程序设计方法学

Chengxu Sheji Fangfaxue

李传湘
陈世鸿
刘海青
编著

武汉大学出版社
WUHAN DAXUE CHUBANSHE

图书在版编目(CIP)数据

程序设计方法学/李传湘,陈世鸿,刘海青编著.—武汉:武汉大学出版社,2000.12
ISBN 7-307-03105-1

Ⅰ.程… Ⅱ.①李… ②陈… ③刘… Ⅲ.程序设计—方法—高等学校—教材 Ⅳ.TP311.11

中国版本图书馆 CIP 数据核字(2000)第 51414 号

责任编辑:郭志安 责任校对:杜 强 版式设计:支 笛

出版:武汉大学出版社 (430072 武昌 珞珈山)
(电子邮件:wdp4@whu.edu.cn 网址:www.wdp.whu.edu.cn)
发行:新华书店湖北发行所
印刷:湖北省通山县印刷厂
开本:850×1168 1/32 印张:10.75 字数:277 千字
版次:2000 年 12 月第 1 版 2000 年 12 月第 1 次印刷
ISBN 7-307-03105-1/TP·91 定价:16.50 元

前 言

编著此书的目的在于强调这样一个观点：高品质的软件只有基于软件学的基本理论才成为可能。特别是进入新的世纪之后，计算机软件将渗透到社会的各个领域。社会对软件的需求量还会迅猛增长，软件的开发速度、质量及可靠性将成为人们密切关注的问题。软件的差错将会给社会带来严重的、有时甚至是灾难性的后果。

基于以上观点，我们编著的这本书具有如下特点：

(1) 对计算机科学的软件的基本理论体系作了系统的讨论；

(2) 概要介绍了基本理论的发展过程，重点概述了其间的成果；

(3) 讨论了基本理论及其发展过程对软件实践的深刻影响；

(4) 着重说明了如何以基本理论为基础，建立软件的计算模式、算法结构与程序的形式表示，以及对程序进行正确性证明与有效性估算及测试。

按 N. Wirth 的命题“Programs = Algorithms + Data Structures”，全书 12 章可分为三个部分：控制部分、基本数据结构部分和程序部分。

在引论中讨论了三个基本问题：(1)问题求解可行性理论；(2)问题求解的基本原则；(3)问题求解的一般方法：抽象化与结构化。

在每个部分，首先讨论了其数学模型，并以此为基础，引出与之对应的基本原理、一般方法、结构形式、表示方法及其在程序设计语言中的形式表示。

在控制部分，系统介绍了三种基本的控制模型(有限自动机、下推自动机和图灵机)及其关系；着重说明了三种数学模型的计算能力的等价形式语言体系及控制结构的发展过程。

在基本数据结构部分，讨论了数据的数学模型、数据类型的定义和表示方法，着重说明了抽象数据类型的定义与封装，详细讨论了抽象数据封装在程序设计语言中的机制与实现。

程序部分的讨论基于第一部分。在第一部分关于控制的讨论中，论证了图灵机是现代计算机的计算模型，图灵机的计算能力是现代计算机所能达到的计算能力的理论极限，图灵机施行的运算也是现代计算机能够完成的操作运算。在图灵机中，引导运算的核心是程序，程序是图灵机计算模型的形式表示。程序体现了问题求解过程的算法结构。所以，在这一部分，重点讨论了算法设计的一般问题，通过若干例子，说明了如何从实际问题分析中提取该问题的最佳算法结构的方法，介绍了算法的两种常见的结构(递归与迭代)以及它们生成的基本原则，同时还讨论了程序正确性的测试与形式证明和程序复杂性与有效性分析等问题。

该书是在作者从事多项国家自然科学基金项目、国家"863"高技术项目的研究成果和从事十余年程序设计方法教学的讲稿的基础上撰写的，其目的在于能为读者建立软件的理论基础提供帮助。

本书是第一次出版，由于作者的水平有限，其中错误在所难免，真诚欢迎读者的指正与批评，以期再版时得以改进与完善。

编 著 者

1999.12.20

第1章 引 论

1.1 引言

所谓程序设计方法学，是指程序设计的基本原则。我们希望通过本书，使读者能够理解，对于一个待求解的问题，当构造一个有效的良结构的计算程序时，必须遵循一组基本原则和基本设计步骤。基于这些基本原则，就能够判断程序的正确性，同时还可以提高程序的效率。本书不着重介绍程序设计的个别技巧和方法，因为这些内容属于软件工程的研究范畴。

本书坚持 W. A. Wulf 等学者的基本观点，强调程序的数学基础，从整体上把程序看成是控制部分与数据结构的结合体。因此，本书首先讨论控制部分与数据结构，然后再讨论程序。

本章将简要说明三个问题：

- 问题求解的可行性；
- 问题求解的基本原则；
- 程序的抽象化与结构化原则。

最后一节，介绍本书的基本结构。

1.2 问题求解的可行性

对任何问题的程序设计，是在已预先确定这样一个问题是可以求解的前提下进行的。这里，所谓一个问题的解是指，对应于该

问题的一组已知的输入数据，存在一个符合该问题要求的一个或一组结果数据，这些结果数据称为该**问题的解**。在数学上往往要讨论这个解的存在性。只有问题的解存在，才谈得上求它的解。否则，问题是不可解的，也就不存在计算该问题的程序。在现实世界中，确实存在着一类不能计算的问题。

从计算理论的观点看，一个问题的函数变换有解，且存在一个程序，通过这个程序计算的值恰好等于这个函数的值，则这个函数才是可计算的。**问题求解的可行性**也就是问题的可计算性。

对于一个求解的问题，将其一组已知数据变换成对应的结果数据，这种变换称为一个**函数**。一个问题的所有已知数据的集合称为该问题的定义域 D，对应的所有结果数据的集合称为它的值域 V。若以符号 f 表示该问题的变换函数，则由函数定义，有

$$f: D \to V$$

对于$\forall\ X \in D, \exists\ Y \in V$，使得

$$f(X) = Y$$

这里，X 常为一多维向量，即

$$X = (x_1, x_2, \cdots, x_n) \qquad x_i \in D_i, i = 1, 2, \cdots, n$$

于是，函数 f 的定义域 D 满足关系式

$$D \subseteq D_1 \times D_2 \times \cdots \times D_n$$

若 D_i 是同一类型 D_t，则可以简写为

$$D \subseteq D_t^n$$

函数是对一个问题求解的数学描述，它可以是一个数学公式，或以某种形式表示的结构，如算法结构。这个函数公式或结构称为**问题求解的计算模型**。函数有许多性质值得注意和理解，在此不做深入讨论，而仅说明其**可计算性**。一个问题的函数存在，并不意味着它就是可计算的。计算理论就是研究函数的可计算性。

为了说明函数的可计算性，先系统地了解一下程序的概念。

一般来说，一个**程序** P 是由某种语言 L 的语句表示的有穷语句序列，即

$$P::=S_0,S_1,\cdots,S_n,\quad S_i\in L,i=0,1,\cdots,n$$

P 由 $n+1$ 个语句表示，这 $n+1$ 个语句是按某种顺序排列的，n 是一个有限值，$n+1$ 称为程序 P 的长度，或程序的规模。

程序在运行时，作用于三类变量，它们分别是：输入变量 x_1，$x_2,\cdots,x_k$，中间变量 $Z_1,Z_2,\cdots,Z_n$，及输出变量 $y_1,y_2,\cdots,y_m$。

程序在运行的每个时刻，其某个(些)语句作用于这些变量，在计算机中形成一个瞬时的计算状态 S_{ti}，因此，程序的计算过程就是计算机从一个计算状态 S_{ti} 到下一个计算状态 S_{ti+1}。设想从程序的第一个语句 S_0 的计算所对应的状态 S_{t0} 开始，其中经历若干语句按某种顺序计算到达终止状态 S_{tr}。所以，一个程序又可定义为一个状态序列：

$$P:S_{t0},S_{t1},\cdots,S_{tr}$$

其中 S_{t0} 称为初始状态，S_{tr} 为终止状态，S_{ti+1} 称为 S_{ti} 的后继状态。在这里，应该指出：不同的输入数据，它们在程序 P 中的状态序列可能是不同的。

关于状态，联系到三类变量，可以定义为一个向量 $V=(V_1,\cdots,V_r)$。V 表示某时刻 t 程序中所有变量取值的集合，即令 $r=k+n+m$，且

$$\begin{cases} V_1=x_1,\cdots,V_k=x_k \\ V_{k+1}=z_1,\cdots,V_{k+n}=z_n \\ V_{k+n+1}=y_1,\cdots,V_r=y_m \end{cases}$$

是这些变量在 t 时刻所取之值。

一个问题求解是可行的，则意味着表示它的计算模型的函数是可计算的，它的计算程序是一个有限、有序的状态序列。程序的状态序列表示问题计算模型的动态过程，而程序的语句表示计算模型的静态描述。程序的终止状态，使输出变量 Y 取一个终值 $\psi_P(V_1,\cdots,V_k)$，即：

$$Y=\psi_P(V_1,\cdots,V_k)$$

其中，V_i 是 x_i 所取之输入值($i=1,\cdots,k$)，$Y=(y_1,\cdots,y_m)$。

至此，关于问题求解可有如下的形式化定义：

定义 1.1 一个已知问题的解存在的必要充分条件是

$$f(V_1,\cdots,V_k) \equiv \psi_P(V_1,\cdots,V_k)$$

其中，f 是该问题的映射函数，ψ_P 是计算该问题的程序 P 的计算值，V_i 是 X_i 的输入初始值，

$$X_i \in D_i, i = 1,2,\cdots,k$$

且

$$Y = \psi_P(V_1,\cdots,V_k) \in V$$

此时 f 称为**可计算函数**。

一般情况下，可计算函数可以定义如下：

定义 1.2 一个已知的部分函数 $g(x_1,\cdots,x_m)$，如果存在一个程序 P 计算该函数，则函数 $g(x_1,\cdots,x_m)$ 称为**部分可计算**，此时，有

$$g(V_1,\cdots,V_m) = \psi_P(V_1,\cdots,V_m) \tag{1.1}$$

式(1.1)对于所有的值 $V_1,\cdots,V_m$ 皆成立，同时，它意味着：当 $g(V_1,\cdots,V_m)$ 和 $\psi_P(V_1,\cdots,V_m)$ 有定义时，则两边取相同的值；当其中一个无定义时，另一个也无定义。

如果 $g(x_1,\cdots,x_m)$ 是全函数，且部分可计算，则函数 $g(x_1,\cdots,x_m)$ 称为**可计算**。

部分可计算函数也称为**部分递归函数**，可计算函数称为**递归函数**。

从定义 1.2 出发，可以进入计算理论的全程讨论，但这已超出了本书的范围。我们仅借此定义，转入到程序设计方法学的讨论。

这个定义说明，一个问题的求解是可行的，则它存在某种形式的数学模型——函数，即存在一个程序，使得程序的计算结果即为函数的值。因此，要对一个问题求解，首先必须构造这个问题的计算模型。

以往，不少人在面对一个新问题时，一开始就着手编制求解该问题的计算程序，这种编制程序的方法称为试错法。即，首先为该

问题设置一个初始程序，然后通过反复计算，找出其错误，修改已有的程序，希望最终得到完满的结果。这样的程序，即使生成了也必然会包含许多错误，有时甚至是严重的灾难性错误。这种修修补补拼凑的程序必然难以理解、维护，且效率极低。

问题求解过程或程序设计过程，反映一种逻辑协调的结构化过程，可表示为一种工程结构。以这种方法作指导，对问题进行系统说明、分析设计和编程实现。这样设计的程序具有如下特征：

- 易于理解，结构透明；
- 以模块实现程序结构，易于维护和排除故障，可靠性增强；
- 模块化结构易于达到高效率；
- 程序开发时间比试错法显著减少。

1.3　问题求解的基本原则

定义 1.1 说明，一个可求解的问题，它的函数变换与程序满足关系

$$f(V_1,\cdots,V_n)=\psi_P(V_1,\cdots,V_n)$$

其中函数是问题的计算模型。程序表示问题的计算过程，它计算的值完全等于函数变换所得到的值，且在函数的定义域中，如果一个元 $x\in D$ 对 f 是有定义的，则 $\psi_P(x)$ 也是有定义的，反之，x 对 f 没有定义，则 $\psi_P(x)$ 也没有定义，所以，函数的变换过程和程序的计算过程是完全等价的。事实上，程序的计算过程可以认为是函数变换用计算机求解的实现，而函数表示的计算模型又是程序设计的基础。图 1-1 进一步表明问题求解的逻辑关系。

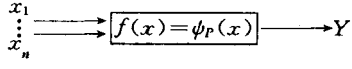

图 1-1　问题求解的逻辑模型

图 1-1 表示输入数据 X 和输出数据 Y 的逻辑关系，同时也说明，Y 值是通过函数 f 的一个变换过程，由输入 $X=(X_1,\cdots,X_n)$ 生成的。为实现这个变换过程，并最后把这个变换过程转换成可以计算的程序，必须遵循以下的**问题求解的基本原则**：

(1) 正确理解和精确定义求解的问题；

(2) 识别和列举问题求解的所有输入数据和最终的输出数据，并设定对应的输入变量和输出变量；

(3) 依据输入数据和输出数据，进一步精确定义问题，并确定输入数据是否足以完成问题的求解；

(4) 为问题的求解设定一个步骤序列，描述问题的函数变换，这个步骤序列称为问题求解的第一级算法；

(5) 求精算法，即对复杂的步骤回到第(1)步，再分解定义出相应的中间变量，逐步形成多级算法，一直到算法易于用某种程序设计语言表述的程度为止。在算法求精过程中，即时引入程序计算的中间变量；

(6)把最终的算法转换成程序。

问题求解的基本原则，实际上是问题的函数生成过程，或者说，是问题求解的算法生成过程，并最终生成程序的基本原则。

首先简单分析一下其正确性，然后再说明使用这些基本原则生成算法的过程。

基本原则的正确性，乃是基于计算理论有关函数构造的基本理论。这里，仅介绍其基本概念和基本结论，不作详细分析和证明，目的在于集中对这个基本原则本身的理解。

计算理论认为，一个函数可以是由其他函数组合而成，或由自身递归生成。

如果一个函数的输出是另一个函数的输入，则这两个函数构成一个组合函数。如有函数 f 和 g，且 g 的输出是 f 的输入，则

$$h(x)=f(g(x))$$

$h(x)$是由 g 和 f 构成的组合函数。

一般地，组合函数可形式化表述如下：

定义 1.3 若函数 f 有 k 个变量，函数 $g_1,g_2,\cdots,g_k$ 是 n 个变量的函数，令

$$h(x_1,\cdots,x_n)=f(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n))$$

则 h 是 f 和 $g_1,g_2,\cdots,g_k$ **的组合函数。**

在该定义中，$f,g_1,\cdots,g_k$ 不一定是全函数。但是，如果 $f,g_1,\cdots,g_k$ 有定义，即存在 $z_1=g_1(x_1,\cdots,x_n),\cdots,z_k=g_k(x_1,\cdots,x_n)$，则 $f(z_1,\cdots,z_k)$ 也是有定义的。

由此定义可引申出定理 1.1。

定理 1.1 如果 h 是(部分)可计算函数 $f,g_1,\cdots,g_k$ 的组合函数，则 h 也是**部分可计算函数。**

这个定理的正确性在此不予讨论，请参见[5](以下定理同此)。

在定理 1.1 中，若 $f,g_1,\cdots,g_k$ 是全函数，则 h 也是全函数。

函数的另一种构造，就是递归构造。递归构造是表述函数以其自身作为因子组合。它的基本形式是：

$$\begin{cases} h(0)=K \\ h(t+1)=g(t,h(t)) \end{cases} \tag{1.2}$$

在式(1.2)中，当递归变量 $t=0$ 时，函数 $h(0)$ 取一个确定的值 K，g 是两个变量的全函数。h 称为由 g 生成的递归运算。

对于式(1.2)，又有定理 1.2。

定理 1.2 若 h 是由 g 得到，(1.2)式成立，且 g 是可计算的，则 h 也是可计算的。

对于递归构造，同样可以给出一般的定义。

定义 1.4 若 h 是 $n+1$ 个变量的函数，f 是 n 个变量的全函数，g 是 $n+2$ 个变量的全函数，h 是由 f 和 g **递归生成**，则表示为：

$$\begin{cases} h(x_1,\cdots,x_n,0)=f(x_1,\cdots,x_n) \\ h(x_1,\cdots,x_n,t+1)=g(t,h(x_1,\cdots,x_n,t),x_1,\cdots,x_n) \end{cases} \tag{1.3}$$

对于定义 1.4 的一般递归运算，同样可引申出如下结论。

定理 1.3 *若 h 是由 f 和 g 以式(1.3)生成的，且 f 和 g 函数可计算，则 h 也是可计算的。*

函数构造的概念及其三个基本结论，是这个基本原则正确性的基础，同时，也为基本原则提供了指导。即运用基本原则进行函数分解时，必须遵循三个定理中关于函数的约束。

下面，形式化说明如何运用基本原则构造问题的算法。本节的最后，还要介绍将问题分解形成算法的过程表示。

算法构造也是一个反复进行的递归过程，这一过程是在问题正确定义的前提下开始的，即首先遵循问题求解的基本原则第(1)、(2)、(3)条，对问题或子问题进行正确地说明和定义，然后才进入原则(4)，这条原则的功能在于形成问题或子问题的第一级算法。其方法就是对问题的函数或问题本身进行分解，把它分解成若干子函数或子问题。例如，欲研究一部汽车的运行过程，无须首先去研究组成它的几百个零件的性能，而是先把它分解成若干个子问题或子部件。如汽车可分为五个部分，分别是：发动机、传动装置、轮、底盘和控制器。这五部分在汽车运行过程中提供不同的功能，它们的协同运动才使汽车得以正常运行。因此，如果以 h 表示汽车的运动函数，g_1, g_2, g_3, g_4, g_5 分别表示这五个部分的运动函数，若 $g_1, \cdots, g_5$ 是 n 个变量的函数，则汽车运动函数 h 可以表示为：

$$h(x_1, \cdots, x_n) = f(g_1(x_1, \cdots, x_n), \cdots, g_5(x_1, \cdots, x_n))$$

其中，f 是一个函数，表示施加于 $g_1, \cdots, g_5$ 的运算的逻辑关系。

在原则(4)中，第一级算法的形成，就是把问题分解成子问题，并把这些子问题按其逻辑关系组合起来，施以规定的运算。

这样，一个原本复杂的问题，通过原则(4)，就化解成对子函数的结果的运算或对子问题结果的处理。如对汽车运动的讨论，就化解成为对它的五个部分运动以及它们协同运动的讨论。

显然，一个复杂的问题，在作了第一次分解后，得到一组子问

题或子函数。在这些子问题或子函数中，有一些仍很复杂，有一些可能已经很简单，无须再分解。但对那些仍很复杂的子问题或子函数，还需作进一步的分解，把它们再分解成子子问题或子子函数，这就是原则(5)规定的算法求精的含义。这一个原则确定那些有待于进一步分解的子问题，并保持一定顺序，使它们一个个地都能接受再处理，也就是说，对每个再处理的子问题，使它回到原则(1)，并依次进行到原则(4)。在原则(4)，对进入的子问题进行分解，形成算法，又进入到原则(5)，而原则(5)的任务仍然是要判别此时的子子问题或子子函数是否还有待于再求精，若是，则进一步对这些子问题求精，否则进行下一个有待求解的子问题的继续求精处理。

综上所述，该基本原则提供了一个建立问题求解的全过程，也就是问题分解的多级算法结构。这个过程一直进行到所有子问题或子函数不用再分解(已经足够简单)，或者已无法再分解成其他子函数或子问题为止。

最后，原则(6)规定把已形成的算法转化成程序，即以某种程序设计语言表示这个算法结构成为程序。

上述问题求解过程可以表示为算法 1-1 所示的递归分-合算法。

算法 1-1　问题求解的递归分-合算法

```
PROCEDURE Solve(X);
  BEGIN
    IF X 可以分解成 n 个部分(n≥2)
      THEN BEGIN
             分解 X 成 X1,…,Xn;
             Solve(X1);…;Solve(Xn);
             组合 X1,…,Xn 的解成 X 的解
           END
      ELSE 求解 X
  END;
```

1.4 抽象化概念

所谓**抽象化**，是指在约定的环境中，提取事物的本质，隐去其内部特征。事物的本质特征，在其所处的环境中，体现事物的个性，事物的个性是在环境中识别事物的标志。被抽象化的事物称为**抽象体**或**模块**。**结构化**是指以模块为构件，按一定逻辑关系组合成某种结构。

抽象化是人们认识事物的一种主要思维方式，是引导软件学理论不断发展完善的基本概念和基本方法。软件抽象包含了数据抽象、控制抽象和程序结构的抽象。

上一节，讨论汽车运动过程的问题时，就是运用了这种思维方式。从宏观上，运动过程可看成是汽车的五个组成部分协同运动的组合，从而分解成五个组成部分，它们为运动过程提供的作用分别是：

- 发动机提供驱动机车的动力；
- 传动装置把动力传输到轮子；
- 轮提供底盘、方向控制和运动滚动支撑；
- 底盘提供结构的依托和负载的支撑；
- 控制器控制运动的方向和力。

这五个部分协同工作，完成汽车的运动过程，但各自具有独立的功能（过程），可以视为系统的一个独立事物或模块，可各自分别加以研究。这样，将有助于更深刻地认识汽车的运动过程，且易于提高其功能与效率。

这就是所说的在约定环境中（汽车的内部世界），运用抽象化的方法，把它分解成了五个组成部分。

由这个例子引申到问题求解原则，则可以认为，问题求解的基本原则就是对问题内部进行抽象化，使其分解出或抽取出若干个具有一定独立性又彼此相关的子问题或子函数。问题求解基本原

则反复运用分解过程，实质上是在重复做一件事，即对问题或子问题在其内部实现抽象化。

实际上，抽象化也是降低程序设计复杂性的一种基本手段。

程序的复杂性伴随着程序的规模而迅速非线性地增大。例如，假设一个100行的程序需要花一周的时间，那么对于一个中等规模约5000行(100行程序的50倍)的程序，它所花费的时间是否也是50倍，即大约一年的时间呢？软件工程关于模块划分的理论告诉我们，情况并非如此。模块内聚度和模块间耦合度是模块独立性的两个主要指标。模块内聚度降低带来的复杂性的降低有可能被模块间耦合度的增加所抵销，甚至导致复杂性的增加。这就是说，如果能把5000行的程序分解成50个100行的小程序，这些小程序是彼此联结着的，分得越细，这种联结越复杂，尽管每个100行的小程序只要1周的时间，但是除了这50周的时间之外，还必须花更多的时间把它们联结起来，因此，完成整个5000行程序的设计，就不仅仅是50周的时间所能完成的，可能需要两年甚至更长的时间。

控制程序的复杂性，避免它随着规模增大而非线性上升，是软件设计的核心问题。解决这个问题的基本方法就是运用抽象化的方法，对问题实现模块化划分。

模块化划分，形成问题的子问题，每个子问题是相对独立的，这些问题有自己的复杂性，但相对整个问题的复杂度要低得多。从程序结构来看，每个子问题形成整个程序结构的一个构件，这个构件称为一个模块。程序的算法结构，就是一个由模块联结成的层次结构。在软件工程中，把这种设计方法归结为软件工程程序设计方法学，该方法学的基本表述为：自顶向下，逐步求精，模块化层次结构设计。

上一节阐述的问题求解的基本原则与软件工程学是等价的。从问题到子问题、到子子问题的逐层分解，就体现了自顶向下、逐步求精的思想。把每次划分的子问题看成一模块，则问题与划分的

子问题就构成了一个二层联结，如图 1-2 所示。

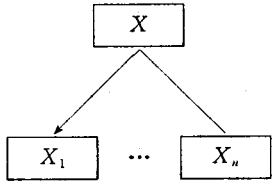

图 1-2 模块层次结构

在图 1-2 中，X 称为父结点，$X_1, X_2, \cdots, X_n$ 称为子结点。其中，任何一个子结点 $X_i(i=1,\cdots,n)$，如果仍很复杂，又可再分解成 $X_{i1}, \cdots, X_{ik}$ k 个子结点，对每个 $X_{ij}(j=1,\cdots,k)$ 再分解下去，直至所有的子结点不能或不用再分解为止，即生成了问题 X 的模块的层次结构，这个模块层次结构就是 X 的映射函数。它的正确性仍然是基于计算理论的函数构造的三个基本定理。只要这种分解在有限次内结束，其正确性就易于用这三个定理证明，在此不详细说明。

认识世界的客观规律是求解现实问题的基本前提。电子工程师有欧姆定律，物理工程师有力学定律，机械工程师有牛顿定律，软件工程师应该遵循什么定律呢，这就是本节最后要讨论的一个重要问题，即抽象的实现问题，也就是面向什么因子来抽象。站在不同基点(因子)进行抽象，将导致不同的程序设计方法学。

根据 N. Writh 关于程序的定义，**程序＝算法＋数据结构**，表示一个程序由两个部分组成。其中，“算法”表示问题的算法结构，它通常描述程序中控制信息流动的路径，“数据结构”表示程序中出现的所有数据结构，也就是控制信息的流动引起在相应路径结点上的操作规程施加于到达此点的数据结构，并对该数据进行处理。Writh 关于程序的定义就是软件工程师在设计软件(程序)时

应遵循的定律。图 1-3 表示了程序中控制信息、操作与数据结构的示意图。

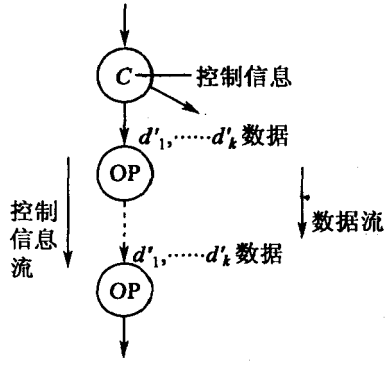

图 1-3　程序运行的信息流程图

在图 1-3 中，C 为控制信息，它决定通过该结点后，计算路径的选择，在图中同时有两股信息流运动，即控制信息流和数据流，在每个结点还有规定的相应操作 OP（操作）、数据和对象（功能＋数据）。在软件工程中，面向不同的因子形成三种不同的程序设计方法，即**面向功能的设计方法，面向数据的设计方法和面向对象的设计方法**。这三种设计方法由于面向不同的因子，而生成三种类型的模块。面向功能的设计方法以问题的分解为基础，每个模块是一个具有独立功能的模块；面向数据的设计方法，是以问题的数据组织为基础，分解的模块是数据结构模块；面向对象的设计方法，是把问题分解成一个个对象，对象是算法加数据结构的封闭单元。因此，在面向对象的设计方法中，应以如下方式看待 Wirth 定律：**对象＝算法＋数据结构，程序＝对象＋对象联结**。联结过程遵循一个确定的算法。