CHAPTER 3　第3章

动态测试技术

与静态测试不同，动态测试需要首先设计测试用例，然后一次或多次运行被测软件，并通过分析软件运行结果与期望结果的差异，来分析被测软件是否满足要求。

前面已经介绍，软件测试有多种分类方法。从是否关注被测程序的内部结构和实现细节的角度，软件测试可分为白盒测试、黑盒测试，以及灰盒测试。

白盒测试利用程序设计的内部逻辑和控制结构生成测试用例，进行软件测试；黑盒测试方法主要通过分析规格说明中被测软件输入和输出的有关描述来设计测试用例，不需要了解被测软件的实现细节；灰盒测试是介于白盒测试和黑盒测试之间的一种测试方法，基于程序运行时的外部表现并结合程序内部逻辑结构来设计测试用例，采集程序外部输出和外部接口数据以及路径执行信息来衡量测试结果，对软件程序的外部需求及内部路径都进行检验。

3.1　白盒测试

3.1.1　概述

白盒测试(white-box testing)也称结构测试、逻辑驱动测试或基于程序的测试。根据GB/T 11457—2006，结构测试(structural testing)是“侧重于系统或部件内部机制的测试。类型包括分支测试、路径测试、语句测试”。白盒测试将测试形象地比喻成把程序放在一个透明的盒子里，如图3-1所示，测试人员了解被测程序的内部结构，利用程序的内部逻辑结

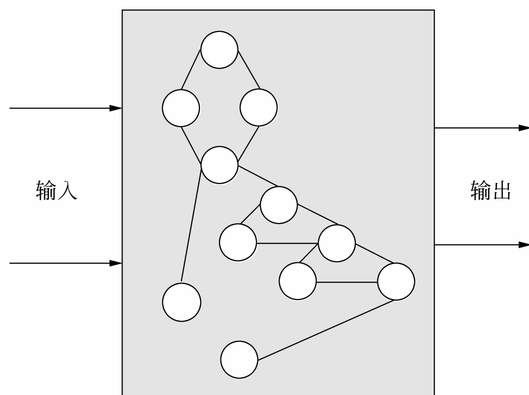

图3-1　白盒测试的基本原理图

构和相关信息，对程序的结构和路径进行测试。白盒测试是从程序设计者的角度进行的测试。

白盒测试的方法总体上可分为静态方法和动态方法两大类。静态方法是不实际执行程序而进行的测试，主要是检查程序代码或文档的表示和描述是否一致、符合要求以及有无冲突或歧义。文档审查、代码审查、静态分析、代码走查等都属于静态测试方法，已在第2章中进行了介绍。

动态测试的主要特点是当软件在真实的或模拟的环境中执行之前、之后及执行当中，对软件行为进行分析。动态测试时，软件在受控的环境下使用特定的期望结果进行正式运行，显示其在检查状态下是正确还是不正确。在本节后续小节中，将主要介绍动态白盒测试技术，包括基本路径测试、控制结构测试和其他的技术。

基本路径测试对程序设计复杂度进行合理度量，并以此为指导来定义一个基本路径集合。基本路径测试对所有独立路径进行测试，这些独立路径能组成程序的任意一条路径，因此能够满足分支测试的要求。基本路径测试通过对基本路径集生成测试用例，保证程序中的每条语句在测试中至少被执行一次。

控制结构测试是由基本路径测试演化而来的，对程序中语句或指令的执行顺序进行控制，其主要目标是选择测试用例以满足代码的各种覆盖准则。控制结构测试通常包括对判定的测试、对循环的测试、对数据流的测试等。

下面给出一段代码示例，本节后续小节将结合该代码开展相关介绍。

问题描述：NextDate 是一个函数，作用是根据输入的日期（年、月、日）计算后一天的日期。假设 NextDate 函数接收的输入值均为合法值，对输入值是否合法的判断在其他函数中完成，在此不再列出。

函数实现：列出 NextDate 函数的代码如下：

```
    /// <summary>
    /// 日期结构体
    /// </summary>
    public struct MyDate
    {
        public int year;
        public int month;
        public int day;
    }

    /// <summary>
    /// 计算下一天的日期
    /// </summary>
    /// <param name="date">当前日期</param>
    /// <returns>下一天</returns>
    MyDate NextDate(MyDate date)
    {
1       int year=date.year;
2       int month=date.month;
3       int day=date.day;
```

```
//计算当月的天数
4int lastday=30;                          //定义月的天数(最后一天)
5    if (month==1 || month==3 || month==5 || month==7
         || month==8 || month==10 || month==12)
6      {//有 7 个月的天数为 31 天
7          lastday=31;
8      }
9     else
10     {
11         if (month==2)
12         {//计算 2 月的天数,应为 28 天或 29 天(当闰年时)
13             if ((year %4==0 && year %100 !=0) || (year %400==0))
14             {//闰年
15                 lastday=29;
16             }
17             else
18             {//非闰年
19                 lastday=28;
20             }
21         }
22         else
23         {//其他月份为 30 天
24         }
25     }

26     MyDate tomorrow=new MyDate();  //下一天
27     tomorrow.year=year;
28     tomorrow.month=month;
29     tomorrow.day=day;

30     if (day==lastday)
31     {//如果当天是月末
32         tomorrow.day=1;             //下一天是下个月的第一天
33         if (month <12)
34         {//未跨年
35             tomorrow.month++;       //月份加 1
36         }
37         else
38         {//跨年
39             tomorrow.month=1;       //月份为下一年的第一个月(1 月)
40             tomorrow.year++;        //年份加 1
41         }
42     }
43     else
44     {//如果当天不是月末(是普通日期)
45         tomorrow.day++;             //日期加 1
46     }

47     return tomorrow;
   }
```

3.1.2 白盒测试基础

1. 控制流图和程序图

白盒测试依赖于程序的结构，需定义一种程序的表示方式。控制流图是一种刻画程序结构和逻辑流的方法，任何过程设计都可以转换为控制流图。

在控制流图中，线条和箭头表示流控制，称为边；圆圈表示一个或多个动作，称为节点；由边和节点围成的范围称为区域。如果一个节点包含判定条件，称为谓词节点。不同程序结构的控制流图表示如图 3-2 所示。

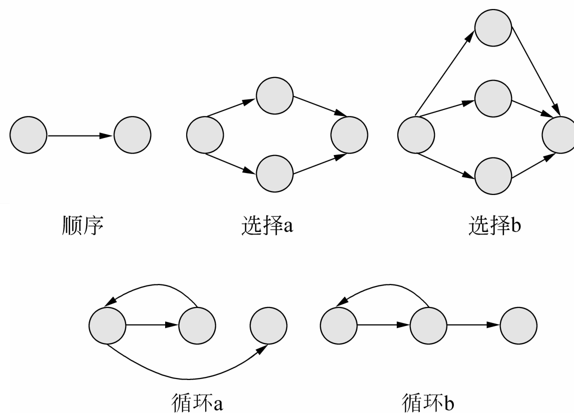

图 3-2 程序图的图形符号

程序图可以看作是压缩后的控制流图，也是一种特殊形式的有向图，上述 NextDate 函数的程序图如图 3-3 所示，图中数字对应源代码中的语句编号，K1、K2、K3 为语句段，其中 K1 表示语句 1～4，K2 表示语句 26～29，K3 表示语句 39～40。

程序图中每个节点代表一段语句片段(包含一条或多条语句)，每条有向边表示程序执行的走向。对一段程序源代码构造其对应的程序图，应遵循如下的压缩原则：

(1) 剔除注释语句，注释不参与实际程序执行，对程序结构不产生任何影响；

(2) 剔除数据变量的声明语句，此处的声明语句特指不进行初始化、只声明了变量类型的语句，进行初始化或赋值的语句不在此列；

(3) 所有连续的串行语句压缩为一个节点，即忽略子路径上经过的语句条数，无论某条子路径包含多少语句，只要不存在执行分支，一律压缩为一个节点，与变量无关；

(4) 所有循环次数压缩为一次循环，即忽略循环次数，无论某个循环结构将循环多少次，仅考虑执行循环体和不执行循环体这两种情况，与程序拓扑无关。

NextDate 的压缩后的程序图如图 3-4 所示，图中每个字母表示一段或一条语句，对应关系如下所示。在未特别指明的情况下，下文将以图 3-4 作为示例进行计算和介绍。

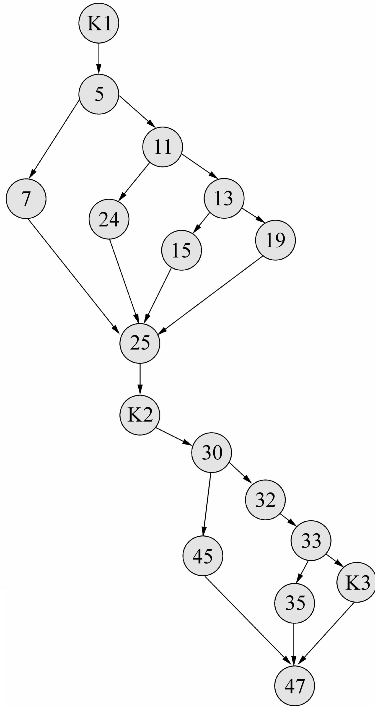

图 3-3　NextDate 的程序图

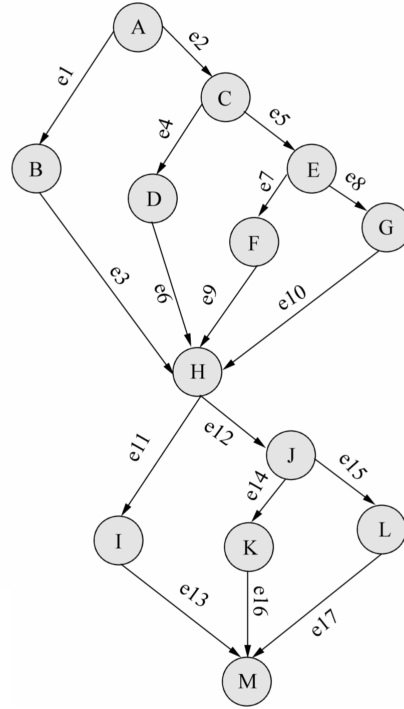

图 3-4　NextDate 的程序图(压缩后)

```
    /// <summary>
    /// 日期结构体
    /// </summary>
    public struct MyDate
    {
        public int year;
        public int month;
        public int day;
    }

    /// <summary>
    /// 计算下一天的日期
    /// </summary>
    /// <param name="date">当前日期</param>
    /// <returns>下一天</returns>
    MyDate NextDate(MyDate date)
    {
1       int year=date.year;                                           ┐
2       int month=date.month;                                         │
3       int day=date.day;                                             │
        //计算当月的天数                                              ├ A
4       int lastday=30;                    //定义月的天数(最后一天)   │
5       if (month==1 || month==3 || month==5 || month==7              ┘
            || month==8 || month==10 || month==12)
```

```
6      {//有 7 个月的天数为 31 天
7          lastday=31;                                                       B
8      }
9      else
10      {                                                               }
11         if (month==2)                                                     C
12         {//计算 2 月的天数,应为 28 天或 29 天(当闰年时)                  }
13             if ((year %4==0 && year %100 !=0) || (year %400==0))      } E
14             {//闰年
15                 lastday=29;                                               F
16             }
17             else
18             {//非闰年
19                 lastday=28;                                               G
20             }
21         }
22         else
23         {//其他月份为 30 天
24         }                                                           } D
25      }
26      MyDate tomorrow=new MyDate();     //下一天
27      tomorrow.year=year;
28      tomorrow.month=month;                                                H
29      tomorrow.day=day;
30      if (day==lastday)
31      {//如果当天是月末
32          tomorrow.day=1;               //下一天是下个月的第一天
33          if (month <12)                                                   J
34          {//未跨年
35              tomorrow.month++;   //月份加 1                               K
36          }
37          else
38          {//跨年
39              tomorrow.month=1;   //月份为下一年的第一个月(1 月)
40              tomorrow.year++;    //年份加 1                               L
41          }
42      }
43      else
44      {//如果当天不是月末(是普通日期)
45          tomorrow.day++;         //日期加 1                             }I
46      }
47      return tomorrow;
   }                                                                       }M
```

2. 圈复杂度

圈复杂度,又称环复杂度或 McCabe 复杂性度量,是一种对程序结构复杂度的度量模型,由 McCabe 于 1982 年提出,其基本思想是基于判定节点对程序图封闭圈数目造成的影响来衡量程序的复杂程度。根据圈复杂度,可以得出一个程序结构中独立路径的数目,以及确保每条语句被执行一次要生成的测试用例数目的上限。

计算一个程序的圈复杂度，有 3 种方式：观察法、公式法和谓词节点法。下面以图 3-4 为例分别介绍。

1）观察法

观察法指根据圈复杂度的定义，直观观察程序图将二维平面分隔成的封闭区域和开放区域的个数。对于图 3-4，其中封闭区域有 5 个：

区域 1，由节点 A、B、C、D、H 所围成；

区域 2，由节点 C、D、E、F、H 所围成；

区域 3，由节点 E、F、G、H 所围成；

区域 4，由节点 H、I、J、K、M 所围成；

区域 5，由节点 J、K、L、M 所围成；

另外还有 1 个外部的开发区域。因此，该程序图的圈复杂度为 6。

2）公式法

可利用程序图中边、节点和开放区域的数目，使用如下公式计算圈复杂度：

$$v(G) = e - n + 2p$$

其中，$v(G)$表示圈复杂度，e 表示图中边的数目，n 表示图中节点的数目，p 表示图中未连接部分（即不封闭区域）的数目。

对于图 3-4，$v(G)=17-13+2\times1=6$，该程序图的圈复杂度为 6。

3）谓词节点法

如果程序图中一个节点包含判定条件，称为谓词节点。可利用独立谓词节点的数目，使用如下公式计算圈复杂度：

$$v(G) = D + 1$$

其中，D 表示图中独立谓词节点的数目。请注意，所谓的独立谓词节点数目不应简单地认为是程序图中判定节点的个数，事实上，分支大于 2 的判定节点可以被拆分成多个仅含两个分支的判定节点。因此，当程序中的判定节点均为两分支的判定时（包括循环节点），可将每个判定节点视为一个独立谓词节点；而当判定节点为 n（大于 2）的分支时，该判定节点应视为（$n-1$）个独立谓词节点。

对于图 3-4，节点 A、C、E、H、J 为独立谓词节点，一共有 5 个，所以，该程序图的圈复杂度为 6。

3. 图矩阵

图矩阵是一种有用的基本路径分析工具。图矩阵是一个方阵，其大小（行和列的数目）为流图中节点的数目，行或列表示流图的节点，矩阵中的元素表示流图的边。表 3-1 给出了图 3-4 的图矩阵，可以看到图矩阵就是一个表示流图的表格。

表 3-1 NextDate 程序图的图矩阵

	A	B	C	D	E	F	G	H	I	J	K	L	M
A		e1	e2										
B								e3					
C				e4	e5								

续表

	A	B	C	D	E	F	G	H	I	J	K	L	M
D								e6					
E						e7	e8						
F								e9					
G								e10					
H									e11	e12			
I													e13
J											e14	e15	
K													e16
L													e17
M													

给图矩阵中每个元素增加一个连接权值，图矩阵可被用于评估程序控制结构。权值可以表示不同的含义，比如：节点之间的连接存在或不存在，通常用“1”或“0”表示；一个连接(边)被执行的概率；在经历一个连接时所需的处理时间；在处理连接时所需的内存或其他资源等。作为示意，把表 3-1 中的边用“1”替代，表示该连接存在，给出表 3-1 的连接矩阵，如表 3-2 所示。连接矩阵可用于计算程序的圈复杂度。

表 3-2　连接矩阵及圈复杂度

	A	B	C	D	E	F	G	H	I	J	K	L	M	连接
A	0	1	1	0	0	0	0	0	0	0	0	0	0	2－1＝1
B	0	0	0	0	0	0	0	1	0	0	0	0	0	1－1＝0
C	0	0	0	1	1	0	0	0	0	0	0	0	0	2－1＝1
D	0	0	0	0	0	0	0	1	0	0	0	0	0	1－1＝0
E	0	0	0	0	0	1	1	0	0	0	0	0	0	2－1＝1
F	0	0	0	0	0	0	0	1	0	0	0	0	0	1－1＝0
G	0	0	0	0	0	0	0	1	0	0	0	0	0	1－1＝0
H	0	0	0	0	0	0	0	0	1	1	0	0	0	2－1＝1
I	0	0	0	0	0	0	0	0	0	0	0	0	1	1－1＝0
J	0	0	0	0	0	0	0	0	0	0	1	1	0	2－1＝1
K	0	0	0	0	0	0	0	0	0	0	0	0	1	1－1＝0
L	0	0	0	0	0	0	0	0	0	0	0	0	1	1－1＝0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0
连接总数														5
圈复杂度														5＋1＝6

3.1.3　基本路径测试

1. 基本原理

从程序入口到程序出口之间存在许多可能的路径。对于判断语句，路径数目可能会加倍；对于 switch-case 语句，路径数目依赖于分支数目；对于循环语句，路径数目随着循环变量值增加而增大。总之，对于一段即使是比较简单的程序，要达到完全路径覆盖也是非常困难的。基本路径测试是一种较好的完全路径覆盖的变通方法。

基本路径测试的基本原理是：如果将全路径集合看作是一个向量空间，把从全路径集合中抽取的一组线性无关的独立路径看作是一组向量基，根据基于向量空间和向量基的理论可知，全路径集合中的所有路径可由这组独立路径的某种组合方式来遍历，因此，只需对这组独立路径进行了测试，就等价于对全路径进行了测试。这组独立路径就成为基路径或基本路径。基本路径测试的基本原理如图 3-5 所示。

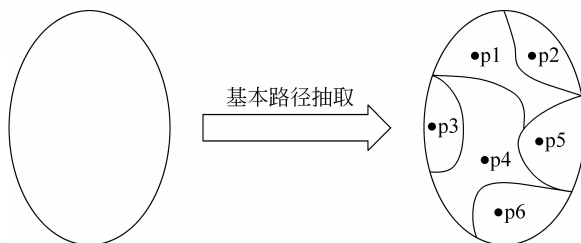

图 3-5　基本路径测试的原理

基本路径测试的目标是：

（1）测试的完备性，即通过对独立路径的测试达到对所有路径的测试覆盖；

（2）测试的无冗余性。每条路径都是独立的，设计的测试用例之间不存在冗余。

抽取的基本路径需满足如下要求：

（1）任意两条基本路径线性无关；

（2）所有基本路径的并是整个向量空间，即任意一条路径都可以转化为某一条或几条基本路径的组合遍历。

2. 测试设计

基本路径测试的主要步骤包括：以代码/设计为基础画出程序图；计算基本路径集合的规模；抽取基本路径；生成测试用例。结合 NextDate 程序进行介绍。

1）画出程序图

根据 NextDate 程序代码（此处略，请见前面章节），得到程序图，见图 3-4。

2）计算基本路径集合的规模

基本路径的个数等于圈复杂度。对于 NextDate 程序，基于程序图可得圈复杂度是 6，其基本路径集合的规模也为 6。

3）抽取基本路径

第一，确定主路径。从所有路径中，找到一条最复杂的路径作为主路径。所谓的最复杂体现在：

（1）主路径应包含尽可能多的判定节点（包括条件判定和循环判定节点），判定节点越多，路径越复杂。

（2）主路径应包含尽可能复杂的判定表达式，判定表达式包含的变量数量和“与”“或”关系越多，路径越复杂；当判定节点相同时，取判定表达式复杂的为主路径。

（3）主路径应对应尽可能高的执行概率，每个判定节点取不同分支的概率并不相同，当不同路径包含相同数量的判定节点时，可根据一定规则来计算每条路径的执行概率（比如，所包含的所有判定分支执行概率的乘积），执行概率越高的路径越复杂。

（4）主路径应包含尽可能多的语句，在相同执行概率的情况下，比较路径所包含的原始语句的数量，取语句数量多的为主路径。

第二，根据主路径抽取其他基本路径。

基于主路径，依次在该路径的每个判定节点处执行一个新的分支，构建一条新的基本路径，直至找到足够的基本路径数。当主路径上所有的判定节点处的每个分支都已覆盖，但仍不能达到指定数量的基本路径时，应查找程序中尚未完全覆盖的判定分支并构建基本路径。构建基本路径时，仍可按照判断表达式的复杂度、路径执行概率、路径包含语句数等原则进行。

对于 NextDate 程序，其基本路径集合为：

Path1：A-C-E-G-H-J-K-M（主路径）；

Path2：A-B-H-J-K-M（在判定节点 A 处执行 e1 分支）；

Path3：A-C-D-H-J-K-M（在判定节点 C 处执行 e4 分支）；

Path4：A-C-E-F-H-J-K-M（在判定节点 E 处执行 e7 分支）；

Path5：A-C-E-G-H-I-M（在判定节点 H 处执行 e11 分支）；

Path6：A-C-E-G-H-J-L-M（在判定节点 J 处执行 e15 分支）。

4）处理不可行路径

上面步骤完全是基于程序图进行的，而由于业务逻辑、程序缺陷等原因，得到的基本路径有可能是不可行路径，需要对不可行路径进行处理，通常有 2 种处理方法：

（1）在不可行路径中，找出所有判定节点，进入另一个分支，直到得到符合业务逻辑的路径为止；

（2）找到不可行路径，通过人工干预方式得到一条符合业务逻辑的路径。

对于 NextDate 程序，Path6 执行 A-C-E-G 语句，说明函数输入为 2 月（并且非闰年），从而不可能再执行判定节点 J 处的 e15 分支到达 L（跨年，只有 12 月才可以），因此，Path6 为不可行路径。

由于所有判定节点的每个分支都已覆盖，通过人工干预方式选定一条新路径作为 Path6：

Path6：A-B-H-I-M（发生几率最大）。

5）设计测试用例

根据上面得到的基本路径设计对应的测试用例。

对于 NextDate 程序，针对基本路径 Path1～Path6，每条路径至少设计一个测试用例，得到测试用例集合如表 3-3 所示。