第5章 算法与程序设计

内容与要求

本章介绍算法的基本概念，算法的复杂度的概念和意义(时间复杂度与空间复杂度)、程序设计方法与风格、结构化程序设计、面向对象的程序设计方法、对象、方法、属性及继承与多态性。

算法的基本概念：其中包括算法的基本特征、基本要素、基本设计方法和算法的时间复杂度和空间复杂度。

程序设计的方法与风格：其中包括源程序文档化、数据说明的方法、语句的结构、输入和输出的风格。

结构化程序设计：其中包括程序设计的原则、结构化程序的基本结构、结构化程序设计应注意的问题。

面向对象的程序设计：其中包括面向对象方法和面向对象方法的概念。

重点和难点

本章重点是算法的基本概念、程序设计方法、结构化程序设计、面向对象的程序设计方法。难点是算法的时间复杂度与空间复杂度。

算法是解决问题的步骤，程序是算法的代码实现。算法要依靠程序来完成功能。程序需要算法作为灵魂。

程序是结果，算法是手段(为编写出好程序所使用的运算方法)。同样编写一个功能的程序，使用不同的算法可以让程序的体积、效率差很多。所以算法是编程的精华所在。

5.1 算 法

5.1.1 算法的基本概念

所谓算法是指解题方案的准确而完整的描述。

对于一个问题，如果可以通过一个计算机程序，在有限的存储空间内运行有限长的时间而得到正确的结果，则称这个问题是算法可解的。但算法不等于程序，也不等于计算方法。当然，程序也可以作为一种描述，但通常还需考虑很多与方法和分析无关的细节问题，这是因为在编写程序时要受到计算机系统环境的限制。通常程序的编制不可能优先于算法的设计。

1. 算法的基本特征

作为一个算法，一般具有以下几个特征。

(1) 可行性(Effectiveness)。针对实际问题设计的算法，人们总是希望得到满意的结果。但一个算法又总是在某个特定的计算工具上执行的，因此，算法在执行过程中往往要受到计算工具的限制，使执行结果产生偏差。例如，在进行数值计算时，如果某计算工具具有7位有效数字(如程序设计语言中的单精度运算)，则在计算下列三个量

$$a=10^{12},\ b=1,\ c=-10^{12}$$

的和时，如果采用不同的运算顺序，就会得到不同的结果，即

$$a+b+c=10^{12}+1+(-10^{12})=0$$

$$a+c+b=10^{12}+(-10^{12})+1=1$$

而在数学上，$a+b+c$ 与 $a+c+b$ 是完全等价的。因此，算法与计算公式是有差别的。在设计一个算法时，必须要考虑它的可行性，否则是不会得到满意结果的。

(2) 确定性(Definiteness)。算法的确定性，是指算法的每一个步骤都必须是有明确定义的，不允许有模棱两可的解释，也不允许有多义性。这一性质也反映了算法与数学公式的明显差别。在解决实际问题时，可能会出现这样的情况：针对某种特殊问题，数学公式是正确的，但按此数学公式设计的计算过程可能会使计算机系统无所适从。这是因为根据数学公式设计的计算过程只考虑了正常使用的情况，而当出现异常情况时，此计算机就不能适应了。

(3) 有穷性(Finiteness)。算法有穷性，是指算法必须能在有限的时间内完成，即算法必须能在执行有限个步骤之后终止。数学中的无穷级数，在实际计算时只能取有限项，即计算无穷级数值的过程只能是有穷的。因此一个数的无穷级数表示只是一个计算公式，而根据精度要求确定的计算过程才是有穷的算法。

算法的有穷性还应包括合理的执行时间的含义。因为，如果一个算法需要执行千万年，显然失去了实用价值。

(4) 输入(Input)。一个算法有零个或多个输入，这些输入取自于某些特定的对象集合。

(5) 输出(Output)。一个算法有一个或多个输出。这些输出是同输入有着某些特定关系的量。

一个算法是否有效，还取决于为算法所提供的情报是否足够。通常，算法中的各种运算总是施加到各个运算对象上，而这些运算对象又可能具有某种初始状态，这是算法执行的起点或是依据。因此，一个算法的执行结果总是与输入的初始数据有关，不同的输入将会有不同的结果输出。当输入不够或是输入错误时，算法本身也就无法执行或导致执行有错。一般来说，当算法拥有足够的情报时，此算法才是有效的，而当提供的情报不够时，算法可能无效。

综上所述，所谓算法，是一组严谨的定义运算顺序的规则，并且每一个规则都是有效的，且是确定的，此顺序将在有限的次数下终止。

2. 算法的基本要素

一个算法通常由两种基本要素组成：一是对数据对象的运算和操作，二是算法的控制结构。

(1) 算法中对数据的运算和操作。每个算法实际上是按解题要求从环境能进行的所有操作中选择合适的操作所组成的一组指令序列。因此计算机算法就是计算机能处理的操作所组成的指令序列。

通常,计算机可以执行的基本操作是以指令的形式描述的。一个计算机系统能执行的所有指令集合称为该计算机系统的指令系统。计算机程序就是按解题要求从计算机指令系统中选择合适的指令所组成的指令序列。在一般的计算机系统中,基本的运算和操作有以下四种:

① 算术运算:主要包括加、减、乘、除等运算。

② 逻辑运算:主要包括“与”“或”“非”等运算。

③ 关系运算:主要包括“大于”“小于”“等于”“不等于”等运算。

④ 数据传输:主要包括赋值、输入、输出等操作。

前面提到,计算机程序也可以作为算法的一种描述,但由于在编辑计算机程序通常要考虑很多与方法和分析无关的细节问题(如语法规则),因此,在设计算法之初,通常并不直接用计算机程序来描述算法,而是用别的描述工具(如流程图,专门的算法描述语言,甚至用自然语言)来描述算法。但不管用哪种工具来描述算法,算法的设计一般都应从上述四种基本操作考虑按解题要求,从这些基本操作中选择合适的操作组成解题的操作序列。算法的主要特征着重于算法的动态执行,它区别于传统的着重于静态描述或按演绎方式求解问题的过程。传统的演绎数学是以公理系统为基础的,问题的求解过程是通过有限次推演来完成的,每次推演都将对问题作进一步的描述,如此不断推演直到直接将解描述出来为止;而计算机算法则是用一些最基本的操作,通过对已知条件一步一步地加工和变换,从而实现解题目标。

(2) 算法的控制结构。一个算法的功能不仅仅取决于所选用的操作,而且还与各操作之间的执行顺序有关。算法中各操作之间的执行顺序称为算法的控制结构。

算法的控制结构给出了算法的基本框架,它不仅决定了算法中各操作的执行顺序,而且也直接反映了算法的设计是否符合结构化原则。描述算法的工具通常有传统流程图、N-S结构化流程图、算法描述语言等。一个算法一般都可以用顺序、选择、循环 3 种基本控制结构组合而成。

【例 5.1】 有黑和蓝两个墨水瓶,但却错把黑墨水装在了蓝墨水瓶子里,而蓝墨水错装在黑墨水瓶子里,要求将其互换。

这是一个非数值运算问题。因为两个瓶子的墨水不能直接交换,所以,解决这一类问题的关键是需要借助第三个墨水瓶。设第三个墨水瓶为白色,其交换步骤如下:

① 将黑瓶中的蓝墨水装入白瓶中;

② 将蓝瓶中的黑墨水装入黑瓶中;

③ 将白瓶中的蓝墨水装入蓝瓶中;

④ 交换结束。

【例 5.2】 计算函数 $f(x)$的值。函数 $f(x)$为:

$$f(x)=\begin{cases} bx+a & x \leqslant a \\ ax+b & x > a \end{cases}$$ 其中,a,b 为常数。

本题是一个数值运算问题。其中 $f(x)$代表要计算的函数值,有两个不同的表达式,根

据 x 的取值决定采用哪一个算式。根据计算机具有逻辑判断的基本功能，用计算机算法描述如下：

① 将 a，b 和 x 的值输入到计算机；

② 判断 x 是否$\leqslant a$？如果条件成立，执行第③步，否则执行第④步；

③ 按表达式 $bx+a$，计算出结果存放到 f 中，然后执行第⑤步；

④ 按表达式 $ax+b$，计算出结果存放到 f 中，然后执行第⑤步；

⑤ 输出 f 的值；

⑥ 算法结束。

由上述两个简单的例子可以看出，一个算法由若干操作步骤构成，并且，任何简单或复杂的算法都是由基本功能操作和控制结构这两个要素组成。算法的控制结构决定了算法的执行顺序。

3. 算法设计的基本方法

计算机解题的过程实际上是实现计算机算法的过程；计算机算法不同于人工处理的方法，下面列举出工程上常用的几种算法设计，在实际应用时，各种方法之间往往存在着一定的联系。

(1) 列举法。列举法的基本思想是根据提出的问题，列举所有可能的情况，并用问题中给定的条件检验哪些是需要的，哪些是不需要的。求解不定方程的问题常用此法。列举法是计算机基础算法之一。

列举法的特点是算法比较简单，但当列举的可能情况较多时，执行列举算法的工作量将会很大。因此，在用列举法设计算法时，使方案优化、尽量减少运算工作量是应该重点注意的问题。

列举原理是计算机应用领域中十分重要的原理；因为许多实际问题若采用人工列举是不可思议的，但由于计算机的运算速度快并擅长重复操作，计算机便可以轻而易举地进行大量列举；因此，列举法虽然笨拙、原始、运算量大，但在许多实际问题中(如查找、搜索等问题)局部使用列举法还是十分有效的。

(2) 归纳法。归纳法的基本思想是通过列举少量的特殊情况，经过分析找出一般的关系。从本质上讲，归纳法就是通过观察一些简单而特殊的情况，最后总结出一般性的结论。

归纳是一种抽象，即从一个实际问题的特殊现象中总结出一般关系；但由于在归纳的过程中不可能对所有的因素进行一一列举，因此，最后由归纳得到的结论不是完全可靠的，还需要进一步加以必要的证明。

(3) 递推。递推是指从已知的初始条件出发，逐次推出所要求的中间结果和最后结果。其中初始条件或者问题本身已经给定，或者通过对问题的分析与化简而确定。递推本质上也属于归纳法，工程上许多递推关系式实际上是通过对实际问题的分析与归纳而得到的，因此，递推关系式往往是归纳的结果。

递推算法经常用于数值计算；但对于数值型的递推算法必须要注意数值计算的稳定性问题，因为计算机中数值的表示往往是有界的，而数学意义上数值是无界的。

(4) 递归。人们在解决一些复杂问题时，为了降低问题的复杂程度(如问题的规模等)，一般总是将问题逐层分解，最后归结为一些最简单的问题。这种将问题逐层分解的过程，实际上并没有对问题进行求解，而只是当解决了最后那些最简单的问题后，再沿着原来分解的

逆过程逐步进行综合，这就是递归的基本思想。

递归分为直接递归与间接递归两种。例如有算法 P 和 Q，如果 P 显式地调用自己则称为直接递归，如果 P 先调用 Q，而 Q 又调用 P 则称为间接递归。

(5) 减半递推技术。实际问题的复杂程度往往与问题的规模有着密切的联系，因此，利用分治法解决这类实际问题非常有效。工程上常用的分治法是减半递推技术。

所谓“减半”是指将问题的规模减半，而问题的性质不变；所谓“递推”是指重复“减半”的过程。

【例 5.3】 设方程 $f(x)=0$ 在$[a,b]$上有实根，且 $f(a)$与 $f(b)$异号。利用二分法求其在$[a,b]$上的一个实根。

减半递推过程如下：

首先取给定区间的中点 $c=(a+b)/2$；然后判断 $f(c)$是否为 0，若为 0，则 c 就是根，结束；否则根据以下原则将原区间减半：

若 $f(a)*f(c)<0$，则取原区间的前半部分；

若 $f(b)*f(c)<0$，则取原区间的后半部分。

最后判定减半后的区间长度是否已经很小；若是很小，则取$(a+b)/2$ 为根的近似值，否则重复上述减半过程(很小的概念取决于计算机的精度)。

(6) 回溯法。在工程上，有些实际问题很难归纳出一组简单的递推公式或直观的求解步骤，并且也不能进行无限的列举。对于这类问题，一种有效的方法是“试”。通过对问题的分析，找出一个解决问题的线索，然后沿着这个线索逐步试探，若试探成功，就得到问题的解，若试探失败，就逐步回退，换别的路线再逐步试探。常用于处理复杂数据结构。

5.1.2 算法复杂度

算法的复杂度包括时间复杂度和空间复杂度。

1. 算法的时间复杂度

所谓算法的时间复杂度是指执行算法所需要的计算工作量。算法所执行的基本运算次数与计算机硬件、软件因素无关。算法所执行的基本运算次数与问题的规模有关。对于一个固定的规模，算法所执行的基本运算次数还可能与特定的输入有关。

算法的工作量用算法所执行的基本运算次数来度量。算法所执行的基本运算次数是问题的规模函数。即算法的工作量$=f(n)$。

例如，在 N×N 矩阵相乘的算法中，整个算法的执行时间与该基本操作(乘法)重复执行的次数 n^3 成正比，也就是时间复杂度为 n^3，表示为 $f(n)=O(n^3)$。

在有些情况下，算法中的基本操作重复执行的次数还依据问题的输入数据集的不同而不同。例如在选择升序排序的算法中，当要排序的一组数初始序列为自小至大有序时，基本操作的执行次数为 0；当初始序列为自大至小有序时，基本操作的执行次数为 $n(n-1)/2$。对这类算法的分析，可以采用以下两种方法来分析。

(1) 平均性态(Average Behavior)。所谓平均性态是指在各种特定输入下，用基本运算次数的加权平均值来度量算法的工作量。

设 x 是所有可能输入中的某个特定输入，$p(x)$是 x 出现的概率(即输入为 x 的概率)，$t(x)$是算法在输入为 x 时所执行的基本运算次数，则算法的平均性态定义为

$$A(n)=\sum_{x\in Dn}P(x)t(x)$$

其中 Dn 表示当规模为 n 时，算法执行的所有可能输入的集合。

(2) 最坏情况复杂性(Worst-case Complexity)。所谓最坏情况分析，是指在规模为 n 时，算法所执行的基本运算的最大次数。

$$W(n)=\max_{x\in Dn}\{t(x)\}$$

显然，$W(n)$比 $A(n)$计算容易，$W(n)$更有实际意义。

2. 算法的空间复杂度

一个算法的空间复杂度，一般是指执行这个算法所需要的内存空间。

一个算法所占用的存储空间包括算法程序所占用的空间、输入的初始数据所占用的存储空间以及算法执行过程中所需要的额外空间。其中额外空间包括算法程序执行过程中的工作单元以及某种数据结构所需要的附加存储空间(例如，在链式结构中，除了要存储数据本身外，还需要存储链接信息)。如果额外空间量相对于问题规模来说是常数，则称该算法是原地(in place)工作的。在许多实际问题中，为了减少算法所占的存储空间，通常采用压缩存储技术，以便尽量减少不必要的额外空间。

类似于时间复杂度的讨论，一个算法的空间复杂度作为算法所需存储空间的量度，记作：

$$S(n)=O(f(n))$$

其中 n 为问题的规模(或大小)，空间复杂度也是问题规模 n 的函数。一个算法所占用的存储空间包括算法程序所占的空间，输入的初始数据所占的存储空间，以及算法执行过程中所需要的额外空间。其中额外空间包括算法程序执行过程中的工作单元以及某种数据结构所需要的附加存储空间。如果额外空间量相对于问题规模来说是常数，则称该算法是原地(In place)工作的。在许多实际问题中，为了减少算法所占的存储空间，通常采用压缩存储技术，以便尽量减少不必要的额外空间。

5.2 程序设计的方法与风格

程序设计是一门技术，需要相应的理论、技术、方法和工具来支持，程序设计主要经历了结构化程序设计和面向对象程序设计的发展阶段。

程序设计风格会深刻影响软件的质量和可维护性，良好的程序设计风格可以使程序结构清晰合理，使程序代码便于维护。程序设计的风格总体而言应该强调简单和清晰，程序必须是可以理解的。可以认为，著名的“清晰第一，效率第二”的论点已成为当今主导的程序设计风格。要形成良好的程序设计风格，主要应注重和考虑下述一些因素。

1. 源程序文档化

源程序文档化主要考虑以下几点：

(1) 符号名的命名规则：符号名的命名应具有一定的实际含义，以便理解程序功能。

(2) 正确的程序注释：程序注释一般分为序言性注释和功能性注释。序言性注释常位于程序开头部分，它包括程序标题、程序功能说明、主要算法、接口说明、程序位置、开发简历、程序设计者、复审者、复审日期及修改日期等。功能性注释一般嵌在源程序体之中，用于描述其后的语句或程序的主要功能。

（3）程序的视觉组织：在程序中利用空格、空行、缩进等技巧使程序的层次结构清晰一目了然。

2. 数据说明的方法

（1）数据说明的次序规范化。

（2）说明语句中变量安排有序化。

（3）使用注释来说明复杂数据的结构。

3. 语句的结构

（1）在一行内只写一条语句。

（2）程序编写应优先考虑清晰性。

（3）除非对效率有特殊要求，程序编写要做到清晰第一，效率第二。

（4）首先要保证程序的正确，然后才要求提高速度。

（5）避免使用临时变量而使程序的可读性下降。

（6）避免不必要的转移。

（7）尽可能使用库函数。

（8）避免采用复杂的条件语句。

（9）尽量减少使用“否定”条件的条件语句。

（10）数据结构要有利于程序的简化。

（11）要模块化，使模块功能尽可能单一化。

（12）利用信息隐蔽，确保每一个模块的独立性。

（13）从数据出发去构造程序。

（14）不要修补不好的程序，要重新编写程序。

4. 输入和输出的风格

（1）对所有的输出数据都要检验数据的合法性。

（2）检查输入项的各种重要组合的合理性。

（3）输入格式要简单，以使得输入的步骤和操作尽可能简单。

（4）输入数据时，应允许使用自由格式。

（5）应允许默认值。

（6）输入一批数据时，最好使用输入结束标志。

（7）在以交互式输入/输出方式进行输入时，要在屏幕上使用提示信息明确提示输入的内容与格式，同时在数据输入过程中和输入结束时，应在屏幕上显示状态信息。

（8）当程序设计语言对输入格式有严格要求时，应保持输入格式与输入语言的一致性；给所有的输出加注释，并实际输出报表格式。

5.3 结构化程序设计

5.3.1 程序设计的原则

程序设计的方法的主要原则可以概括为：自顶向下、逐步求精、模块化、限制使用 goto 语句。

（1）自顶向下：即先考虑总体，后考虑细节；先考虑全局目标，后考虑局部目标。这种

程序结构按功能划分为若干个基本模块，这些模块形成一个树状结构。

（2）逐步求精：对复杂问题，应设计一些子目标做过渡，逐步细化。

这种设计方法的过程是将问题求解由抽象逐步具体化的过程。用这种方法分解复杂问题，直到把复杂问题分解为可以直接用程序语言的基本语句结构表达出来为止。这种方法就叫作“自顶而下，逐步求精”。在向下一层展开之前应仔细检查本层设计是否正确，只有上一层是正确的才能向下细化。如果每一层设计都是正确的，则整个算法就是正确的。

（3）模块化：模块化是把程序要解决的总目标分解为分目标，再进一步分解为具体的小目标，把每个小目标称为一个模块。

（4）限制使用 GOTO 语句。

5.3.2 结构化程序的基本结构

1966 年，Boehm 和 Jacopini 证明了程序设计语言仅仅使用顺序、选择和重复（循环）三种基本控制结构就足以表达出各种形式结构的程序设计方法。采用结构化程序设计方法编写程序，可使程序结构良好、易读、易理解、易维护，从而可以提高编程工作的效率，降低软件开发的成本。

1. 顺序结构

顺序结构是一种简单的程序设计结构，顺序结构自始至终严格按照程序中语句的先后顺序逐条执行，是最基本、最常用的结构形式；它是程序设计中的必备；如图 5-1(a)所示。

2. 选择结构

又称为分支结构，它包括简单选择和多分支选择结构，如图 5-1(b)所示。

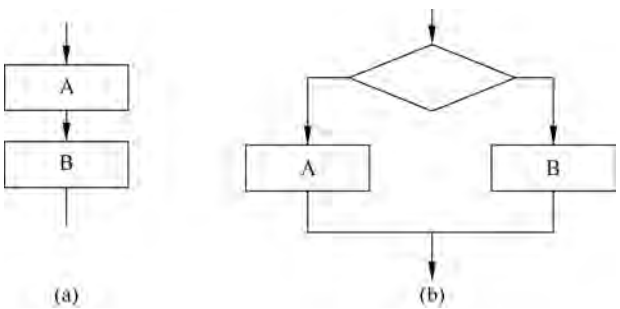

图 5-1　顺序与选择结构

3. 循环结构

循环结构又称为重复结构，它根据给定的条件，判断是否需要重复执行某一相同功能的程序段。在程序设计语言中，重复结构对应两类循环语句，对先判断后执行的循环体称为当型循环结构，对先执行循环体后判断的称为直到型循环结构，如图 5-2 所示。

结构化程序设计的优点：

- 程序的可读性好，易于维护。
- 提高了编程效率，降低了开发成本。

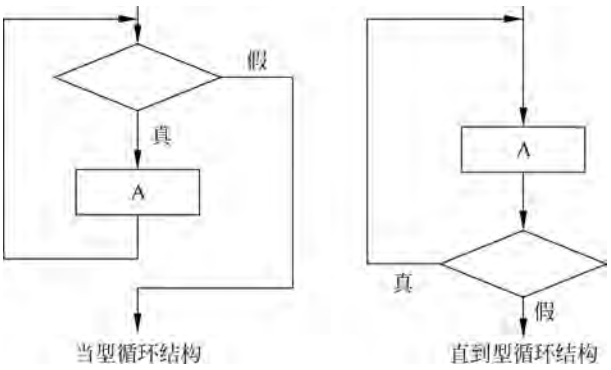

图 5-2　循环结构

5.3.3　结构化程序设计应注意的问题

在结构化程序设计的具体实施中，要注意把握如下要素：

(1) 使用程序设计语言中的顺序、选择、循环等有限的控制结构表示程序的控制逻辑；

(2) 选用的控制结构只准许有一个入口和一个出口；

(3) 程序语句组成容易识别的块，每块只有一个入口和一个出口；

(4) 复杂结构应该用嵌套的基本控制结构进行组合嵌套来实现；

(5) 语言中没有的控制结构，应该采用前后一致的方法来模拟；

(6) 严格控制非结构化语句(如 goto、break、continue 语句)的使用。除非以下情况：

- 使用后可以大大提高程序的效率，而且不但不影响程序可读性，反而使程序机构更加清晰时，才考虑使用；
- 用一个非结构化的程序设计语言去实现一个结构化的构造，当然目前此类情况已是微乎其微。

5.4　面向对象的程序设计

5.4.1　关于面向对象方法

客观世界中任何一个事物都可以被看成是一个对象，对象是现实世界事物或个体的抽象表示，抽象的结果不仅包括事物个体的属性，还包括事物的操作。属性值表示了对象的内部状态。面向对象方法的本质就是主张从客观世界固有的事物出发来构造系统，提倡用人类在现实生活中常用的思维方法来认识、理解和描述客观事物，强调最终建立的系统能够映射问题域，也就是说，系统中的对象以及对象之间的关系能够如实地反映问题域中固有事物及其关系。从计算机的角度来看，面向对象就是运用对象、类、继承、封装、消息、结构与连接等面向对象的概念对问题进行分析、求解的系统开发技术。

面向对象方法有以下几个主要优点：

(1) 与人类习惯的思维方法一致。

(2) 稳定性好。

(3) 可重用性好。

(4) 易于开发大型软件产品。

(5) 可维护性好。

5.4.2 面向对象方法的基本概念

面向对象的程序设计方法中涉及的对象是系统中用来描述客观事物的一个实体，是构成系统的一个基本单位，它由一组表示其静态特征的属性和它执行的一组操作组成。面向对象方法学中的对象是由描述该对象属性的数据以及可以对这些数据施加的所有操作封装在一起构成的统一体。对象可以做的操作表示它的动态行为，在面向对象分析和面向对象设计中，通常把对象的操作称为方法或服务。属性在设计对象时确定，一般只能通过执行对象的操作来改变。对象有一些基本特点：标识唯一性，分类性，多态性，封装性，模块独立性好。

1. 对象(Object)

对象是面向对象方法中最基本的概念。对象可以用来表示客观世界中的任何实体，也就是说，应用领域中有意义的、与所要解决的问题有关系的任何事物都可以作为对象。总之，对象是对问题域中某个实体的抽象。

2. 类(Class)和实例(Instance)

类是对具有共同特征的对象的进一步抽象。将属性和操作相似的对象归为类，也就是说，类是具有共同属性、共同方法的对象的集合。所以，类是对象的抽象，它描述了属于该对象类型的所有对象的性质，而一个对象则是其对应类的实例。如杨树、柳树、枫树等是具体的树，抽象之后得到“树”这个类。类具有属性，属性是状态的抽象，如一棵杨树的高度是10m，柳树是8m，树则抽象出一个属性“高度”。类具有操作，它是对象行为的抽象。

3. 继承(Inheritance)

继承是使用已有的类定义作为基础来建立新类的定义技术。已有的类可当作基类来引用，则新类相应地可当作派生类来引用。面向对象软件技术的许多强有力的功能和突出的优点，都来源于把类组成一个层次结构的系统：一个类的上层可以有父类，下层可以有子类。这种层次结构系统的一个重要性质是继承性，一个类直接继承其父类的描述或特性，子类自动地共享基类中定义的数据和方法。

继承关系模拟了现实世界的一般与特殊的关系。它允许我们在已有的类的特性基础上构造新类。被继承的类称为基类(父类)，在基类的基础上新建立的类称为派生类(子类)。派生类的特性比基类的特性更细致。

继承关系可以表述为：派生类是基类。因此可以说：动物是生物。生物比动物具有更一般的特性。

4. 聚合(Aggregation)

聚合模拟了现实世界的部分与整体的关系。它允许利用现有的类组成新类。比如说汽车，它是由发动机、变速箱、底盘等组成，那么我们就可以利用发动机、变速箱、底盘等类聚合成一个新的类：汽车类。

5. 消息(Message)

消息是一个实例与另一个实例之间传递的信息，它请求对象执行某一处理或回答某一要求的信息，它统一了数据流和控制流。消息中包含传递者的要求，它告诉接受者需要做哪

些处理，但并不指示接受者应该怎么样完成这些处理。消息完全由接受者解释，接受者独立决定采用什么方式完成所需的处理，发送者对接受者不起任何控制作用。一个对象能接受不同形式、不同内容的多个消息；相同形式的消息可以送往不同的对象，不同的对象对于形式相同的消息可以有不同的解释，能够做出不同的反映。一个对象可以同时往多个对象传递消息，两个对象也可以同时向某个对象传递消息。

消息是对象之间交互的唯一途径，一个对象要想使用其他对象的服务，必须向该对象发送服务请求消息。而接收服务请求的对象必须对请求做出响应。

例如：当我们向银行系统的账号对象发送取款消息时，账号对象将根据消息中携带的取款金额对客户的账号进行取款操作：验证账号余额，如果账号余额足够，并且操作成功，对象将把执行成功的消息返回给服务请求的发送对象，否则发送交易失败消息。

6. 多态性（Polymorphism）

多态性是指在一般类中定义的属性或行为，被特殊类继承之后，可以具有不同的数据类型或表现出不同的行为。

多态性机制不仅增加了面向对象软件系统的灵活性，进一步减少了信息冗余，而且显著提高了软件的可重用性和可扩充性。当扩充系统功能增加新的实体类型时，只需派生出与新实体类相应的新的子类，完全无须修改原有的程序代码，甚至不需要重新编译原有的程序。利用多态性，用户能够发送一般形式的消息，而将所有的实现细节都留给接受消息的对象。

例如，在两个类 Male（男性）和 Female（女性）都有一项属性为 Friend。一个人的朋友必须属于类 Male（男性）、Female（女性）二者其一，这是一个多态性的情况。因为 Friend 指向两个类之一的实例。

本章小结

算法是解决问题的步骤；程序是算法的代码。实现算法要依靠程序来完成功能；程序需要算法作为灵魂。

程序是结果，算法是手段（为编写出好程序所使用的运算方法）。同样编写一个功能的程序，使用不同的算法可以让程序的体积、效率差很多。所以算法是编程的精华所在。

算法＋数据结构＝应用程序。

算法是程序设计的核心，算法的好坏在很大程度上决定了一个程序的效率。一个好的算法可以降低程序运行的时间复杂度和空间复杂度。先选出一个好的算法，再配合以一种适宜的数据结构，这样程序的效率会大大提高。

算法和程序都是指令的有限序列。程序是算法，而算法不一定是程序。

【注释】

流程图：以特定的图形符号加上说明，表示算法的图，称为流程图。

N-S 结构化流程图：全部算法写在一个矩形阵内，在框内还可以包含其他框的流程图形式。

算法描述语言：算法可采用多种描述语言来描述，例如，自然语言、计算机语言或某些伪语言。各种描述语言在对问题的描述能力方面存在一定的差异。