# 实验1反射读取注解信息

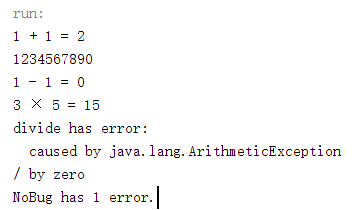
一、实验目的：

1、理解注解的作用。

2、了解反射机制。

1. 实验内容：

1、请在【代码一】~【代码五】处，填写适当的代码，实现对NoBug类中使用@Check注解的方法进行测试，生成的测试报告内容如下：



【实验步骤】

1. 新建一个名为”ex1”的项目（Java应用程序）。在项目中，新建Java类，命名为“Check”。在Check.java文件中，定义@Check注解。

//【代码一】

//定义名为@Check的注解，该注解只能定义在方法上，并且保留策略为//RetentionPolicy.RUNTIME。

1. 新建Java类，命名为“NoBug”。在NoBug.java文件中，定义被测试的NoBug类，给该类中的方法加上@Check注解。

public class NoBug {

@Check

public void print() {

System.out.println("1234567890");

}

@Check

public void add() {

System.out.println("1 + 1 = " + (1 + 1));

}

@Check

public void subtract() {

System.out.println("1 - 1 = " + (1 - 1));

}

@Check

public void multiply() {

System.out.println("3 × 5 = " + (3 \* 5));

}

@Check

public void divide() {

System.out.println("6 / 0= " + (6 / 0));

}

public void manifesto() {

System.out.println("我写的程序没有 bug!");

}

}

1. 新建Java类，命名为“TestTool”。在TestTool.java文件中，执行使用了@Check注解的方法，并生成测试报告。

//【代码二】导入包

public class TestTool {

public static void main(String[] args) throws Exception {

Class clazz = NoBug.class;

//【代码三】通过clazz创建名为noBug的NoBug实例对象

//【代码四】获取NoBug类声明的方法

//用来记录测试产生的 log 信息

StringBuilder log = new StringBuilder();

// 记录异常的次数

int errornum = 0;

for (Method m : methods) {

if (/\*【代码五】\*/) { // 只有被@Check 标注过的方法才进行测试

try {

m.invoke(noBug);

} catch (Exception e) {

errornum++;

log.append(m.getName());

log.append(" ");

log.append("has error:");

log.append("\n caused by ");

//记录测试过程中，发生的异常的名称

log.append(e.getCause().getClass().getName());

log.append("\n");

//记录测试过程中，发生的异常的具体信息

log.append(e.getCause().getMessage());

log.append("\n");

}

}

}

//获取被测试的类名（不包含包名）

log.append(clazz.getSimpleName());

log.append(" has ");

log.append(errornum);

log.append(" error.");

// 生成测试报告

System.out.println(log.toString());

}

}

【选读】2、Java反射获取注解，并拼接为SQL命令。请阅读下面代码，进一步理解注解的作用。

（1）先建两个注解 分别为 Table 和 Column。

1）Table.java

import java.lang.annotation.\*;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.TYPE)

public @interface Table {

String value();

}

2）Column.java

import java.lang.annotation.\*;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.FIELD)

public @interface Column {

String value();

}

（2）在Person类中使用注解

@Table("person")

public class Person {

@Column("name")

private String name;

@Column("user\_name")

private String userName;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

}

（3）在SqlJoin中读取注解，以实现SQL命令的拼接。

import java.lang.reflect.Field;

import java.lang.reflect.Method;

public class SqlJoin {

public static void main(String[] args) throws Exception {

Person p1 = new Person();

p1.setName("ai");

p1.setUserName("aiai");

String str = Query(p1);

System.out.println(str);

}

public static String Query(Person person) throws Exception {

StringBuilder sb = new StringBuilder();

//通过反射获取person对象

Class p = person.getClass();

//判断此p是不是Table注解类

boolean exist = p.isAnnotationPresent(Table.class);

if (!exist) {

return null;

}

//如果是，强制转换为Table

Table table = (Table) p.getAnnotation(Table.class);

String tableName = table.value();

//取值拼接SQL语句

sb.append("select \* from ").append(tableName).append(" where 1=1");

//通过p获取对象的所有字段

Field[] fArray = p.getDeclaredFields();

for (Field field : fArray) {

//判断是否存在Column注解类

boolean fExist = field.isAnnotationPresent(Column.class);

if (!fExist) {

return null;

}

Column column = field.getAnnotation(Column.class);

//获取字段上方注解的内容

String columnName = column.value();

//获取字段的名字

String fieldName = field.getName();

Object fieldValue = null;

String getMethodName = "get" + fieldName.substring(0, 1).toUpperCase() + fieldName.substring(1);

Method method = p.getMethod(getMethodName);

fieldValue = method.invoke(person);

sb.append(" and ").append(columnName).append("=").append(fieldValue);

}

return sb.toString();

}

}

（4）执行SqlJoin，输出结果为：

select \* from person where 1=1 and name=ai and user\_name=aiai