第五章节 测试分类

1. 为什么要对软件测试进行分类

软件测试是软件生命周期中的一个重要环节，具有较高的复杂性，对于软件测试，可以从不同的角度加以分类，使开发者在软件开发过程中的不同层次、不同阶段对测试工作进行更好的执行和管理测试的分类方法

2. 按照测试目标分类

2.1 界面测试

软件只是一种工具，软件与人的信息交流是通过界面来进行的，界面是软件与用户交流的最直接的一层，界面的设计决定了用户对我们设计的软件的第一印象；界面如同人的面孔，具有吸引用户的直接优势，设计合理的界面能给用户带来轻松愉悦的感受。

如果不严格按照设计图来进行界面测试，结果可能就会出现小岳岳版本~

界面测试（简称UI测试），指按照界面的需求（一般是UI设计稿）和界面的设计规则，对我们软件界面所展示的全部内容进行测试和检查，一般包括如下内容：

- 验证界面内容显示的完整性，一致性，准确性，友好性。比如界面内容对屏幕大小的自适应，换行，内容是否全部清晰展示；
- 验证整个界面布局和排版是否合理，不同板块字体的设计，图片的展示是否符合需求；
- 对界面不同控件的测试，比如，对话框，文本框，滚动条，选项按钮等是否可以正常使用，有效和
- 无效的状态是否设计合理；
- 界面的布局和色调符合当下时事的发展。

找茬~ 请你根据给定的设计图找出实现的页面有哪些界面问题~

2.2 功能测试

功能测试就是对产品的各功能进行验证，根据功能测试用例，逐项测试，检查产品是否达到用户要求的功能。

根据产品特性、操作描述和用户方案，测试一个产品的特性和可操作行为以确定它们满足设计需求。本地化软件的功能测试，用于验证应用程序或网站对目标用户能正确工作。使用适当的平台、浏览器和测试脚本，以保证目标用户的体验将足够好，就像应用程序是专门为该市场开发的一样。功能测试是为了确保程序以期望的方式运行而按功能要求对软件进行的测试，通过对一个系统的所有的特性和功能都进行测试确保符合需求和规范。

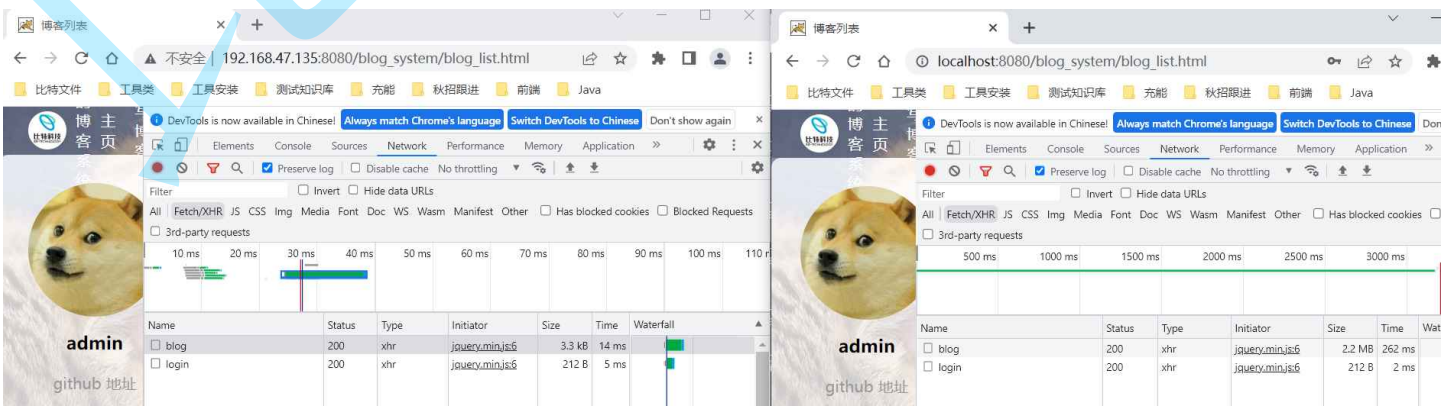
如何进行功能测试?

设计功能测试用例，参考产品规格说明书进行用例的编写，具体的测试用例需要使用黑盒设计测试用例的方法，如等价类、边界值、判定表法、正交法、场景法、错误猜测法等。

2.3 性能测试

我们在使用软件的时候有时会碰到软件网页打开时越来越慢，查询数据时很长时间才显示列表，软件运行越来越慢等问题，这些问题都是系统的性能问题引起的。

以博客系统为例，看看两种场景下哪种性能比较好?



要进行软件产品的性能问题，要对产品的性能需求进行分析，然后基于系统的性能需求和系统架构，完成性能测试的设计和执行，最后要进行持续的性能调优。

2.4 可靠性测试

可靠性（Availability）即可用性，是指系统正常运行的能力或者程度，一般用正常向用户提供软件服务 的时间占总时间的百分比表示。

可靠性 = 正常运行时间/（正常运行时间+非正常运行时间）*100%

隔壁村有个人叫老王

让老王请吃饭，要求了十次，但是他只请了一次，那么我们说老王的可靠性是10%，那么我们称老王这人不可靠

如果让老王请吃饭，要求了十次，每次他都请客了，可靠性是100%，那么老王是个可靠的人（纯纯大冤种~~）

系统非正常运行的时间可能是由于硬件，软件，网络故障或任何其他因素（如断电）造成的，这些因素 能让系统停止工作，或者连接中断不能被访问，或者性能急剧降低导致不能使用软件现有的服务等。

可用性指标一般要求达到4个或5个“9”，即99.99%或者99.999%

如果可用性达到99.99%，对于一个全年不间断（7*24的方式）运行的系统，意味着全年（252600min)不能

正常工作的时间只有52min，不到一个小时。

如果可用性达到99.999%，意味着全年不能正常工作的时间只有5min。

不同的应用系统，可用性的要求是不一样的，非实时性的信息系统或一般网站要求都很低，99%和99.5%就可以了，但是军事系统，要求则很高；

2.5 安全性测试

安全性是指信息安全，是指计算机系统或网络保护用户数据隐私，完整，保护数据正常传输和抵御黑客，病毒攻击的能力。

安全性测试属于非功能性测试很重要的一个方面，系统常见的安全漏洞和威胁如下

- 输入域，如输入恶性或者带有病毒的脚本或长字符串；
- 代码中的安全性问题，如SQL/XML注入
- 不安全的数据存储或者传递
- 数据文件，邮件文件，系统配置文件等里面有危害系统的信息或者数据；
- 有问题的访问控制，权限分配等
- 假冒ID：身份欺骗
- 篡改，对数据的恶意修改，破坏数据的完整性

安全性测试的方法有代码评审，渗透测试，安全运维等，常用的静态安全测试工具有，Coverity，IBM Appscan Source，HPFortify，常用的动态安全测试有OWASP的ZAP，HP WebInspect等。其中静态安全测试是常用的安全性测试的方法。

安全如此重要，咱们的私人电脑需要安装杀毒软件吗.....（铁汁多少有点抬举自己了嗷~）

2.6 易用性测试

许多产品都应用人体工程学的研究成果，是产品在使用起来更加灵活和，舒适。软件产品也始终关注用

户体验，让用户获得舒适，易用的体验，针对软件这方面的测试称之为易用性测试。

易用性在ISO25020标准中指容易发现，容易学习和容易使用。易用性包含七个要素：符合标准和规范，直观性，一致性，灵活性，舒适性，正确性和实用性。我们主要讨论以下几个方面

1，标准性和规范性

对于现有的软件运行平台，通常其UI标准已经不知不觉地被确立了，成为大家的共识。多数用户已经习惯并且接受了这些标准和规范，或者说已经认同了这些信息所代表的的含义。比如安装软件的界面的外观，在什么场合使用恰当的对话框等。

所以用户界面上的各中信息应该符合规范和习惯，否则用户使用起来会不舒适，并得不到用户的认可。

测试人员需要把与标准规范，习惯不一致的问题报告为缺陷

2，直观性

用户界面的直观性，要求软件功能特性易懂，清晰。用户界面布局合理，对操作的响应在用户的预期之中。比如数据统计结果用报表的形式（条形图，扇形图等）展示清晰直观；现在主流的很多搜索引擎和日历的设计也有直观性的特点；

3，灵活性

软件可以有不同的选项以满足不同使用习惯的用户来完成相同的功能。但是灵活性的设计要把握好度，

不然可能由于太多的用户状态和方式的选择，增加了软件设计的复杂性，和程序实现的难度。 例如手机键盘有九宫格和全键盘，还支持手写，满足了不同用户的需求

4，舒适性

舒适性主要强调界面友好，美观，操作过程顺畅，色彩用运恰当，按钮的立体感等。

3. 按照执行方式分类

3.1 静态测试

所谓静态测试（static testing）就是**不实际运行**被测软件，而只是**静态地检查程序代码、界面或文档**中可能存在的错误的过程。

不以测试数据的执行而是**对测试对象的分析过程**，仅通过分析或检查源程序的设计、内部结构、逻辑、代码风格和规格等来检查程序的正确性。

常见的静态测试方式有代码走查，代码扫描工具等。

3.2 动态测试

动态测试（dynamic testing），指的是**实际运行被测程序，输入相应的测试数据，检查实际输出结果和预期结果是否相符**的过程，所以判断一个测试属于动态测试还是静态的，唯一的标准就是看是否运行程序。

大多数软件测试工作都属于动态测试

4. 按照测试方法

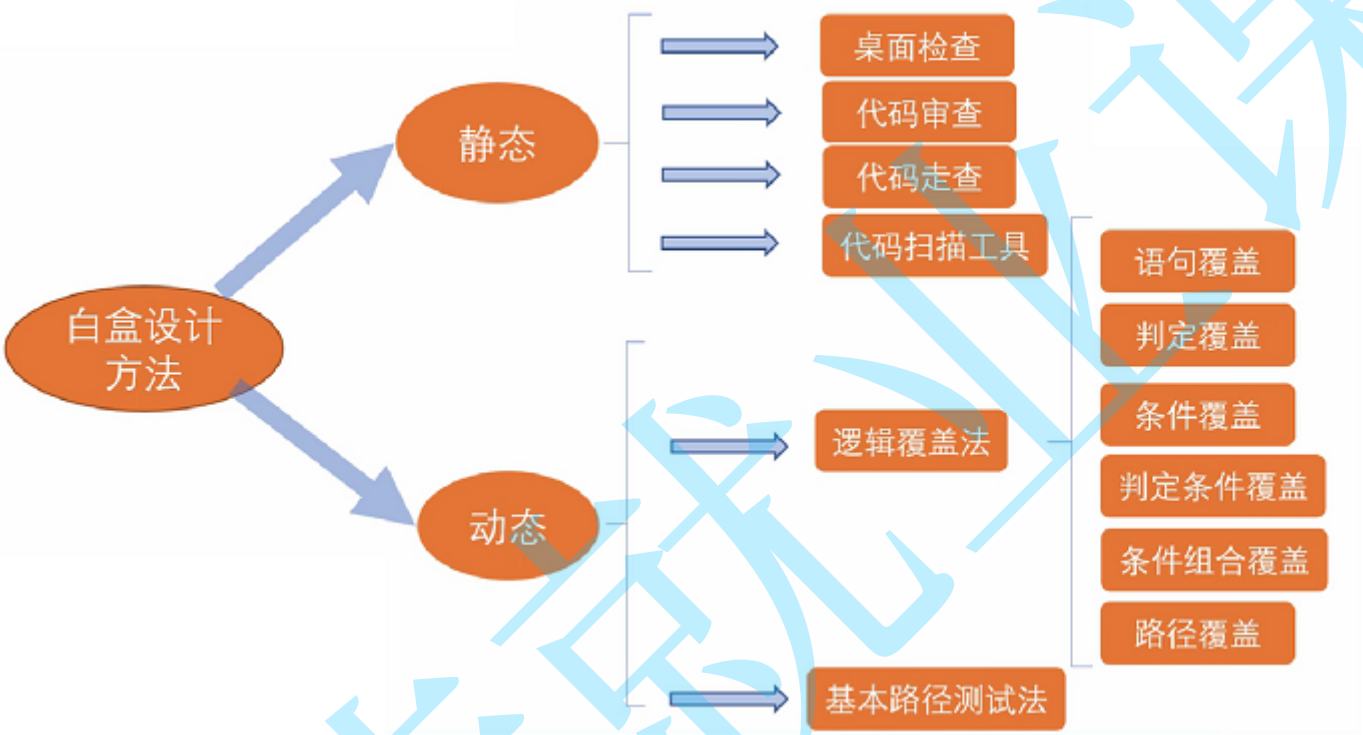
4.1 白盒测试

白盒测试又称为结构测试或逻辑测试，它一般用来分析程序的内部结构，针对程序的逻辑结构来设计测

试用例进行测试。

白盒测试的测试目的是，通过检查软件内部的逻辑结构，对软件中的逻辑路径进行覆盖测试；在程序不

同地方设立检查点，检查程序的状态，以确定实际运行状态与预期状态是否一致。



白盒测试主要分为静态测试和动态测试两种。静态测试常见于桌面检查、代码审查、代码走查、代码扫描工具

动态测试方法主要包含六种测试方法：**语句覆盖、判定覆盖、条件覆盖、判定条件覆盖、条件组合覆盖、路径覆盖**。

给出简单的案例，接下来了解一下白盒测试方法的概念和使用

```
if   A  and  B
     then  action1
if   c  and  D
     then  action2
```

4.1.1 语句覆盖

每个语句至少执行一次。

针对A and B：A为T且B为T

针对C or D：C为T或者D为T

得出用例：

用例1：A为T，B为T，C为T，D为F

4.1.2 判定覆盖

A and B 要为T => A=T B=T ①

A and B 要为F => A=T B=F 或者A=F B=T或者 A=F B=F ②

C or D 要为T => C=T D=T/F 或者 C=T/F D=T ③

C or D 要为F => C=F D=F ④

得出用例：

用例1：A=T B=T C=T D=F 满足 ①③

用例2：A=T B=F C=F D=F 满足 ②④

4.1.3 条件覆盖

A： T F

B： T F

C： T F

D： T F

⑤ ⑥

得出用例：

用例1：A=T B=T C=T D=T ⑤

用例2：A=F B=F C=F D=F ⑥

4.1.4 判定条件覆盖

结合判定覆盖和条件覆盖。

得出用例：

用例1：A=T B=T C=T D=T 满足①③⑤

用例2：A=F B=F C=F D=F 满足②④⑥

4.1.5 条件组合覆盖

A B | C D

T T | T T

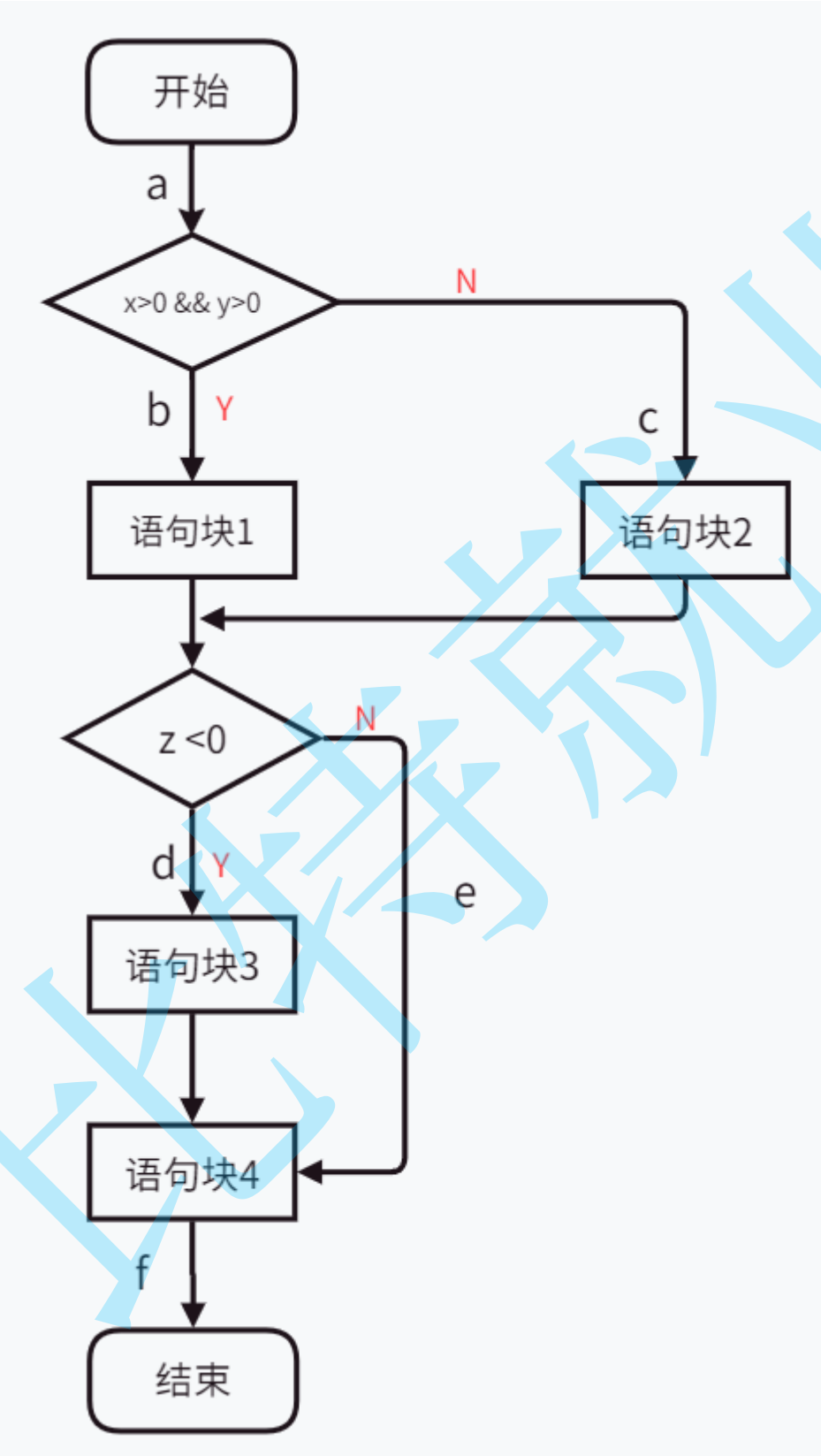
T F | T F

F T | F T

F F | F F

每行就可以是一个用例，一共四个用例。

4.1.6 路径覆盖



判定条件定义如下：

（1）if(x>0 && y>0)判定：记为P1

（2）if(z < 0)判定：记为P2

（3）x > 0：记为C1

（4）y > 0：记为C2

（5）z < 0：记为C3

测试用例设计：

数据	C1	C2	C3	P1	P2	路径
{x=3,y=3,z=-2}	T	T	T	T	T	a-b-d-f
{x=-3,y=3,z=-2}	F	T	T	F	T	a-c-d-f
{x=3,y=3,z=2}	T	T	F	T	F	a-b-e-f
{x=-3,y=15,z= 2}	F	T	F	F	F	a-c-e-f

总结：

- 白盒测试主要应用于单元测试阶段
- 先执行静态设计用例的方法，再执行动态设计测试用例的方法
- 设计用例一般使用路径测试，重点模块追加使用逻辑覆盖方法

4.2 黑盒测试

黑盒测试就是在完全不考虑程序逻辑和内部结构的情况下，检查系统功能是否按照需求规格说明书的规 定正常使用、是否能适当的接收输入数据而输出正确的结果，满足规范需求。

所以，黑盒测试又称之为数据驱动测试，只注重软件的功能

黑盒测试的优点

不需要了解程序内部的代码以及实现，不关注软件内部的实现。

从用户角度出发设计测试用例，很容易的知道用户会用到哪些功能，会遇到哪些问题，锻炼测试人 员的产品思维

测试用例是基于软件需求开发文档，不容易遗漏软件需求文档中需要测试的功能。

黑盒测试的缺点是不可能覆盖所有代码。

黑盒测试用到的测试方法有，等价类，边界值，因果图，场景法，错误猜测法等

4.3 灰盒测试

灰盒测试，是介于白盒测试与黑盒测试之间的一种测试，灰盒测试多用于集成测试阶段，不仅关注输出、输入的正确性，同时也关注程序内部的情况。

但是，灰盒测试没有白盒测试详细和完整，黑盒测试是覆盖产品范围最广的测试，因此灰盒测试基本是不能够替代黑盒测试，否则需要很大的代价，设计非常多的用例。

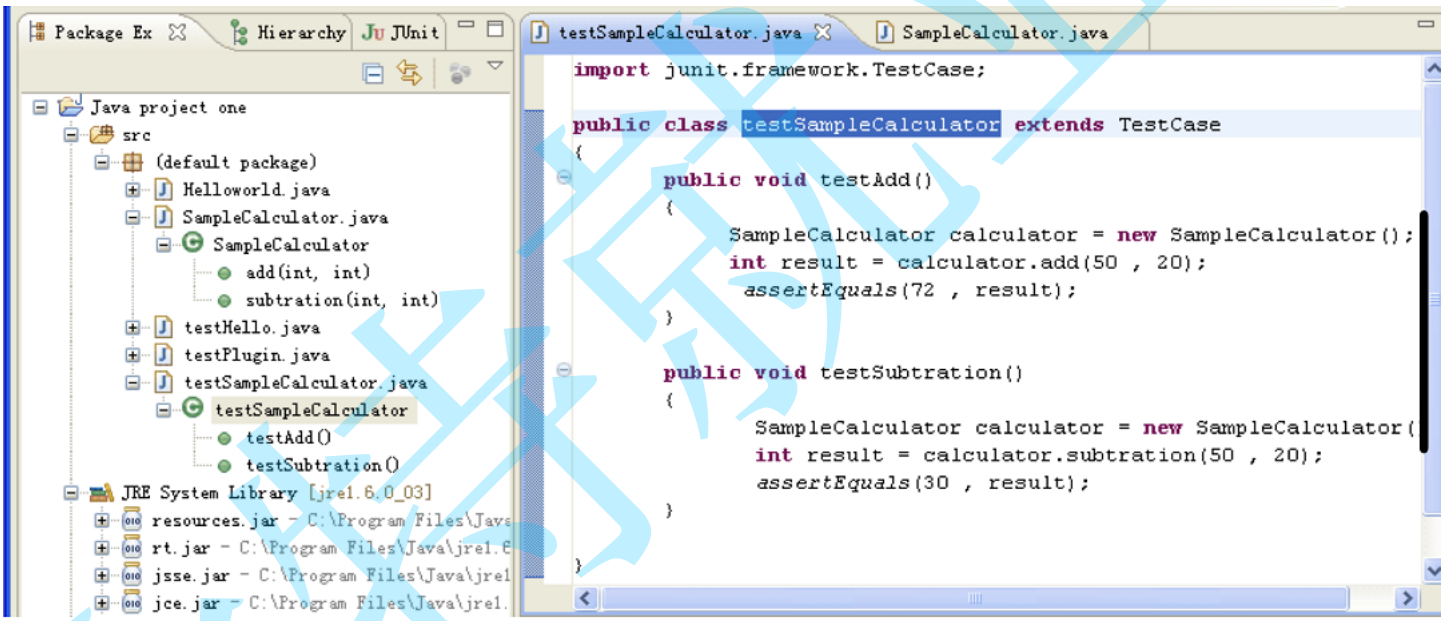
常见面试题：你知道的测试方法有哪些？哪种用的比较多？

常见的测试方法有黑盒测试，白盒测试和灰盒测试。开发人员主要用白盒测试和灰盒测试，测试人员主要用白盒测试和黑盒测试。对于测试人员来说，相较于白盒测试，黑盒测试用的更多一些。

5. 按照测试阶段分类

5.1 单元测试

与编码同步进行，针对软件最小组成单元进行测试，主要采用白盒测试方法，从被测对象的内部结构出发设计测试用例



到底怎么才算“最小单元”呢，最小单元实际是认为定义的，一个方法，一个类都可以理解为“最小单元”。

- 测试阶段：编码后或者编码前（TDD）
- 测试对象：最小模块
- 测试人员：白盒测试工程师或开发工程师
- 测试依据：代码和注释+详细设计文档
- 测试方法：白盒测试
- 测试内容：模块接口测试、局部数据结构测试、路径测试、错误处理测试、边界测试

针对上面给出的冒泡排序，我们尝试实现一个简单的单元测试。

```java
public class Main {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n; i++) {
            // 每轮遍历将最大的数移到末尾
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j+1]) {
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }
    public static void main(String[] args) {
        // 排序无序数组
        Test01();
        // 排序有序数组
        Test02();
        // 排序空数组
        Test03();
        // 排序有重复数组数组
        Test04();
    }

    private static void Test04() {
        int[] act_array1= {1, 1, 29, 12, 12, 9, 9};
        int[] expect_array1 = {1, 1, 9, 9, 12, 12, 29};
        //排序无序数组
        bubbleSort(act_array1);
        boolean isSame = Arrays.equals(act_array1, expect_array1);//判断两个数组内容是不是一样
        if(isSame == false) {
            System.out.println("测试不通过");
        } else {
            System.out.println("测试通过");
        }
    }

    private static void Test03() {
        int[] act_array1= {};
        int[] expect_array1 = {};
        //排序无序数组
        bubbleSort(act_array1);
```

```
44          boolean isSame = Arrays.equals(act_array1, expect_array1);//判断两个数组
    内容是不是一样
45          if(isSame == false) {
46              System.out.println("测试不通过");
47          } else {
48              System.out.println("测试通过");
49          }
50      }
51
52      private static void Test02() {
53          int[] act_array1= {1, 2, 3, 4, 5};
54          int[] expect_array1 = {1, 2, 3, 4, 5};
55          //排序无序数组
56          bubbleSort(act_array1);
57          boolean isSame = Arrays.equals(act_array1, expect_array1);//判断两个数组
    内容是不是一样
58          if(isSame == false) {
59              System.out.println("测试不通过");
60          } else {
61              System.out.println("测试通过");
62          }
63      }
64
65      private static void Test01() {
66          int[] act_array1= {64, 34, 25, 12, 22, 11, 90};
67          int[] expect_array1 = {11, 12, 22, 25, 34, 64, 90};
68          //排序无序数组
69          bubbleSort(act_array1);
70          boolean isSame = Arrays.equals(act_array1, expect_array1);//判断两个数组
    内容是不是一样
71          if(isSame == false) {
72              System.out.println("测试不通过");
73          } else {
74              System.out.println("测试通过");
75          }
76      }
77  }
```

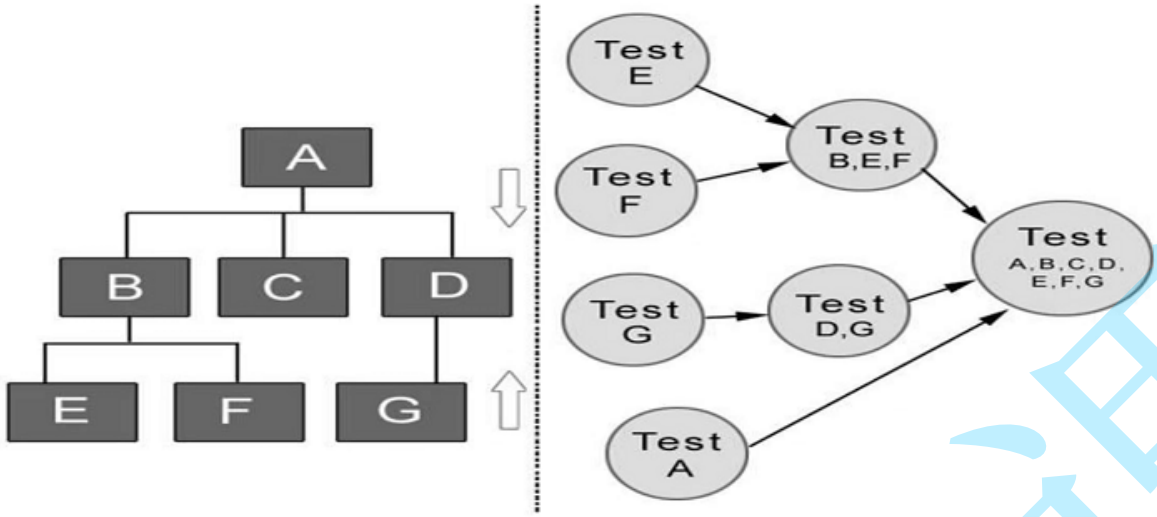
java中也有很多单元测试框架，如Junit，Junit提供了非常多注解和断言函数，有效提升开发单元测试脚本的效率，感兴趣的同学课下去探索~

```
1  import org.junit.jupiter.api.Test;
2  import static org.junit.jupiter.api.Assertions.assertArrayEquals;
3
4  public class BubbleSortTest {
```

```
5
6   @Test
7   public void testBubbleSort() {
8     // 测试用例：正常情况下的冒泡排序
9     int[] arr = {5, 3, 9, 1, 7};
10    int[] expected = {1, 3, 5, 7, 9};
11    BubbleSort.bubbleSort(arr);
12    assertArrayEquals(expected, arr);
13  }
14
15  @Test
16  public void testBubbleSortEmptyArray() {
17    // 测试用例：空数组的冒泡排序
18    int[] arr = {};
19    int[] expected = {};
20    BubbleSort.bubbleSort(arr);
21    assertArrayEquals(expected, arr);
22  }
23
24  @Test
25  public void testBubbleSortAlreadySorted() {
26    // 测试用例：已经有序的数组，排序后应该保持不变
27    int[] arr = {1, 2, 3, 4, 5};
28    int[] expected = {1, 2, 3, 4, 5};
29    BubbleSort.bubbleSort(arr);
30    assertArrayEquals(expected, arr);
31  }
32
33  @Test
34  public void testBubbleSortWithDuplicates() {
35    // 测试用例：包含重复元素的数组
36    int[] arr = {4, 2, 4, 1, 3, 2};
37    int[] expected = {1, 2, 2, 3, 4, 4};
38    BubbleSort.bubbleSort(arr);
39    assertArrayEquals(expected, arr);
40  }
41 }
```

5.2 集成测试

集成测试也称联合测试（联调）、组装测试，将程序模块采用适当的集成策略组装起来，对系统的接口 及集成后的功能进行正确性检测的测试工作。集成主要目的是检查软件单位之间的接口是否正确。



- 测试阶段：一般单元测试之后进行
- 测试对象：模块间的接口
- 测试人员：白盒测试工程师或开发工程师
- 测试依据：单元测试的模块+概要设计文档
- 测试方法：黑盒测试与白盒测试相结合
- 测试内容：模块之间数据传输、模块之间功能冲突、模块组装功能正确性、全局数据结构、单模块缺陷对系统的影响

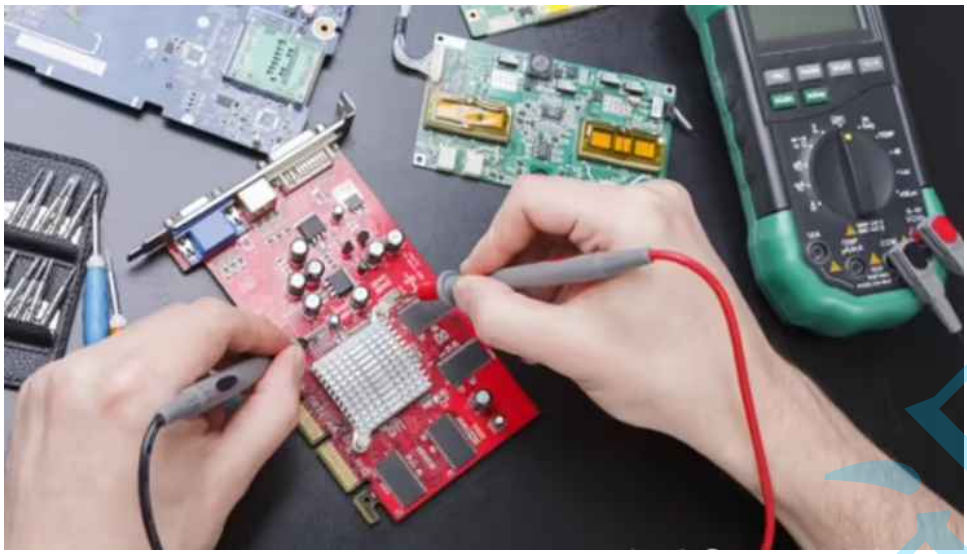
5.3 系统测试

对通过集成测试的系统进行整体测试，验证系统功能性和非功能性需求的实现。

- 测试阶段：集成测试通过之后
- 测试对象：整个系统（软、硬件）
- 测试人员：黑盒测试工程师
- 测试依据：需求规格说明文档
- 测试方法：黑盒测试
- 测试内容：功能、界面、可靠性、易用性、性能、兼容性、安全性等

5.3.1 冒烟测试

这一术语源自硬件行业。对一个硬件或硬件组件进行更改或修复后，直接给设备加电。如果没有冒烟，则该组件就通过了测试。在软件中，“冒烟测试”这一术语描述的是在将代码更改嵌入到产品的源树中之前对这些更改进行验证的过程。在检查了代码后，冒烟测试是确定和修复软件缺陷的最经济有效的方法。冒烟测试设计用于确认代码中的更改会按预期运行，且不会破坏整个版本的稳定性。



冒烟测试的对象是每一个新编译的需要正式测试的软件版本，目的是确认软件主要功能和核心流程正常，在正式进行系统测试之前执行。冒烟测试一般在开发人员开发完毕后提交给测试人员来进行测试时，先进行冒烟测试，保证基本功能正常，不阻碍后续的测试。

如果冒烟测试通过，则测试人员开始进行正式的系统测试，如果不通过，则测试人员可以让开发人员重新修复代码直到冒烟测试通过，再开始进行系统测试。

在生活中，

购买一个电视，首先会通电，查看电视是否能够运行。

购买一个水杯，首先会灌水，查看水杯是否漏水。

在工作中，假如有一个博客系统项目提测了，冒烟测试即只需要测试系统是否能够成功打开，主流程是否可以走通即可。

5.3.2 回归测试

回归测试是指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误。

在整个软件测试过程中占有很大的工作量比重，软件开发的各个阶段都会进行多次回归测试。随着系统的庞大，回归测试的成本越来越大，通过选择正确的回归测试策略来改进回归测试的效率和有效性是很有意义的。

回归测试主要由人工测试和自动化测试进行。

在实际工作中，回归测试需要反复进行，当测试者一次又一次地完成相同的测试时，这些回归测试将变得非常令人厌烦，而在大多数回归测试需要手工完成的时候尤其如此，因此，需要通过自动测试来实现重复的和一致的回归测试。通过测试自动化可以提高回归测试效率。为了支持多种回归测试策略，自动测试工具应该是通用的和灵活的，以便满足达到不同回归测试目标的要求。

5.4 验收测试

针对用户需求，对通过系统测试的软件进行交付性测试，以确定系统是否满足验收标准，由用户或其他授权机构决定是否接受系统。验收测试是部署软件之前的最后一个测试操作。它是技术测试的

最后一个阶段，也称为**交付测试**。验收测试的目的是确保软件准备就绪，按照项目合同、任务书、双方约定的验收依据文档，向软件购买都展示该软件系统满足原始需求。

- 测试阶段：系统测试通过之后
- 测试对象：整个系统（包括软硬件）。
- 测试人员：主要是最终用户或者需求方。
- 测试依据：用户需求、验收标准
- 测试方法：黑盒测试
- 测试内容：同系统测试(功能...各类文档等)

单元测试，集成测试，系统测试，回归测试之间的关系

关于车企生产车到客户开上小汽车的过程中~

造车需要原材料，如车轮、发动机等零部件不是车企自己制造出来的，而是通过购买零部件来造车。

对买来的零部件进行检查，零部件是否符合造车标准（**单元测试**）

零件确认完毕，接下来就是复杂的造车工艺，将零部件集成起来构成了一辆车，并初步检查拼车的车是否能正常运作（**集成测试**）

一辆车成型之后并不意味着就可以直接销售给客户了，需要车企专业的测试人员进行详细而完整的测试。（**系统测试**）

专业的测试人员对企业测试完毕，通过测试的汽车将会在车展或者4S店进行展示，供用户进行选择和购买。用户在选择汽车的过程中也会对车外观以及性能等方面进行校验（验收**测试**）

除了以上阶段外，还有两个非常重要，在工作中经常会听到：冒烟测试和回归测试

6. 按照是否手工测试

6.1 手工测试(Manual testing)

手工测试就是由人去一个一个的输入用例，然后观察结果，和机器测试相对应，属于比较原始但是必须的一个步骤。

6.2 自动化测试(Automation Testing)

就是在预设条件下运行系统或应用程序，评估运行结果，预先条件应包括正常条件和异常条件。简单说 自动化测试是把以人为驱动的测试行为转化为机器执行的一种过程。 自动化测试比如功能测试自动化、性能测试自动化、安全测试自动化。 自动化测试按照测试对象来分，还可以分为接口测试、UI测

试等。接口测试的ROI（产出投入比）要比UI测试高。（这里了解一下，等到将自动化的时候再详细展开）

6.3 自动化测试和手工测试优缺点

	自动化测试	手工测试
优点	• 节省成本 • 提高测试人员执行工作效率 • 保障软件的质量	• 对测试人员技术要求没有自动化技术要求高 • 可以进行发散性测试
缺点	• 对测试人员技术要求较高 • 不能发散性测试	• 效率低 • 人员、时间成本比起自动化测试都比较高

7. 按照实施组织划分

大型通用软件，在正式发布前，通常需要执行Alpha和Beta测试

7.1 α测试(Alpha Testing)

α测试又叫内测或者叫a测，其实都是一个涵义

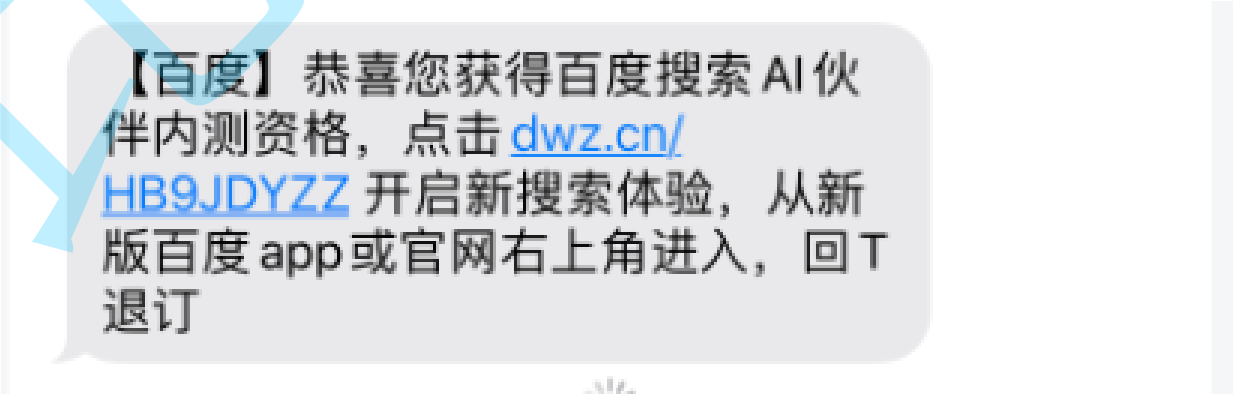
α测试通常是公司内部的用户在模拟实际操作环境下进行的测试。α测试的目的是评价软件产品的FLURPS(即功能、可使用性、可靠性、性能和支持)。 α测试不能由程序员或测试员完成。

7.2 β测试(Beta Testing)

β测试又叫公测或者叫b测

β测试由软件的最终用户们在一个或多个场所进行，这里就可以理解为，β测试是正式用户中的一部分，他们在任意的场合来使用软件，目的是为了发现软件是否存在一系列的问题

通常会发送一些邀请码，来邀请用户参与项目测试



α测试与β测试的区别

测试的场所不同	α测试是在公司内部进行测试的,但是β测试是在用户环境下进行测试的 α测试的环境是受开发方控制的,用户的数量相对比较少,时间比较集中 β测试的环境是不受开发方控制的,用户数量相对比较多,时间不集中
测试执行时机不	通常是α测试通过后，在进行β测试
测试持续时间长短不同	α测试时间没有β测试持续时间长

举个例子：

7.3 第三方测试

第三方软件测试是指由独立的第三方公司或组织进行的软件测试活动。

https://www.zmtests.com/

通过第三方测试，可以确保软件的质量，节省成本，确保软件尽快上线

8. 按照测试地域划分

按照测试地域划分，一般会将测试划分为国际化测试和本地测试

8.1 国际化测试

简言之，测试人员需要测试软件在不同语言和地区是否能正常工作

墨西哥

中国

国际化测试需要关注软件的哪些特性：

- 布局
- 时间
- 日期

- 数字格式
- 货币
- 机器型号
- .....

8.2 本地测试

之前所讲的都属于本地测试