第 5 章 chapter 5

系 统 思 维

民可使由之，不可使知之。

Enable people to follow the way, without them having to understand [the internals of] it.

——孔子(Confucius)，551-479 BCE

计算思维的核心是抽象。

At the heart of computational thinking is abstraction.

——阿尔弗雷德·阿霍(Alfred Aho)与杰佛里·乌尔曼(Jeffrey Ullman)，2022

计算机科学研究的计算过程需要在计算系统中运行。前面 4 章讲述的数字符号操作、计算模型、计算逻辑、算法、程序，都需要通过计算系统才能得以表达，并实际运行起来发挥作用。也就是说，系统思维使计算过程变得实用。系统思维涉及科学、工程、艺术，计算系统设计者往往被称为架构师(architect)。

系统思维(systems thinking)的要点是：**通过抽象，将模块组合成为系统，无缝执行计算过程**。更准确地说，系统思维的要点是通过巧妙地定义和使用计算抽象，将部件组合成为计算系统，该系统流畅地运行所需的计算过程。模块就是精心设计的系统部件，即实践了信息隐藏原理的抽象。

本章通过一系列例子，集中系统地讨论抽象化、模块化、无缝衔接 3 个概念。抽象化主要采用了软件的例子，模块化和无缝衔接主要采用了硬件的例子。这些例子整合起来回答了一个问题：如何设计从逻辑门到应用软件多层抽象形成的计算机软硬件技术栈。

5.1 系统思维一览

什么是**系统**呢？简单的回答是：系统就是计算机。更全面的回答是：计算系统是一个**平台**(platform)，它主要提供 3 种能力。

(1) 提供资源。为所支持的计算过程的执行提供硬件、软件、数据资源。

(2) 提供接口。为用户提供开发和使用应用程序的接口抽象(编程抽象)。

(3) 忠实执行。将程序转换为系统能够理解的步骤序列，比特精准地执行每一步骤。

系统是计算模型、计算逻辑、算法、程序的实用载体和整体具象。因此，不仅桌面计算机、笔记本计算机、智能手机是系统，微信这样的应用软件平台也是系统，因为用户可以在微

信上开发各种各样的应用程序。

如何设计一个系统，使得它执行的计算过程变得实用呢？我们可以从高德纳测试得到启示：[The computer] repeats back exactly what I tell it。人告诉计算机做什么，而计算机忠实地回应人。这里，计算机是实用的系统，人通过应用代码和用户操作告诉计算机做什么。因此，我们需要追求3个目标。

(1) **周到性**(being thorough)。周到地考虑系统执行的所有可能的应用场景，保证计算过程的正确执行或报错。不周到的例子包括：明明用户的应用代码和操作过程无误，但系统执行计算过程时停在中间(死机)，或系统产生错误结果但作为正确结果输出，计算机没有忠实地回应人。

(2) **整体性**(being systematic)。系统思维产出一个系统整体，用一套统一的抽象来支持万千应用场景，而不是每个场景对应一个周到但随意堆砌的系统。采用罗列堆砌、随机应变方法也可能构建出系统，但造成的后果往往是：人与计算机之间的命令和回应可能出现大量随意而为的异构繁杂情况，系统不实用。

(3) **应对复杂性**(coping with complexity)。即使只考虑一个场景和一套系统，例如微信系统，系统复杂性仍可能很高。必须**通过抽象将系统分解成模块、将模块组合成为系统**，使得系统复杂性降低到设计者、开发者、使用者能够驾驭的程度。不然的话，人告诉计算机和计算机回应人，都可能变得很复杂，系统也不实用。

本章首先用计算机历史上的创新实例说明周到性、整体性、应对复杂性3个目标，从目标角度对系统思维有一个了解。然后介绍抽象化、模块化、无缝衔接3个系统思维要点。这3个要点是系统思维不同角度的体现。抽象是最本质的考虑，模块与无缝衔接是补充。

5.1.1 周到性

系统思维要求全面周到地考虑计算过程在系统上的端到端执行的所有可能，覆盖从第一步(初端)到最后一步(末端)的全部情况，保证计算过程的正确执行或报错。系统抽象不能忽略必要的细节。我们用两个例子展示周到性的具体含义。

1. 考虑所有应用场景

【实例5.1】 度量超级计算机的计算速度。

超级计算机是世界上速度最快的计算机。根据贝尔定律的计算机发展第一条路线，我们希望超级计算机的速度每10年增加1000倍。其中，计算速度定义为：运行典型应用时，实际观察到的计算机执行的每秒浮点运算数(floating-point operations per second，FLOPS)。但是，超级计算机执行成千上万种应用程序，哪些才是典型应用呢？总不能在设计时将这成千上万种应用程序都执行一遍吧？

超级计算社区提出了一种**基准程序集**(benchmarks)测试方法。这种聪明的系统思维方法如图5.1所示，它采用仅含4个程序的基准程序集，就能知道一台新计算机是不是速度提高了1000倍。它的计算机科学技术道理如下。

(1) **局部性原理**。计算系统存在局部性现象(locality)。**时间局部性**(temporal locality)是指当前执行的指令或数据在不久的将来很可能会被再次使用。空间局部性(spatial locality)是指假如计算机使用某个地址的指令或数据，相邻地址的指令或数据很可

能也会被使用。计算机的速度与局部性密切相关。

（2）**四角包围**。找出图 5.1 所示的 4 个角落案例(corner cases)作为典型应用。为什么它们是典型的、具有代表性呢？这 4 个基准程序代表了关于时间局部性和空间局部性的 4 种典型情况：①Linpack 程序的时间局部性和空间局部性两者都高；②RandomAccess 程序的时间局部性和空间局部性两者都低；③FFT 程序的时间局部性高而空间局部性低；④PTRAN 程序的时间局部性低而空间局部性高。其他成千上万个应用程序的局部性行为应该位于这四者形成的方框里边。

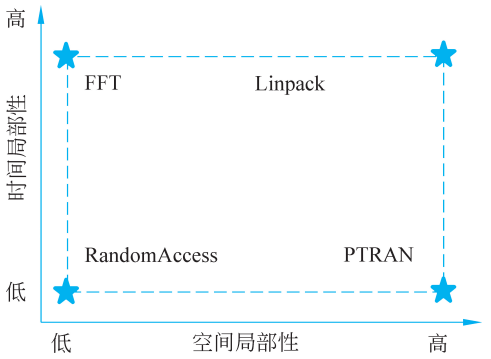


图 5.1　利用时间局部性和空间局部性的基准程序集

也就是说，如果在运行这 4 种基准程序时，新计算机的速度都提高了 1000 倍，我们有实证理由相信，新计算机比原来的超级计算机快 1000 倍。

2. 不能忽略必要细节

【实例 5.2】 数的大端表示与小端表示之争。

有些细节不涉及技术的优劣或创新，但必须明确规定。一个例子是计算机发展史上曾出现过数的大端表示与小端表示之争(big endian versus little endian)。假设我们要在计算机内存中存放一个 32 位整数$(1078018627)_{10}=(01000000010000010100001001000011)_2$。这个整数的十六进制值是 0x40414243，包含 4 字节，Byte0 是最高有效字节 01000000＝0x40，Byte1 是 01000001＝0x41，Byte2 是 01000010＝0x42，Byte3 是最低有效字节 01000011＝0x43。当代计算机都是字节寻址的(byte addressable)，需要用 4 个连续字节地址，记为 A、$A+1$、$A+2$、$A+3$，来存放这个 32 位整数的 4 字节。

问题是，以什么顺序放置这 4 字节？

（1）**大端**(big endian)派认为，应该是地址 A 放**最高**有效字节 Byte0，$A+1$ 放 Byte1，$A+2$ 放 Byte2，$A+3$ 放 Byte3。

（2）**小端**(little endian)派认为，应该反过来，A 放**最低**有效字节 Byte3，$A+1$ 放 Byte2，$A+2$ 放 Byte1，$A+3$ 放 Byte0。

这两种选择没有对错优劣，只需确定一个就行(图 5.2)。但必须明确规定一个，不能忽略这个细节。当一个计算过程涉及使用不同表示的部件时，还需要提供自动转换机制。实践中，两种选择都在使用。常见例子如下。

（1）TCP/IP 互联网、MIPS 处理器属于大端派。

（2）ARM 处理器、RISC-V 处理器、x86 处理器属于小端派。

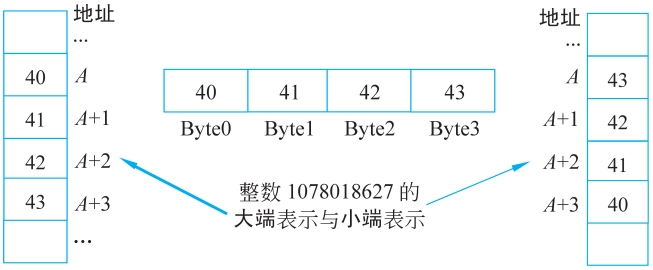


图 5.2 大端表示与小端表示示意

5.1.2 整体性

整体性也称为系统性。系统是一个整体，而不是一堆部件的罗列堆砌；需要全局考虑和权衡取舍（trade off），用一套统一的方法来支持万千应用场景，而不是每个场景对应一个计算系统。中国古人将这种思想称为“**以一耦万**”。英文则用 systematic（系统地），而不是 ad hoc（随意地、随机应变地），来形容这种系统思维。

同时，系统性不是僵化的教条，不是用于阻碍创新，更不是遏制应用的丰富性和多样性。构造计算系统是有条理、有门道的。但是具体是什么条理、门道就是创新的重要目标。成功的条理、门道、诀窍往往体现为系统抽象。理解和设计系统富有令人激动的挑战性。系统思维不是让计算系统设计者机械地执行一些教条方法，而是要求创造性和综合性。这也是为什么在英文中，计算系统设计者被称为 **architect**（直译为建造师或建筑师，信息技术领域则称为**架构师**）。本章介绍两种思路，即**周期**（cycle）思路和**层次**（layer）思路。每一个系统抽象在运行时往往会经历几个阶段，它们构成该抽象的周期。例如，**指令流水线**有对应的**指令周期**。我们将在 5.4.1 节讨论周期思路。层次思路体现为一个重要的整体性方法，即技术栈（抽象栈）方法。

如何用一套抽象来支持万千应用场景？如何实现“以一耦万”，体现整体性？

【实例 5.3】 多个抽象层次组成技术栈整体。

一台笔记本计算机具有数十亿个晶体管部件，如何整体描述它呢？我们用多层抽象的总体构成了一个软硬件**技术栈**（stack of layers），包含软件和硬件。它也称为**抽象栈**，是一套抽象的体现。图 5.3 展示了一个典型的计算机技术栈的部分主要抽象。较高层次的抽象会使用或包含较低层次的抽象。例如，处理器包含很多晶体管。

按照抽象层次从高到低，我们可用如下方式理解一台笔记本计算机。

（1）最高的系统抽象层次是编程语言。例如，用户使用 Go 语言编写快速排序程序。Go 语言编译器将快速排序程序变换成一串指令。最基本、最底层的计算机软件抽象是指令。操作系统、编译器和应用软件都是由指令组成的。开发出来的程序是静态的代码。当应用程序的指令代码被启动执行后，程序就变成了“活的”进程。**进程**是运行中的程序。

（2）有颜色部分是计算机的硬件。它符合冯·诺依曼模型，由处理器、存储器和输入输出设备组成。当代计算机的处理器一般都包含多个中央处理器（CPU），每个 CPU 称为一个**核**（core）。因此，当代处理器一般是**多核处理器**（multicore processor）。除了包含 CPU

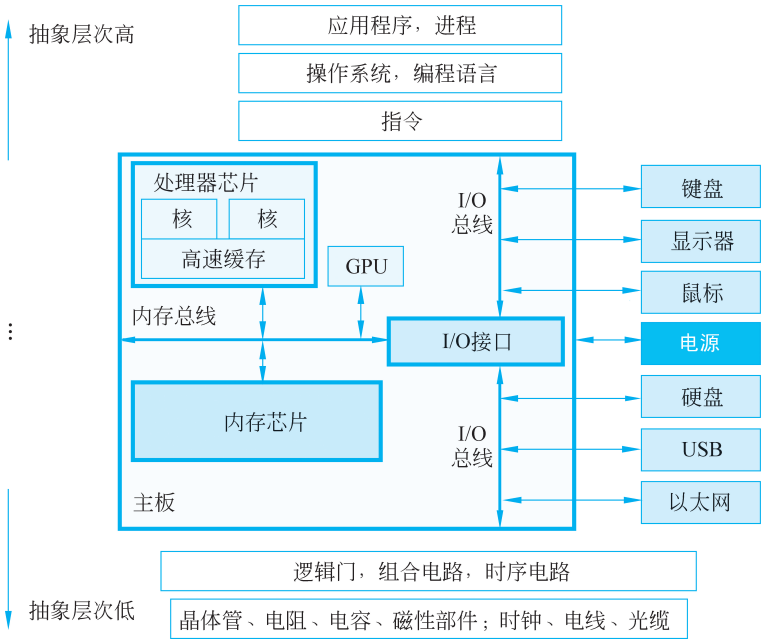


图 5.3　一台计算机的技术栈：抽象层次示意

以外，当代处理器一般还包含**高速缓存**(cache)，它是比存储器容量更小但速度更快的存储子系统。存储器(memory)也称为**内存**。这是相对于硬盘(hard disk)这样的**外存**来说的。

(3) 计算机硬件可对比 1.3.2 节中的笔记本计算机硬件拆解实物图片。处理器、存储器、接口电路、GPU(graphic processing unit，图形处理器，有时简称显卡)等部件通常由半导体集成电路实现，这些集成电路又称为**芯片**(microchip 或 chip)。这些部件通过**总线**(bus)相连，包括内存总线和 I/O 总线。计算机一般会有一个主机板，简称**主板**或**母板**(motherboard)，它是计算机的主要电路板，实现了总线，通过焊接或其他方式连接处理器、存储器、接口电路等芯片，并通过接口电路连接各种 I/O 设备。

(4) 处理器、存储器、GPU 等芯片是较大的抽象，它们是由较低层次的逻辑门、组合电路、时序电路抽象组成的。更低的抽象是晶体管、电容、电阻等元器件，以及电线和光缆等物理连线。

技术栈的软件部分称为**软件栈**(software stack)。根据功能的不同，计算机软件可以粗略地分成 3 类。

(1) **应用软件**种类最多，包括办公软件、电子商务软件、通信软件、行业软件、游戏软件等。

(2) 常见的**中间件**(middleware)包括数据库管理软件、万维网浏览器、深度学习应用框架(application framework)等，它们在应用软件和系统软件之间建立一种桥梁。近年来，业界往往将中间件和系统软件合称为基础软件(infrastructure software)，它们和硬件一起为应用开发者和用户提供了基础设施。

(3) **系统软件**(system software)还可细分成三层抽象(表 5.1)。最贴近计算机硬件的是一些小巧的软件。它们实现一些最基本的功能，通常“固化”在只读存储器芯片中，因此称

为**固件**(firmware)。系统软件和硬件一起提供一个**平台**(platform),管理和优化计算机硬件资源的使用,并为应用软件开发提供抽象程度更高的表达能力。科学计算领域常见"x86处理器+Linux操作系统"平台,桌面办公常见"x86处理器+Windows操作系统"平台,移动互联网领域常见"ARM处理器+安卓操作系统"平台。

表5.1 计算机软件栈例子

软件类型			例子
应用软件			科学计算、企业计算、个人计算;办公软件、搜索引擎、微信、抖音
基础软件	中间件	数据库、Web服务器、浏览器、应用框架	MySQL、OceanBase、WebServer.go、Chrome、Safari、TensorFlow
	系统软件	编程语言及其编译器或解释器	C、Go、JavaScript、Python、Shell
		操作系统	Linux、Android、iOS、Windows
		固件	BIOS、UEFI

【实例5.4】 辨别从高到低的抽象层次,形成计算机系统的整体概念。

表5.2是由一套多层抽象形成的计算机整体技术栈(抽象栈),是第5章的重点内容。

表5.2 计算机抽象栈

软件	数据类型	比特、字节、整数、数组、切片、BMP图像文件
	算法	巧妙的信息变换方法。例如信息隐藏算法
	程序	算法的代码实现。例如实现信息隐藏算法的hide.go程序
	进程	运行时的程序。例如在Linux环境中的hide进程
	指令	程序的最小单位,计算机能够直接执行
软件与硬件的桥接模型:**冯·诺依曼体系结构**		
硬件	指令流水线	每条指令都通过"取指—译码—执行"3个操作阶段组成的指令流水线得以自动执行。所有指令都由这一种机制执行,指令流水线由若干时钟周期组成
	时序电路	等同于**自动机**,说明每一个时钟周期的操作。时序电路由组合电路与存储单元组成,理论上等同于**图灵机**
	组合电路	实现二值逻辑表达式(**布尔逻辑表达式**)

抽象使得计算机科学成为一门优美的学科。抽象的本质是采用一套机制、一套概念来解决该层次的所有问题,应对系统的复杂多样性,以不变的抽象应对系统的万变,即**以一耦万**。下面是一些例子。

(1) 操作数字符号变换信息的方法有无穷多种,但都可以用高德纳定义的**算法**表达。

(2) 算法有无穷多种,但都可以用Go语言编写**程序**实现。这些高级语言程序都可以被编译成为机器语言程序(二进制**指令**程序)由计算机执行。

(3) 程序有无穷多种,但运行时都变成了**进程**这种抽象,被操作系统调度执行。

(4) 所有指令都由**指令流水线**这一种机制执行。

(5) 指令流水线的每个步骤的执行都等同于**自动机**的一次变换(状态转换)。

（6）自动机的变换逻辑可由**组合电路**和**时序电路**实现。

抽象可以组合，形成更强大的抽象。从最底层的组合电路，一直到最上层的应用系统，各级抽象提供表达能力越来越强、离用户越来越近的功能。

（1）最下层是由逻辑门组成的组合电路，实现布尔逻辑功能。组合电路由晶体管、电阻、电容等各种电路部件和各种连线组成。

（2）组合电路与存储单元组合起来，形成时序电路，实现自动机功能。

（3）多个时序电路组合起来，形成指令流水线，实现处理器自动执行指令的功能。

（4）处理器加上存储器、输入输出设备，构成了计算机硬件整机。

（5）冯·诺依曼体系结构提供了软件与硬件的桥接模型。

（6）计算机硬件整机能够执行其指令集，从而自动执行程序。

（7）在操作系统管理下，提供了进程功能。

5.1.3 应对复杂性

研究、理解和使用计算系统的最大挑战是应对系统的复杂性。应对系统复杂性挑战的主要思维方法是抽象化，特别是“用一套系统抽象支持万千应用场景”，即“以一耦万”。注意，**系统复杂性**（system complexity）不同于**算法复杂度**（algorithmic complexity）。

【实例 5.5】 微信系统的复杂性。

让我们粗略地估算一下腾讯微信系统涉及的晶体管个数，可以稍微认识一下该系统有多么复杂。微信系统有上亿用户在线。假设每个用户使用一台智能手机或笔记本计算机这样的终端设备，那么有上亿颗处理器芯片在同时工作，执行微信的计算过程。这还不包括微信云端系统及互联网系统。

每颗处理器芯片大约包括 20 亿个晶体管。如果将每个晶体管看成一个住房，晶体管间的连线看成道路（从高速公路一直到公寓楼道），一颗芯片的电路图比全中国的米级地图（显示了全中国全部的道路及每一套住房）还要复杂。整个微信系统包括 20 亿亿个晶体管，比全世界的米级地图复杂得多。显然，理解和设计微信系统需要系统思维，不能从每一个晶体管做起，更不能简单地堆砌 20 亿亿个晶体管。

惠勒间接原理——“以一耦万”的 4 个实例

我们讨论 4 个计算机科学史上的创新实例，展示如何通过精心设计的系统抽象应对系统复杂性，如何“以一耦万”。主要有 4 种因素影响系统复杂性。

（1）**多**：系统规模大，部件多，如上述微信例子。

（2）**杂**：系统的部件多种多样，异构性大。

（3）**乱**：系统的组织杂乱无章。典型例子是软件中的面条代码（spaghetti code）。

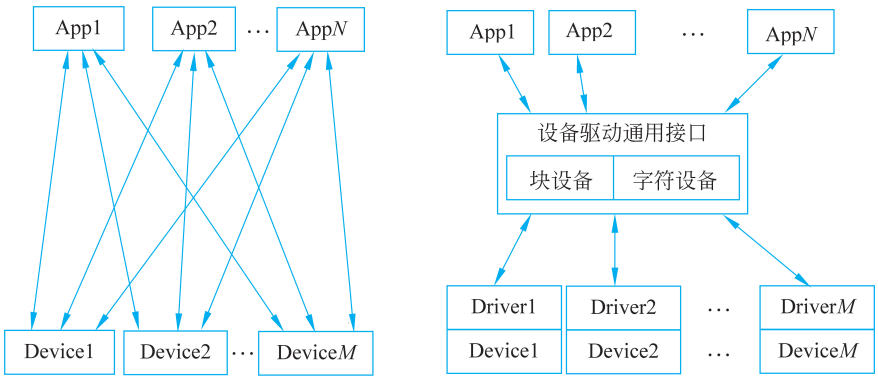
（4）**变**：系统的组织或部件随意变化。

这 4 个实例分别针对其中一种因素。它们都采用了计算机科学中的一个基本抽象思路，称为**惠勒间接原理**：任何计算机科学问题都可以通过另一个间接层解决（Any problem in computer science can be solved with another level of indirection）。汇编语言的发明者大卫·惠勒（David Wheeler）提出了这个看起来很夸张的原理。巴特勒·兰普森（Butler Lampson）在 1993 年的图灵奖演说中引用了这个原理，使它成为一句智慧类名言。

【实例 5.6】 驱动程序：如何应对“多”造成的系统复杂性。

今天世界上有数十亿用户和计算机，他们运行着百万种应用程序，使用着百万种 I/O 设备。注意，这里说的应用程序和 I/O 设备是“种”，不是“个”。全班数百名同学，每名同学开发 10 个 Go 程序，全部数千个 Go 程序都属于一种程序。

这些众多的应用程序如何与众多的 I/O 设备实现比特精准的交互？一种暴力方法是：每一种程序直接与每一种 I/O 设备交互。这也是早期计算机的做法，每种程序直接用指令控制每一个 I/O 设备。但是，这样会导致 $M \times N$ 种交互，其中 M、N 是百万量级。这是巨大的系统复杂性，需要考虑万亿种交互。计算机界发明了一种系统抽象，称为 I/O 设备**驱动程序**(device driver)，添加了一个间接层(图 5.4)，将原来的 $M \times N$ 种交互的系统复杂性降低为 $M+N$ 种交互的系统复杂性，即从万亿量级降到了百万量级。



(a) 暴力交互法：$M \times N$复杂性　　(b) 驱动程序法：$M+N$复杂性

图 5.4 设备驱动程序抽象降低复杂性的示意

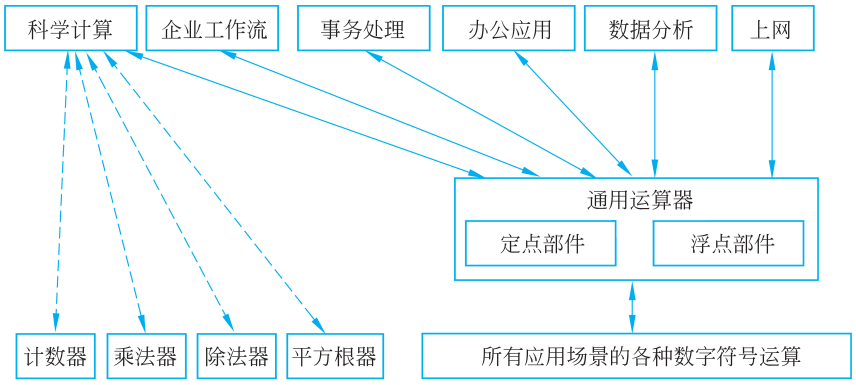
有了驱动程序抽象，应用程序不是直接与 I/O 设备打交道，而是访问操作系统提供的设备驱动通用接口(generic device driver interface)。它提供两类接口，分别针对**字符设备**(character device)和**块设备**(block device)。命令行界面的键盘和显示器是字符设备，每次输入或输出交互传递一个字符。硬盘和 U 盘等通过文件访问的设备是块设备，每次输入或输出交互传递一个数据块，例如一幅图片的部分数据。假设某厂商研制一个新设备 Device1。它会开发一个相应的驱动程序 Driver1，其中实现了新设备的设备驱动通用接口，以及各种必要的底层操作细节。这些细节应用层看不见也不关心。

事实上，驱动程序抽象带来的复杂性降低程度还要大得多。研制一个新设备时，往往只需要针对某些计算机系统开发驱动程序，不需要关心上层的应用程序。计算机的操作系统和编程语言会自动地支持应用程序访问设备。例如，假设某厂商研制了一款新的显示器，它只需要为使用该显示器的计算机系统(可能只有几十种)开发驱动程序。在这些计算机上运行的各种 Go 程序可以通过 fmt.Println 等函数访问该显示器。

【实例 5.7】 通用运算器：如何应对“杂”造成的系统复杂性。

在计算机发展历史中，一个关键的抽象化工作是确定运算器抽象。具体的问题是：一个处理器中的运算器应该如何设计，才能产生一个通用处理器，高效地支持各种各样的混杂应用场景？图 5.5 列出了两种思路。

早期的计算机采用了比较随意的(ad hoc)方式实现多种多类的运算。例如，1945 年的



(a) 早期计算机的运算器设计　　(b) 当代计算机的通用运算器设计

图 5.5　通用运算器抽象降低复杂性的示意

第一台数字电子计算机“埃尼阿克”没有一个通用运算器，而是包括了计数器、乘法器、除法器、平方根器等多种杂异的硬件运算单元，能够支持一些数值计算（科学计算）。早期的计算机不能解决通用运算器问题。

当代计算机则需要支持科学计算、企业工作流、事务处理、办公应用、数据分析、游戏、上网、社交网络等多种应用场景，需要通用运算器。1964 年的 IBM 360 计算机系统地回答了这个通用运算器问题，提出了**定点部件**和**浮点部件**硬件抽象，使得 IBM 360 成为了通用计算机。简略地讲，定点部件高效地实现带符号或无符号整数的加、减、乘、除，与、或、非，左右移位运算，而浮点部件高效地实现有限精度实数的加、减、乘、除运算。

用定点部件和浮点部件这一套硬件抽象支持所有的应用所需的运算，而不是为科学计算设计几种运算器，又为企业计算设计几种运算器，再为上网游戏设计几种运算器，这就是“以一耦万”的体现，即“用一套通用抽象支持多个具体应用需求”。今天的 x86、ARM 及 2014 年推出的 RISC-V 处理器指令系统体系结构，都继承了这种硬件抽象。当然，这种通用运算器思想不是教条。近年有些计算机在通用运算器之外，还包括各种加速器硬件，如向量部件、张量处理器、图形处理器等。

抽象化需要人们的辛勤努力和聪明才智。加拿大计算机科学家威廉·卡汗（William Kahan）从 20 世纪 70 年代开始长期致力于计算机浮点运算部件抽象的研究，并促成了 IEEE 754 浮点算术标准的建立。我们今天使用的计算机，包括超级计算机、微机、平板计算机、智能手机，都用到了他的研究成果。卡汗教授也因此在 1989 年获得了图灵奖。

【实例 5.8】　并行进位加法器，应对“乱”造成的系统复杂性。

根据布尔逻辑的范式定理，任何一个布尔表达式可以通过与、或、非三级门电路实现。但是，这样的结果只具有理论通用性，不实用。直接使用范式定理得到的布尔电路很混乱，没有利用到该电路的特性知识（即该电路的条理和门道），既难以实现又难以理解。

在 5.3.2 节，我们将讨论更加真实的加法器组合电路，它引入了两组间接变量 G 和 P，同时算出所有的进位比特，更加有条理地实现了并行进位加法器，且只需要 4 级逻辑运算。教师在课堂讲解时可展示并行进位加法器的电路图。

【实例 5.9】 桥接模型，应对“变”造成的系统复杂性。

计算机领域的变化很快，已经演变成具有成千上万种计算机和百万种应用程序的动态变化的生态系统，每年甚至每月都有众多的新型计算机和新应用程序出现。计算机领域是如何应对这种剧烈变化呢？一个利器是冯·诺依曼体系结构，它在众多的计算机硬件和应用程序软件之间提供了一个**桥接模型**(bridging model)。

根据调查，2020 级的全班数百名同学拥有 10 种以上笔记本计算机平台。使用的操作系统包括 Windows、iOS、Linux 等，使用的处理器包括 Intel、AMD、苹果 M1 等。当这些同学开发 Go 程序时，他们看到的是一种计算机、10 种计算机，还是数百种计算机？

回答是：同学们只看见一种计算机，即冯·诺依曼模型计算机。他们并不用关心具体的笔记本计算机平台细节。针对冯·诺依曼计算机开发出来的程序，在任何内存足够大的真实计算机上运行，其执行时间的差别只有 $O(1)$。计算机产品不断变化，但冯·诺依曼模型不变。

作为对比，图灵机就不适合作为桥接模型。在图灵机上开发出来的程序，在真实计算机上运行，其执行时间的差别可能高达 $O(n^4)$，其中 n 是问题规模。

5.2 抽 象 化

抽象化是所有科学技术学科共有的方法。那么，什么是计算机科学的抽象化特色呢？就是可自动执行的、比特精准的信息变换抽象，即计算抽象。计算机科学的抽象可大致分为 3 类，即数据抽象、控制抽象和模块抽象。**数据抽象**(data abstraction)是某一类数据及其操作，如各种运算操作和访存操作。数据抽象也称为数据类型(data type)。通常，针对这些数据抽象的多个操作步骤组合起来才能解决一个问题。控制多个步骤如何组合起来实现计算过程的抽象称为**控制抽象**(control abstraction)，它确定某个步骤何时激活。**模块抽象**描述一个系统的子系统单元，往往同时包含数据抽象与控制抽象。有些学者将模块抽象也纳入控制抽象。在讨论这 3 种抽象之前，我们先指出任何计算抽象都具备的 3 个性质。

5.2.1 抽象三性质

计算抽象既是计算机科学最重要的方法，也是最重要的产物。作为动名词的抽象也被称为**抽象化**，而抽象化的产物也称为**抽象**，两者对应的英文都是 abstraction。

抽象化的要点是：一个系统可从多个层次(或多个角度、多个视野)理解，每个层次聚焦于考虑有限的、该层次特有的本质问题，并提取出一套精确规定的抽象，统一处理该层次所有的计算过程，解决这些特有问题。其他问题则留给别的层次考虑。该层次甚至看不见这些其他问题，因此也可以忽略与这些问题相关的所有细节。换句话说，抽象化和抽象具备 3 个性质，称为**抽象三性质**，英文缩写为 COG，即**齿轮**性质。

(1) **受限性**(Constrained)：抽象化意味着“聚焦本质，忽略细节”，意味着限制。通常做法是从某抽象层次理解一个计算系统的本质，每个抽象仅仅考虑该层次特有的本质问题，忽略或隐藏不必要的其他问题和细节。受限性，即忽略或隐藏不必要细节的能力，是抽象有助于应对复杂性的主要原因。