计算机科学笔记

theqiong.com

2015 年 2 月 26 日

Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1

Overview

1.1 Turing Machine

从数学和哲学上来对计算进行定义时，计算机可以被看成是一个图灵模型，计算机科学可以被定义为“和计算机相关的问题”。

1937 年，Alan Turing 首次提出了图灵模型，所有的计算都可以在图灵机上执行，而且图灵模型建立在人们进行计算过程的行为上，这些行为可以抽象到图灵模型中。

在引入图灵模型之前，计算机被定义为一个数据处理器的黑盒，可以接收输入数据、处理数据并产生输出数据，因此只能表示用于完成特定任务的专用计算机（或者处理器）。

图灵模型增加了一个额外的元素（程序），从而可以适用于通用计算机。其中，程序用于指示计算机对数据进行处理的指令集合。

基于通用图灵机建造的计算机都是在存储器中存储数据，并且输出数据依赖输入数据和程序的结合作用。

- 对于相同的数据输入，如果改变程序，则可以产生不同的输出；
- 对于相同的程序，如果改变数据输入，则可以产生不同的输出；
- 如果输入数据和程序保持不变，则输出结果也将不变。

通用图灵机是对现代计算机的首次描述，因此现代计算机和通用图灵机可以执行任何可计算的运算，用户仅需要提供数据以及处理数据的程序。

1.2 Von Neumann

在 1944 ～ 1945 年期间，von Neumann 指出程序指令和数据在逻辑上是相同的，因此程序也可以保存在存储器中。

现代计算机都是以 von Neumann 模型为基础的，并且可以划分为 4 个子系统：存储器、算术逻辑单元、控制单元和输入/输出单元。

- 存储器用于保存在处理过程中的数据和程序；
- 算术逻辑单元用于执行算术运行和逻辑运算；
- 控制单元用于对存储器、算术逻辑单元和输入/输出等子系统进行控制操作；
- 输入子系统负责从计算机外部读入数据和程序；
- 输出子系统负责将计算机的处理结果写入到计算机外部。

早期的计算机系统结构中，只有数据才保存在存储器中，但是 von Neumann 模型要求程序也必须存储在内存中，从而导致了和早期计算机系统结构的完全不同。

在 von Neumann 模型中，完成某一任务的程序通过操作一系列的开关或改变其配线来实现。

在现代计算机系统结构中，数据和程序具有相同的格式，因此存储单元可以存储程序及其响应数据，实际上它们都是以位模式（0 和 1 序列）存储在内存中的。

另外，von Neumann 模型中的程序由一组数量有限的指令组成，并且指令本身是顺序执行的，任何时刻只能有一条指令处于执行状态。

具体来说，控制单元从内存中读取一条指令，译码器解释指令，然后处理器执行指令。实际上，一条指令也可能请求控制单元来跳转到其前面或后面的指令并继续执行，但是指令仍然是顺序执行的，跳转指令本身也是在指令的执行序列中。

von Neumann 模型的初始条件就是指令的顺序执行，而且计算机已经进化为以最高效的顺序来执行程序指令。

1.3 Hardware

von Neumann 模型将现代计算机清楚地定义为数据处理机，其硬件也都基于 von Neumann 模型，分别用于接收输入数据，处理并输出相应的结果。

基于 von Neumann 模型的计算机都包含硬件、软件和数据，虽然 von Neumann 模型并没有定义数据存储格式，但是二进制计算机中的数据都是以二进制格式存储在计算机内部的，只是在计算机外部可以针对不同的用户表现为不同的形式。

计算机以及数据处理表示法开创了数据组织技术，这样就可以在将数据存储到计算机中之前，有效地将数据组织成不同的实体和格式。

为了对数据组织进行进一步抽象，通过数据结构可以将组合相同或不同类型的数据到特定的类属中。例如，文件系统的本质可以认为是抽象数据类型。

在引入文件系统之后，数据被组织为许多小的单元，并且进一步由这些小的单元组成更大的单元，这样针对不同的目的也就产生了不同的文件系统结构等。

在现代计算机科学中，原子数据汇集成记录、文件和数据库等。

1.3.1 CPU

在大多数系统结构中，所有的数据操作都由 CPU 完成，CPU（Central Process Unit）由三个组成部分组成。

- 算术逻辑单元（ALU）
- 控制单元
- 寄存器组

ALU 对数据进行逻辑、移位和算术运算。

- 逻辑运算包括非、与、或和异或等，它们把输入数据作为二进制模式，运算的结果也是二进制模式。
- 移位运算包括逻辑移位和算术移位，其中：
  - 逻辑移位运算用于对二进制位进行向右或向左的移位；
  - 算术移位运算用于对整数执行用 2 除操作或乘一个整数。
- 某些专门硬件可以用于提高算术运算的效率。

控制单元控制各个子系统的操作，具体来说控制是通过从控制单元发送到其他子系统的信号来完成的。

寄存器是用于临时存放数据的高速存储设备，常用的寄存器包括：

- 数据寄存器

  数据寄存器用于存储输入数据、中间结果和最终结果。
- 指令寄存器

  CPU 从内存中逐条读取指令时将会把读取出的指令存储在指令寄存器（IR）中，然后再解释和执行指令。
- 程序计数器

  程序寄存器（PC）保存当前正在执行的指令，并且在当前的指令执行完毕后通过对

程序计数器自动加 1 来指向下一条指令的内存地址。

通用计算机使用程序来处理数据，程序和数据都在内存中，CPU 使用重复的机器周期来执行程序中的指令，在任何时刻都只有一条指令在运行。

在每一个机器周期中，CPU 都按照取指令、译码和执行的步骤执行指令。

在取指令阶段，控制单元将下一条要指令的指令复制到 CPU 的指令寄存器中，被复制指令的地址保存到程序计数器中，程序计数器在复制完成后自动加 1 来指向内存中的下一条指令。

在译码阶段，控制单元对指令进行译码并产生一系列系统可以执行的二进制代码。

在执行阶段，控制单元发送指令到 CPU 的相应部件并执行任务，例如控制单元可能通知系统从内存中加载（读）数据项，或者是通知算术逻辑单元对寄存器中的内存进行运算后将结果进行保存。

- 流水线技术允许控制单元同时执行两个或三个阶段来提高吞吐量，这样下一条指令的处理可以在前一条结束前开始。
- 并行处理通过使用多指令流处理多数据流来改善吞吐量。

1.3.2 Memory

计算机中的输入/输出设备可以分为两大类：非存储设备和存储设备。

- 非存储设备（包括键盘、监视器和打印机等）本身不能存储信息，但是可以使得 CPU 和内存和外界通信。
- 存储设备通常分为磁介质（例如磁盘）和光介质等。

主存储器是存储单元的集合，每一个存储单元都有唯一的标识（也就是地址），这样就可以在存储器中通过对应的地址来存取每一个字。

RAM（随机存取存储器）是主存储器的组成部分，这里的随机是指可以使用存储单元地址来随机存取一个数据项，不需要存取位于其前面的所有数据项，从而区别于磁带等存储设备。

- SRAM（静态 RAM）使用触发器门电路来保存数据，而且门可以保持状态（0 或 1），通电时可以在不需要刷新的情况即可保持数据始终存在。
- DRAM（动态 RAM）使用电容器充电来保持状态 1，电容器放电则代表状态 0，而且内存单元需要周期性地刷新来维持电容器的状态。

在硬件层次上，每个字都是通过地址来标识的，所有在存储器中唯一可标识的独立地址单元的总数称为地址空间。例如，一个 8GB、字长为 8 字节的内存的地址空间的范围为

0 到 1 048 576。

Table 1.1: 存储单位

单位	字节数的准确值	字节数的近似值
千字节（KB）	2^{10}（1 024）字节	10^3 字节
兆字节（MB）	2^{20}（1 048 576 字节）字节	10^6 字节
千兆字节（GB）	2^{30}（1 073 741 824）字节	10^9 字节
兆兆字节（TB）	2^{40} 字节	10^{12} 字节

在计算机内部都是以位模式存储数字并进行计算的，存储地址也是使用位模式（即无符号二进制整数）来表示的。

一般来说，起始地址通常是 0，而且如果一个计算机有 N 个字的存储空间，那么就需要有 $\log_2 N$ 位的无符号整数来确定每一个存储单元。

1.3.3 Cache

高速缓冲存储器的存取速度比主存快，但是比 CPU 及其内部的寄存器慢，通常位于 CPU 和主存之间。

高速缓冲存储器在任何时刻都含有主存中一部分内容的副本，因此当 CPU 要存取主存中的一个字时，将按照以下步骤进行：

1. CPU 首先检查高速缓存。
2. 如果要存取的字存在，CPU 将其复制，否则 CPU 将从主存中拷贝一份从需要读取的字开始的数据块，该数据块将覆盖高速缓存中的内容。
3. CPU 存取高速缓存被拷贝该字。

实际上，很有可能 CPU 在下次存取中需要存取上次存取的第一个字的后续字，因此高速缓存可以提高运算的速度。

- 如果字在高速缓存中，就立即存取它；
- 如果字不在高速缓存中，字或整个数据块就会被拷贝到高速缓存中。

根据 80/20 法则，通常计算机需要花费 80% 的时间来读取 20% 的数据，也就是说相同的数据往往被存取多次，这样高速缓存可以通过存储 20% 的数据来提高 80% 的存取速度。

1.3.4 BUS

在现代计算机系统结构中，CPU 和内存之间通常由总线（bus）连接在一起。

总线可以被分为三组线路，分别是数据总线、地址总线和控制总线。

数据总线（Data Bus）的每一根线上每次传送 1 个位的数据，而且线的数量取决于字的大小。

- 32 位计算机使用 32 根数据总线来每次传送 4 个字节，这样同一时刻就可以同时传送 32 位的字。
- 64 位计算机使用 64 根数据总线来每次传送 8 个字节，这样同一时刻就可以同时传送 64 位的字。

地址总线（Address Bus）允许访问存储器中的某个字，线的数量取决于存储空间的大小。例如，如果存储器容量为 2^n 个字，那么地址总线一次需要传送 n 位的地址数据，也就是说地址总线中的线的数量就是 n。

控制总线（Control Bus）负责在中央处理器和内存之间传送信息。例如，必须有一个代码从 CPU 发往内存来指定进行的是读操作还是写操作。

控制总线的数量取决于计算机所需要的控制命令的总数。例如，如果计算机有 2^m 条控制命令，那么 m 条控制总线就就可以定义 2^m 个不同的操作，那么 m 根控制总线就可以满足需求。

输入/输出设备与 CPU 以及内存的本质不同，因此不能直接与连接 CPU 和内存的总线相连。

- 输入/输出设备是机电、光磁设备。
- CPU 和内存都是电子设备。

输入/输出设备的速度要显著低于 CPU 和内存，因此必须通过中介来与 CPU 和内存通信。

一般来说，输入/输出设备通过输入/输出控制器或连接器来与总线沟通，然后再由总线与 CPU 和内存等进行沟通。

输入/输出控制器（或者说接口）可以是串行或并行的设备，用来消除输入/输出设备与 CPU 及内存在本质上的障碍。

- 串行控制器只有一根数据线进行连接。
- 并行控制器需要多根数据线进行连接，一次能同时传送多个位。

SCSI（小型计算机系统接口）是一个 8、16 或 32 线的并行接口，并且提供了菊花链连接。

- 连接链的两端都必须有终结器。
- 每个设备都必须有唯一的地址（目标 ID)。

1394 是一种高速的串行接口，数据采用数据包的形式传送。

- 可以在一条菊花链或树型连接（只用一根线）上连接多达 63 个设备。
- 不需要 SCSI 中的终结器。

USB（通用串行总线）控制器是一种串行控制器，根集线器可以连接多个设备到一个 USB 控制器上。例如，USB2.0 支持多达 127 个设备组成树状拓扑结构连接到一个 USB 控制器上，其中控制器作为树的根，集线器作为中间节点，设备作为末端节点。

USB 控制器（根集线器）与其他集线器的不同在于控制器能感知树中其他集线器的存在，而其他集线器则是被动的设备，它们只是简单地传送数据。

USB 支持热交换设备，因此无需关闭计算机就可以直接移除和连接，当集线器被从系统中移除时，与该集线器相连地所有设备和其他集线器也被移除。

USB 使用 4 根线的电缆，其中两根线（+5 伏和地）用来为低压设备（例如键盘和鼠标）提供电压，高压设备则需要连接到电源上。

集线器从总线取得电压，从而为低压设备提供电压，其他的两根线进行缠绕来减少噪声和传送数据、地址和控制信号。

USB2.0 提供三种传输速率：1.5Mbps、12Mbps 和 480Mbps。

- 低速率可以用于低速设备（例如键盘和鼠标)。
- 中速率用于打印机。
- 高速率用于大容量的存储设备。

USB 控制器以包的形式传输数据，每个包含有地址部分（设备标识)、控制部分和数据部分，所有设备将接收到相同的包，但是只有具有数据包中所定义的地址的设备能够识别和接受包。

通常情况下，CPU 使用相同的总线在主存和输入/输出设备之间读写数据，唯一的不同是指令。

- 如果指令涉及主存中的字，那么数据会在主存和 CPU 之间传送。
- 如果指令涉及输入/输出设备，那么数据会在输入/输出设备和 CPU 之间传送。

输入/输出设备的寻址方式有两种，分别是 I/O 独立寻址和 I/O 存储器映射寻址。

- 在 I/O 独立寻址中，用来从内存读/写的指令完全不同于用来从输入/输出设备的读/写指令。
- 在 I/O 存储器映射寻址中，CPU 把 I/O 控制器中的每个寄存器看作是内存中的字。

为了把数据从 I/O 设备传输到 CPU 和内存，并使 CPU 的操作在某种程度上和输入/输出设备保持同步，现在已经发展出三种不同的方法。

- 程序控制输入/输出
- 中断控制输入/输出
- 直接存储器存取（Direct Memory Access）

在程序控制输入/输出中采用最简单的一种同步，CPU 等待 I/O 设备，而且 CPU 和 I/O 设备之间的数据传输通过程序中的指令实现。

具体来说，当 CPU 遇到一条 I/O 指令时就停止工作并等待，直到数据传输完毕，CPU 不断地查询 I/O 设备地状态来确定数据传输完成。

程序控制输入/输出的问题在于在每一个单元数据被传输时，CPU 都要查询 I/O 设备的状态，而且数据的输入/输出都要经过内存。

在中断控制输入/输出中，CPU 指示 I/O 设备开始传输数据，并且在 I/O 设备准备就绪后通知（中断）CPU，不过数据的输入/输出仍然要经由内存。

直接存储器存取（DMA）用于在高速 I/O 设备间传输大量的数据块（例如磁盘、内存），这样就可以不需要经由 CPU 的数据传输，不过需要单独的 DMA 控制器来承担 CPU 的运算。

DMA 控制器中有寄存器，可以在内存传输前后保存数据块。

当准备好传输数据时，由 DMA 控制器通知 CPU 来获得总线的使用权，CPU 继而停止使用总线并转交给 DMA 控制器使用。

在内存和 DMA 之间的数据传输完成后，CPU 继续使用总线，实际上 CPU 只是在一小段时间内是空闲的，即在 DMA 和内存间传输数据时才会空闲。

1.4 Software

在早期的计算机中，用户都是在计算机外部进行编程的，程序体现为对系列开关的开闭和配线的改变。

von Neumann 模型要求内存必须可以存储程序，而且程序必须是有序的指令集，其中的每一条指令操作一个或者多个数据项。例如，在最简单的相加操作中也包含四个独立的指令集，首先输入两个数据，然后将二者相加并将结果保存在存储器中，最后输出结果。

通过仔细地定义计算机可以使用的不同指令集，可以使指令集具备重用性，这样就可以组合不同的指令来创建任意的程序，每个程序本身可以是不同指令的不同组合。

计算机语言是指令的抽象，早期的计算机语言称为机器语言（位模式），后来进化为使用符合来表示二进制模式。

不同的开发范式产生了不同种类的计算机语言，例如：

- 过程式语言
- 面向对象语言
- 函数式语言

1.5 Algorithm

为了解决问题，首先可以以顺序方式来开发程序，然后可以改进程序算法，从而尽量找到合适的指令序列来解决问题。

用户在完成某一任务时，往往需要遵循程序设计原理和规则，这些原理和规则的综合就是现代软件工程。

操作系统是公用指令的抽象，最初的操作系统仅仅是为程序访问计算机部件提供方便。例如，读入和写入数据的操作就可以被提取出来组成公共的组件。

Chapter 2

Numberic

2.1 Overview

为了把数据存储到计算机中，作为“纸和笔的代表物”，数字系统可以将不同的数字转换为统一的格式。

具体来说，数字系统定义了如何使用特定的符号来表示数字，不同的记数系统有不同的表示方法，其基础都是位置记数法。

在使用位置记数法的数字系统中，数字中的符号所处的位置决定了其表示的值，每个位置有一个位置量与其相关联。

> 使用非位置记数法的记数系统（例如罗马数字）并不用在计算机中，它们仍然使用有限的数字符号，每个符号有一个值，但是符号所处的位置通常与其值无关——每个符号的值是固定的。

Table 2.1: 罗马数字系统的符号取值

符号	I	V	X	L	C	D	M
值	1	5	10	50	100	500	1000

> 玛雅文明发明了位置化的二十进制数字系统，该系统使用的 20 个符号建立在 3 个更简单的符号之上，其先进特征在于它有符号 0。

> 巴比伦文明发展了首个位置化数字系统，在继承了闪族人和阿卡得人的数字系统的基础上，发展出位置化的六十进制数字系统，而且 60 现在仍然用于时间和角度记数中。

在不同的数制之间转换时，数字的值是等价的。

用户可以将十进制数转换为与其等值的任意底（base），其需要两个过程，分别用于整数部分和小数部分。

- 整数部分需要连除；
- 小数部分需要连乘。

现在，Internet 的公共底是 256，因此可以使用 256 个符号来表示该数字系统中的数字，设计者使用十进制数字 0 到 255 来表示其中一个符号。

$$S = \{0, 1, 2, 3, \cdots, 255\}$$

IP 地址系统中的数字总是以 $S_1.S_2.S_3.S_4$ 的形式出现，不过 IP 地址也可以表示为位模式，其中使用 32 位表示一个地址，在点十进制记数法的一个符号占用 8 个数字。

2.2 Data Type

在现代计算机中，数据可以以不同的形式出现（例如数字、文字、音频、图像和视频等），但是所有计算机外部的数据都采用统一的数据表示法转换后才能存储到计算机中。

为了把不同的数据类型存储到计算机中，需要使用统一的方式来将数据存储到计算机内部，并且在从计算机输出时再还原成对应的数据类型。

通常来说，所有的数据都是以位模式存储在计算机内部的，而且现在提供了多种方法来处理数字的正负问题。例如，整数可以使用下面的不同的格式进行存储：

- 无符号格式

  通常计算机都定义了一个最大无符号整数的常量，称为最大无符号整数（$2^n - 1$）。
- 符号加绝对值
- 二进制补码

为了更有效地利用内存，无符号和有符号的整数的存储方式是不同的，因此在对数字类型的数据进行进一步扩充时，实数使用浮点格式进行存储。

计算机使用两种不同的方法（定点和浮点）来表示小数点。

- 定点用于把数字作为整数存储（没有小数部分）；

- 浮点用于把数字作为实数存储（带有小数部分）。

不同的计算机系统引入了相应的编码系统（例如 ASCII 和 Unicode 等）来保存文本数据。

- 通过采样、量化和编码来存储音频数据；
- 通过光栅和矢量模式来存储图像数据；
- 通过图像随时间的变化来存储视频数据。

2.2.1 bit

位（bit，binary digit）是数据存储在计算机中的最小单位 i，本身只能是 0 或 1。

具体来说，位代表设备的两种状态中的一个，一般使用 1 表示高电位，0 表示低电位，因此现代计算机就是使用多种不同的双态设备来存储数据的。

2.2.2 byte

计算机根据位模式来存储数据，位模式（或位流）可以进一步进行扩展。例如，长度为 8 的位模式称为 1 个字节（byte），计算机一次能处理的最长位数称为字（word）。

实际上，数据是以字的位组的形式在内存中传入和传出的。

为了在存储数据时消耗较少的存储空间，可以对数据进行压缩后再存储。

在数据传输和存储时都要进行错误校验和纠正。

2.3 Integer

整数是完整的数字（没有小数部分），并且可以被当作小数点位置固定的数字，小数点固定在最右边。

计算机存储正负数的方式是不同的，定点表示法用于存储整数，其中最高位为 1 表示负数，最高位为 0 表示正数。

2.3.1 Unsigned Integer

无符号整数是永远不会为负的整数。

2.3.2 Signed Integer

在存储有符号整数时，可以使用符号加绝对值格式，其中最高位用于指示符号且其余位定义绝对值，符号和绝对值互相分开。

2.4 Complement

几乎所有的计算机都使用二进制补码表示法来存储位于 n 位存储单元中的有符号整数。

无符号整数的有效范围被分为两个相等的子范围，其中第一个子范围用来表示非负整数，第二个子范围用于表示负整数。

在二进制补码表示法中，最左位决定整数的符号，但是符号和绝对值互相分开。

为了获得二进制补码，可以首先转换成二进制反码，然后把结果加 1。

在十进制数中，与二进制反码对等的称为十进制反码（例如 $1 = 2 - 1$ 和 $9 = 10 - 1$），因此对于 n 位的单元可以使用十进制反码表示数字的范围是：

$$-[(10^n/2) - 1] \sim +[(10^n/2) - 1]$$

带有 n 个数码的十进制反码数字通过下面的方法获得：

- 如果数字为正，十进制反码就是其本身；
- 如果数字为负，将每个数码减 9。

在十进制数中与二进制补码对等的称为十进制补码，因此对于 n 位的单元，使用十进制补码表示数字的范围是：

$$-(10^n/2) \sim +(10^n/2 - 1)$$

带有 n 个数码的十进制补码数字通过下面的方法获得，先求出该数字的十进制反码，接着给结果加 1。

Part II

Network

Chapter 3

Overview

计算机网络是硬件和软件的组合，其中硬件把信号从网络中的一个节点传送到另一个节点，而软件用于使用户可以使用网络服务。

网络的性能可以用多种方式来衡量，例如传输时间、响应时间等。

- 传输时间是信息从一个设备传输到另一个设备所需的时间总量。
- 响应时间是查询和响应之间的时间间隔。

网络的性能依赖于许多因素，例如用户数、传输介质类型、硬件的连接能力和软件的效率等。

除了发送的准确性之外，还可以从发生故障的频率、从故障中恢复的时间、灾难时网络的健壮性等因素来衡量网络的可靠性。

网络的安全问题包括数据保护、防范非授权访问、破坏和修改，实现从数据破坏和数据丢失中恢复的策略和程序等。

3.1 Connection

链路是数据从一个设备传输到另一个设备的通信信道，可能的连接类型包括点对点和多点。

- 点对点连接提供了两个设备间的专用链路，而且链路的整个容量为两个设备的传输所拥有。
- 多点连接（或多站连接）是两个以上的指定设备共享一个链路，因此信道的容量被共享。

3.2 拓扑

网络的拓扑是所有链路和设备（通常为节点）间关系的几何表示，包括网状型、星型、总线型和环型。

实际上，网络的物理拓扑都是两个或多个设备连接到一个链路，一个或多个链路形成拓扑。

- 在网状拓扑中，每个设备都有专用的点对点链路与其他每个设备相连。
- 在星型拓扑中，每个设备都有专用的点对点链路，只与集线器（Hub）相连。
- 在总线拓扑中，所有的设备都连接到总线，而且每个节点使用分支线和连接器与总线相连。
- 在环形拓扑中，每个设备都有专用的点对点链路，并且只与两边的设备相连，信号只是以一个方向沿着环从一个设备传输到另一个设备，直到到达目的地。
-

在环形拓扑中，环中的每个设备连接一个中继器（Router），这样当一个设备收到要发送到另一个设备的信号时，中继器可以重新生成二进制位并传输它们。

星型拓扑比网状拓扑更便宜，而且具有网状拓扑的大多数优点，只是其整个拓扑依赖单个点（即集线器）。

3.3 Router

当两个或多个网络通过路由器连接在一起时，它们就变成了互联网（internet），其中最著名的互联网就是因特网（Internet）。

路由器（Router）是发送数据包（消息），并使其在互联网中传输的连接设备。

- 网络本身可以是一组连接在一起的通信设备（例如计算机和打印机）。
- 互联网本身是能够通过路由器进行互相通信的两个或多个网络。

Chapter 4

Protocol

4.1 TCP

使用计算机解决问题的基础工作是由硬件完成的，用户通过软件可以控制问题求解过程，硬件工作的细节问题由软件层处理。

计算机网络提供的服务也可以类比于解决问题，例如发送电子邮件的任务可以被分解为更小的子任务。

实际上，计算机网络的每层完成一个任务，而且每层都使用更低层的服务。

在计算机网络的最底层，信号或信号组被从源计算机传送到目的计算机，这也是网络协议的最底层。

计算机网络协议允许使用不同技术的 LAN 和 WAN 互相连接到一起，从而可以从一点向另一点传送信息。

原始的 TCP/IP 协议族被定义为 4 层，分别是主机到网络层（或链路层）、互联网层（网络层）、传输层和应用层，现在已经发展为 5 层。

Table 4.1: TCP/IP 协议族

TCP/IP	应用层	5	应用层	原始的 TCP/IP
	传输层	4	传输层	
	网络层	3	互联网层	
	数据链路层	2	主机到网络层	
	物理层	1		

在网络协议分层结构中，每一层调用其直接下层的服务，路由器只使用 TCP/IP 协议族的前三层。

应用层允许用户（人或者软件）访问网络，并提供对电子邮件、远程文件访问和传输、浏览 Internet 等服务的支持。

应用层负责向用户提供服务，而且应用层是唯一一个大多数 Internet 用户能够直观感受的层。例如，客户/服务器体系结构和对等体系结构都可以用于允许不同计算机上的应用程序互相通信，它们都需要应用层的支持。

在客户/服务器体系结构中，每个应用由两个分开但相关的程序（客户端程序和服务器端程序）组成。

- 客户端程序只在需要时运行。
- 服务器端程序需要一直运行。

客户端程序和服务器端程序之间的通信称为进程到进程的通信，其中服务器端进程一直运行，并等待接收客户端进程的请求。

用户运行休眠的客户端程序就可以将其转变为客户端进程，使得客户端进程请求服务，而且将被服务器端进程响应。

一般情况下，服务器端进程可以响应多个客户端进程的请求并返回数据。

4.2 IP

当客户需要向服务器发送请求时，首先需要服务器应用层的地址，而且服务器应用层地址不能用来发送消息的，它只是帮助客户找到服务器的实际地址。

在应用层，客户端位置不需要进行标识，不同的服务器有不同的应用层地址。举例来说，在只知道一个人的姓名时无法向其发送信件，邮局无法仅凭姓名投递信件，还需要收信人的实际地址。

应用层地址可以帮助客户端找到服务器的实际地址，也就是服务器的 IP 地址。

实际上，客户端进程是通过 DNS 服务器来找到服务器的 IP 地址的，DNS 服务器包含将域名匹配到 IP 地址的目录。

客户端准备和发送信息到 DNS 服务器，询问其所需要的服务器的实际 IP 地址，DNS 逐级查找并获取结果，否则返回错误消息。

传输层负责客户和服务器之间进程到进程的消息的传输，并建立客户和服务器的传输层的逻辑通信，因此应用层可以把传输层看作是负责消息传输的代理。