第3章　计算机程序与软件开发

3.1　程序与软件

3.1.1　程序与软件的概念

今天，每一个人都知道，计算机工作离不开程序。离开了程序，计算机就只是一堆金属和塑料。但是，程序又是看不见、摸不着的，所以说程序是计算机的灵魂。

但是并非只有计算机的工作要依靠程序。实际上，做任何事情都需要程序。没有程序，将一事无成。例如做菜，离开了一定的过程，做出来的菜能吃吗？只不过，有些程序是写在纸上，有些程序是记在人的大脑中。而计算机的程序是保存在计算机的存储器中。

由于程序已经成为计算机系统的一个不可分割的组成部分，所以把组成计算机的物理部分称为计算机硬件，而把程序、规程等非物理部分称为计算机软件。这也就是说，计算机软件除了包括程序外，还包括其他，如关于程序开发、使用、维护的文档和规程等。这是计算机程序与软件的一个区别，即软件并非仅仅指程序。

此外，软件往往指多个程序的结合，程序是一个命令执行指令。例如：一个游戏软件包括程序(*.exe)和其他图片(*.bmp 等)、音效(*.wav 等)等附件，那么这个程序(*.exe)与其他文件(图片、音效等)在一起合称为一个软件。

3.1.2　虚拟化与智能化：程序的两大基本功能

对于计算机来说，没有程序将一事无成。但是，今天程序的应用并非还仅仅限于纯粹的计算机，有许多机器和设备中也要靠程序控制。这些程序除了要实现具体的概念外，从更高的层次上认识，可以将它们的功能概括为两点：虚拟化和智能化。

1. 程序的虚拟化功能

虚拟化(virtualization)指对于有限的固定的资源根据不同需求通过重新规划以达到最大利用率的技术。而这些技术中最根本的是使用了软件。例如，利用操作系统中的内存调度程序，可以将部分外存当作内存使用。再如 CPU 的虚拟化技术可以用单 CPU 模拟多 CPU 并行，允许一个平台同时运行多个操作系统，并且应用程序都可以在相互独立的空间内运行而互不影响，从而显著提高计算机的工作效率。

如图 3.1 所示是采用计算机虚拟现实技术制作成的一款 NBA 游戏，它可以让游戏者以 NBA 球员身份体验篮球的乐趣。

现在，虚拟化已经成为一个被广泛使用的概念和技术。数字图书馆、数字校园、数字城市、数字地球等，都是虚拟技术的体现。通过虚拟化，人们可以做在真实环境中很难做或无法做的事情。例如，模拟军事演习、模拟地震灾害、用模拟方法设计原子能反应堆的壁厚，以

图 3.1　采用计算机虚拟现实技术制作而成的一款 NBA 游戏

及在游戏中扮演一位元帅、在虚拟现实环境下进行人体手术等。

所以，计算机技术对于人类的最大贡献在于它开辟了一个虚拟世界。而在这个虚拟世界的背后主要是程序在工作。

2. 程序的智能化功能

程序是人解题思路的描述。人编写了程序，然后把程序安装到计算机系统中，就使计算机成为人的代理从而也有了智能。如果一台机器中嵌入了可以执行程序的芯片，这台机器就具有了一定的智能。

3.1.3　程序＝模型＋表现

编写计算机解题程序，首先要对问题进行抽象，建立问题的模型，然后把这个模型逐步向计算机可以进行的操作转换；当这个模型被全部表现为计算机可以进行的操作时，该问题的求解程序就编写好了。所以，程序就是问题模型的计算机操作表现，可以简单地表示为：

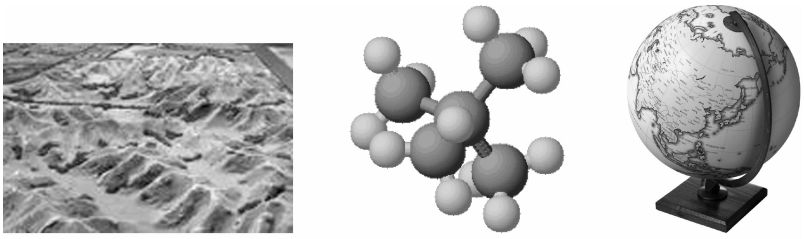
程序＝模型＋表现

1. 模型

模型是对于客观问题抽象的结果。建立模型的目的是为了便于理解、沟通，也便于实现。因为人对复杂问题的理解能力是有限的，不做一些简化，许多大的系统很难理解，达不到共识；不去掉一些枝节，也难以实现。

模型的建立主要取决于两个方面：问题的领域和建立模型所采用的方法，对于不同领域或采用不同的方法，可以建立不同的模型。例如，模拟战争以及建设规划可以采用沙盘模型，如图 3.2(a)所示；进行化学课程教学可以用小球和直棍构建分子结构的实物模型，如图 3.2(b)所示；进行地理教学可以使用地球仪，如图 3.2(c)所示。还有许多问题可以采用公理系统构建的数学模型表示等。为使用计算机进行问题求解而建立的模型通常称为计算模型。

在进行程序设计时，对于不同的客观问题，可以建立不同的计算模型；对于同一客观问题，也可以用不同的方法建立不同的计算模型。从方法的角度，目前广泛采用的计算模型有面向过程的模型(简称过程模型)和面向对象的模型(简称对象模型)。



(a) 军事沙盘　　(b) 分子结构模型　　(c) 地球仪

图 3.2　几种不同的模型

2. 面向过程模型

面向过程模型认为问题从原始态到结果态的变化是通过一个操作过程完成的。为了求解问题，首先需要分析解决问题所需要的操作步骤，然后把这些操作步骤逐步实现。问题的求解模型主要是表现这个操作过程(procedure)或操作流程。在计算机程序中，把操作的对象抽象为数据(data)。所以，过程模型也是用数据流和操作流描述的。

例 3.1　有两个瓶子，一个装有醋，一个装有酒。要求把它们互相换装。

这个问题本身就描述了一个操作过程。不过根据已知条件，首先将两个瓶子用两个变量表示：装醋的瓶子用变量 c 表示，装酒的瓶子用变量 j 表示。于是，初始条件为

$$c=\text{'醋'},j=\text{'酒'};$$

为了交换，要使用第三个瓶子 x。交换的过程用如图 3.3 所示的流程图模型描述。

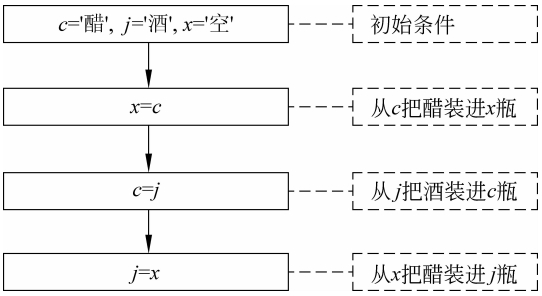


图 3.3　交换两个瓶子内容的流程图模型

所以，过程常常是用将输入转化为输出的步骤描述的。

3. 面向对象模型

面向对象模型是用面向对象的方法建立的计算模型，它认为问题从原始态到结果态的变化是通过组成问题的对象之间的相互作用完成的，而对象间的相互作用是由事件引发的。因此，问题的求解模型主要是表现组成问题的对象，以及相关事件发生的条件和对象的行为。

例 3.2　打手机。这个问题要涉及 4 个对象：两个打手机的人甲和乙、两个手机 A(假定号码为 1333333××××)和 B(假定号码为 1366666××××)。则打手机的过程如下。

(1) 甲执行“拨号”操作，在 A 中输入 B 的号码 1366666××××，即甲通过“拨号”，向

A 传递消息“1366666××××”，A 则启动“暂存”行为。

（2）甲在 A 中按下“呼叫”键，即甲通过按键操作，向 A 传递 “呼叫” 消息（指令），A 则启动“呼叫”行为。

（3）A 按照“1366666××××”找到 B，并向 B 传递“呼叫”消息，启动 B 的“振铃”行为，同时向甲发出“拨号中”的消息，启动甲的“监听”行为，等待接通。

（4）B 用“振铃”向乙传递消息，乙启动按键操作，按下 B 的“通话”键。

（5）B 收到乙的“已接通”消息（指令），停止振铃，启动“通话”操作；同时向 A 发出“接通”消息，使 A 结束“监听”过程，启动“通话”操作。

（6）甲、乙开始通话。

（7）通话中，一方（如甲）向所持手机（如 A）传递“挂断”消息，该手机即执行“挂断”，并向对方手机（如 B）发出“结束”消息。

在这个过程中，每个动作之间有时间序列的关系，如图 3.4 所示。

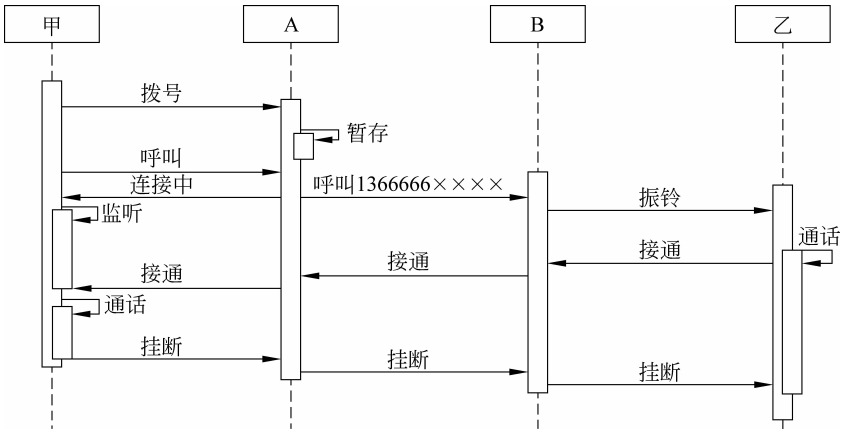


图 3.4　两个人用手机通话过程中的消息传递序列图

对象比流程更加稳定。业务流程的制定需要受到很多条件的限制，甚至程序的效率、运行方式都会反过来影响业务流程。有时候用户也会为了更好地实现商业目的，主动地改变业务流程，而一个流程的变化经常会带来一系列的变化。这就使得按照业务流程设计的程序经常面临变化。此外，对象比流程更加封闭、稳定。因此，对于大型复杂问题采用对象模型比较合适。

4. 表现

将问题的求解模型用计算机能接受的语言描述出来，就成为程序。计算机能接受的语言称为计算机程序设计语言。

课堂讨论

各举一个面向过程和面向对象模型的例子。

3.1.4　程序设计语言

每一个程序都是由人设计的，但要由计算机执行。人熟悉自己的自然语言，而计算机能

理解的是自己的机器语言。那么，人与计算机如何沟通呢？也就是说，人用什么语言编写让计算机执行的程序呢？下面介绍几种程序设计语言。

1. 从机器语言到计算机程序设计语言

计算机诞生之初，人们只能直接用二进制形式的机器语言写程序。机器语言是用 0、1 编码表示的语言，是设计机器时就确定了的。每一条机器语言指令代表了机器可以执行的一个基本操作。使用这样的语言编写程序，难记、难认、难于查错，程序设计效率很低，人们要把大量精力花在对于 0、1 码指令的辨认、记忆和检查上。改进的第一步是把每条机器语言指令用一个便于记忆、辨认的符号代替，就形成了符号语言。表 3.1 是用机器语言和符号语言编写的计算表达式 d＝a×b＋c 的程序。

表 3.1　用机器语言和符号语言编写的计算表达式 d＝a×b＋c 的程序比较

机器语言程序	意　　义	符号语言程序
00000001000000001000	将单元 1000 的数据 a 装入寄存器 0	load 0 a
00000001000100001010	将单元 1010 的数据 b 装入寄存器 1	load 1 b
00000101000000000001	将寄存器 1 的数据 a 与寄存器 0 原有数据 b 相乘	mult 0 1
00000001000100001100	将单元 1100 的数据 c 装入寄存器 1	load 1 c
00000100000000000001	将寄存器 1 的数据 c 加到寄存器 0 原有数据 a×b 上	add 0 1
00000010000000001110	将寄存器 0 里的数据 a×b＋c 存入单元 1110(d)	save 0 d

用符号语言编写的程序，每条指令的意义容易理解和掌握。但是，符号语言指令还是与机器语言指令具有一一对应的关系，仍然是面向机器的，不同的机器有不同的符号语言。编写好的程序，换了一台机器就不能使用，可移植性差。于是后来人们开发了接近人的自然语言的程序设计语言——高级语言。高级程序语言更接近人所习惯的描述形式，更容易被接受，在高级语言（例如 C 语言）的层面上，描述前面同样的程序片段只需一条语句：

```
d=a*b+c;
```

人们把编程语言与机器语言之间的距离称为 Neumann 距离。如图 3.5 所示，高级语言的 Neumann 距离要比符号语言的 Neumann 距离大。

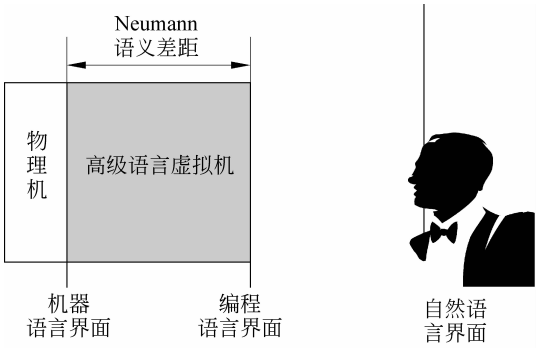


图 3.5　Neumann 距离

2. 高级语言程序从编辑到执行

用高级语言程序编写的程序代码称为源程序代码，源代码是机器不能直接执行的，只有翻译成机器语言表示的目标程序，才能让计算机执行。这个翻译过程依照高级语言的不同，被分为两种方式：编译方式和解释方式。编译方式有点像笔译，是把一个源代码一下子翻译成目标代码；而解释方式有点像口译，即对源代码翻译一句执行一句。

需要说明的是，采用编译形式得到的目标程序还是不能直接执行，因为每个被编译的源代码都不会是完整的程序代码，起码还会使用一些商家开发出来的已经成为目标代码的模块。因此，目标代码必须经过连接变成可执行代码才能被执行。图 3.6 展示了编译型程序从编辑到执行的过程。

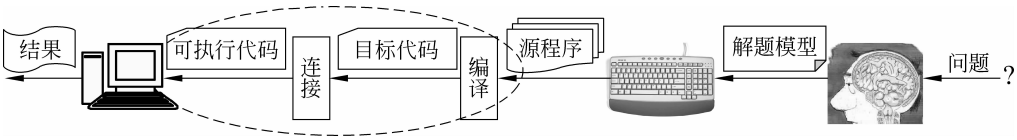


图 3.6　编译型程序从编辑到执行的过程

在所有的高级语言中，有一种特殊的语言是 Java 语言。Java 语言是一种半解释、半编译的语言，即它要先进行编译，形成字节代码——形成一种可以在网络上与平台无关的代码，然后就可以放到网上，再由本地 Java 虚拟机解释执行。图 3.7 展示了一个 Java 源程序文件经过编译、解释的过程。

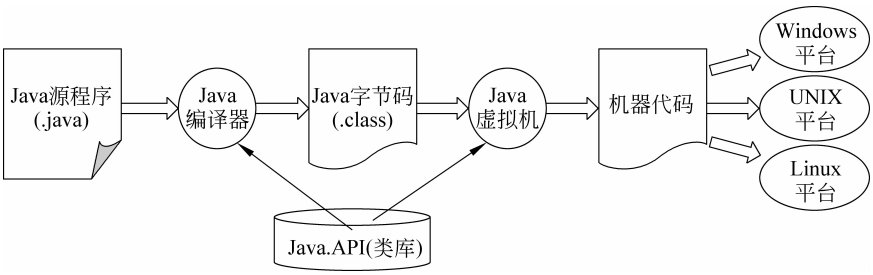


图 3.7　一个 Java 源程序文件的编译、解释过程

3.1.5　软件危机

1. 软件危机及其表现

20 世纪 60 年代以前，计算机刚刚投入实际使用，基本状况是：价值连城、容量很小、速度极慢、机时昂贵。为了节省机时，只能在“效率第一”的思想指导下，把程序编写得非常短小。

随着计算机技术日新月异地发展，计算机的容量和速度不断提高，20 世纪 60 年代之后，大容量、高速度计算机的出现，使计算机的应用范围迅速扩大，软件系统的规模越来越大，复杂程度越来越高，软件可靠性问题也越来越突出。主要表现在以下几个方面。

(1) 软件开发进度难以预测。拖延工期几个月甚至几年的现象并不罕见,这种现象降低了软件开发组织的信誉。例如,丹佛新国际机场投资 1.93 亿美元建立了一个地下行李传送系统,按原定计划要在 1993 年万圣节前启用,但一直到 1994 年 6 月,机场的计划者还无法预测行李系统何时能达到可使机场开放的稳定程度。

(2) 软件开发成本难以控制。投资一再追加,令人难以置信。往往是实际成本比预算成本高出一个数量级。而为了赶进度和节约成本所采取的一些权宜之计又往往损害了软件产品的质量,从而不可避免地会引起用户的不满。

(3) 用户对产品功能难以满足。开发人员和用户之间很难沟通、矛盾很难统一。往往是软件开发人员不能真正了解用户的需求,而用户又不了解计算机求解问题的模式和能力,双方无法用共同熟悉的语言进行交流和描述。在双方互不充分了解的情况下,就仓促上阵设计系统、匆忙着手编写程序,这种"闭门造车"的开发方式必然导致最终的产品不符合用户的实际需要。

(4) 软件产品质量无法保证。系统中的错误难以消除。软件是逻辑产品,质量问题很难以统一的标准度量,因而造成质量控制困难。并且,软件产品并不是没有错误,而是盲目检测很难发现错误,而隐藏下来的错误往往是造成重大事故的隐患。例如 IBM 公司开发 OS/360 系统,共有 4000 多个模块,约 100 万条指令,投入 5000 人年,耗资数亿美元,结果还是延期交付。在交付使用后的系统中仍发现大量(2000 个以上)的错误。

(5) 软件缺少适当的文档资料,软件产品难以维护。软件产品本质上是开发人员的代码化的逻辑思维活动,他人难以替代。除非是开发者本人,否则很难及时检测、排除系统故障。

这些状况引起了开发者与用户之间极大的矛盾,也引起了社会对于软件的不信任,于是人们将之称为软件危机(Software Crisis)。

2. 软件危机的原因分析

分析软件开发危机的种种表现,可以找出软件开发危机的原因。

(1) 软件是一种信息产品,属于柔性生产,与通用性强的硬件相比,软件更具有可延展性和多样化的特点,因此,软件开发过程较难控制。同时,软件是计算机系统中的逻辑部件而不是物理部件,其试制过程就是生产过程,在运行时所出现的软件错误几乎都是在开发时期就存在而一直未被发现的,改正这类错误通常意味着改正或修改原来的设计,这就在客观上使得软件维护远比硬件维护困难。

(2) 开发过程是复杂的逻辑思维过程,其产品取决于软件人员的教育、训练和经验的积累,极大程度地依赖于开发人员高度的智力投入。在缺乏有力的方法学和工具方面的支持的情况下,会过分地依靠程序设计人员在软件开发过程中的技巧和创造性,加剧软件开发产品的个性化。这样,随着软件复杂性急剧地增加,软件的特殊性和人类智力的局限性,导致人们无力处理"复杂问题"。

(3) 随着软件应用范围的扩大,软件规模愈来愈大。大型软件项目需要组织一定的人力共同完成,而多数管理人员缺乏开发大型软件系统的经验,多数软件开发人员又缺乏管理方面的经验。各类人员的信息交流不及时、不准确,有时还会产生误解。软件项目开发人

员不能有效地、独立自主地处理大型软件的全部关系和各个分支，因此容易产生疏漏和错误。

(4) 在软件开发过程中，开发者对于用户需求不清楚或理解不同，特别是用户自己在软件开发出来之前，用户自己也不清楚软件开发的具体需求或者无法描述清楚；再加上用户需求往往是变化的，甚至在软件开发过程中，会不断地提出修改软件开发功能、界面、支撑环境等方面的要求。

3. 解决软件危机的途径

实际上，几乎从计算机诞生的那一天起，软件危机就出现了。但是到了 20 世纪 60 年代中后期才引起了人们的关注。为此，人们找到了两种解决途径：一是科学途径，另一是工程途径。

(1) 科学途径——程序设计方法学。程序设计方法学从科学的角度研究如何解决程序设计中出现的问题。基本内容涉及程序推导、程序综合、程序设计自动化研究、并发程序设计、分布式程序设计、函数式程序设计、语义学、程序逻辑、形式化规格说明、公理化系统等课题。

程序设计方法学也与软件工程关系密切。方法学对软件的研制和维护起指导作用。

(2) 工程途径——软件工程。“软件工程”的概念是 1968 年在德国召开的 NATO (North Atlantic Treaty Organization，北大西洋公约组织)会议上首次提出的，目的是通过用工程化的原则和方法对软件开发过程进行管理，来克服软件危机，建立与系统化软件生产有关的概念、原则、方法、技术和工具，指导和支持软件系统的生产活动，以期达到降低软件生产成本、改进软件产品质量、提高软件生产率水平的目标。软件工程包括方法、工具和规范三个要素。

- 软件工程方法。为软件开发提供“如何做”的技术，是完成软件工程项目的技术手段，主要涉及软件开发的原则与策略、软件过程模型、软件标准与质量管理、软件开发的组织与项目管理等。
- 软件工具。是人类在开发软件的活动中智力和体力的扩展和延伸，为软件工程方法提供了自动的或半自动的软件支撑环境。
- 软件工程规范。为提高软件的可靠性，根据经验对软件结构、文档、开发过程等进行的必要约束。

程序设计方法学与软件工程关系密切、相互促进。

课堂讨论

你有解决软件危机的良策吗？

3.2 数据结构＋算法＝程序

简单地说，算法就是解题的思路，数据就是程序处理的对象。进行程序设计的过程，就是组织思路和组织数据的过程。组织数据要解决两个问题：一是数据的逻辑结构——在程

序中，数据以什么样的逻辑形式进行组织，比较容易处理——算法简单；另一个是数据的物理结构——数据用什么样的方式存储更为经济。

把程序抽象为“算法＋ 数据结构”，是著名瑞士计算机科学家Niklaus Wirth(1934—，见图 3.8)于 1976 年在他以公式命名的惊世之作——《算法＋ 数据结构＝程序》中提出的。

随着程序规模的不断扩大，处理的数据量也急剧增长，这时数据结构重要性就非常突出，通常要先决定数据结构，在此基础上再进行算法设计。为了强调数据结构的重要性，人们往往把这个公式修改为

数据结构＋算法＝程序

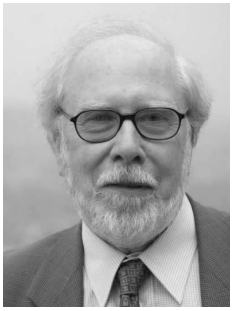


图 3.8　Niklaus Wirth

3.2.1　算法

1. 算法的概念及其特征

算法(algorithm)是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令，算法代表着用系统的方法描述解决问题的策略机制。也就是说，能够对一定规范的输入，在有限时间内获得所要求的输出。如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、空间或效率来完成同样的任务。一个算法的优劣可以用空间复杂度与时间复杂度来衡量。

一个算法应该具有以下 5 个基本特征。

(1) 有穷性(finiteness)。算法必须能在执行有限个步骤之后终止。

(2) 确切性(definiteness)。算法的每一步骤必须有确切的定义。

(3) 输入(input)。一个算法有 0 个或多个输入，以刻画运算对象的初始情况。0 个输入是指算法本身定出了初始条件。

(4) 输出(output)。一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的。

(5) 可行性(effectiveness，也称之为有效性)。即算法中执行的任何计算步都是可以被分解为基本的可执行的操作步，即每个计算步都可以在有限时间内完成。

“算法”的中文名称在公元前 1 世纪成书的算经《周髀》(唐初规定它为国子监明算科的教材之一，遂改名《周髀算经》，如图 3.9 所示)中已经出现，它的西文名称来自公元 9 世纪的波斯数学家花拉子米(Al-Khowarizmi)的名字。原为“algorism”，意思是阿拉伯数字的运算法则，在 18 世纪演变为“algorithm”。因为 Al-Khwarizmi 编写了第一本用阿拉伯语在伊斯兰世界介绍印度数字和记数法的著作，在书中提出了算法的概念。

图 3.9　《周髀》算经

一个非常著名的古老算法是求两个整数的最大公约数的欧几里得算法。这个算法最早出现在大约公元前 350—公元前 300 年欧几里得写成的《Elements》(几何原本)中。欧几里得算法也称为辗转相除法。具体思路如下。

已知两个整数 x 和 y，用 mod(x,y)或者 x mod y 表示 x 被 y 除后所得的余数。两个已知数 a 和 b(假设 $a>b$)的最大公约数的计算过程如下。

设 $r_1=\mathrm{mod}(a,b)$，再设 $r_2=\mathrm{mod}(b,r_1)$，$r_3=\mathrm{mod}(r_1,r_2)$，一直计算下去，直到 $\mathrm{mod}(r_{n-1},r_n)=0$，这时 r_n 就是 a 和 b 的最大公约数。

例 3.3 计算 91 和 52 的最大公约数，求解过程如下。

(1) mod(91,52)=39

(2) mod(52,39)=13

(3) mod(39,13)=0

所以，13 就是 91 和 52 的最大公约数。

课堂讨论

举一个生活中的例子说明算法 5 个特征的意义。

2. 算法的描述工具

常用的算法描述工具有：程序流程图、程序设计语言、自然语言与伪代码等。

(1) 用流程图来描述算法。流程图语言是人们经常用来描述算法的工具。流程图用规定式样的图形、指向线和文字说明组合起来表示算法。其优点是直观、清晰、易懂，便于检查、修改和交流。

(2) 用程序设计语言描述算法。用计算机程序设计语言表示算法显得清晰、简明、一步到位，写出的算法能由计算机处理。事实上，由程序设计语言描述的算法就是计算机程序(一般表现为一个子程序或程序片段)。

(3) 伪代码描述算法。伪代码实际上是一种主要让人看的程序，它结合了自然语言表示和程序设计语言的优点，丢弃了程序设计语言中的烦琐细节，保留了程序设计语言中的关键的流程控制结构，再适当辅之以自然语言进行描述。

例 3.4 闰年的判定。判断闰年的条件如下。

(1) 能被 4 整除，但是不能被 100 整除的年份是闰年。

(2) 能同时被 100 和 400 整除的年份是闰年。

用伪代码描述的算法如下。

```
begin
    input(k)  //输入 k,k 表示当前年份
    if ((k mod 4=0) and (k mod 100 ≠ 0)) or ((k mod 100=0) and (k mod 400=0)) then
        output("是闰年")
    else
        output("不是闰年")
    end if
end
```

3. 穷举与迭代——计算机算法设计的两种基本方法

算法设计是一个创造性的过程，人们在长期的算法研究中创造了许多设计算法的方法