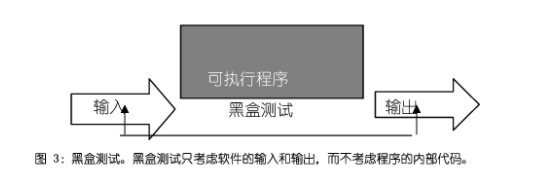
# 黑盒测试

## 定义：

黑盒测试(也称为功能测试)是忽略系统或组件的内部机制，只关注响应选定的输入和执行条件而生成的输出的测试。



黑盒测试也称为功能测试和行为测试，侧重于根据程序的功能需求确定程序是否完成了它应该做的事情。

黑盒测试试图在以下类别的代码的外部行为中发现错误

(1) 不正确或缺少功能;

(2)接口错误;

(3) 接口使用的数据结构错误;

(4) 行为或表现错误;

(5) 初始化和终止错误。

通过此测试，我们可以确定功能是否按照规范工作。但是，重要的是要注意，再多的测试都不能明确地证

## 优缺点

### 黑盒测试的优点：

适用于功能测试、可用性测试及可接受性测试;对照说明书测试程序功能;可测试长的、复杂的程序的工作逻辑，易被理解。[5] 

### 黑盒测试的缺点：

不可能进行完全的、毫无遗漏的输入测试，有一些软件Bug或人为设置的故障通过黑盒测试是无法检测出来的。正是因为黑盒测试的测试数据来自规格说明书，这一方法的主要缺点是它依赖于规格说明书的正确性。实际上，人们并不能保证规格说明书完全正确。如在规格说明书中规定了多余的功能，或是漏掉了某些功能，这对于黑盒测试来说是完全无能为力的。[5]

## 测试方法

从理论上讲，黑盒测试只有采用穷举输入测试，把所有可能的输入都作为测试情况考虑，才能查出[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F)中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但可能的输入进行测试。这样看来，完全测试是不可能的，所以我们要进行有针对性的测试，通过制定测试案例指导测试的实施，保证[软件测试](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E6%B5%8B%E8%AF%95)有组织、按步骤，以及有计划地进行。黑盒测试行为必须能够加以量化，才能真正保证[软件质量](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E8%B4%A8%E9%87%8F)，而[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)就是将测试行为具体量化的方法之一。具体的黑盒[测试用例设计](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B%E8%AE%BE%E8%AE%A1)方法包括等价类划分法、[边界值分析法](https://baike.baidu.com/item/%E8%BE%B9%E7%95%8C%E5%80%BC%E5%88%86%E6%9E%90%E6%B3%95/4137943)、[错误推测法](https://baike.baidu.com/item/%E9%94%99%E8%AF%AF%E6%8E%A8%E6%B5%8B%E6%B3%95/788810)、[因果图法](https://baike.baidu.com/item/%E5%9B%A0%E6%9E%9C%E5%9B%BE%E6%B3%95)、判定[表驱动](https://baike.baidu.com/item/%E8%A1%A8%E9%A9%B1%E5%8A%A8)法、正交试验设计法、功能图法、[场景](https://baike.baidu.com/item/%E5%9C%BA%E6%99%AF)法等。

[等价类划分](https://baike.baidu.com/item/%E7%AD%89%E4%BB%B7%E7%B1%BB%E5%88%92%E5%88%86/4219313)的办法是把[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F)的输入域划分成若干部分（子集），然后从每个部分中选取少数代表性数据作为测试[用例](https://baike.baidu.com/item/%E7%94%A8%E4%BE%8B" \t "_blank)。每一类的代表性数据在测试中的作用等价于这一类中的其他值。该方法是一种重要的，常用的黑盒[测试用例设计](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B%E8%AE%BE%E8%AE%A1)方法。

**划分等价类**

等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F)中的错误都是等效的，并合理地假定：测试某等价类的代表值就等于对这一类其它值的测试。因此，可以把全部输入数据合理划分为若干等价类，在每一个等价类中取一个数据作为测试的输入条件，就可以用少量代表性的测试数据.取得较好的测试结果.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。

有效等价类：是指对于[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F" \t "_blank)的规格说明来说是合理的，有意义的输入数据构成的集合.利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。

[无效等价类](https://baike.baidu.com/item/%E6%97%A0%E6%95%88%E7%AD%89%E4%BB%B7%E7%B1%BB)：与有效等价类的定义恰巧相反。

设计[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B" \t "_blank)时，要同时考虑这两种等价类.因为，软件不仅要能接收合理的数据，也要能经受意外的考验.这样的测试才能确保软件具有更高的可靠性。

划分等价类的方法：下面给出六条确定等价类的原则。

①在输入条件规定了取值范围或值的个数的情况下，则可以确立一个有效等价类和两个无效等价类。[2]

②在输入条件规定了输入值的集合或者规定了“必须如何”的条件的情况下，可确立一个有效等价类和一个[无效等价类](https://baike.baidu.com/item/%E6%97%A0%E6%95%88%E7%AD%89%E4%BB%B7%E7%B1%BB" \t "_blank)。

③在输入条件是一个[布尔量](https://baike.baidu.com/item/%E5%B8%83%E5%B0%94%E9%87%8F)的情况下，可确定一个有效等价类和一个无效等价类。

④在规定了输入数据的一组值（假定n个），并且[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F" \t "_blank)要对每一个输入值分别处理的情况下，可确立n个有效等价类和一个无效等价类。

⑤在规定了输入数据必须遵守的规则的情况下，可确立一个有效等价类（符合规则）和若干个[无效等价类](https://baike.baidu.com/item/%E6%97%A0%E6%95%88%E7%AD%89%E4%BB%B7%E7%B1%BB" \t "_blank)（从不同角度违反规则）。

⑥在确知已划分的等价类中各元素在[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F)处理中的方式不同的情况下，则应再将该等价类进一步的划分为更小的等价类。

**边界值分析法**

边界值分析是通过选择等价类边界的[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B" \t "_blank)。边界值分析法不仅重视输入条件边界，而且也必须考虑输出域边界。它是对等价类划分方法的补充。

（1）边界值分析方法的考虑：

长期的测试工作经验告诉我们，大量的错误是发生在输入或输出范围的边界上，而不是发生在输入输出范围的内部.因此针对各种边界情况设计[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B" \t "_blank)，可以查出更多的错误。

使用边界值分析方法设计[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)，首先应确定边界情况.通常输入和输出等价类的边界，就是应着重测试的边界情况.应当选取正好等于，刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值作为测试数据。

（2）基于边界值分析方法选择[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B" \t "_blank)的原则：

1）如果输入条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。

2）如果输入条件规定了值的个数，则用最大个数，最小个数，比最小个数少一，比最大个数多一的数作为测试数据。

3）根据规格说明的每个输出条件，使用前面的原则1）。

4）根据规格说明的每个输出条件，应用前面的原则2）。

5）如果[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F" \t "_blank)的规格说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)。

6）如果[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F" \t "_blank)中使用了一个内部[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84)，则应当选择这个内部数据结构的边界上的值作为[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)。

7）分析规格说明，找出其它可能的边界条件。

**错误推测法**

错误推测法是基于经验和直觉推测[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F)中所有可能存在的各种错误，从而有针对性的设计[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)的方法。

错误推测方法的基本思想： 列举出[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F)中所有可能有的错误和容易发生错误的特殊情况，根据他们选择[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)。 例如，在[单元测试](https://baike.baidu.com/item/%E5%8D%95%E5%85%83%E6%B5%8B%E8%AF%95" \t "_blank)时曾列出的许多在模块中常见的错误。以前产品测试中曾经发现的错误等，这些就是经验的总结。还有，输入数据和输出数据为0的情况。 输入表格为空格或输入表格只有一行. 这些都是容易发生错误的情况。可选择这些情况下的例子作为测试用例。

**因果图法**

前面介绍的等价类划分方法和边界值分析方法，都是着重考虑输入条件，但未考虑输入条件之间的联系，相互组合等。 考虑输入条件之间的相互组合，可能会产生一些新的情况。但要检查输入条件的组合不是一件容易的事情，即使把所有输入条件划分成等价类，他们之间的组合情况也相当多。因此必须考虑采用一种适合于描述对于多种条件的组合，相应产生多个动作的形式来考虑设计[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B" \t "_blank)，这就需要利用[因果图](https://baike.baidu.com/item/%E5%9B%A0%E6%9E%9C%E5%9B%BE)（逻辑模型）。

[因果图](https://baike.baidu.com/item/%E5%9B%A0%E6%9E%9C%E5%9B%BE)方法最终生成的就是判定表。它适合于检查[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F)输入条件的各种组合情况。

生成测试用例：

(1) 分析软件规格说明描述中，哪些是原因（即输入条件或输入条件的等价类），哪些是结果（即输出条件），并给每个原因和结果赋予一个标识符。

(2) 分析软件规格说明描述中的语义。找出原因与结果之间，原因与原因之间对应的关系. 根据这些关系，画出[因果图](https://baike.baidu.com/item/%E5%9B%A0%E6%9E%9C%E5%9B%BE" \t "_blank)。

(3) 由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现. 为表明这些特殊情况，在[因果图](https://baike.baidu.com/item/%E5%9B%A0%E6%9E%9C%E5%9B%BE" \t "_blank)上用一些记号标明约束或限制条件。

(4) 把[因果图](https://baike.baidu.com/item/%E5%9B%A0%E6%9E%9C%E5%9B%BE" \t "_blank)转换为[判定表](https://baike.baidu.com/item/%E5%88%A4%E5%AE%9A%E8%A1%A8)。

(5) 把[判定表](https://baike.baidu.com/item/%E5%88%A4%E5%AE%9A%E8%A1%A8" \t "_blank)的每一列拿出来作为依据，设计[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)。

从[因果图](https://baike.baidu.com/item/%E5%9B%A0%E6%9E%9C%E5%9B%BE" \t "_blank)生成的[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)（局部，组合关系下的）包括了所有输入数据的取TRUE与取FALSE的情况，构成的测试用例数目达到最少，且测试用例数目随输入数据数目的增加而线性地增加。

前面[因果图](https://baike.baidu.com/item/%E5%9B%A0%E6%9E%9C%E5%9B%BE" \t "_blank)方法中已经用到了[判定表](https://baike.baidu.com/item/%E5%88%A4%E5%AE%9A%E8%A1%A8)。判定表（Decision Table）是分析和表达多逻辑条件下执行不同操作的情况下的工具.在程序设计发展的初期，判定表就已被当作编写程序的[辅助工具](https://baike.baidu.com/item/%E8%BE%85%E5%8A%A9%E5%B7%A5%E5%85%B7" \t "_blank)了.由于它可以把复杂的逻辑关系和多种条件组合的情况表达得既具体又明确。

**判定表组成法**

条件桩（Condition Stub）：列出了问题的所有条件.通常认为列出的条件的次序无关紧要。

动作桩（Action Stub）：列出了问题规定可能采取的操作.这些操作的排列顺序没有约束。

条件项（Condition Entry）：列出针对它左列条件的取值.在所有可能情况下的真假值。

动作项（Action Entry）：列出在条件项的各种取值情况下应该采取的动作。

规则：任何一个条件组合的特定取值及其相应要执行的操作.在[判定表](https://baike.baidu.com/item/%E5%88%A4%E5%AE%9A%E8%A1%A8" \t "_blank)中贯穿条件项和动作项的一列就是一条规则.显然，判定表中列出多少组条件取值，也就有多少条规则，既条件项和动作项有多少列。

判定表的建立步骤：

①确定规则的个数。假如有n个条件.每个条件有两个取值（0，1），故有2n种规则。

②列出所有的条件桩和动作桩。

③填入条件项。

④填入动作项.等到初始判定表。

⑤简化.合并相似规则（相同动作）。

B. Beizer 指出了适合使用[判定表](https://baike.baidu.com/item/%E5%88%A4%E5%AE%9A%E8%A1%A8)设计[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)的条件：

①规格说明以[判定表](https://baike.baidu.com/item/%E5%88%A4%E5%AE%9A%E8%A1%A8)形式给出，或很容易转换成判定表。

②条件的排列顺序不会也不影响执行哪些操作。

③规则的排列顺序不会也不影响执行哪些操作。

④每当某一规则的条件已经满足，并确定要执行的操作后，不必检验别的规则。

⑤如果某一规则得到满足要执行多个操作，这些操作的执行顺序无关紧要。

**正交试验设计**

就是使用已经造好了的正交[表格](https://baike.baidu.com/item/%E8%A1%A8%E6%A0%BC)来安排试验并进行数据分析的一种方法，目的是用最少的[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)达到最高的测试覆盖率。[2]

场景法

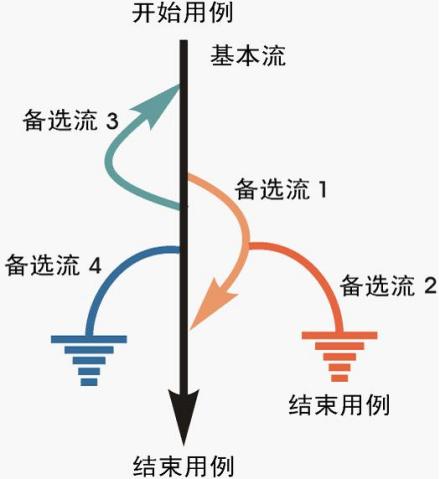
[](https://baike.baidu.com/pic/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/934030/0/91ef76c6a7efce1b3b0b1a23af51f3deb48f6533?fr=lemma&ct=single)

图1基本流和备选流

软件几乎都是用事件触发来控制流程的，事件触发的情景便形成了场景，而同一事件不同的触发顺序和处理结果就形成事件流。这种在软件设计方面的思想也可以引入到软件测试中，可以比较生动地描绘出事件触发时的情景，有利于测试设计者设计测试用例，同时使测试用例更容易理解和执行。

基本流和备选流：如图1所示，图1中经过用例的每条路径都用基本流和备选流来表示，直黑线表示基本流，是经过用例的最简单的路径。备选流用不同的色彩表示，一个备选流可能从基本流开始，在某个特定条件下执行，然后重新加入基本流中（如备选流1和3）；也可能起源于另一个备选流（如备选流2），或者终止用例而不再重新加入到某个流（如备选流2和4）。