SAMS 北京科海培训中心

Visual Basic™ for Windows™
实用编程指南

[美] D. F. Scott 著
周少柏 查良钿 编译
金 奇 审校

清华大学出版社

目 录

第二部分 从概念到创造

第三部分 实验性程序设计

第一部分 程序设计过程

第一章 使用语言进行计算

让我们回忆一下上半世纪中有关计算的科学幻想。无论是小说还是电影，你都会看到科幻作家使人类和计算机对话。在科幻计算中，和语言相比，图标不是功能很强的工具。

90年代的程序员可能认为图符和图形的设计是用户通信的基本工具，但不可否认程序设计语言仍然是桥接人类和机器的重要通信工具。它之所以能成为通信的基本构件（思维的原型）是因为通信涉及更多的文字而不是图画。那么，软件市场商弄错了吗？以描述图标作为他们的联编原理的程序员，是不是忽视了向用户提供更为清晰的方法？

本章展示两个处于初始构造阶段的大型 Visual Basic 应用程序，当你仔细地研究它们之后，你会注意到我的重点是讨论高级语言计算与使用组装软件不同之处。你可以看到，语言为解题和自动化任务提供许多独特的方法。

1.1 为什么使用 Visual Basic 语言？

Visual Basic 的目标是生成独特的问题求解应用程序。高级程序设计语言为计算机构造了第一个真正的用户程序。当计算机硬件开发者认为所有计算机用户应成为程序员的时候，组装的软件应用程序就已经出现。但是，虽然有了无数的包装软件，高级语言仍保留着某些重要的领域。

商业界喜欢用高级语言而不是软件程序包构造定制的应用程序，主要原因如下：

- 商业界当前完成工作任务的方法未能用当前选择的预组装软件成功地自动化。
- 商业界中工作人员执行作业的各种方法，由于管理上的安排或个人嗜好，经常变动或调整，而当作业改变时，程序也必须改良。
- 没有经过全面训练的计算机操作工作人员，当他们学会更多有关计算的问题时，可能想使用随时增加功能的程序。假设用户都是些程序员，则希望他们能适应现成的软件和电子表格和其他软件程序包。
- 为了竞争、占领市场和鼓舞士气，公司有权使用有它们自己特色的软件。如果公司选择秘密的商业策略，更不应采用竞争对手所用的同类市场化软件。
- 面向特殊顾客的公司要求高度灵活的软件系统。

1.1.1 为初学者定义 Visual Basic

基于 Microsoft 的 Visual Basic 市场化方法，你可以认为它仅仅是另一种定制的应用程序。该公司不强调“选中和单击”的计算原理（不象 Word for Windows 和 Excel）。对于那些询问 Visual Basic 实际是什么的客户，标准的答复是：Visual Basic 是一种程序设计的指令系

统，它可以用于定制事务应用程序。

在商业或办公环境中，Visual Basic 的真正目的是把正常执行的事务模型化。VB 应用程序能实现自动化以加快和优化每日的任务，使工作少一些重复、也许还更有趣。此外，VB 应用程序不增加用户更多的工作。某些组装软件增加了办公室的工作量，而且工作耗费在维护计算机系统而不是运行事务。Visual Basic 程序员必须避免建造耗时和要求高维护成本的程序包；否则，就没有必要使用它。

人们对 Visual Basic（或任一程序设计语言）最普通的误解是：它是智能系统的主要成分。可能由于含混的市场印象，人们相信 Visual Basic 和其他语言能研究问题、进行分析和提供结论。高级程序设计语言不是智能的；它们不为你阐述任何事情。作为一个成熟的程序员，你应该避免误解，作为程序员，向 Visual Basic 说明是你的责任。作为 Visual Basic 应用程序的作者，我尽量不考虑我是在用 Visual Basic 进行通信。

图 1.1 为高级语言的通信模型。这里的椭圆形对象代表会话的各方，而前头代表会话的事件。电话工程师用通信模型把假想的双方或多方的会话联系起来。在这个特殊模型中，通信模型展示程序员和程序用户之间的双方会话。

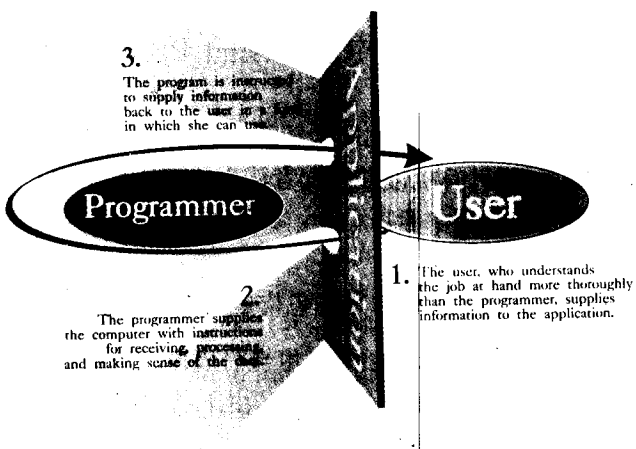

图 1.1 高级语言的通信模型

请注意，事件由用户启动和结束；程序员逻辑上作为他们自身和用户之间数据传输的向导。这不是程序和用户之间或程序员和程序之间的会话。完整的程序设计就是人们之间的会话。这个特殊会话的特点就是程序员预先指示程序如何应答用户所有的咨询。必须预先编制好如何应答用户向应用程序发出的所有可能询问。

图 1.1 把计算机描绘为遮挡程序员和用户两者视线的一堵墙。作为程序的作者，你可能感觉到就象通过磁带录音机和用户说话。在 Visual Basic 的情况下这特别真实（比标准 C++和 QBasic 更真实）。因为系统中有一个创新的事件驱动机制。这个系统“能”使用户更容易输入数据。当设计程序时，设想你已和用户交互而不是和计算机交互。

下面的实例展示了在程序员和用户之间的通信过程中高级程序设计语言的作用。

1.1.2 零售商品库存跟踪系统(实例 1)

若你的客户是一个零售产品商或联营商,他们希望编写一个零售货物的库存、购销和跟踪系统。这种机构比较小,往往是同一个人从供应商或批发商那里购买货物、销售这些货物,及分析这种货物的销售史。因此,采购、开发票、销售点和跟踪模块都不必完全分离。

该程序的用户需要使用的功能是:他们应能检查库存以观察产品是否在库中、放置在什么地方及它的销售价格等。如果产品不在库中,而顾客现在需要它,则用户现在有权初始化采购进程。在任何时候,用户必须能对特殊种类的产品和特定的产品考查销售的情况。

实例 1 的目标如下:

- 主要的库存模型维护一个全部销售货物的数据库。
- 顾客表保存关于带有特权和邮政订货的顾客信息。
- 采购模型维持一个由不同批发商提供的各种货物的最新价格表。这能使购买者确定当前最好的贸易策略。
- 发票和销售点模型使当前库存表可即时更改。
- 销售跟踪模型使用户知道特定货物的销售情况,包括考虑不同的货物分类和商标名。

首要的工作是设计窗体和窗口,而数据是程序的最重要部分,必须首先考虑好。所以,首先设计全程范围的数据变量。组成这个程序主体的数据元素是多变的库存货物。首先让我们研究一下库存货物的特征。除软件工具外,书本、硬件、视频元件和办公家具等都可以作为例子。库存货物的主要特征是它的名称、销售成本和货架价格。

货架中的一格可能有专门的序号。如果制造商对有限的货物提供特殊的优惠。譬如说,对专门标记和编号的货物提供优惠价格。这时,必须在顾客的发票中记录序号。然后,顾客可以复印该发票,并把它送给制造商作为购买的证明。再者,假定顾客返回有缺陷的货物,系统将"不卖"这货物并把它放回库中,原来发票的某处也必须记录这个序号。购销部门需要这个序号以便把该货物退回给制造商(简称为"退回确认"或"R. A.")。

图 1.2 示出在作决定过程中的原始分段。必须区分货物和产品,所以需要两个编目的数据文件。"货物"是货架上的特定对象或盒子,而"产品"是一组货物的名称。虽然两种编目有相关的元素,但为提高效率,应保持最少的重复。

Visual Basic 处理存储中数据数组的功能还比较弱。解释程序提供三种数据存取方案;其中两个可应用于本程序。顺序存取方案要求把整个数组送入内存,完全在内存中改变或修补数据。当关闭文件时,才把该数组全部存回磁盘。当在网络上编写 Visual Basic 应用程序时,顺序存取是危险的方案。如果两个客户或伙伴沿着网络把同样的数据文件装入存储数组,而每一伙伴改变数组时没有注意到其他伙伴的改变,那么,最后保存的数据文件将重写先前保留的文件,这意味着丢失了一半的修改。无论如何,文件变成 100%的失效。

当需要一次读一个记录时,随机存取方法是比较好的。Visual Basic 允许程序员在引用变量时采用面向对象的语法。Microsoft 发现它很难命名用 Type 子句声明变量这种概念。在 Visual Basic 1,Microsoft 使用术语"用户定义变量",尽管是程序员(不是用户)进行定义。至于 Visual Basic 2 ,Microsoft 试用名字"记录变量",因为在 Visual Basic 的随机存取数据文件

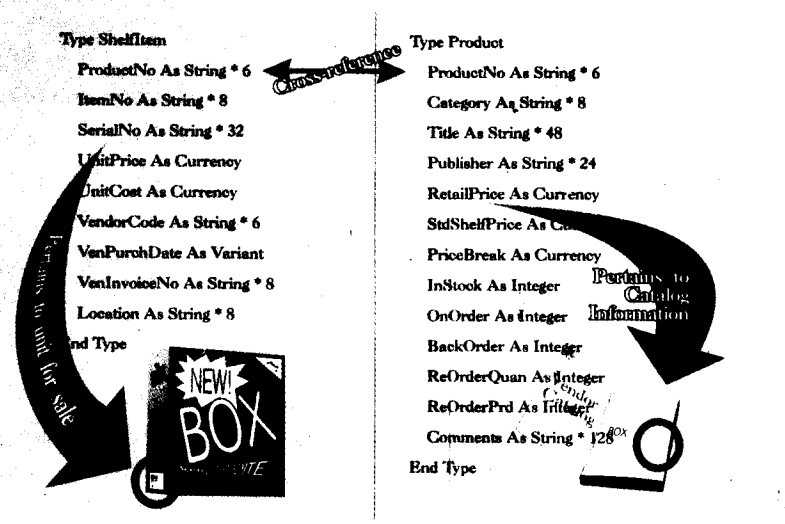

图 1.2 一致性标识的危机

方案中，这涉及到数据记录的构成元素。由于变量不包含记录，Microsoft 在术语的选择上可能有些混淆。因此，本书使用术语“组合变量”以指称在全程或总模块中用 Type 子句声明的变量结构。

> **技术注记** 如果多数据文件中含有相关字段时，选择随机存取方法。

组合变量方案允许在记录中查找数据字段时，按先主题，后字段的顺序进行。有如下的语法：

```
subject. field
```

从数据库工程师的观点看，前面引用的第一项是它的主题，因为可以使用 subject 作为分类，访问记录中所有的字段。但是，从软件工程师的观点看，引用的第一项是对象；每一字段组成“事件”的一个特征，它按对象编址，因此必须把对象分类，不能混淆。

在任何情况下，都可以利用 Visual Basic 中的 Type 子句描述数据表格和它们的构成字段。在图 1.2 中，可以看出作为货架中的盒子和作为销售产品的货物之间的不同。为 Visual Basic 货物 INVENT1. MAK 编写的前几行代码是对库存系统两个基本数据文件结构的初始估算。我已在程序中加进了一些注释，以说明每一字段在它的组合变量中的作用。

```
Type ShelfItem
  ProductNo As String  * 6        ' Key field
  SerialNo As String  * 32        ' Number given the item
                                  ' by its manufacturer
  UnitPrice As Currency           ' Shelf price give the item set
                                  ' by vendor
  UnitCost As Currency            '  Actual Cost of the item set
                                  '  by Vendor
```

```
    VendorCode As String * 6        'Code for the vendor that
                                    'sold store this item
    VenPurchDate As Variant         'Date store purchased the
                                    'item from vendor
    VenInvoice As String * 8        'Invoice number of prior
                                    'purchase from vendor
End Type

Type Product
    ProductNo As String * 6         Key field
    Category As Integer             Store-defined code for shelf
                                    'category
    Title As String * 48            'Official title for the
                                    'product
    Publisher As String * 6         'Store-defined code for
                                    'product publisher
    RetailPrice As Currency         'Manufacturer-suggested
                                    'retail price
    OnOrder As Integer              'Amount of product currently
                                    'on order
    BackOrder As Integer            'Amount of product vendors
                                    'placed on back-order
    ReOrderQuan As Integer          'Recommended reorder quantity
                                    'on regular basis
    ReOrderPrd As Integer           'Store code for regular
                                    'reorder
    Comments As String * 128        'Arbitrary comments from
                                    'any user
End Type
```

所有应用程序(扩展的或其他)都在内存或盘上运行数据库,甚至对于只有四种功能的计算器也是如此。存储中数据文件的内容是一个表格。为把内容组织好,表格须包含一系列等长的记录。每个在 Type 声明中出现的成分变量的规则定义了 ShelfItem 和 Product 数据文件的记录长度和结构。记录中的每一变量构成记录的字段。图 1.3 示出怎样安排对象变量 ShelfItem 的数据表格的内容。

现在,请注意两个互相关联的组合变量结构,只有一个变量 ProductNO 在两种类型中同时声明。在此方案中,ProductNO 包含一个 6 数字的标识号。它代表产品的名称。这是个关键的字段,它标识涉及该产品的所有数据表格中的产品。因为它是关键字段。它唯一地标识该产品。

关键字段技术对某些数据库或数据文件方案的结构是很重要的。

技术摘要:关键数据字段

定义:在 Visual Basic 中,关键字段作为一组组合变量的单个独特的成分元素来实现。关键字段可能包含数字或字母的数据。根据定义,它出现在 Visual Basic 货物不只一个的组合变量中。当组合变量代表 Visual Basic 的数据文件时,Type 子句声明的成分元素各自定义数据文件的一个记录。每一数据文件的记录和相同的货物或对象有关。通常,记录中的成分元素用作为标识符——例如,人的社会保险号或汽车的车牌号。可以在多个组合变量中使用同样的标识符,以代表同一个人或货物,采用这种方法,你不必存储货物的所有特征。这些货物遍及 Visual Basic 工程,在存储的大量组合变量中,或在磁盘面积和繁杂的数据文件里。你也

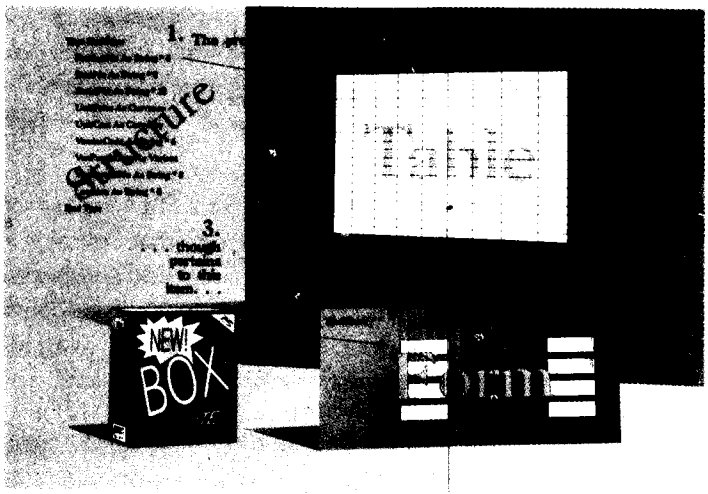

图 1.3 数据表格的双重标识

可以使用关键字段作为排序过程的索引字段。

执行：使用 Type 子句在全程模块中声明一组组合变量。这些组合变量的目标彼此相关，因而它们的 Type 子句都包含一个(只有一个)成分变量。这个变量的名字、定义、和声明是相同的。考虑到形式，关键字段是 Type 子句中声明的第一个组合变量。这些子句使用关键字段作为它们的标识符。

例子：假设你正在为快速润滑汽车服务性企业编写应用程序。该应用程序使用两个数据文件，一个文件包含所有已预先入库的汽车的服务记录，而另一个则包含每辆汽车的发票。你可以使用交通工具的牌照编号，它在大多数汽车的档泥板或在发动机罩下，作为标识符。

两个和同一辆汽车的记录有关的 Type 子句如下：

```
Type ServiceRecord
    VID As String * 36
    ServiceCode(16) As Integer
    ServiceDate(16) As Integer
End Type

Type Invoice
    VID As String * 36
    Number As String * 6
    EntryDate As Variant
    BillAmount As Currency
    Received As Currency
    PymtCode As Integer
End Type

Global CarHistory(1000) As ServiceRecord
Global CustomerHistory(1000) As Invoice
```

你可以使用已经在 ServiceRecord 文件中处理的数据内容生成第一张顾客发票。用下面的指令设置 Invoice 文件的新关键字段：

CustomerHistory(newcar%).VID=CarHistory (thisCar%).VID

其中 newcar%是指向 ServicRecord 文件中最后一个记录的指针，而 thisCar%是指向发票窗体上当前发票记录的指针(指针反映出数据库或数据文件中记录的位置)。

在应用程序的后部：假定这个公司的会计师抽出发票记录以查看与高收费相符的服务。你可以建立指向当前汽车的指针 thisCar%，使用这指令：

Carno $ =CustomerHistory 9thisCar%).VID

现在，你可以使用变量 Carno $ 作为指针进行搜索，以便匹配 CarHistory (searchcar%).VID 的内容。

提醒：为正常地实现数据文件中的关键字段，关键字段中每一表项的内容必须唯一。应该在你的应用程序中实现安全功能，以确保没有两个关键字段的表项相同。为高效地进行工作，你应使程序在每次添加记录时重排序数据文件，并使用关键字段作为排序的索引字段。采用这种方法，如果两个关键字段表项相同，在数据表被排序之后，它们将彼此相邻。

对于货物 INVENT1.MAK，许多内含组合变量的关键字段是 ProductNO，它代表货架上产品的名称或专用标识。

根据采购代理人的看法，一些批发商可能把价格降到低于当前产品目录所印出的价格，目的在于进行商业上的竞争。当作出采购决策，采购代理人接收把价格降低的那些批发商代理人的电话，并对比每一单项的当前目录价格。这些特殊的处理可能难以归类为每箱的价格，它们可能为买三样东西，或一样 A 和两样 B 而专门降价。许多目录已经对每一货物分类标价，如果订单的总金额是在某一数目(譬如，1000 美元)之上，则可降低价格。现在零售商品市场中的货物价格是从不固定的；事实上，它是灵活可变的。你必须以某种方式为价格的变化编码。至此，你可以认为产品价格保留在独立的数据文件中。

在这些细节问题上，试考虑一下使用预组装应用程序工作时你该如何做，这样做又有些什么不同。在 80 年代，使用电子表格程序作为数据库管理程序是很普遍的。用户写库存“程序”(一组松散相关的宏)的电子表格列出所有和库存产品有关的数据字段，每项一行，每字段一列。在库存应用模型中，需要分离货物数据文件和产品数据文件，但带有某些两文件共享和重复的标识字段。使用电子表格，那两个文件可以是两个独立的表，也可以是一个大表格中的两段。电子表格用户需要维护这两个表格的关系或为不同的行记录单元的指针，可以想像得到其中的困难。没有什么比文件格式更能说明电子表格作为商业自动化系统的低效。

现在，让我们暂不讨论实例 1，等到第三章“应用程序模型”时再讨论它。当你重新使用这个程序时，你将明白如何建立和分类应用程序中的各种窗体。稍后，你将看到如何设计和使用此程序的更为复杂的数据结构。

1.2 语言的通用性

Florian Coulmas(他在德国 Dusseldorf 大学教授语言)论证说，书写的语言是作为对金钱符号的一种方法而创造的。在 1989 年他的一篇有关正文符号进展的优秀论文“世界的书写系统”中，Culmas 提供论据指出，公元前 8000 年小亚细亚的苏门答腊人使用不同形状的石头代表货币单位的记号。苏门答腊人使用软陶片记录他们的市场交易，把每一种记号刻入陶

片，以使每一独特的形状代表他们用于交易的“硬币”值。后来，他们使用棍来“锻造”硬币的模型，于是，诞生了书写的文字通信。运行现代的记帐系统同样需要书写的正文。

Coulmas 定义说，文字是一种符号系统，它代表符号的信息，互相连接的信息并没有直接的意义。在玛雅文化中，人的图画真实地代表人；但在现代的英语字母表中，p—e—r—s—o—n 只是表示抽象的概念，6 个字母中没有一个类似（或假设类似）于人体的任何部分。在文字学中，符号依赖于它们彼此间的位置关系代表信息或概念。

Visual Basic 借用英语的术语和表达方式，不能认为是文字，虽然在最主要的方面很类似；它使用符号代表或模型化各种情况，而其中符号及它们的位置互相没有直接的联系。可以论证代数起源于苏门答腊人的记帐。代数学使用符号表示已知和未知值，就象表示苏门答腊人硬币的模型一样。Visual Basic 使用代数和英语的组合来表示作为数据的信息，以及表示作为代码的数据元素中的变化关系。

1.3 转向 Expressor 计算器

下面介绍一个可编程计算器应用程序，它是我在另一本书中建立的。为理解全面的程序设计可从其中取得经验——你首先考虑的是什么、你试图做什么 、你自己修改了些什么、什么时候你受挫、什么时候工作使你满意，最后，完成全部工作。

1.3.1 Expressor 计算器（实例 2）

这个实例的目标是建立一个模块集。人们可以把它综合到面向商业或科学的程序中。向用户提供部分数字、部分文字的屏幕操作装置，看样子象普通计算器，但有完整电子表格那样的求解公式的特色。用户可以在另一个 Visual Basic 应用程序或另一个 Windows 应用程序中安装这个模块集，“拨出”一个公式以帮助解决特定的、敏感的问题。例如，财会或几何问题。选择公式将以口语方式表达。在公式求解以后，答案出现在计算器的读出屏，并且可以传送给主应用程序。

下面说明 Expressor 如何工作。这个计算器的表面有一系列作为基本存储的槽。计算器保存一个已存储公式的表，它们的变量具有名字，也是存储好的。当用户选择求解的公式时，该公式的每一成分变量就出现在这些槽中。在它们的旁边，用户可以为求解公式键入值。（在后来的修订版中，Expressor 可解某一范围内的值）。用户可以使用标准的计算器登录系统取得值，并随后把该值从显示器拷贝到存储槽，或者直接把值打入槽中，这一应用程序的目的是使用户对公式有更口语化的理解。

计算器是 Microsoft′s Reference for Visual Basic 中的测试应用程序之一。我完全忽略它和几十个 Visual Basic 有关的文章所发表的其他计算器应用程序之间的同异。但希望能发现更行之有效的方法。也许允许初始时的不确定，将可能学到更多有关编辑的经验，而无须借助正文的帮助。图 1.4 示出 Expressor 原始的外观——有点像便携式电话机的支架。

不管 Visual Basic 所描述的上下文、变量作用域、以及程序模块之间有何差异，也不管设计窗体所用的那些工具，解释程序向程序员提供了凿刀似的等价物。使用它，你可以每次一片地雕刻 Visual Basic 项目程序的大理石块。往往是在过程的初始时的工作比在综合阶段的工作更有意义。

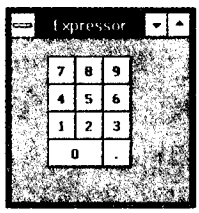

图 1.4　再生的 Expressor

1.3.2　核心上下文

对于 Expressor 程序的应用模型，首先要给出按钮的 Name 属性，从顶行开始是按钮 7，按钮 8，按钮 9 等等。为按钮 7 编写一过程，它把数字 7 放在读出屏(称为 Readout 的标签控件)的上方。下面是为整个程序编写的第一段指令。

```
Sub Button7_Click ()
Readout.Caption = Readout.Caption + "7"
End Sub
```

往后，可应用逻辑推理。不允许用户无限地添加数字，对读出屏的范围应有某些合理的限制。这里使用 Len()函数限制读出屏为 19 个数字，如下：

```
Sub Button7_Click ()
If Len(Readout.Caption) < 20 Then
  Readout.Caption = Readout.Caption + "7"
End If
End Sub
```

> **技术注记**　在子句中间打入缩进控制指令(像前面的 If-Then 子句)有助于标识子句的存在和范围。制表符不影响 Visual Basic 程序的执行。

对于命名为 Button Point 的小数点按钮，可以把先前的过程拷贝和粘贴到 Sub Button Point_Click()框架中。但是，仅当不多于一个小数点时，读出屏中的数才是有意义的。因此，程序应知道是否已有小数点。逻辑上，过程可使用 Val()函数把 Readout 标签的字母数字内容转换为数字值。在出错的情况下，例如在显示中有太多的小数点，可调用错误俘获例程。显然唯一不能成功转换为值的是 Buttonpoint。其他可发送到读出屏的字符都是数字。除了实现上的麻烦之外，未来开发的 Expressor 的其他类型字符也许不能用 Val()把 Readout.Caption 翻译为数值，例如，像“C”那样的十六进制数字。

> **技术注记**　在可能的情况下，你的程序应比现在能做更多的工作。不要限制你的逻辑思维能力以便使程序提供多种功能。

再者，对 Roadout.Caption 的内容再求值将影响程序效率。它的动作似乎有点固执，不断地询问解释程序，“这是一个数吗？现在怎样？如果要做方法又怎样？”如此等等。如果一

次性告诉程序读出屏中有一个小数点，“请不要在那里加其他东西。”那可能简单些。

在 Visual Basic 中，当指令的目的确实需要询问解释程序有关程序中某些事情的状态时，你从解释程序接收到的应答纯粹是符号性的。作为程序员，你可以随意为这些符号形成代码。换句话说，对“这个框是红的还是白的？”或“此刻联机的是哪一个用户？”的应答将由解释程序使用你编写的代码作为简短说明。但是，如果你提出的问题答案措词为 yes 或 no、true 或 false、on 或 off，则应答的代码已经存在，即解释程序已能识别。

无论你使用什么样的符号系统，有一条适用于符号象征学的总规则：由于符号的数目是有限和完整的，你用以表示应答符号的变量是一个整变量。因为可能应答的数目总是完整的（你不能回答一半），不需要任何变量的部分存储空间。表示 yes/no 或 true/false 回答的变量（换句话说，二元的状态）是一个标志变量。

技术摘要：标志变量

定义：标志变量是任意命名的项。在程序中它的值任何时候都代表双态之一。在这种情况下，双态可等价于作比较时 true/false 状态、yes/no 回答、either/or 应答，或任何要求一种或它种表示的状态。标志变量独立于当前存在于 Visual Basic 程序中任一窗体的任何图形对象。

执行：Visual Basic 2 的程序员表示真假状态的最普通的方法是使用语言化的内部常数对：True 和 False，On 和 Off，或 Yes 和 No。在每一对偶中，前一常数的值总是－1，后一常数的值总是 0。在明显地使用－1 和 0 不足以描述的情况下，可以在模块或全程模块（如果提供的话）的声明段选择声明常数的替代符号。下面是某些例举的声明：

```
Const BLACK = -1
Const WHITE = 0

Const IN = -1
Const OUT = 0
```

属于 True/False，BLACK/WHITE 和 IN/OUT 情况的标志变量的声明如下：

```
Static state As integer
Dim side As Integer
Global direction As Integer
```

以传统的表达式赋值方式把值赋与标志变量：

```
state = True
```

在比较表达式中可以使用标志变量：

```
If side = BLACK Then
   .
   .
   .
End If
```

在任何类型的条件子句中标志变量都是有用的：

```
Select Case direction
    Case IN
```

```
    .
    .
  Case OUT
    .
    .
    .
End Select
```

提醒：标志变量如果不作为局部变量声明是最有用的。因为严格的局部变量在执行 End Sub 或 Exit Sub 时重新初始化(它的值如被声明为数；则重置为 0)。作为标志变量，0 是有意义的，所以数值的局部变量的初始值总是 False 或 NO。如果标志变量没有在模块级形式地声明为 Global Dim，或在过程级声明为 static，则 Visual Basic 2 给它以缺省的类型 Vaviant。这意味着标志变量无论怎样都没有值(包括 0)，直到指令明确地给它一个值为止。在逻辑比较中引用未初始化或未声明的标志变量时(像 If Not state then，特殊地，if state 不等于任何东西)，逻辑表达式 Not State 的值为 True，就像 if state＝0，或 False 或 NO 那样。

对 Const 语句使用 LEFT 和 RIGHT 作为常数有些危险，因为这些字也是 Visual Basic 词汇中的保留字。Left 和 Right 是 Left $ ()和 Right $ ()函数的一部分(现在，在所有字符串函数中，$ 是任选的)，而. Left 属性则是图形对象的坐标系统的一部分。

为 ButtonPoint_Click()建立的标志变量是 point_lock，它的值可以置为 True 或 False。估计只有这个过程利用 point_lock，可把它声明为 Static。这里是该过程的代码：

```
Sub ButtonPoint_Click ()
Static point_lock As Integer
If point_lock = False Then
    Readout. Caption = Readout. Caption + "."
    point_lock = True
End If
End Sub
```

这个过程记下 Point_lock 的值。一旦执行条件子句之后，它将不能再被执行。直到某些其他过程把 Point_lock 置为 False 为止。可以想像，那可能是 Clear 按钮的过程。

这会引起另外一个问题：程序怎样才知道什么时候一个数结束而另一个数开始呢？显然功能按钮必须发送某些信号：当计算器的用户揿压“plus”或“times”时，这意味着他(她)将要把当前值键入显示器。

首先，我们引进一个作为标志变量的 ready，当键入的值正常终止时把它置为 True。可用以下代码成功地实现：

```
Sub Button7_Click ()
If Len(Readout. Caption) < 20 and ready = False Then
    Readout. Caption = Readout. Caption + "7"
Else
    Readout. Caption = "7"
End If
End Sub
```

增加第二行使之不可能把当前值加“7”，除非 ready＝False。如果附加到数学功能按钮的过程把 ready 置为 True，则下一个打入的“7”，将清除读出屏，只留下了值中的第一个数字“7”。

重温这个过程可得出以下结论：为什么要用 If—then 语句进行两次比较(通过 And 连接)，什么时候你可以用一个语句代替？如果 ready 是读出屏中字符的总数(以前用 Len (Readout. Caption)表示)并且 ready 的值为正(非零)，它必然是正在接受的输入。如果用户揿压了功能按钮，终止值的登录，该按钮的_Click 过程可以通过置 ready 为 0(或假，但由于 ready 从未置为 True，这种语法可能有点麻烦)，发出终止的信号。

技术注记 如果它的正/零值可表示 True/false 二元状态，可以把该变量看成是“虚”标志。

考虑把变量 ready 作为虚标志实现之后，过程变成下面的样子：

```
Sub Button7_Click ()
If ready < 20 Then
    Readout. Caption = Readout. Caption + "7"
    ready = ready +1
Else
    Readout. Caption = "7"
    ready = 1
End If
End Sub
```

完成的 Sub ButtonPoint_Click()过程如下：

```
Sub ButtonPoint_Click ()
Static point_lock As Integer
If point_lock = False Then And ready < 20 Then
    Readout. Caption = Readout. Caption + "."
    point_lock = True
    ready = ready + 1
End If
assess_readout
End Sub
```

对于 Visual Basic，在行中发现的不是机器已有词汇的术语，就认为是对过程的调用。例如，调用 assess_readout。

下面是完整的过程：

```
Sub assess_readout ()
assess_value = Val(Readout. Caption)
End Sub
```

这里只有一条指令，其目的是为了把正文框的说明词翻译为数值。在当前情况下不一定要写成过程，提供 Sub assess_readout 的原因是为了使程序在以后更容易更新。那时，实现值的转换过程远比单条指令复杂得多。

如果只编写 Expressor I，我可能取消这个过程并用它的指令替代它，我原来就是这样写的。坦白地说，在当前的程序正文中该过程完全是不必要的。但是，提供 Sub assess_readout ()从成功修改 Expressor 中得到补偿：留下以后程序容易更新的手段。