOrm.UpdateTest

SQL语句: update users set name=?,age=?,email=?,tel=? where id=?
占位符的值: [name3, 21, name3@outlook.com, 68728374927492749, 12]

User更新成功
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.598 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.949 s
[INFO] Finished at: 2020-04-08T21:22:32+08:00
[INFO] ------------------------------------------------------------------------
```

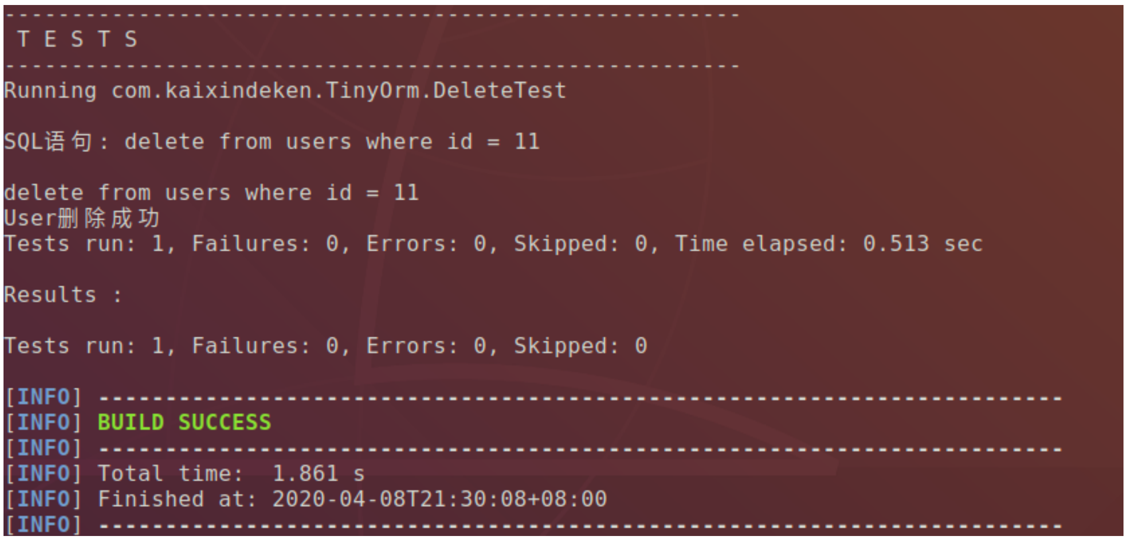

test.users [@cdb-o2cgbooa.cd.tencentcdb.com]
console [@cdb-o2cgbooa.cd.tencentcdb.com]
3 rows
Tx: Auto
Comma-…d (CSV)
test.users
<Filter Criteria>

	id	name	age	email	tel
1	10	name1	19	name1@qq.com	2345678987656789
2	11	name2	20	name2@163.com	8372837429834792
3	12	name3	21	name3@outlook.com	68728374927492749

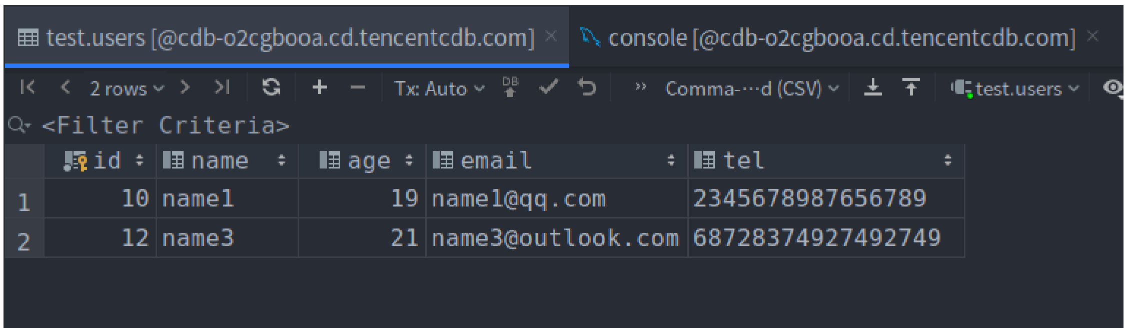
- 实现 `delete` 语句，在 `DeleteImpl.java` 打印指令下添加

```
        PreparedStatement ps =
JdbcDAOHelper.getCon().prepareStatement(sql.toString());
        //执行sql
        ps.execute();
        JdbcDAOHelper.release(ps,null);
        System.out.println(sql+"\n"+tClass.getSimpleName()+"删除成功");
```

- 改一下 `DeleteTest.java` 的 `id` 为 11，执行 `mvn test -Dtest=DeleteTest` ，查看一下结果。



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.kaixindeken.TinyOrm.DeleteTest

SQL语句: delete from users where id = 11

delete from users where id = 11
User删除成功
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.513 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.861 s
[INFO] Finished at: 2020-04-08T21:30:08+08:00
[INFO] ------------------------------------------------------------------------
```


test.users [@cdb-o2cgbooa.cd.tencentcdb.com]
console [@cdb-o2cgbooa.cd.tencentcdb.com]
2 rows
Tx: Auto
Comma-…d (CSV)
test.users
<Filter Criteria>

```
   id  name   age  email              tel
1  10  name1  19   name1@qq.com       2345678987656789
2  12  name3  21   name3@outlook.com  68728374927492749
```

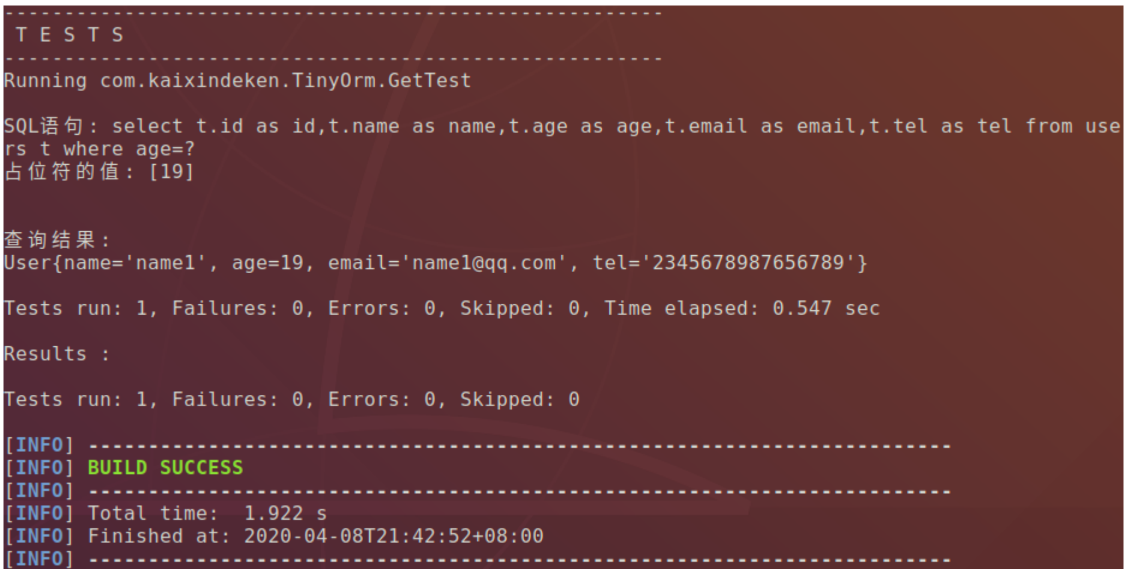
- 为了实现 `get` 方法，需要使用 `ResultSet` 结果集，并能根据结果集初始化对象，在 `Core.java` 中添加方法

```
//根据结果集初始化对象
    public void initObject(T t, Field[] fields, ResultSet rs)
            throws SQLException, IntrospectionException,
IllegalAccessException, InvocationTargetException {
        for (Field field: fields) {
            String property_name = field.getName();
            Object param_value = null;
            Class<?> clazz_field = field.getType();
            if (clazz_field == String.class){
                param_value = rs.getString(property_name);
            }else if (clazz_field == short.class || clazz_field ==
Short.class){
                param_value = rs.getShort(property_name);
            }else if (clazz_field == int.class || clazz_field ==
Integer.class){
                param_value = rs.getInt(property_name);
            }else if(clazz_field == long.class || clazz_field == Long.class)
{
                param_value = rs.getLong(property_name);
            }else if (clazz_field == float.class || clazz_field ==
Float.class){
                param_value = rs.getFloat(property_name);
            }else if(clazz_field == double.class || clazz_field ==
Double.class){
                param_value = rs.getDouble(property_name);
            }else if (clazz_field == byte.class || clazz_field ==
Byte.class){
                param_value = rs.getByte(property_name);
            }else if (clazz_field == char.class || clazz_field ==
Character.class){
                param_value = rs.getCharacterStream(property_name);
            }else if (clazz_field.isArray()){
                //逗号分隔字符串
                param_value = rs.getString(property_name).split(",");
            }
            PropertyDescriptor pd = new
PropertyDescriptor(property_name,t.getClass());
            pd.getWriteMethod().invoke(t,param_value);
        }
    }
```

- 实现 `get` 方法，在 `GetImpl.java` 打印指令下添加

```
        PreparedStatement ps;
        //设置参数占位符的值
        if (values != null){
            ps = JdbcDAOHelper.getCon().prepareStatement(sql);
            new Core<T>().setParam(values,ps,true);
        }else{
            ps=JdbcDAOHelper.getCon().prepareStatement(sql);
        }
        //执行sql
        ResultSet rs = ps.executeQuery();
        while(rs.next()){
            T t = tClass.newInstance();
            new Core<T>().initObject(t,fields,rs);
            list.add(t);
        }

        //释放资源
        JdbcDAOHelper.release(ps,rs);

        return list;
```

- 更改 `GetIMpl.java` 的 `age` 为 `19`， 执行 `mvn test -Dtest=GetTest`，查看结果



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.kaixindeken.TinyOrm.GetTest

SQL语句：select t.id as id,t.name as name,t.age as age,t.email as email,t.tel as tel from use
rs t where age=?
占位符的值：[19]


查询结果：
User{name='name1', age=19, email='name1@qq.com', tel='2345678987656789'}

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.547 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.922 s
[INFO] Finished at: 2020-04-08T21:42:52+08:00
[INFO] ------------------------------------------------------------------------
```

- 到此，所有代码就都实现啦，啦啦啦啦啦啦啦。
- 点击此处查看源代码