由鸿铭
E-mail：hithongming@163.com

【软件构造】第七章第五节 测试与测试优先编程

第七章第五节 测试与测试优先编程

- 确保程序正确性/健壮性的最普遍的手段：测试
  - 设计测试用例
  - 用JUnit写测试程序
  - 自动化测试过程

Outline

- 测试和测试优先编程
- 黑盒测试
  - 等价类划分
  - 边界值分析
- 代码覆盖度
- 用注释形式撰写测试策略
- JUnit测试用例写法

Notes

测试和测试优先编程

【测试的定义】

- 测试：发现程序中的错误 提高程序正确性的信心
- 程序正确确认的基本方法:
  - 形式化推理
  - 代码评审
  - 测试
- 测试是提高软件质量的重要手段
  - 确认是否可达到可用的级别
  - 关注系统某一侧面的质量特性
  - 是否满足需求
  - 是否正确响应所有需求
  - 性能是否可接受
  - 是否可用
  - 可否正确部署安装
  - 是否达到期望

【测试的分类】

- 单元测试

- 集成测试
- 系统测试
- 回归测试
- 验收测试

黑盒测试

- 白盒测试:对程序内部代码结构的测试 只关注代码内部的问题
- 黑盒测试:对程序外部表现出来的行为的测试 采用两个方法
  - 等价划分 将程序可能的输入进行分类 划分为不同集合 包括不合法数据
    - 等价类划分可有两种不同的情况：有效等价类和无效等价类。
    - 若一组对象自反、对称、传递，则为等价类
    - 可产生相似结果的输入集合中的一个可代替整个集合
    - 同理，对输出也可以划分等价类
    - 极端：每个分区只有一个测试用例，覆盖所有分区
  - 边界值分析方法 边界值分析法是对输入输出的边界值进行测试一种黑盒测试方法，是对等价类分析法的补充。
    - 错误通常隐藏在边界中，如一位偏移、边界值需单独处理等
    - 找到有效数据和无效数据的分界点（最大值、最小值），对该分界点以及两边的值分别单独进行测试。
    - 等价类划分法可以挑选等价类范围内任意一个数据作为代表，而边界值分析法要求每个边界值都要作为测试条件。
- 测试困难
  - 软件行为在离散输入空间中差异巨大
    - 大多数正确 少数错误
    - bug出现不遵循特定概率分布
  - 无统计规律可循

代码覆盖度

- 定义：已有的测试用例有多大程度覆盖了被测程序；
- 代码覆盖度越低，测试越不充分；但要做到很高的代码覆盖度，需要更多的测试用例，测试代价高；
- 代码覆盖率高的程序在测试期间执行了更多的源代码，与低代码覆盖率的程序相比，包含未检测到的软件错误的可能性较低
- 基本覆盖标准：函数覆盖；语句覆盖、分支覆盖、

条件覆盖、 路径覆盖

- 测试效果：路径 > 分支 > 语句
- 测试难度：路径 > 分支 > 语句

部分转自 长安蒹葭的博客

以注释的形式撰写测试策略

- “测试策略”通俗来讲就是6个字：“测什么”和“怎么测”。测试策略非常重要，需要在程序中显性记录下来。
- 目的：在代码评审过程中，其他人能够理解你的测试，并评判测试是否充分
- 在测试类的顶端写策略

```
/*
 * Testing strategy
 *
 * Partition the inputs as follows:
 * text.length(): 0, 1, > 1
 * start:         0, 1, 1 < start < text.length(),
 *                text.length() - 1, text.length()
 * text.length()-start: 0, 1, even > 1, odd > 1
 *
 * Include even- and odd-length reversals because
 * only odd has a middle element that doesn't move.
 *
 * Exhaustive Cartesian coverage of partitions.
 */
```

- 在每个测试方法前说明测试用例是如何选择的

```
// covers test.length() = 0,
//        start = 0 = text.length(),
//        text.length()-start = 0
@Test public void testEmpty() {
    assertEquals("", reverseEnd("", 0));
}
```

JUnit 测试用例写法

- Junit单元测试是依据 注释中@Test 之前的方法编写的
- JUnit测试经常调用多次方法，使用 assertEqual || assertTrue || assertFalse 来检查结果
- @Before：准备测试、完成初始化，每个测试方法前执行一次
- @After：清理现场，每个测试方法后执行一次
- @Test：表明测试方法，内含Assert语句
  - 第一个参数是预期结果、第二个参数实施及结果；
  - 如果断言失败，该测试方法直接返回，Junit记录该测试的失败；
  - 一个测试方法失败，其他测试方法仍运行
  - @Test(expected = *.class)：对错误的测试，expected的属性值是一个异常
  - @Test(timeout = xxx)：测试方法在制定的时间之内没有运行完则失败
- @ignore：忽略测试方法
- 栗子：

```
1 public class Calculator {
2     public int evaluate(String expression) {
3         int sum = 0;
4         for (String summand: expression.split("\\+"))
5             sum += Integer.valueOf(summand);
6         return sum;
7     }
8 }
9 ----------------------------------------
10 import static org.junit.Assert.assertEquals;
11 import org.junit.Test;
12
13 public class CalculatorTest {
14 @Test
```

```
15 public void evaluatesExpression() {
16         Calculator calculator = new Calculator();
17         int sum = calculator.evaluate("1+2+3");
18         assertEquals(6, sum);
19     }
20 }
```

```
assertArrayEquals("failure - byte arrays not same", expected, actual);
assertEquals("failure - strings are not equal", "text", "text");
assertFalse("failure - should be false", false);
assertNotNull("should not be null", new Object());
assertNotSame("should not be same Object", new Object(), new Object());
assertNull("should be null", null);
assertSame("should be same", aNumber, aNumber);
assertTrue("failure - should be true", true);

assertThat("albumen", both(containsString("a")).and(containsString("b")));
assertThat(Arrays.asList("one", "two", "three"), hasItems("one", "three"));
assertThat("good", allOf(equalTo("good"), startsWith("good")));
```

查看实际值是否满足指定的条件，条件使用Hamcrest Matchers匹配符进行匹配