

# Introduction to CS

2019年11月27日 10:40

计算机科学与技术：它是什么，为什么学习它？

---

第一个问题

$1+1=?$

关于第一个问题的一些问题

你能从数学上(形式上)定义数字“1”吗?

你可以定义添加的“ $1+1=?$ ”(加法关系/函数 $(x+y)$ )在数学上(形式上)?

你能从数学上(形式上)证明“ $1+1=2$ ”这个等式吗?

你能以数学(正式)的方式计算加法“ $1+1=?$ ”吗?

你能制造一台自动计算机来计算加法“ $1+1=?$ ”吗?

你能制造一台自动计算机来证明“ $1+1=2$ ”这个等式吗?

更多的问题

“从数学(形式上)定义某物”是什么意思?

“用数学(正式)证明某事”是什么意思?

“计算某物”是什么意思?

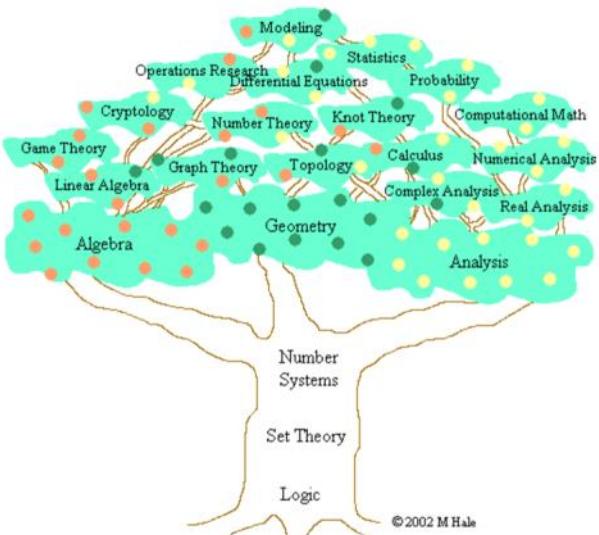
什么是“数学(形式)方法”?

什么是“自动化计算机”?

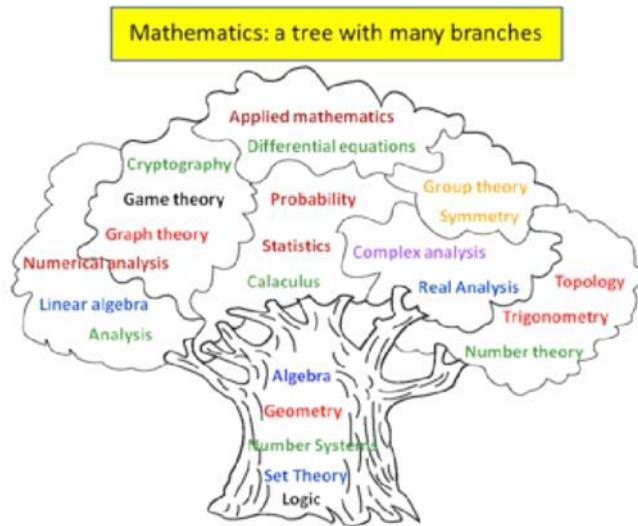
“制造”意味着什么吗?

---

“算术上的 (形式上) ” 到底意味着什么



数学可以被认为是一棵有着很多分支的数



“拟议科学和技术领域的国际标准命名法”，

11. 逻辑, 12. 数学

21. 天文学和天体物理学, 22. 物理,

23. 化学, 24. 生命科学、25. 地球与空间科学

31. 农业科学、32. 医学科学, 33. 技术科学

1203. 计算机科学3

304. 计算机技术

计算机已经到了非常小的分支中去了

集合set的概念(从朴素集合论的观点来看)

集合论的基本思想是把一组对象看作一个单独的实体，并把它表示出来。

集合是不同different对象(称为集合的元素element)的集合，它们可以彼此区分。

集合的例子:{1,2,3}, {1,2,3,a, b, c}, {1, 2, 3, ...})

非集合的例子:{1,2,2,3}, {1,2,3,a, b, c, c}

$x \in (\notin) A$ :  $x$ 是(不是)是 $A$ 的一个元素,  $x$ 属于(不属于)  $A$ 。

空集( $\emptyset$ )

空集没有元素。

$\emptyset = df (\forall x)(x \notin \emptyset)$ (对于所有 $x$ ,  $x$ 都不是 $\emptyset$ 中的元素)

集合的概念

有限集finite set

有限集:包括有限元素的集合。

一个有限集的size(表示为 $|A|$ ):  $a$ 元素的数量

( $|\emptyset| = df 0$ )

无限集infinite set

无限集合:包括无限元素的集合。

Note: 无限集没有“大小”的概念。

集合的扩展定义:枚举所有元素。

集合的内涵定义: 定义所有元素都具有的属性。

Ex:  $N_{10000} = df\{x | x \in N \wedge x < 10000\}$

---

子集subset和包含inclusion

集合A称为集合B的子集，如果集合A的所有元素都是集合B的元素。

- ◆ **Inclusion relation:**  $A \subseteq B =_{df} (\forall x)(x \in A \Rightarrow x \in B).$
- ◆ **For any set A,  $A \subseteq A, \emptyset \subseteq A.$**
- ◆ **Proper subset:**  $A \subset B =_{df} (A \subseteq B) \wedge (A \neq B).$

集合的幂集power set

集合A的幂集是包含集合A的所有子集的集合。

$$P(A) =_{df} \{x | x \subseteq A\}, 2^A =_{df} \{x | x \subseteq A\}$$

Ex:  $A = \{1, 2, 3\}, P(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$



---

#### ♣ Equivalence of two sets

- ◆  $A = B =_{df} (A \subseteq B) \wedge (B \subseteq A)$
- ◆  $A \neq B =_{df} \neg(A = B) =_{df} \neg(A \subseteq B) \vee \neg(B \subseteq A)$

#### ♣ Join (Union) of sets

- ◆  $A \cup B =_{df} \{x | (x \in A) \vee (x \in B)\}$

#### ♣ Meet (Intersection) of sets

- ◆  $A \cap B =_{df} \{x | (x \in A) \wedge (x \in B)\}$

#### ♣ Difference of sets

- ◆  $A - B =_{df} \{x | (x \in A) \wedge (x \notin B)\}$

#### ♣ Symmetric difference of sets

- ◆  $A \oplus B =_{df} \{x | ((x \in A) \wedge (x \notin B)) \vee ((x \notin A) \wedge (x \in B))\}$

那个下面角标df是什么意思，不是很懂

还是说就没什么意思

---

如何以数学的方法定义自然数

- ◆  $0 =_{\text{df}} \emptyset$  (define 0 as the empty set)
  - ◆  $1 =_{\text{df}} \{0\} = \{\emptyset\} = \emptyset \cup \{\emptyset\} = 0 \cup \{0\}$
  - ◆  $2 =_{\text{df}} \{0, 1\} = \{\emptyset, \{\emptyset\}\} = \emptyset \cup \{\emptyset\} \cup \{\emptyset \cup \{\emptyset\}\}$   
 $= 0 \cup \{0\} \cup \{1\} = 1 \cup \{1\}$
  - ◆  $3 =_{\text{df}} \{0, 1, 2\} = 0 \cup \{0\} \cup \{1\} \cup \{2\} = 2 \cup \{2\}$
  - ◆  $4 =_{\text{df}} \{0, 1, 2, 3\} = 0 \cup \{0\} \cup \{1\} \cup \{2\} \cup \{3\} = 3 \cup \{3\}$
  - ◆  $5 =_{\text{df}} \{0, 1, 2, 3, 4\} = 0 \cup \{0\} \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\} = 4 \cup \{4\}$   
 $\vdots$
  - ◆  $n + 1 =_{\text{df}} n \cup \{n\}$   
 $\vdots$
- 

有序对ordered pair

- ◆  $(a, b) =_{\text{df}} \{\{a\}, \{a, b\}\}$

笛卡尔积，集合的直接积artesian product

- ◆  $A \times B =_{\text{df}} \{(a, b) \mid a \in A, b \in B\}$
  - ◆ Ex:  $A = \{1, 2, 3\}, B = \{x, y, z\}$   
 $A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z), (3, x), (3, y), (3, z)\}$
  - ◆  $A_1 \times A_2 \times \dots \times A_n =_{\text{df}} \{(a_1, a_2, \dots, a_n) \mid a_1 \in A_1, a_2 \in A_2, \dots, a_n \in A_n\}$
  - ◆ **Cartesian power** (exponentiation):  $A^n =_{\text{df}} A \times A \times \dots \times A$
- 

关联relation的概念

二元关系binary relation

从集合A(源soucre, 从集合from-set)到集合B(目标traget, 到集合to-set)  
的二元关系R定义为

$$R: A \rightarrow B =_{\text{df}} R \subseteq A \times B.$$

任何二元关系都是有序对的集合。

一个二元关系R: A → B定义了一个可能有很多实例的抽象关系(相关的两个集合A和B)。

AxB的任意具体(显式枚举所有元素)子集定义了从a到b的具体二进制关系。

二元关系的定义域domain和值域range

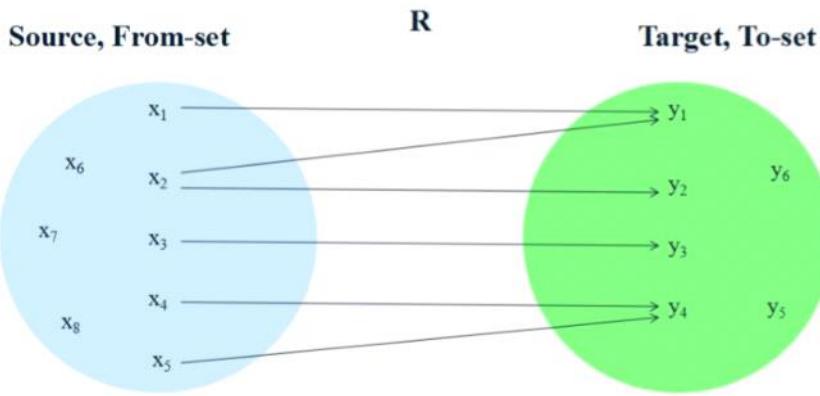
$$\text{Domain: } \text{dom}(R) =_{\text{df}} \{a \mid (\exists b)((a, b) \in R)\}, \text{dom}(R) \subseteq A$$

$$\text{Range: } \text{ran}(R) =_{\text{df}} \{b \mid (\exists a)((a, b) \in R)\}, \text{ran}(R) \subseteq B$$

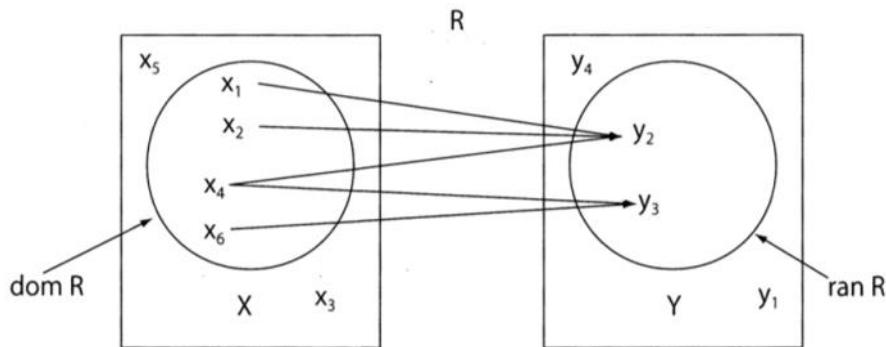

---

关系:源、从集、目标、到集

从源到目标的关系



定义域和值域



一个y可以对应多个x, 一个x只能对应一个y

如何从数学上定义加法关系addition relation?

◆ Let  $N$  be the set of natural numbers

◆  $AddR: N \rightarrow N =_{df} AddR \subseteq N \times N$ .

加法关系 (小于5的自然数)

◆  $AddR_5: N_5 \rightarrow N_5$   
 $=_{df} \{ (0,0), (0,1), (0,2), (0,3), (0,4),$   
 $(1,0), (1,1), (1,2), (1,3), (0,4),$   
 $(2,0), (2,1), (2,2), (2,3), (2,4),$   
 $(3,0), (3,1), (3,2), (3,3), (3,4),$   
 $(4,0), (4,1), (4,2), (4,3), (4,4) \}$

函数function的概念

二元函数binary function

一个从集合A到集合B的二元函数定义为

$f: A \rightarrow B =_{df} f \subseteq A \times B \wedge$   
 $(\forall x)(\forall y)(\forall z)((x \in A \wedge y \in B \wedge z \in B) \Rightarrow (((x, y) \in f \wedge (x, z) \in f) \Rightarrow$   
 $y = z))$

任何二元函数都是一个二元关系(反之则不一定为真)和一组有序对。

二进制函数 $f: A \rightarrow B$ 定义了一个可能有很多实例的抽象函数。 $A \times B$ 的一些子

集定义了从A到B的具体二进制函数。

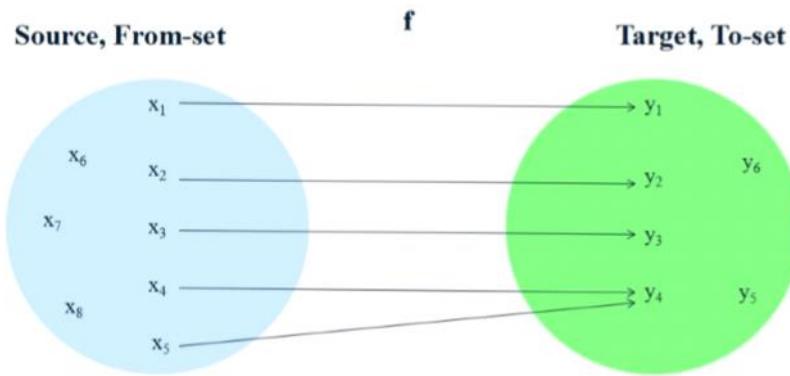
二元函数的定义域和值域

**Domain:**  $\text{dom}(f) = \{a \mid (\exists b)((a, b) \in f)\}$ ,  $\text{dom}(f) \subseteq A$

**Range:**  $\text{ran}(f) = \{b \mid (\exists a)((a, b) \in f)\}$ ,  $\text{ran}(f) \subseteq B$

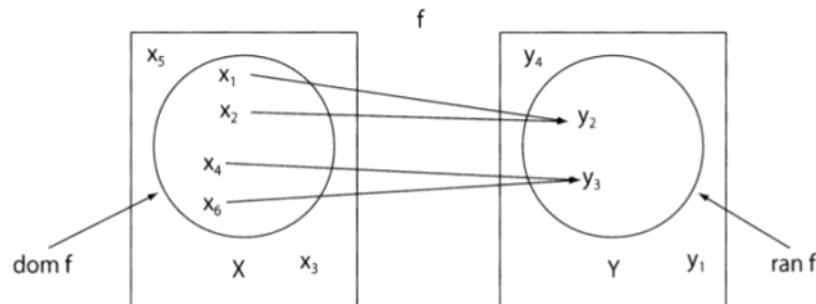
---

二元函数



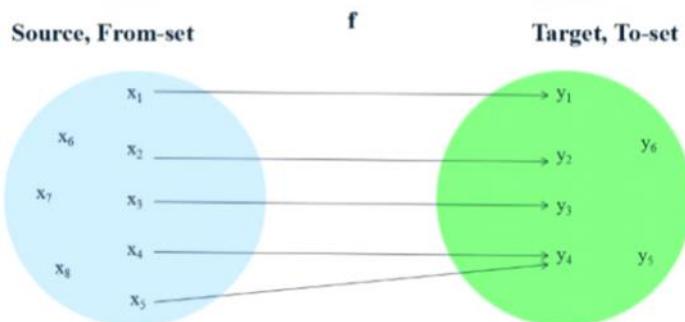
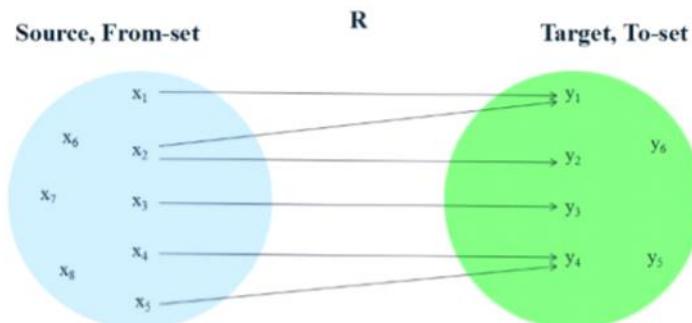
---

二元函数的定义域和值域



---

关联relation和函数function的区别



关联中x可以对应多个y, y可以对应多个x

而函数中x只能对应一个y, y可以对应多个x

如何定义加法函数

**Let  $N$  be the set of natural numbers.**

$AddF: N \rightarrow N \rightarrow N =_{df} AddF \subseteq N \times N \times N.$

加法函数 (自然数小于5的加法)

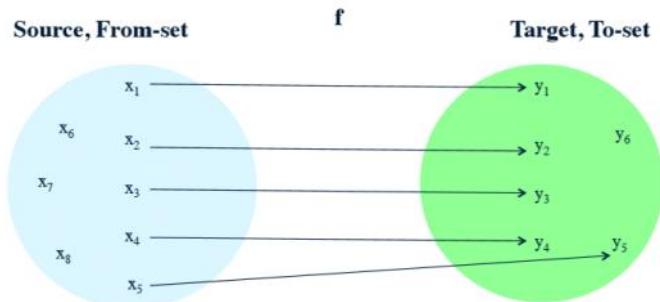
$$\diamond N_5 = \{0, 1, 2, 3, 4\}, N_9 = \{0, 1, 2, 3, 4, \dots, 8\}$$

$$\diamond AddF_5: N_5 \rightarrow N_5 \rightarrow N_9 \\ =_{df} \{(0,0,0), (0,1,1), (0,2,2), (0,3,3), (0,4,4), \\ (1,0,1), (1,1,2), (1,2,3), (1,3,4), (1,4,5), \\ (2,0,2), (2,1,3), (2,2,4), (2,3,5), (2,4,6), \\ (3,0,3), (3,1,4), (3,2,5), (3,3,6), (3,4,7), \\ (4,0,4), (4,1,5), (4,2,6), (4,3,7), (4,4,8)\}$$

单射 injection、满射 surjection、双射 bijection

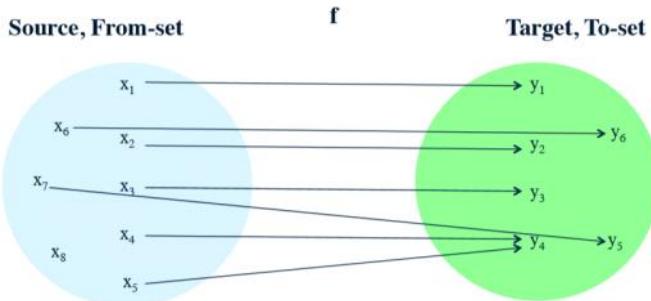
单射

一个单射函数，或“一对一”，是一个将源的不同值映射到目标的不同值的函数。



满射

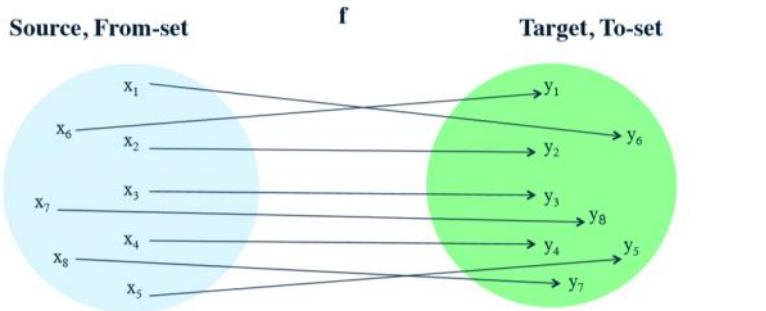
一个满射函数，或“on-to”，是一个函数，其范围是其target集合的全部。



双射

一个双射函数，或“一对一”“对应”是一个函数，它以一对一的关系将源的每个元素映射到目标的每个元素上。

双射是既是单射又是满射。



# CS: what is it and why study it

2019年11月30日 10:49

计算机科学与技术：它是什么，为什么为什么要学习它

---

CS: 它是什么

要回答“什么是CS?”这个问题，我们必须考虑以下更基本的问题：

什么是科学？

电脑是什么？

本课程将详细回答第二个问题。

Notes

从科学或工程学的观点来看，“计算机科学”是一门学科的错误名称。

“计算科学computational science”、“计算科学computation science”或“计算机工程computer engineering”是更好的名称。

---

争论点

“计算机科学是科学吗？”

“计算机科学真的是科学吗？”“为什么？为什么不？”

“计算机科学”是一门“自然科学”吗？

计算机科学是一门工程学科吗？

计算机科学：它真的是一门科学吗？

它是关于什么的科学？

“计算机科学不是一门科学”

“计算机科学不是真正的科学”

Note: 《计算机科学与技术》 《计算机科学与工程》

---

科学和工程/技术的对比

科学的各种定义

“知晓的状态或事实；对特定的或隐含的事物的认识或认识；此外，广泛的参考，知识(或多或少广泛)作为个人属性。”

“与良心相对或结合，强调对真理的理论感知和道德信念之间的区别”

“通过学习获得的知识；熟悉或掌握任何学习部门。还有各种各样的知识。”

“知识或研究的一个特定分支；一个公认的学术部门。”

“更受限制的意义：研究关注的一个分支连接身体的证明真理或观察到的事实综合系统地分类和或多或少受到一般的法律，和包括值得信赖的方法对新事实的发现在自己的领域。”

“以各种‘科学’为例的知识或智力活动。”

“在现代应用中，通常被视为‘自然和物理科学’的同义词，因此仅限于与物质宇宙现象及其规律有关的研究分支，有时隐含着对纯数学的排斥。这是现在在日常使用中占主导地位的意思。”

自17世纪以来作为自然科学特征的一种方法，包括系统的观察、测量和实验，以及对假设的提出、检验和修正。

“任何有关物质世界及其现象的知识体系，包括公正的观察和系统的实验。一般来说，一门科学涉及到对知识的追求，包括一般真理或基本法则的运作。”

### 工程的各种定义

动词工程师的动作;由工程师或其职业所做的工作

“工程师职业的艺术和科学。”

“将科学应用于自然资源的最佳转化为人类的使用。”定义的字段已经工程师职业发展委员会,在美国,创造性应用的科学原理设计或开发结构,机器,设备,或制造工艺,或有效利用它们单独或结合构造或操作相同的充分认知自己的设计;或预测它们在特定操作条件下的行为;所有这些都是预期的功能。经济运行和生命财产安全。”

### 技术的各种定义

“关于一种或多种艺术的论述或论述;对实用艺术或工业艺术的科学研究。

集体实用艺术。

“一种特殊的实用或工业艺术。”

“将科学知识应用于人类生活的实际目的，或如有时所说，应用于人类环境的改变和控制。”

---

### 科学

知道或认识某一类事物的状态或事实。

物体或现象。无偏观测，系统实验，识别，描述，实验和理论调查，解释，预测。

### 工程

将科学和数学原理应用于实际目的，如设计、制造和操作高效、经济的结构、机器、过程和系统，以使这些人造物体具有所要求的性能，并因此能正常工作。

### 技术

把科学知识应用于实践是为了解决问题或达到某种目的。

### 科学

什么?(目标不一定明确定义)为什么?

这些形容词均有“发现、观察、调查、测量、解释、预测”之意。

科学家可以享受“科学游戏”。 =

## 工程/技术

为了什么?(目标必须明确定义)如何实现?

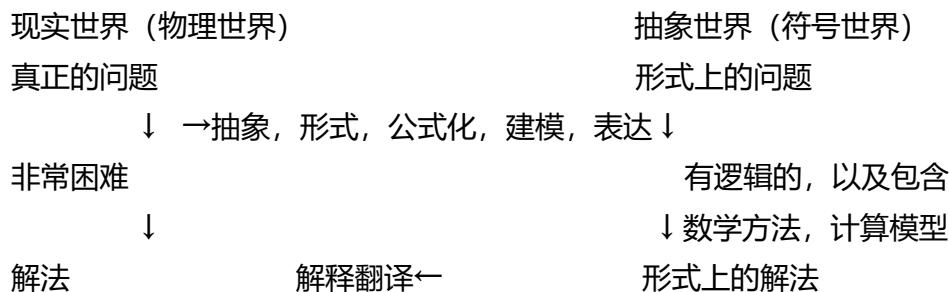
设计、计划、设计、开发、测试、评价有用、可靠、安全/可靠、健壮/强壮、有效、高效、廉价、适应力强

不允许工程师玩“工程/技术游戏”。

科学与工程/技术科学之间的关系是

科学是工程/技术的基础。

工程技术促进科学进步。



## 什么是CS

研究计算机的构造、操作、程序设计和应用的一门学科。

研究计算机及其设计，以及计算机的计算computation、数据处理data processing和系统控制systems control方面的应用，包括计算机硬件hardware、软件software和编程programming的设计和开发。

计算机的研究，它们的基本原理和用途。

它包括下列主题:

编程;

信息结构;

软件工程;

编程语言;

编译器和操作系统;

硬件设计与测试;

计算机系统架构;

计算机网络和分布式系统;

系统分析与设计;

信息、系统和计算理论;

- **programming;**
- **information structures;**
- **software engineering;**
- **programming languages;**
- **compilers and operating systems;**
- **hardware design and testing;**
- **computer system architecture;**
- **computer networks and distributed systems;**
- **system analysis and design;**
- **theories of information, systems, and computation;**

计算机的研究，它们的基本原理和用途。它包括以下主题:

应用数学与电子学;

计算技术(如图形、仿真、人工智能和神经网络);

应用程序;

计算的社会、经济、组织、政治、法律和历史方面。

“这并不是一个严格意义上的科学是一门学科采用科学的方法来解释自然和社会现象(虽然与物理连接、心理学和行为科学),而是宽松意义上的系统的基础理论知识。”

“由于它最终涉及到有关设计和建造有用系统的实际问题，在成本和可接受性的限制下，它既是一门科学，又是一门工程。”

---

计算学科computind discipline

离散结构

编程基础

算法和复杂性

体系结构组织和操作系统

网络中心计算

编程语言

人机交互

图形和可视化

计算智能系统

信息管理

社会及专业事务

软件工程

计算科学

Discrete Structures
Programming Fundamentals
Algorithms and Complexity
Architecture and Organization
Operating Systems
Net-Centric Computing
Programming Languages
Human-Computer Interaction
Graphics and Visual Computing
Intelligent Systems
Information Management
Social and Professional Issues
Software Engineering
Computational Science

“计算机科学的学科是计算机专业人员在工作中使用的知识和实践的主体。”

“计算机科学这门学科诞生于20世纪40年代初，它融合了算法理论、数学逻辑和存储程序电子计算机的发明到20世纪60年代初，已经有了足够的知识体系来建立第一批学术部门和学位项目。这门学科也被称为计算机科学与工程、计算和信息学。

Note:

第一个计算机科学系于1962年在普渡大学和斯坦福大学成立。

1965年，宾夕法尼亚大学授予了第一个计算机科学博士学位。

计算机科学课程由ACM在1968年出版

计算的知识体系经常被描述为对描述和转换信息的算法过程的系统研究：它们的理论、分析、设计、效率、实现和应用。

“所有计算的根本fundamental问题是，什么东西可以(有效地efficiently)自动化automated?”

CS: 它不是什么？

CS不仅仅是编程

编程只是计算机科学众多理论和技术中的一种。

计算机科学家/软件工程师不是程序员

计算机科学家/软件工程师要做的工作比程序员多得多。

计算机理论不是纯数学，也不是应用数学理论

Theoretical CS只是应用逻辑和离散的从数学到计算理论。

在应用数学时，计算机科学需要更多的实际考虑。

CS: 为什么要学习它？

这是必不可少的，绝对必要的在现代，没有任何规程不调用CS，并且可以在没有CS的情况下进行。

在我们的日常生活中，无处不在的计算是“为用户提供随时随地的计算方

式，使他们可以使用计算系统，甚至不考虑他们”。

**它很重要**

计算方法已成为几乎所有学科的主要研究方法。“计算逻辑”、“计算数学”,“计算几何”、“计算语言学”、“计算物理学”、“计算力学”、“计算流体动力学”、“计算化学”、“计算生物学”、“计算天文学”、“计算经济学”……

**它是创新**

自1950年代诞生以来。计算机科学是最具创新性和发展最快的学科，在21世纪，这一趋势必将继续。

**它很有效effective**

CS提供并将继续提供解决现实世界中各种问题的有效方法。

**它很高效efficient**

CS提供并将继续提供高性能计算，以满足实际应用中的效率要求。

**它为你提供工作**

有许多不同的CS工作，包括研究、开发和管理。

---

**计算机协会ACM**

ACM是世界上最大的教育和科学计算学会，它提供的资源将计算作为一门科学和一门专业来推进。ACM提供计算领域首屈一指的数字图书馆，并为其成员和计算专业提供领先的出版物、会议和职业资源。

**ACM图灵奖**

是计算机界的“诺贝尔奖”

---

# Computation: what is it and why study it

2019年11月30日 11:27

计算：它是什么，为什么学习它

---

计算：它是什么？

计算computation：一种计算的行为或工序

事实：在各种权威的百科全书、字典等都没有对计算的准确定义

现在：没有一个能被所有学者或条例接受的关于计算的定义

---

计算的历史起源

被Hilbert提出的关于“可判定问题” decision problem要求一种有效的方法来确定经典微积分中任意给定公式的有效性。

有效的方法：有效的程序，有效的计算，意味着是有限的计算步骤a finite step way (algorithm)

逻辑是CS的父母

正是“可判定问题”导致了对计算这一观念的注意，进而引发了对计算能力computability的一系列研究

---

最早关于计算的一些模型

递归函数，微积分，图灵机Turing machine，波斯特系统

一些图灵机计算的模型

一些图灵完备和/或图灵等价的计算模型：组合逻辑，重写系统，寄存器机器

其他关于计算的模型

是否存在一切其他比图灵机更加高端的模型？

---

人类怎么计算

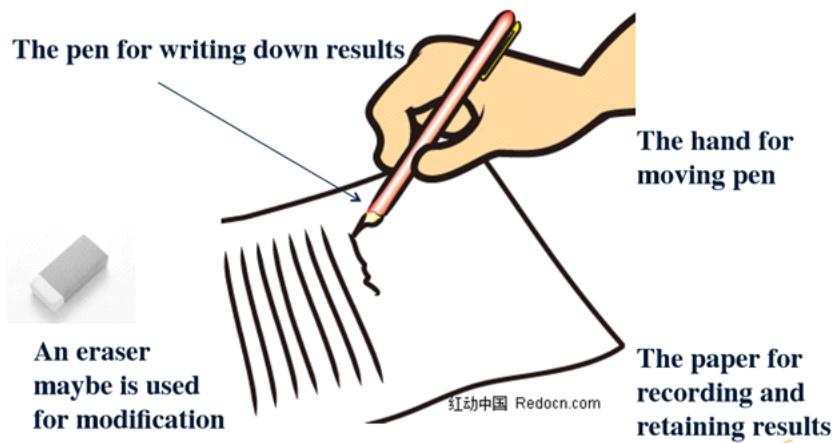
关键点：我们有一个脑袋来操纵计算

一支笔来写下结果

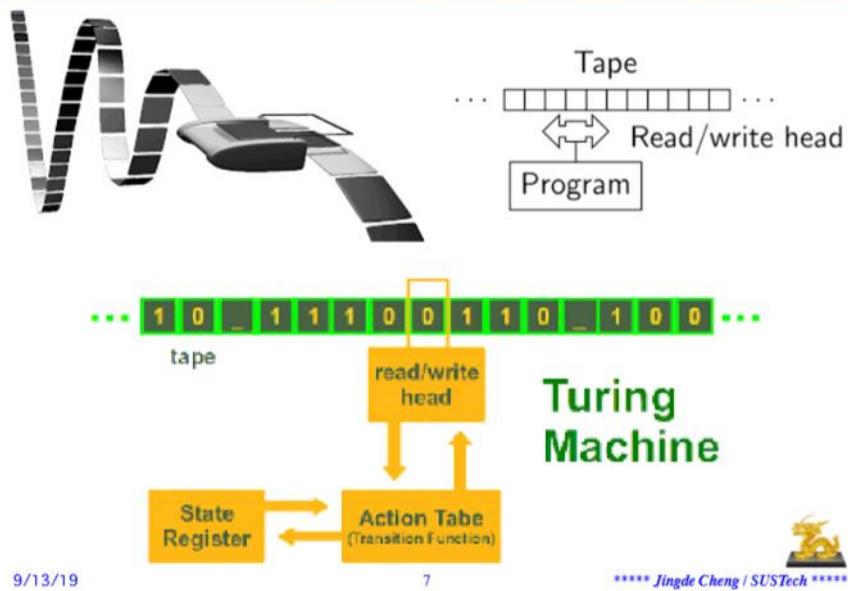
一个手来移动笔

一个橡皮可能可以用于修改

一张纸来记录或者保留结果



图灵机：一个理论的计算模型



### 纸带tape

图灵模型有一个无穷的纸带infinite tape，它被分成了无穷多个小隔间 infinite cells，每一个小隔间都可以包含任何一个有限的符号的集合，它有无限的记忆

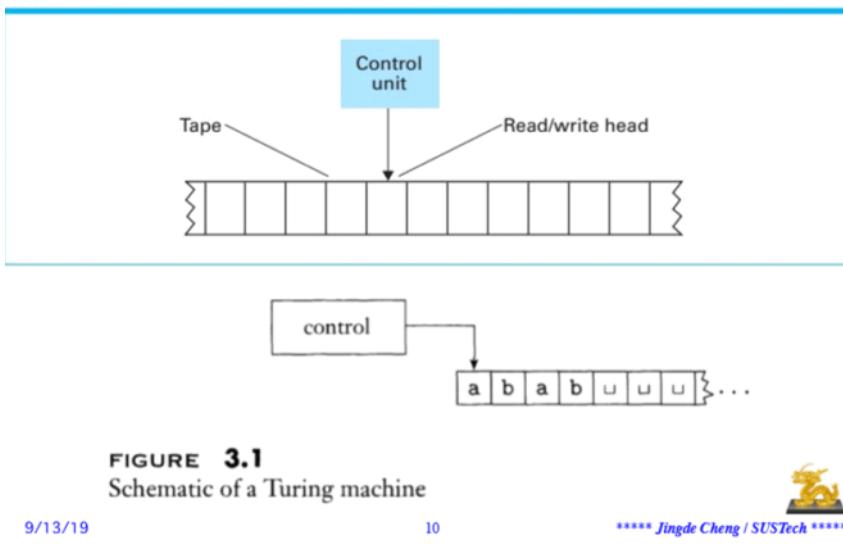
### 读写头read/write head

图灵模型有一个可以读和写符号以及可以移动纸带的读写头，它的活动被一个控制单元所控制

### 控制单元control unit

图灵模型有一个可以控制读写头的控制单元

Figure 12.2 The components of a Turing machine



9/13/19

10

\*\*\*\*\* Jingde Cheng / SUSTech \*\*\*\*\*



### 初始状态initial state

图灵机有一个初始/输入状态使得纸带只包含了输出串并且是空白的  
一个图灵机的计算工序从它的initial state开始

### 接受和拒绝状态accept and reject states

图灵机有两种特别的输出状态out put

一个图灵机的计算工序不会停止halt除非它进入了接受状态或者拒绝状态，  
然后程序会输出一个接受accept或者拒绝reject，否则会一直持续下去，永  
不停止

### 计算工序computation process

一个图灵机的计算工序 (带有一个程序)包括了有尽的或者无尽的被图灵机  
的控制单元control unit执行的顺序步骤sequence steps

一个计算工序从初始状态下开始

每一个计算步骤包括了：读取处于当前纸带小隔间的符号，把符号写入这  
个隔间，然后把读写头从当前隔间向左或者向右移动

### 计算工序的结果

一个图灵机的计算工序不会停止halt除非它进入了接受状态或者拒绝状态，  
然后程序会输出一个接受accept或者拒绝reject，否则会一直持续下去，永  
不停止

### 字母表，字符串和语言

#### 字母表alphabets

字母表是一个不空non-empty且有限finite的符号的集合

我们通常会用希腊符号 $\Sigma$ 「指定字母表

例如

- ◆  $\Sigma_1 = \{0, 1\}$
- ◆  $\Sigma_2 = \{a, b, c, \dots, z\}$
- ◆  $\Gamma = \{0, 1, x, y, z\}$

### 字符串string

来源于字母表中的字符串是一个有限的序列符号，经常被一个接着一个写，且不用逗号

对于一个给定的字母表 $\Sigma$ ,  $\Sigma^*$ 表示所有在 $\Sigma$ 中的字符串

### 字符串的长度

如果 $w$ 是一个字符串，那么 $w$ 的长度被写成 $|w|$ ，它含有符号的数量

### 一个空字符串

一个长度为0的字符串被叫做the empty string，且被写作 $\lambda$

### 字符串的反转 the reverse of string

对于一个字符串 $w_1 w_2 \dots w_n$ , 反转字符串 $w$ 被写作

$$w^R = w_n \dots w_3 w_2 w_1$$

### 字符串的连结concatenation

对于一个长为 $m$ 的字符串 $x$ 和一个长为 $n$ 的字符串 $y$ , 它们的连结被写作 $xy$ , 是 $x_1 x_2 \dots x_m y_1 y_2 \dots y_n$

连结一个字符串它自己 $k$ 次, 被叫做 $k$  times, 我们用 $x^k$ 来表示

### 字符串的前缀prefix of string

对于一个字符串 $y$ , 如果存在一个字符串 $z$ 使得 $xz = y$ , 我们说字符串 $x$ 是 $y$ 的一个前缀, 并且如果 $x$ 不等于 $y$ , 我们说 $x$ 是一个 $y$ 的proper prefix

Note: 任何一个字符串都是自己的前缀

### 语言language

对于一个给定的字母表 $\Sigma$ , 一个 $\Sigma$ 的语言, 记为 $L_\Sigma$ □是一系列在 $\Sigma$ 的字符串

对于任何 $\Sigma$ ,  $L_\Sigma \subseteq \Sigma^*$

如果没有一个成员是另一个成员的proper prefix, 那么我们说一个语言是没有前缀prefix-free

对于一个给定的字母表 $\Sigma$ ,  $\Sigma^*$ 是在该字母表里的所有字符串

### 图灵机的例子M1

#### TM M1

让M1是一个TM在语言B下拥有的的测试资格,  $B = \{w\#w \mid w \in \{0, 1\}^*\}$ , 我们想要M1处于接收状态如果它的输入是B中的一个成员, 两个先攻字符串

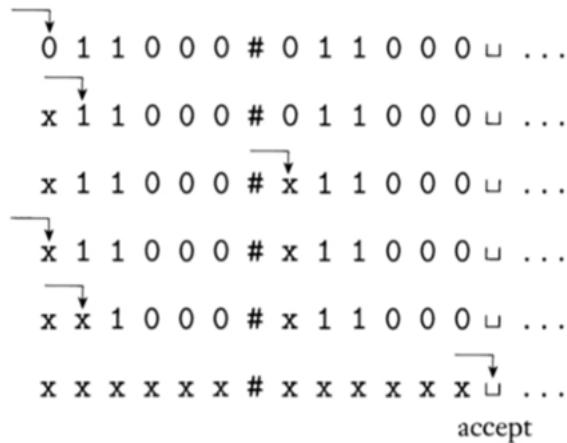
被# (hash) 分隔，然后它会拒绝其他的输入

## 工作方案

通过来回移动#到两边对应的位置来决定它们是否匹配

通过用一些符号来标记纸带的位置来记录哪些位置是对应的

让M1按如下状态工作



**FIGURE 3.2**

Snapshots of Turing machine  $M_1$  computing on input 011000#011000

按上述方法设计  $M_1$ ，让读写头在输入串上多次通过，每一次匹配#两边的一对字符。为了跟踪哪些字符已经被检查过， $M_1$  消去所有已检查过的符号，如果最后所有的符号都被消去，意味着匹配成功， $M_1$  进入接受状态；如果发现一个不匹配，就进入拒绝状态。 $M_1$  的算法如下：

$M_1 =$ “对于输入字符串  $w$ ：

- 1) 在#两边对应的位置上来回移动，检查这些对应位置是否包含相同的字符，如不是，或者没有#，则拒绝，为跟踪对应的字符，消去所有检查过的字符。
- 2) 当#左边的所有字符都被消去时，检查#的右边是否还有字符，如果是则拒绝，否则接受。”

**定义 3.1** 图灵机是一个7元组( $Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}$ )，其中： $Q, \Sigma, \Gamma$ 都是有穷集合，并且

- 1)  $Q$  是状态集。
- 2)  $\Sigma$  是输入字母表，不包括特殊空白符号 $\sqcup$ 。
- 3)  $\Gamma$  是带字母表，其中： $\sqcup \in \Gamma, \Sigma \subseteq \Gamma$ 。
- 4)  $\delta$ :  $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  是转移函数。
- 5)  $q_0 \in Q$  是起始状态。
- 6)  $q_{\text{accept}} \in Q$  是接受状态。
- 7)  $q_{\text{reject}} \in Q$  是拒绝状态，且  $q_{\text{reject}} \neq q_{\text{accept}}$ 。

$q_0$   $q_{\text{reject}}$  和  $q_{\text{accept}}$  都是  $Q$  中的成分

状态集state set  $Q$

输入字母表input alphabet  $\Sigma$

带字母表tape alphabet  $\Gamma$  都是有限集合

$\Sigma \subset \Gamma, \square \notin \Sigma, \square \in \Gamma$ .

转换函数transition function

$$\delta = (Q \times \Gamma) \rightarrow ((Q \times \Gamma) \times \{L, R\})$$

转换函数是一个来自于对于Q,  $\Gamma$ 的笛卡尔（直接）乘积函数到另一个笛卡尔（直接） $(Q \times \Gamma)$  和{L,R}乘积函数

$$(q_i, a_j) \rightarrow (q_{i+1}, a_{j+1}, L|R)$$

这个函数意味着从状态 $q_i$ 和当前的符号 $a_j$ , 下一步符号会变成 $q_{i+1}$ 和 $a_{j+1}$

1, 下一个位置可能是在当前位置的左边 (L) 或者右边 (R)

是转换函数 $\delta$ 代表了计算的步骤

---

如何用一个表来表示转换函数 $\delta$

	0	1	x	
$q_1$	$1, q_2$			
$q_2$		$q_{accept}$		
$q_3$			$q_{reject}$	

图灵机  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$  的计算方式如下：开始时， $M$  以最左边的  $n$  个带方格接收输入  $w = w_1 w_2 \dots w_n \in \Sigma^*$ ，带的其余部分保持空白(即填以空白符)，读写头从最左边的带方格开始运行，注意  $\Sigma$  不含空白符，故出现在带上的第一个空白符表示输入的结束。 $M$  开始运行后，计算根据转移函数所描述的规则进行，如果  $M$  试图将读写头从带的最左端再向左移出，即使转移函数指示的是  $L$ ，读写头也停在原地不动。计算一直持续到它进入接受或拒绝状态，此时停机，如果二者都不发生，则  $M$  将永远运行下去。

图灵机的例子M2

考虑一个M2可以判定 $A = \{0^{2^n} \mid n \geq 0\}$ ，这个语言包括了所有的长度为的幂的“0”们（也就是连续0的个数）

M2: 输入字符串w

- 1.从左到右扫过纸带，扫过每一个0
- 2.如果是在第一阶段，磁带包含一个0 接受
- 3.如果是在第一阶段，磁带包含多个0和0们数量的基数 拒绝
- 4.将磁带头放回磁带的左端
- 5.进入第一阶段

第一阶段每次迭代都将让0们的数量减半，随着机器不断扫荡在第一阶段的磁带上，它会跟踪看到0们的数量是否均匀。如果该数字是奇数且大于1，则输入0的个数不可能是2的幂，

因此机器拒绝。然而，如果0的数量是1，则起始的数量确实是2的幂，此时机器会接受

现在我们给一个关于M2

$(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

的正式定义

$Q = \{q_1, q_2, q_3, q_4, q_{\text{accept}}, q_{\text{reject}}\}$

$\Sigma = \{0\}$

$\Gamma = \{0, x, \_\}$

我们用状态图state diagram来形容 $\delta$  (见图片3.8)

$q_1$ 是开始状态start

M2的例子

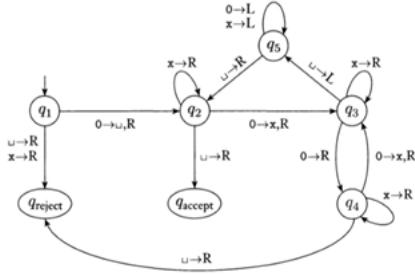


FIGURE 3.8  
State diagram for Turing machine  $M_2$

在这个状态图下，标记 $0 \rightarrow$ 空格，R出现在 $q_1$ 到 $q_2$ 的转换，这个标签的意义是，当在状态 $q_1$ ，且读写头读数是0时，机器移动到状态 $q_2$ ，并写下空格，然后移动读写头去右边。换句话说， $\delta(q_1, 0) = (q_2, \text{空格}, R)$ 。为了清楚起见，我们在 $q_3$ 到 $q_4$ 转换中使用简写 $0 \rightarrow R$ ，来表明当在状态 $q_3$ 读到0时机器会向右移动但是不会改变纸带，所以 $\delta(q_3, 0) = (q_4, 0, R)$

(为什么 $q_1$ 下面有两个判定，而明明x和空格不可能出现在 $q_1$ 的右边，因为这个意义表明你输入错误了或者是根本没有输入字符串，它是在提醒你输入无效)

### 配置configuration

当一个图灵机计算的时候，改变发生在当前的状态，当前纸带的内容，还有当前读写头的位置

这三个项目的集合叫做图灵机的配置

### 配置的表示

对于在纸带字母表中的一个状态 $q$ 还有两个字符串 $u$ 和 $v$ ，我们写作 $u q v$ 是当前状态 $q$ 下的配置，当前的纸带内容是 $uv$ ，而当前读写头的位置是在 $v$ 的第一个符号上，针对 $v$ 的第一个字符

纸带在 $v$ 的最后一个符号后面只有空（啥都没有）

### 格局转移configuration transition：通常的情况

我们说格局C1yield产生于格局C2低端如果图灵机可以合法的从一个单个的步骤从C1变化到C2

对于在 $\Gamma$ 中的a, b和c, 以及在 $\Gamma^*$ 中的v, 以及状态 $q_i$ 和 $q_j$   
 $ua\ q_i\ bv$ 和 $u\ qj\ acv$ 是两个格局 (uv字符串, ab单个字符  $q_iq_j$ 状态)  
 我们说如果有  $\delta(q_i, b) = (q_j, C, L)$  则  $ua\ q_i\ bv$ 产生  $u\ qj\ acv$   
 如果有  $\delta(q_i, b) = (q_j, C, R)$ , 则  $ua\ q_i\ bv$ 产生  $uac\ qj\ v$   
 Note: 图灵机可以通过单个single的步骤从C1到C2

---

格局转移: 左手边结束的特殊例子left-hand end  
 我们会规定一种特殊情况, 纸带的左边是有尽头的, 右边是没有尽头的  
 对于左手边结束的特殊例子  
 如果  $\delta(q_i, b) = (q_j, C, L)$ , 那么  $q_1\ bv$ 产生  $qj\ cv$   
 因为我们阻止机器往左边移动, 因为左手边是纸带的尽头  
 但是如果纸带往右边移动, 那么还是跟常規格局转移是一样的

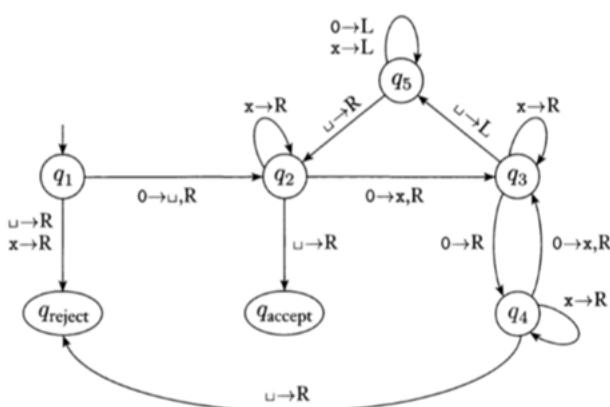
格局转移: 右手边结束的特殊例子right-hand end  
 对于右手边结束的特殊例子里  
 $ua\ q_i$ 等价于  $ua\ q_i\ _$  因为我们假设空格也是配置中指代的一部分, 因此我们可以把这种例子考虑成通常的情况, 头部不再在右手端

---

开始格局start configuration  
 一个图灵机输入一个状态w的开始格局是输入格局 $q_0w$ , 它代表着这个机器处于 $q_0$ 的开始状态下, 并且它的读写头在最左边的位置

接受格局和拒绝格局accepting configuration and reject configuration  
 在一个图灵机的接受格局中, 格局的状态是 $q_{accept}$   
 在一个图灵机的拒绝格局中, 格局的状态是 $q_{reject}$   
 接受格局和拒绝格局是一种停止的格局, 并且不会再进去其他的格局中

---



图灵机M2的状态程序  
 在这个状态程序下, 标签 $0 \rightarrow$ 空格, R出现在从 $q_1$ 到 $q_2$ 的转移。这个标签表明, 当在状态 $q_1$ 且读写头读到的是0的情况下, 机器会去状态2, 写下空格, 并且把读写头向右移动。换句话说,  $\delta(q_1, 0) = (q_2, \_ R)$

为了清楚的表明我们用简写 $0 \rightarrow R$ 来表明从状态 $q_3$ 到 $q_4$ , 这意味着机器当在状态 $q_3$ 读到0的时候, 会向右移动, 但是不会修改纸带, 所以说 $\delta(q_3, 0) = (q_4, 0, R)$

---

这个机器从在最左边的0的左边书写一个空格符号以便它在状态4可以找到最左手边的纸带的尽头。然而我们会经常使用一种更好的符号#作为左手边尽头的定界符号, 我们在这里用空格来保证这个纸带的字母表, 然后运行这个小小状态程序, 例子给了另一种找到最左手边的纸带的尽头的方法选择我们输入了0000给了这个机器一个简单的运行。最开始的格局是 $q_1$ 0000。这个机器的其他输入序列如下, 每一列从左至右读下去

$q_1 0000$	$\sqcup q_5 x 0 x \sqcup$	$\sqcup x q_5 x x \sqcup$
$\sqcup q_2 0000$	$q_5 \sqcup x 0 x \sqcup$	$\sqcup q_5 x x x \sqcup$
$\sqcup x q_3 000$	$\sqcup q_2 x 0 x \sqcup$	$q_5 \sqcup x x x \sqcup$
$\sqcup x 0 q_4 0$	$\sqcup x q_2 0 x \sqcup$	$\sqcup q_2 x x x \sqcup$
$\sqcup x 0 x q_3 \sqcup$	$\sqcup x x q_3 x \sqcup$	$\sqcup x q_2 x x \sqcup$
$\sqcup x 0 q_5 x \sqcup$	$\sqcup x x x q_3 \sqcup$	$\sqcup x x q_2 x \sqcup$
$\sqcup x q_5 0 x \sqcup$	$\sqcup x x x q_5 x \sqcup$	$\sqcup x x x q_2 \sqcup$
		$\sqcup x x x \sqcup q_{\text{accept}}$

接下来是一个关于

- $Q = \{q_1, \dots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$ ,
- $\Sigma = \{0, 1, \#\}$ , and  $\Gamma = \{0, 1, \#, x, \sqcup\}$ .

的正式描述, 在上面讲到了图灵机的一种非正式描述来定义一种语言  
 $B = \{w\#w \mid w \in \{0, 1\}^*\}$ .

- $Q = \{q_1, \dots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$ ,
- $\Sigma = \{0, 1, \#\}$ , and  $\Gamma = \{0, 1, \#, x, \sqcup\}$ .

我们描述状态图表来描述 $\delta$

开始, 接受和拒绝状态是分别用 $q_1$ ,  $q_{\text{accept}}$ ,  $q_{\text{reject}}$ 来表示

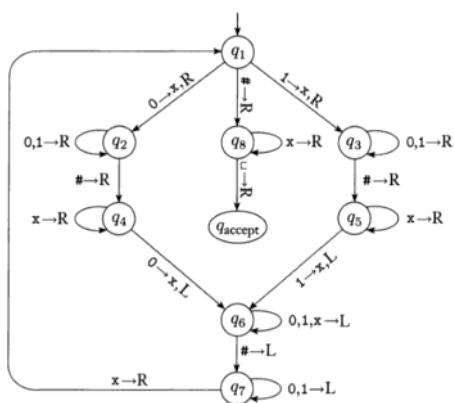


FIGURE 3.10  
State diagram for Turing machine  $M_1$

在图3.10中, 它描述了图灵机 $M_1$ 的状态图表, 你可以看到标签 $0, 1 \rightarrow R$ 在从 $q_3$ 到它自己的转换中。这个标签表明当机器在状态 $q_3$ 中读到0或者1的时候, 它会呆在状态 $q_3$ 然后向右边移动, 同时它改变在纸带上的符号

阶段1是一个从q1到q6的实施implement，然后阶段2是继续这个状态，为了简化图片，我们不会展示拒绝状态或者转换进入了拒绝状态。当状态缺少特定符号的传出转换时，这些转换将隐式地发生。因此，因为在状态q5不会有箭头和#出现，如果一个#出现在在状态q5的读写头下，它就直接去了状态qreject。为了完整，我们说读写头在读到拒绝状态后都会向右移动

---

Computation计算：什么是计算？

它是一个有限finite的计算步骤

它是一个被先前规定好的程序所命令的步骤

Note：操作的观点

为什么学习它？

为了调查计算的通法

为了找到一些原则来设计和实施，让计算机自动执行计算工序

为了找到有效的和高效的计算方法

---

不可计算工序 incomputable functions

没有有效的方法（有限的步骤）来计算这类工序

Note：图灵机无法停止

可计算工序：computable functions

存在有效的方法（有限的步骤）来计算这类工序

真实可计算工序

这是一个“真实”有效的方法（任务有意义或者有限的步骤）来计算这类工序

“really”或者“task-meaningfully”是什么意思？

---

计算时间的比较

n	Time (n)	$\log_2 n$	Time ( $\log_2 n$ )	$n^2$	Time ( $n^2$ )
2	0.002 sec	1	0.001 sec	4	0.004 sec
16	0.016 sec	4	0.004 sec	256	0.256 sec
64	0.064 sec	6	0.006 sec	4096	4.1 sec
256	0.256 sec	8	0.008 sec	65536	1 min 5 sec
1024	1 sec	10	0.010 sec	1048576	17 min 28 sec
4096	4.1 sec	12	0.012 sec	16777216	4 hours 40 min
16384	16.4 sec	14	0.014 sec	268435456	3 days 2 hours 34 min
65536	1 min 5 sec	16	0.016 sec	4294967296	49 days 17 hours
262144	4 min 22 sec	18	0.018 sec	68719476736	2 years 65 days
1000000	16 min 40 sec	20	0.020 sec	1000000000000	31 years 259 days
6	0.006 sec	3	0.003 sec	36	0.03 sec
30	0.03 sec	5	0.005 sec	900	0.9 sec
1000000000	11 days 14 hours	30	0.03 sec	$10^{18}$	33,000,000 years



我们来看一个有关“可计算”问题所需演算步骤的数量化比较。我们考虑A,B,C三个“可计算”问题，假设为了算出最终结果A问题需要n步演算、B问题需要 $n^2$ （n的2次方）步演算、而C问题需要 $\log_2 n$ （以2为底n的对数）步演算，并且还假设这些问题在某种现代通用计算机上被实施计算时每个演算步骤花费的时间为0.001秒，那么关于这三个问题之演算步骤及花费时间的一个数量化对比如下：

A	$n=2$	$n=4$	$n=6$	$n=30$	$n=109$
B	$n^2=4$	$n^2=16$	$n^2=36$	$n^2=900$	$n^2=1018$
C	$\log_2 n=1$	$\log_2 n=2$	$\log_2 n=3$	$\log_2 n=5$	$\log_2 n=30$
A	0.002秒	0.004秒	0.006秒	0.030秒	约11天14小时
B	0.004秒	0.016秒	0.036秒	0.9秒	约33,000,000年
C	0.001秒	0.002秒	0.003秒	0.005秒	0.030秒

由上面这个表我们可以看出，以30(36)步和0.03(0.36)秒为基准，三个问题之间的差距是巨大的。

表中这些数是估计的数量级，并且是从各种资料中精选得到的，天体物理学中许多大数见 Freeman Dyson 的文章 Time Without End: Physics and Biology in an Open Universe, Reviews of Modern Physics, v.52, n.3, 1979.7, 447~460。汽车事故的死亡人数是根据 1993 年交通部统计数据每百万人中有 163 起死亡事故和人均寿命为 69.7 年计算出来的。

表 1-1 大数

物理模拟量	大数
每天被闪电杀死的可能性	90 亿 ( $2^{39}$ ) 分之一
赢得国家发行彩票头等奖的可能性	4 百万 ( $2^{22}$ ) 分之一
赢得国家发行彩票头等奖并且在同一天被闪电杀死的可能性	$1/2^{38}$
每年淹死的可能性	59 000 ( $2^{18}$ ) 分之一
1993 年在美国交通事故中死亡的可能性	6 100 ( $2^{13}$ ) 分之一
一生在美国死于交通事故的可能性	$88 \cdot (2^7)$ 分之一
到下一个冰川年代的时间	14 000 ( $2^{11}$ ) 年
到太阳变成新星的时间	$10^6 \cdot (2^{26})$ 年
行星的年龄	$10^9 \cdot (2^{26})$ 年
宇宙的年龄	$10^{15} \cdot (2^{24})$ 年
行星中的原子数	$10^{51} \cdot (2^{36})$
太阳中的原子数	$10^{57} \cdot (2^{100})$
银河系中的原子数	$10^{87} \cdot (2^{223})$
宇宙中的原子数（黑粒子除外）	$10^{117} \cdot (2^{265})$
宇宙的体积	$10^{144} \cdot (2^{260}) \text{ cm}^3$
如果宇宙是封闭的：	
宇宙的生命期	$10^{11} \cdot (2^{27})$ 年
	$10^{18} \cdot (2^{61})$ 秒
如果宇宙是开放的：	
到小行星冷却下来的时间	$10^{14} \cdot (2^{47})$ 年
到行星脱离星系的时间	$10^{15} \cdot (2^{50})$ 年
到行星脱离银河系的时间	$10^{19} \cdot (2^{64})$ 年
到由引力线引起的轨道倾变的时间	$10^{20} \cdot (2^{67})$ 年
到由散播过程引起黑洞湮没的时间	$10^{64} \cdot (2^{213})$ 年
到所有物质在 0°时都为液体的时间	$10^{65} \cdot (2^{214})$ 年
到所有物质都转变成铁的时间	$10^{29}$ 年
到所有物质都收缩为黑洞的时间	$10^{35}$ 年

# Computability

2019年11月30日 12:42

## 可计算性

---

### 基础问题

什么样的计算工序是可计算的?

为什么有些计算工序不可计算?

计算机能做什么, 不能做什么?

是不是有些问题计算机不能计算处理?

可计算性computability: 它是什么?

它是计算机科学的理论基础

它是可计算性中将多种问题区分为可以计算和不可以计算的理论

可计算性的理论是计算复杂性学说的基础, 它介绍了很多种在后面会谈到的观点

---

可计算性: 为什么要学习它?

为了知道计算机能做和不能做什么的准则

我必须去学习计算复杂性的理论, 从而知道什么样的东西计算机可以真正做到 (有效的, 高效的), 但是可计算性理论是计算复杂性理论

computational complexity 的基础

---

### 有穷自动机模型

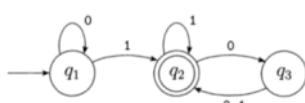


FIGURE 1.4  
A finite automaton called  $M_1$  that has three states

Figure 1.4 is called the **state diagram** of  $M_1$ . It has three **states**, labeled  $q_1$ ,  $q_2$ , and  $q_3$ . The **start state**,  $q_1$ , is indicated by the arrow pointing at it from nowhere. The **accept state**,  $q_2$ , is the one with a double circle. The arrows going from one state to another are called **transitions**.

For example, when we feed the input string 1101 to the machine  $M_1$  in Figure 1.4, the processing proceeds as follows.

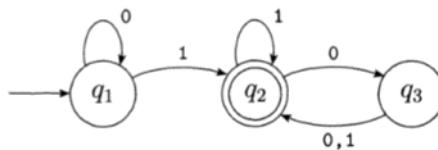
1. Start in state  $q_1$ .
  2. Read 1, follow transition from  $q_1$  to  $q_2$ .
  3. Read 1, follow transition from  $q_2$  to  $q_2$ .
  4. Read 0, follow transition from  $q_2$  to  $q_3$ .
  5. Read 1, follow transition from  $q_3$  to  $q_2$ .
  6. Accept because  $M_1$  is in an accept state  $q_2$  at the end of the input.
- 

一个有限的自动机器automata是一个五元的

1.  $Q$  is a finite set called the **states**,
2.  $\Sigma$  is a finite set called the **alphabet**,
3.  $\delta: Q \times \Sigma \rightarrow Q$  is the **transition function**,<sup>1</sup>
4.  $q_0 \in Q$  is the **start state**, and
5.  $F \subseteq Q$  is the **set of accept states**.<sup>2</sup>

1.转换函数 $\delta$ 是来自于Q的Cartesian product, 然后 $\Sigma (Q \times \Sigma) \rightarrow Q$

2.接受状态也叫final states



**FIGURE 1.6**

The finite automaton  $M_1$

We can describe  $M_1$  formally by writing  $M_1 = (Q, \Sigma, \delta, q_1, F)$ , where

1.  $Q = \{q_1, q_2, q_3\}$ ,
2.  $\Sigma = \{0,1\}$ ,
3.  $\delta$  is described as

	0	1
q1	q1	q2
q2	q3	q2
q3	q2	q2

4.  $q_1$  is the start state, and
5.  $F = \{q_2\}$ .

### 机器的语言language of machine

如果A是机器M可以接受的所有字符串, 那么我们说A is the language of machine M, 然后写下 $L(M) = A$ , 然后也可以说M recognizes A 或者M accepts A

### 关于使用有限的自动机器计算定义

Let  $M = (Q, \Sigma, \delta, q_0, F)$  be a finite automaton and let  $w = w_1 w_2 \dots w_n$  be a string where each  $w_i$  is a member of the alphabet  $\Sigma$ . Then  $M$  accepts  $w$  if a sequence of states  $r_0, r_1, \dots, r_n$  in  $Q$  exists with three conditions:

1.  $r_0 = q_0$ ,
2.  $\delta(r_i, w_{i+1}) = r_{i+1}$ , for  $i = 0, \dots, n - 1$ , and
3.  $r_n \in F$ .

情况1说明了机器从开始状态开始

情况2说明了机器根据转换公式从一个状态到另一个状态

情况3说明了机器如果它停在了接受状态, 那么机器接受它的输入, 这个时候我们说M recognizes language A if  $A = \{w \mid M \text{ accepts } w\}$

### 图灵机的语言

如果一格局的序列C1, C2, ...Ck存在, 那么一个图灵机M accepts 输入w, 满足

1.C1是机器M在输入w后的第一个转换

2.每一个 $C_i$ 都会进入 $C_{i+1}$

3. $C_k$ 是一个接受格局 accept configuration

M接受的一系列的字符串叫做the language of M, 或者the language is recognized by M, 写作 $L(M)$

定义一个图灵机，和若干个语言关联

---

三种图灵机的可能输出

当我们从输入端开始启动一个图灵机后，有三种可能的输出：机可能 accept, reject或者infinite loop（永远不停止，没有必要的重复着相同的行动）

Note

通常我们很难去发现一个机器是在永远的循环还是在读写了很长的时间但是会在某一刻停下来的情况

---

图灵可识别语言 Turing-recognizable language

如果一些图灵机可以识别一种语言，那么这种语言叫做Turing-recognizable (recursively enumerable)

定义了一个语言，去和若干个图灵机关联

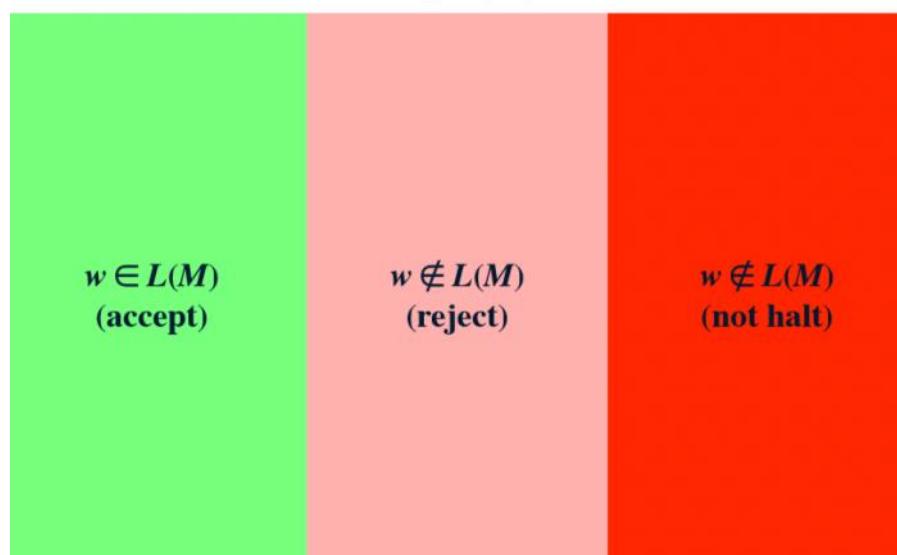
Note

让A作为图灵机M的一种语言，即 $L(M) = A$ ，因此，A是图灵可识别语言

对于任何属于A的元素w，图灵机M必须停止并且接受w，但是对于不属于A的元素w，图灵机M可能停止，或者拒绝w，或者不停止

---

$$w \in \Sigma^*, L(M) \subset \Sigma^*$$



$\Sigma^*$ 字母表

W字符串

$L(M)$  图灵机M接受的字符串

后面两个的划分间接表达了图灵机可以解决什么和不可以解决什么

---

$M$  在输入  $w$  上的起始格局是格局  $q_0 w$ , 表示机器处于起始状态  $q_0$ , 并且读写头处于带子的最左端位置, 在接受格局里, 状态是  $q_{\text{accept}}$ , 在拒绝格局里, 状态是  $q_{\text{reject}}$ 。接受和拒绝状态都是停机状态, 它们都不再产生新的格局。因为机器只在接收或拒绝状态下才停机, 因此可以等价地将转移函数简化为  $\delta: Q' \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ , 其中  $Q'$  是取消状态  $q_{\text{accept}}$  与  $q_{\text{reject}}$  的  $Q$ 。图灵机  $M$  接受输入  $w$ , 如果存在格局的序列  $C_1, C_2, \dots, C_k$  使得:

1)  $C_1$  是  $M$  在输入  $w$  上的起始格局;

2) 每一个  $C_i$  产生  $C_{i+1}$ ;

3)  $C_k$  是接受格局。

$M$  接受的字符串的集合称为  $M$  的语言, 或被  $M$  识别的语言, 记为  $L(M)$ 。

定义 3.2 如果一个语言能被某一图灵机识别, 则称该语言是图灵可识别的 (Turing-recognizable)<sup>⊕</sup>。

---

## 图灵机判定器deciders

当一个图灵机对它的输入一定可以停下来, (不可能永不停止) 的时候, 它叫做判定器decider

一个能识别某些语言的图灵判定器也可以叫做可判定某些语言

图灵可判定语言Turing-decidable language

一个叫做图灵可判定语言的语言或者可以被某些图灵机简单的判定的

Notes

每一个可判定decidable语言一定是图灵可判定语言, 但是一些图灵可判定语言并不是可判定decidable语言, 因为可能没有判定器

一个语言的“图灵可判定性”是这个语言的固有属性intrinsic property

---

## 图灵可判定语言Turing-decidable language

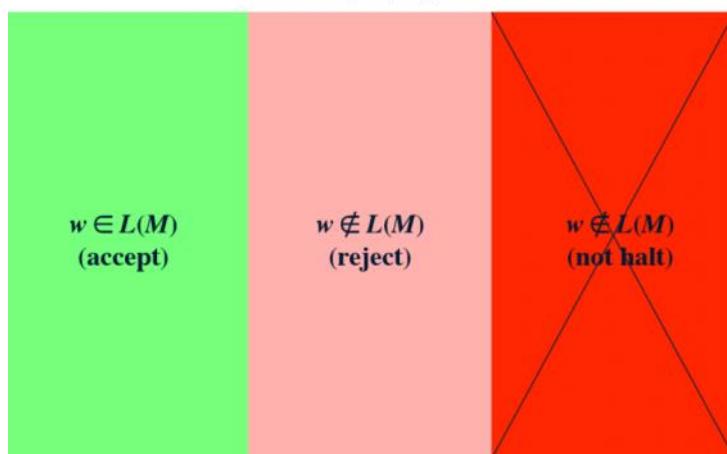
如果一个语言图灵机可以判定, 那么这个语言被叫做图灵可判定语言

Notes

一个语言的图灵可判定性是该语言的内在性质

对于一个给定的语言  $L$ , 我们考虑是否有一个图灵机能够判定  $L$ , 如果有, 我们说  $L$  图灵可判定的

$$w \in \Sigma^*, L(M) \subset \Sigma^*$$



---

## 图灵可识别语言

一种语言的图灵可识别性是这种语言的内在性质

对于一个给定的语言  $L$ , 我们考虑是否存在一个图灵机可以接受accept

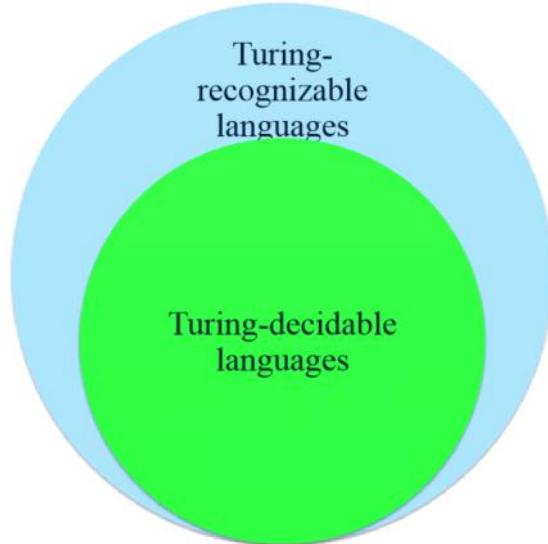
$L$ , 如果可以, 我们说 $L$ 是图灵可识别的

图灵可判定语言

一种语言的图灵可判定性是这种语言的内在性质

对于一个给定的语言 $L$ , 我们考虑是否存在一个图灵机可以判定decide

$L$ , 如果可以, 我们说 $L$ 是图灵可判定的



在输入上运行一个 TM 时, 可能出现三种结果: 接受、拒绝或循环。这里循环仅仅指机器不停机, 而不一定是这个词的字面所指的那样, 永远以同样方式重复同样的步骤。循环动作可能是简单的, 也可能是复杂的, 但都不会导致停机状态。

对于一个输入, 图灵机有两种方式不接受它: 一种是进入拒绝状态而拒绝它, 另一种是进入循环。有时候, 很难区分机器是进入了循环还是需要耗费长时间的运行, 因此, 我们更喜欢对所有输入都停机的图灵机, 它们永不循环, 称这种机器为判定器, 因为它们总能决定是接受还是拒绝, 也称识别某个语言的判定器判定该语言。

**定义 3.3** 如果一个语言能被某一图灵机判定, 则称它是图灵可判定的(Turing decidable), 简称可判定的(decidable)<sup>④</sup>。

### The notion of algorithm 算法的概念

直到20世纪的时候, 算法这个概念都没有被准确定义了下来

黑尔波特的23个问题

根据它可以被有限的操作决定下来, 这个问题设计了算法algorithm的概念, 它想知道是否一个多项式可以内的整数根

这个问题的必须由一个清楚的“算法”定义来解决

算法概念的必要性

证明一个算法不存在需要一个对于算法清楚的定义

### Church' s Lambda演算 ( $\lambda$ -计算)

Lambda演算是数学逻辑上为了表达计算是基于运用变量的捆绑和减少为基础的抽象和应用的一个正式的体系

它是一个等价于图灵机的计算通用模型

### The Church-Turing thesis

直观的算法的概念 等价于 图灵模型的算法概念

这个理论陈述了: 一个问题是在算法上的可解决 (或者存在算法) 是等价于它是图灵可判定性的

四个计算模型是近似等价的，一个可计算，另外一个一定也可以被计算出来，所以所有计算模型是等价的，这也就提出了算法的概念，图灵可计算性也就等于算法等于 $\lambda$ -计算

---

## 黑尔波特的十个问题

让 $D=\{p \mid p\text{是一个有着整数根的多项式}\}$

这个问题问出了本质：是否集合 $D$ 是可判定的（答案是不可以）

$D$ 是图灵可判定的

让 $D_1=\{p \mid p\text{是一个多项式除以}x, \text{有一个积分根}\}$

Eg  $p$ 是一个只有一个变量的多项式

Here is a TM  $M_1$  that recognizes  $D_1$ :

$M_1 = \text{"On input } \langle p \rangle: \text{ where } p \text{ is a polynomial over the variable } x.$

1. Evaluate  $p$  with  $x$  set successively to the values  $0, 1, -1, 2, -2, 3, -3, \dots$ . If at any point the polynomial evaluates to 0, accept."

If  $p$  has an integral root,  $M_1$  eventually will find it and accept. If  $p$  does not have an integral root,  $M_1$  will run forever. For the multivariable case, we can present a similar TM  $M$  that recognizes  $D$ . Here,  $M$  goes through all possible settings of its variables to integral values.

---

## 一个基础的问题

当我们描述算法时，什么是正确的细节的分层？

三个分层

Formal description正式的描述（最底层）：用图灵机的状态来拼写，转换函数等等，这是最底层最细节的描述层次

Implementation description实施的描述：运用英语语言来描述图灵机为什么移动读写头，储存数据于纸带上，在这个层次上我们不会给关于状态和转换函数的细节

High-level description高级描述：用英语语言来描述一种算法，无视掉执行细节，在这个层次下，我们不需要说图灵机是如何安排它的纸带和读写头的

---

## encoding of objects编码的对象

这个标准通常是将一串字符串输入到图灵机中，因此，如果我们想要提供一个不是已输入的字符串对象的对象，我们必须首先说明这个对象也是一个字符串

字符串可以代表多项式，图片，语法，自动装置，和任何对象的组合。

一个图灵机可以被设计成编码这些对象

图灵机编码的格式和概念

编码一个对象 $O$ ：那么我们在字符串上写 $\langle O \rangle$

如果我们含有多个对象 $O_1, O_2 \dots O_k$ ，我们用单个的字符串 $\langle O_1, O_2, \dots O_k \rangle$ 表示

---

## 格式

我们用引号内文本的缩进段 indented segment of text within quotes 来描述图灵机的算法

我们把算法分成几个阶段，每一个阶段都包含着图灵机计算的很多单个步骤

我们用更多的信息来表明程序块的结构

---

表述图灵机的算法

算法的第一行是写下要输入进图灵机什么东西

如果想输入一个简单的w，那么输入的东西要变成一个字符串

如果输入的东西是编码的对象  $\langle A \rangle$ ，那么图灵机会首先自动检测这个输入的东西是否能像预料的一样表明正常的编码一个对象，如果不行就拒绝它

---

假设A是由表示无向图的所有连接的字符串组成的语言。回想一下，如果每个节点都可以通过沿着图的边移动而从其他节点到达，那么图就是连通的。我们写作

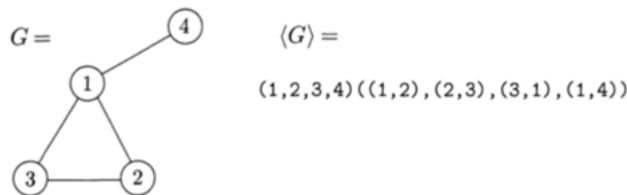
$A = \{(G) | G\text{是一个连通无向图}\}$ 。

下面是决定A的图灵机的高级描述。

$M = \text{输入}(G)$ 、一个图G的编码:

1. 选择G的第一个节点并标记。
  2. 重复以下步骤，直到没有新节点被标记：
  3. 对于G中的每个节点，如果它被一条边连接到一个已经标记的节点上，则标记它。
  4. 扫描G的所有节点，确定是否都有标记。如果是，接受；否则，拒绝
- 

作为额外的实践，让我们检查一些图灵机器m的实现级别的细节。通常我们不会在将来给出这个级别的细节，您也不需要这样做，除非在练习中特别要求这样做。首先，我们必须理解  $\langle G \rangle$  如何将图G编码为字符串。考虑这样一种编码，即G的节点列表后面跟着G的边列表。每个节点都是一个十进制数，每条边都是十进制数对，表示该边的两个端点处的节点。下图描述了此图及其编码。



**FIGURE 3.24**  
A graph  $G$  and its encoding  $\langle G \rangle$

当M接收到输入(G)时，它首先检查输入是否是某个图的正确编码。为了做到这一点，M扫描磁带以确保有两个列表，并且它们的形式是正确的。第一个列表应该是不同的十进制数的列表，第二个列表应该是十进

制数对的列表。然后M检查一些东西。首先，节点列表不应该包含重复，其次，出现在边缘列表上的每个节点也应该出现在节点列表上。对于第一个，我们可以使用例3.12中为TM M4提供的检查元素差异的过程。类似的方法也适用于第二次检查。如果输入通过了这些检查，那么它就是某个图g的编码。这个验证完成了输入检查，M进入到阶段1。

对于阶段1，M用最左边数字上的点标记第一个节点。

对于第二阶段，M扫描节点列表以找到一个没有点的节点n1，并通过不同的标记来标记它~例如，通过在第一个符号下划线来标记它。然后M再次扫描列表，找到一个点节点，并给它加下划线。

现在M扫描边的列表。对于每条边，M测试带下划线的两个节点n1和n2是否出现在该边中。如果是，则M点n1，删除下划线，并从阶段2开始。如果不是，M检查列表上的下一条边。如果没有更多的边，{n1, n2}不是g的边，那么M将n2上的下划线移动到下一个点节点，现在将这个节点称为n2。它重复本段中的步骤，与前面一样，检查新对{n1, n2}是否为边。如果没有更多的点节点，则n1不附加到任何点节点。然后M设置下划线，使n1是下一个无点节点，n2是第一个有点节点，并重复本段中的步骤。如果没有更多的未点状节点，M就无法找到任何要点状的新节点，因此它将转到阶段4。

---

### 不可判定语言A TM undecidable language

这个关于决定是否一个图灵机接受一个给定的输入字符串的问题是不可判定的

A TM是不可判定的，没有一个图灵机可以作为一个判定器来判定A TM

A TM是图灵可识别的

下面的图灵机U可以识别A TM

U=输入<M, w>，M是图灵机而w是字符串

1.模仿M输入w

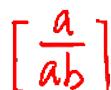
2.如果M可以进入接受状态，那么接受，如果M进入拒绝状态，那么拒绝

Note

如果M循环w，那么U将会循环<M, w>，这就是为什么它不能判定A TM

---

我们可以用益智游戏来说明这个问题，我们从一个多米诺骨牌的收集开始，每一个包含了两个字符串，每个都一边上，一个单独的多米诺骨牌看起来像



然后一系列多米诺骨牌看起来像

$$\left\{ \left[ \frac{b}{ca} \right], \left[ \frac{a}{ab} \right], \left[ \frac{ca}{a} \right], \left[ \frac{abc}{c} \right] \right\}$$

这个任务是为了列出一系列多米诺骨牌，这样我们在上面读到的字符串和在下面的字符串是相同的，这个清单叫做matcb

例如，下面列出的满足这个益智游戏

$$\left[ \frac{a}{ab} \right] \left[ \frac{b}{ca} \right] \left[ \frac{ca}{a} \right] \left[ \frac{a}{ab} \right] \left[ \frac{abc}{c} \right]$$

读完上面的字符串我们得到了abcaaabc，它跟我们在下面读到的是一样的，我们也可以用变换多米诺的排列来描述这个匹配

波斯特对应问题post correspondence problem (PCP)

为了决定是否一系列多米诺存在一个匹配

一个PCP的例子是像一个多米诺的收集：

$$P = \left\{ \left[ \frac{t_1}{b_1} \right], \left[ \frac{t_2}{b_2} \right], \dots, \left[ \frac{t_k}{b_k} \right] \right\}$$

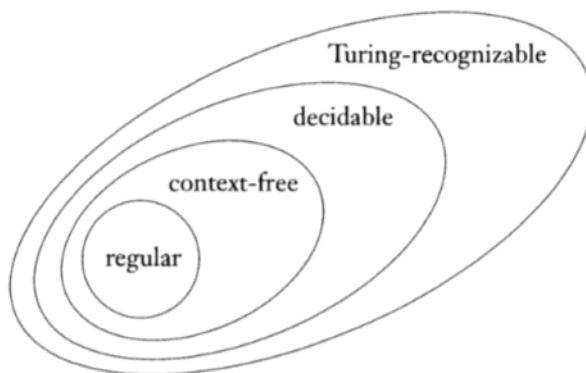
然后匹配是一个序列i1, i2...il, 满足ti1ti2...til=bi1bi2...bil

这个问题是来决定是否P存在一个匹配

PCP的不可判定性

PCP={<P>|P是一个拥有匹配的PCP例子}

PCP语言不是图灵可判定的，没有一个图灵机可以作为判定器判定PCP



**FIGURE 4.10**  
The relationship among classes of languages

可计算的countable

如果C是有限的，或者C和自然数是存在双射映射的，那么这个集合C被称为可计算的（真数的集合是不可计算的）

鸽舍原理The pigeonhole principle

如果n只鸽舍被n+1只鸽子抢夺，那么至少一个鸽舍有两只鸽子占据

图灵不可识别语言

因为真数的集合是不可计算的，有不可计算的许多语言  
有很多不可计算的语言，但是有可计算的数目的图灵机，一定存在图灵  
不可识别语言

---

### 对于语言的补集The complement of a language

对于一个在 $\Sigma$ 内的语言 $L$ ,  $L$ 的补集是在 $\Sigma$ 内的所有不是 $L$ 的字符串

写作

$$\Sigma^* = L \cup L^c \quad L \cap L^c = \emptyset$$

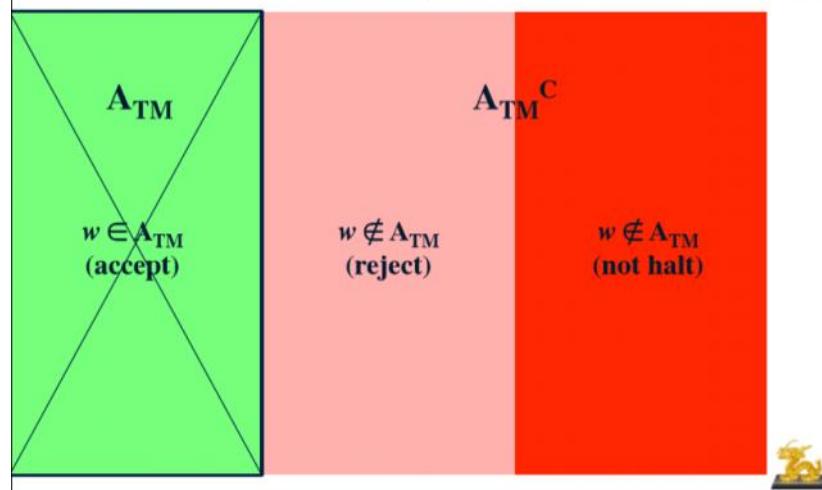
### 图灵不可识别语言

如果 $L$ 和 $L^c$ 同时都是图灵可识别的，那么一个语言 $L$ 是可识别的

Note: 有两种图灵机 $M$ 和 $M^c$ 满足 $M$ 可以识别 $L$ ,  $M^c$ 可以识别 $L^c$

A TM的补集不是图灵可识别的，因为A TM是不可判定并且不可识别的

There is a TM  $U$  such that  $L(U)(A_{\text{TM}}) \subset \Sigma^*$ , for any  $w \in \Sigma^*$ ,  $w$  may be:



# Computational complexity

2019年12月4日 20:08

## 计算复杂性

---

### 基础问题

- 1.什么是实际可以算的计算工序
- 2.计算机到底能做什么
- 3.为什么有些问题难以计算而有些比较简单？

### 计算复杂性：它是什么？

它提供了用于可计算的问题的复杂性的定量分类的方法以及定量测量的方法

它是CS中理论上的最基础

是计算复杂性理论才根据他们的计算难度，把问题分成了很多明确的定义

---

### 计算复杂性：为什么要学习它

为了知道计算机到底可以做什么

为了评判定量分类和尺度的定量测量

为了找到一些高效的算法来解决现实世界中的可计算问题

为了找到计算上有难度的问题，来设计密码系统

---

### 计算上解决的vs实际上可解决的

即使原则上一个问题是可以判定且可以计算上解决的，如果它需要大量的时间和空间，它也可能在实际上无法解决

### 计算复杂性理论

一个理论上关于时间，记忆（空间）或者其他为了解决计算问题所需的资源的调查

### 基础问题

哪些资源需要被衡量

这些资源是怎样被衡量的

---

### 时间复杂性理论time complexity

它介绍了一种通过测量时间来解决问题

它根据解决问题所需要的时间来划分问题

### 空间复杂性理论space complexity

它介绍了一种通过测量记忆/空间来解决问题

它根据问题占有的记忆/空间来划分问题

---

衡量一个算法通常的时间复杂性

我们想要去衡量一个算法通常的时间复杂性而不是对于一个特定的执行  
在一个特定的电脑上所需的时间复杂性

Note: 真正一个算法的特殊执行的跑程序时间，取决于跑算法的特殊的  
电脑

跑程序时间被跑步骤所取代

在某些电脑，我们计算一个算法的跑的步骤而不是跑的时间

Note: 一个算法的跑步骤是这个算法的固有特性而不会取决于电脑来跑  
这个算法

---

关于输入的函数来看时间复杂性

一个算法跑步骤需要的特定的输入可能会依据一些量

举个简单的例子，我们纯粹通过一个函数的字符串的长度来代表输入，  
并且考虑任何其他的量，来计算/衡量一个算法的跑步骤

输入的长度来看算法的复杂性（最主要的因素）

在上面考虑内隐含的本质上的假设

字符串的长度代表了输入中最重要的因素，它影响计算问题的复杂性

输入越长，计算的问题越复杂

---

重点定义

定义 7.1 令  $M$  是一个在所有输入上都停机的确定型图灵机。 $M$  的运行时间或者时间复杂  
度，是一个函数  $f: \mathbb{N} \rightarrow \mathbb{N}$ ，其中  $\mathbb{N}$  是非负整数集合， $f(n)$  是  $M$  在所有长度为  $n$  的输入上运行  
时所经过的最大步数。若  $f(n)$  是  $M$  的运行时间，则称  $M$  在时间  $f(n)$  内运行， $M$  是  $f(n)$  时间图  
灵机。通常使用  $n$  表示输入的长度。

---

最坏的分析例子

对于一个特定长度的输入，考虑跑得最久的时间

最坏例子跑的程序给了我们对于任何输入所需时间的上限

知道它能够给我们一个保证：这种算法不会一直跑下去

平均的分析例子

对于一个特定长度的输入，考虑跑的时间的平均值

平均的分析范围是有局限性的，因为一个特定问题的平均输入可能看起  
来不太清楚

---

渐进分析Asymptotic analysis

因为一个算法确切的跑时间/步骤是一个复杂的表达，我们经常用一个方  
便的量来估计，叫做

渐进符号：Big-O 符号 必考

我们在看一个算法的跑时间/步骤的时候，只考虑最高次项，且不管最高  
项的系数和其他低次项，因为最高次项控制阶数，它的尺寸或者复杂性  
只由最高项的阶数决定

我们只考虑最高项的阶数，其他项和最高项的系数都不考虑

Ex: For  $f(n) = 6n^3 + 2n^2 + 20n + 45$ , 我们记作  $f(n) = O(n^3)$

---

**定义 7.2** 设  $f$  和  $g$  是两个函数  $f, g: \mathcal{N} \rightarrow \mathbb{R}^+$ 。称  $f(n) = O(g(n))$ , 若存在正整数  $c$  和  $n_0$ , 使得对所有  $n \geq n_0$  有  
$$f(n) \leq cg(n)$$
当  $f(n) = O(g(n))$  时, 称  $g(n)$  是  $f(n)$  的上界(upper bound), 或更准确地说,  $g(n)$  是  $f(n)$  的渐近上界, 以强调没有考虑常数因子。

$f(n) = O(g(n))$  意味着如果我们忽视了常数因素的差异, 啊么  $f$  小于等于  $g$

你可能认为  $O$  代表了一个抑制常数

让  $f_1(n) = 5n^3 + 2n^2 + 22n + 6$ , 然后选择它的最高次项  $5n^3$  且不考虑它的系数 5, 我们有  $f_1(n) = O(n^3)$

让常数  $c=6$  然后  $n_0=10$  (或者 9, 8, 7, 6) 然后对于每一个  $n \geq 10$  或者  $9 \geq 7 \geq 6$  有  $5n^3 + 2n^2 + 22n + 6 \leq 6n^3$

此外  $f_1(n) = O(n^4)$ , 因为  $n^4$  比  $n^3$  大所以也是  $f_1$  的渐进上界

然而,  $f_1(n) = O(n^2)$ , 即使不考虑我们给的的  $c$  和  $n_0$ , 定义仍然无法满足这类情况

---

### 多项式边界 Polynomial bound

$f(n) = O(n^c)$  的边界对于任何真数  $c$  大于 0 叫做多项式边界

### 指数边界 Exponential bound

$f(n) = O(2^{n^c})$  的边界对于任何真数  $c$  大于 0 叫做指数边界

Note: 2 可以被其它数替换

上界可以开始比原函数小, 但是随着变量个数的增加, 最终上界函数一定大于原函数, 即当  $n$  趋近于  $+\infty$  的时候, 恒有上界函数大于原函数

---

**定义 7.5** 设  $f$  和  $g$  是两个函数  $f, g: \mathcal{N} \rightarrow \mathbb{R}^+$ 。如果  
$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$
则称  $f(n) = o(g(n))$ 。换言之,  $f(n) = o(g(n))$  意味着对于任何实数  $c > 0$ , 存在一个数  $n_0$ , 使得对所有  $n \geq n_0$ ,  $f(n) < cg(n)$ 。

### Small-o 符号

Small-o 符号是用来代表一个函数渐进上小于其它的函数

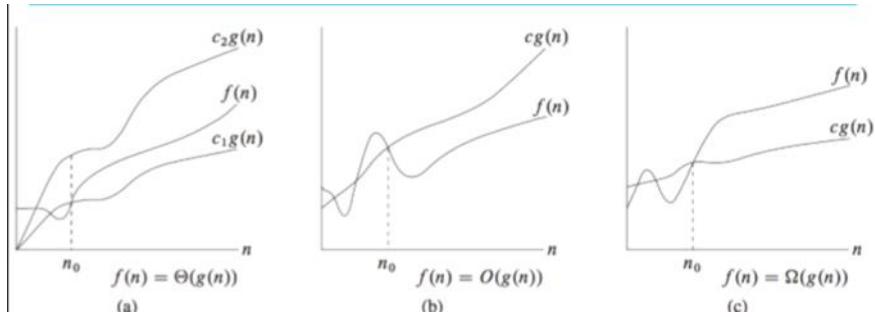
### Big-O 符号

Big-O 符号是用来代表一个函数渐进上小于等于其他函数

### Big-O 和 Small-o 的区别

### $\leq$ 和 $<$ 的区别

---



## 时间复杂性的阶级

让  $t: N \rightarrow R^+$  作为一个函数,  $R^+$  是一系列非负真数

我们可以定义时间复杂性阶级  $\text{TIME}(t(n))$ , 采集任何被  $O(t(n))$  时间/步骤

图灵机所决定的语言

这个图灵机是设定出来用 Big-O 定义时间复杂性的

### Notes

时间复杂性是一个语言/问题的阶级而不是关于图灵机的

可能有许多时间复杂性阶级, 因为会有很多函数

不同的时间/步骤图灵机来定义一种语言的时间复杂性可能会产生不同的阶级

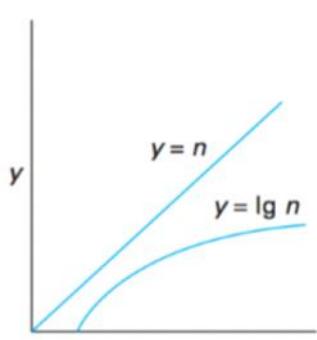
## 线性时间复杂性

$O(n)$  时间被称作线性时间 linear time

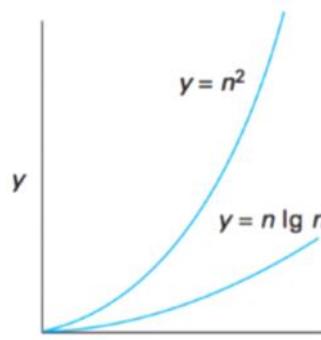
关于正规语言 regular language 的一个重要的事实

任何一个语言可以被一个单带确定性图灵机判定成  $O(n \log n)$  的语言是正规语言

### Graphs of the mathematical expressions $n$ , $\lg n$ , $n \lg n$ , and $n^2$



a.  $n$  versus  $\lg n$



b.  $n^2$  versus  $n \lg n$

我们来看一个有关“可计算”问题所需演算步骤的数量化比较。我们考虑A,B,C三个“可计算”问题，假设为了算出最终结果A问题需要n步演算、B问题需要 $n^2$ （n的2次方）步演算、而C问题需要 $\log_2 n$ （以2为底n的对数）步演算，并且还假设这些问题在某种现代通用计算机上被实施计算时每个演算步骤花费的时间为0.001秒，那么关于这三个问题之演算步骤及花费时间的一个数量化对比如下：

A	n=2	n=4	n=6	n=30	n=109
B	$n^2=4$	$n^2=16$	$n^2=36$	$n^2=900$	$n^2=1018$
C	$\log_2 n=1$	$\log_2 n=2$	$\log_2 n=3$	$\log_2 n=5$	$\log_2 n=30$
A	0.002秒	0.004秒	0.006秒	0.030秒	约11天14小时
B	0.004秒	0.016秒	0.036秒	0.9秒	约33,000,000年
C	0.001秒	0.002秒	0.003秒	0.005秒	0.030秒

由上面这个表我们可以看出，以30(36)步和0.03(0.36)秒为基准，三个问题之间的差距是巨大的。

## 比较重要的概念

### 一个关于可计算性和计算复杂性理论重要的区别

可计算性：所有的可行的计算模型都是等价的，每一个语言都是同一阶级的（一个能算，另一个也能算）

计算复杂性：计算模型的选择会影响时间复杂性

我们用什么模型来衡量计算复杂性

时间需求在不同的判定模型中不会有很大的区别

如果我们的分级系统对复杂性中小的区别不是很敏感，那么判定模型的选择不是很重要

## 科学家与工程师

为什么vs怎么做

理论vs应用

方法论vs工具

The following table summarizes some classes of commonly encountered time complexities. In the table,  $\text{poly}(x) = x^{O(1)}$ , i.e., polynomial in  $x$ .

Name	Complexity class	Running time ( $f(n)$ )	Examples of running times	Example algorithms
constant time	$O(1)$	10		Determining if an integer (represented in binary) is even or odd
inverse Ackermann time	$O(\alpha(n))$			Amortized time per operation using a disjoint set
iterated logarithmic time	$O(\log^* n)$			Distributed coloring of cycles
log-logarithmic	$O(\log \log n)$			Amortized time per operation using a bounded priority queue <sup>[2]</sup>
logarithmic time	$O(\log n)$	$\log n, \log(n^2)$		Binary search
polylogarithmic time	$\text{poly}(\log n)$	$(\log n)^2$		
fractional power	$O(n^c)$ where $0 < c < 1$	$n^{1/2}, n^{2/3}$		Searching in a kd-tree
linear time	$O(n)$	$n$		Finding the smallest or largest item in an unsorted array, Kadane's algorithm
$\gamma n \log \gamma n$ time	$O(n \log^* n)$			Seidel's polygon triangulation algorithm.
quasilinear time	$O(n \log n)$	$n \log n, \log n!$		Fastest possible comparison sort; Fast Fourier transform
quadratic time	$O(n^2)$	$n^2$		Bubble sort; Insertion sort; Direct convolution
cubic time	$O(n^3)$	$n^3$		Naive multiplication of two $n \times n$ matrices. Calculating partial correlation.
polynomial time	P	$2^{\text{poly}(n)} = \text{poly}(n)$	$n, n \log n, n^{10}$	Karmarkar's algorithm for linear programming; AKS primality test
quasi-polynomial time	QP	$2^{\text{poly}(\log n)}$	$n^{\log \log n}, n^{\log \log \log n}$	Best-known $O(\log^2 n)$ -approximation algorithm for the directed Steiner tree problem.
sub-exponential time (first definition)	SUBEXP	$O(2^{\epsilon n})$ for all $\epsilon > 0$	$O(2^{\log n \log \log n})$	Assuming complexity theoretic conjectures, BPP is contained in SUBEXP <sup>[3]</sup>
sub-exponential time (second definition)		$2^{\text{poly}(n)}$	$2^{n^{1/3}}$	Best-known algorithm for integer factorization and graph isomorphism
exponential time (with linear exponent)	E	$2^{\text{O}(n)}$	$1.1^n, 10^n$	Solving the traveling salesman problem using dynamic programming
exponential time	EXPTIME	$2^{\text{poly}(n)}$	$2^n, 2^{\sqrt{n}}$	Solving matrix chain multiplication via brute-force search
factorial time		$O(n!)$	$n!$	Solving the traveling salesman problem via brute-force search
double exponential time	2-EXPTIME	$2^{\text{poly}(n)}$	$2^{2^n}$	Deciding the truth of a given statement in Presburger arithmetic

P 是确定型单带图灵机在多项式时间内可判定的语言类。换言之，

$$P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$$

---

为什么阶级P本质上很重要

所有合理的确定性计算模型都多项式的等价，里面的任何一个可以模仿另一个，仅仅一个多项式增加了跑的时间

对于任何多项式等价于确定的单带图灵机的计算模型，P是不变的

P粗略的对应可以真实在电脑上可解问题的阶级

Notes

P是一个数学上稳定的阶级

P与实际应用上是相关的

---

检验者Verifier

语言A的检验者是一种算法V

$A = \{w \mid V \text{接受 } \langle w, c \rangle \text{ 对于某些字符串 } c\}$

我们仅仅从w的长度来衡量检验者的时间，所以一个多项式时间检验者在长度w内跑多项式时间

如果语言A有一个多项式时间检验者，那么它叫做多项式可检验的

Note:

一个检验者可以使用额外的信息，为了检验一个字符串w是A中的成员，用c来表示以上的定义。这种信息被叫做一个证明certificate/proof A中的成员

---

# Algorithm

2019年12月4日 20:09

## 算法

描述：本算法假定输入是两个正整数，目的是要计算这两个数的最大公约数。

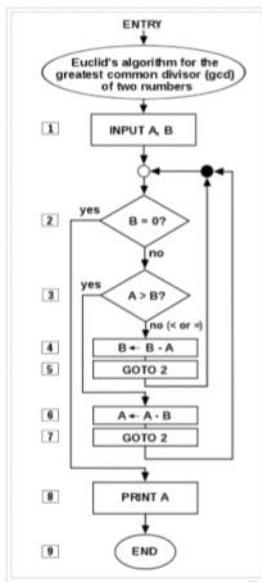
过程：

步骤 1 将这两个数中较大的一个和较小的一个分别赋予 $M$ 和 $N$ 。

步骤 2 用 $M$ 除以 $N$ ，余数设为 $R$ 。

步骤 3 如果 $R$ 不为0，那么将 $N$ 的值赋予 $M$ ，并将 $R$ 的值赋予 $N$ ，然后回到步骤2；否则最大公约数就是 $N$ 当前被赋予的值。

### 欧几里得算法



### 一个问题

从这个算法中你想到了什么

什么是你需要观察到的

算法作为计算目标

算法有输入

算法在每个步骤中有特定的命令序列

算法有三个基本控制结构：序列，条件分支，和循环

算法一定会停止

算法有一个输出

算法是计算的核心

没有计算是不含有算法的

算法是用于当代计算系统的所有技术的核心

算法是解决问题的核心

计算，可计算性计算复杂性只是提供了观念上和理论上的框架，然而算法提供了具体精确的解决多种问题的手段

算法的计算机科学的引擎机

很多计算机科学的程序都是由新的算法引领的

## 算法是计算机科学的引擎机

算法定义：一个可以终止过程的一组有序的，无歧义的，可执行的步骤的集合

简单来说，所谓算法(algorithm)就是定义良好的计算过程，它取一个或一组值作为输入，并产生出一个或一组值作为输出。亦即，算法就是一系列的计算步骤，用来将输入数据转换成输出结果。

我们还可以将算法看作是一种工具，用来解决一个具有良好规格说明的计算问题。有关该问题的表述可以用通用的语言，来规定所需的输入/输出关系。与之对应的算法则描述了一个特定的计算过程，用于实现这一输入/输出关系。

## 算法一定是没有歧义的

Polya列表的第二步中的最后一句说，最终应该得到解决方案。在计算领域，这种解决方案被称为算法。算法是在有限的时间内用有限的数据解决问题或子问题的一套指令。这个定义暗示，算法中的指令是明确的。在计算领域中，必须明确地描述人类解决方案中暗含的条件。例如，在日常生活中，我们不会仔细考虑一个总出现的解决方案。此外，如果一个解决方案要求处理的信息量比我们能够处理得多，它也不会被选用。在计算机解决方案中必须明确这些限制，因此算法的定义包括它们。

为了有效（能够）的解决问题effectively

任何有意义的计算都有算法

为了有效率的解决问题efficiently

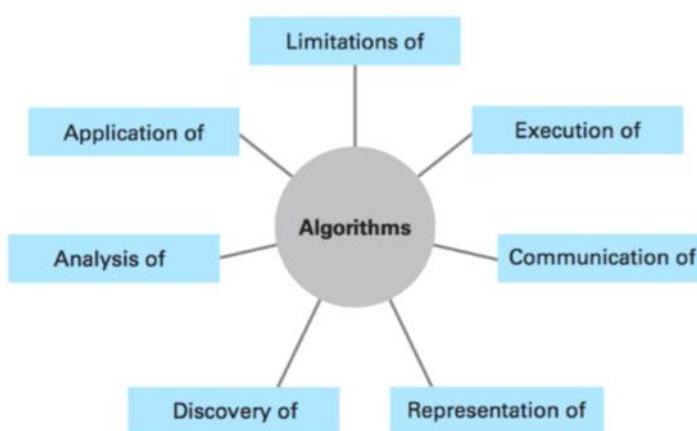
一些可计算的复杂的问题需要有效率的算法

计算时间是有限制的资源，空间和记忆也是一样，我们应该聪明的使用这些资源

为了变成一个专业人员

拥有牢固的算法基础和技术是把真正有技术的专家和初学者区别开的特点

### The central role of algorithms in computer science



## 流程图Flowcharts

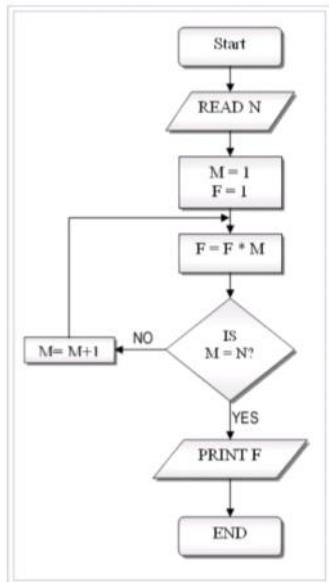
流程图是一个代表算法或者工序，使它们多种多样的形状代表了计算的步骤/行为，然后它的箭头代表了控制转换的方向

### Note

对于一个复杂的程序，它的流程图会变得很乱，然后会让这个算法看起来更复杂。

来很复杂

ANSI/SQ Shape	Name	Description
→	Flowline (Arrowhead) <sup>[14]</sup>	Shows the process's order of operation. A line coming from one symbol and pointing at another. <sup>[14]</sup> Arrowheads are added if the flow is not the standard top-to-bottom, left-to-right. <sup>[14]</sup>
○	Terminal <sup>[14]</sup>	Indicates the beginning and ending of a program or sub-process. Represented as a stadium, <sup>[14]</sup> oval or rounded (tiled) rectangle. They usually contain the word "Start" or "End", or another phrase signaling the start or end of a process, such as "submit inquiry" or "receive product".
□	Process <sup>[14]</sup>	Represents a set of operations that changes value, form, or location of data. Represented as a rectangle. <sup>[14]</sup>
◇	Decision <sup>[14]</sup>	Shows a conditional operation that determines which one of the two paths the program will take. <sup>[14]</sup> The operation is commonly a yes/no question or true/false test. Represented as a diamond (rombua). <sup>[14]</sup>



## 最基本的东西Primitives

计算机科学建立了一组定义良好的(在语法和语义上)构建块，可以从这些构建块构建算法表示。这样的构建块称为原语。

为这些原语分配精确的定义消除了许多歧义问题，并要求根据这些原语描述的算法建立统一的详细级别。

原语集合和说明如何组合原语以表示更复杂思想的规则集合构成了一种编程语言。

Figure 5.2 Folding a bird from a square piece of paper

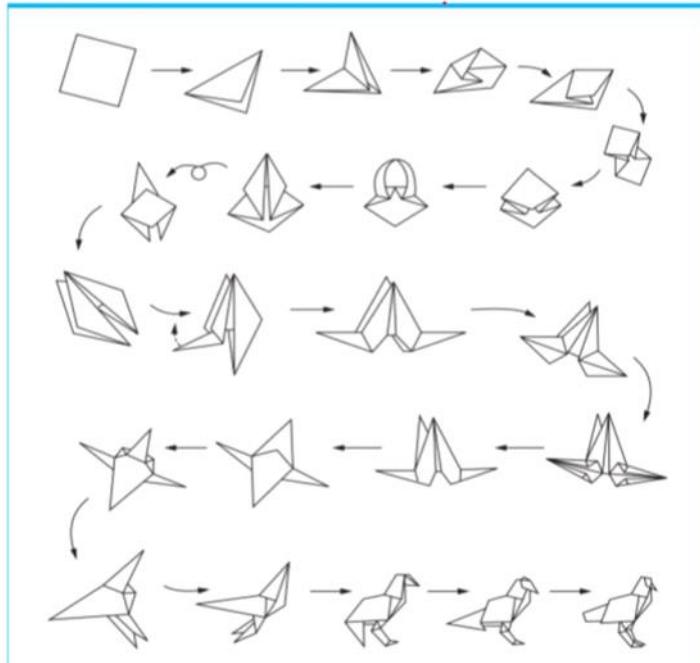
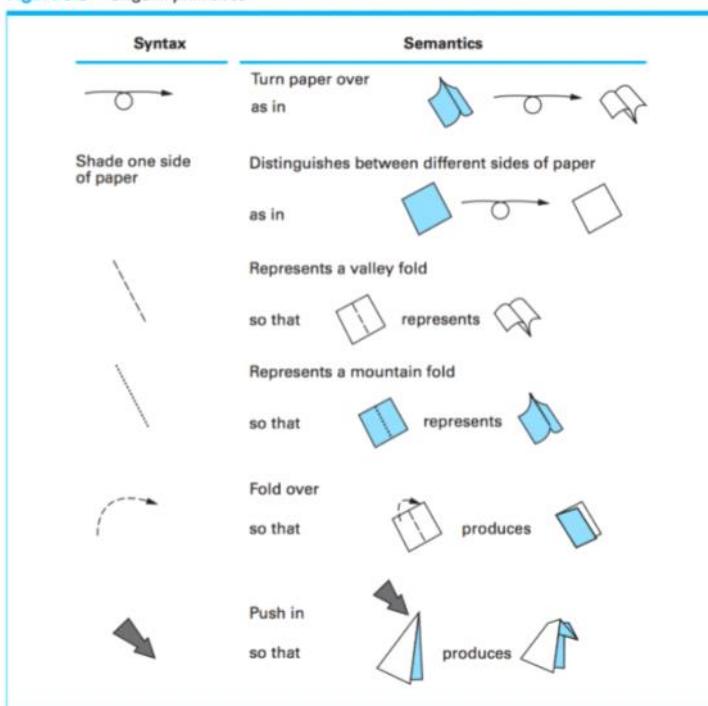


Figure 5.3 Origami primitives



## 伪代码pseudocode

通常，伪代码是一种可以把想法非正式的在算法工序中表达出来的符号系统

获得伪代码一种方式就是简单的放宽对正式语言的要求，只要在最终版本里算法可以被表示出来

它的目的不需要把我们的讨论局限于特定的编程语言，而是去考虑算法的发展和展示

### Note

一个伪代码应该尽可能越简单越好，但不是更简单（不能产生歧义）

---

必考！

伪代码Pseudocode最初的例子

名字←变量表达

如果 (条件) 那么 (活动) 不然 (活动)

当 (条件) 做 (活动)

产生名字 (参数)

伪代码算法的例子

Procedure Greetings

Count ←3;

While (Count>0) do

    (print the message "Hello" and

    Count←Count-1                  破坏性赋值

Note: 分号; 代表了顺序执行 sequential execution 标准的表达方法

Count: 一般叫做计数器

### Pseudocode primitive examples

- ◆ name ← expression
  - ◆ if (condition) then (activity) else (activity)
  - ◆ while (condition) do (activity)
  - ◆ procedure name (parameters)
- 

分类问题定义

输入: 一系列n的数字<a<sub>1</sub>,a<sub>2</sub>,...a<sub>n</sub>>

输出: 一串排列<a'<sub>1</sub> a'<sub>2</sub> ... a'<sub>n</sub> >满足a'<sub>1</sub> ≤ a'<sub>2</sub> ≤ ... ≤ a'<sub>n</sub>

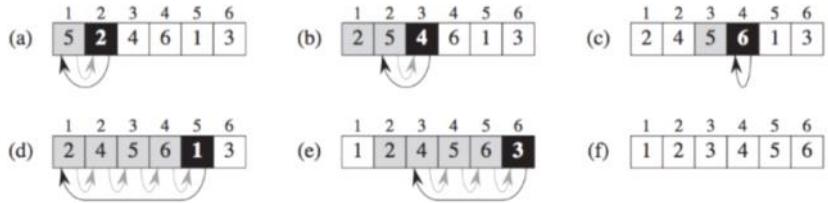
把你手中的卡牌排序

我们从反放在桌上的卡牌开始抽取

然后我们每一次移动一张卡, 把它插在正确的位置

为了找到卡的正确的顺序, 我们从左到右比较每一张在手里的卡





//+后面的东西是辅助说明,电脑不看

**INSERTION-SORT( $A$ )**

```

1  for  $j = 2$  to  $A.length$  对于数组第 $j$ 个, 从第2个取到 $A$ 的长度
2    key =  $A[j]$  临时变量 =  $A[j]$ 
3    // Insert  $A[j]$  into the sorted sequence  $A[1..j - 1]$ .
4     $i = j - 1$  赋值  $i=j-1$ 
5    while  $i > 0$  and  $A[i] > key$  条件满足
6       $A[i + 1] = A[i]$  紧挨了和上一个 $j-1$ 的位置
7       $i = i - 1$  如果不满足直接进入第8行
8     $A[i + 1] = key$  临时变量放置位置

```

## 计算问题的解决流程

### Analysis and specification phase

- |                          |   |
|--------------------------|---|
| Analyze<br>Specification | Understand (define) the problem.<br>Specify the problem that the program is to solve. |
|--------------------------|---|

### Algorithm development phase

- |                   |   |
|-------------------|---|
| Develop algorithm | Develop a logical sequence of steps to be used to solve the problem.          |
| Test algorithm    | Follow the steps as outlined to see if the solution truly solves the problem. |

### Implementation phase

- |      |  |
|------|--|
| Code | Translate the algorithm (the general solution) into a programming language.                                      |
| Test | Have the computer follow the instructions. Check the results and make corrections until the answers are correct. |

### Maintenance phase

- |                 |  |
|-----------------|--|
| Use<br>Maintain | Use the program.<br>Modify the program to meet changing requirements or to correct any errors. |
|-----------------|--|

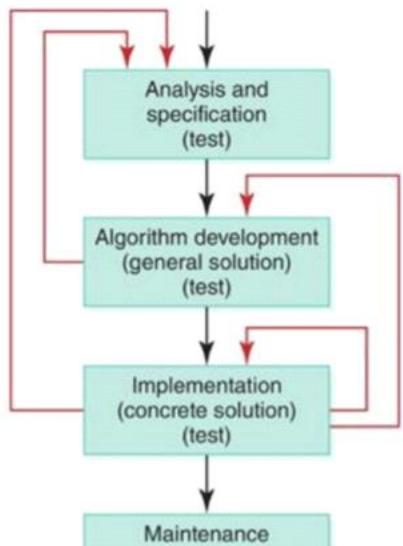


Figure 5.6 The sequential search algorithm in pseudocode

```

procedure Search (List, TargetValue)
if (List empty)
    then
        (Declare search a failure)
else
    (Select the first entry in List to be TestEntry;
     while (TargetValue > TestEntry and
            there remain entries to be considered)
        do (Select the next entry in List as TestEntry.);
        if (TargetValue = TestEntry)
            then (Declare search a success.)
            else (Declare search a failure.)
    ) end if

```

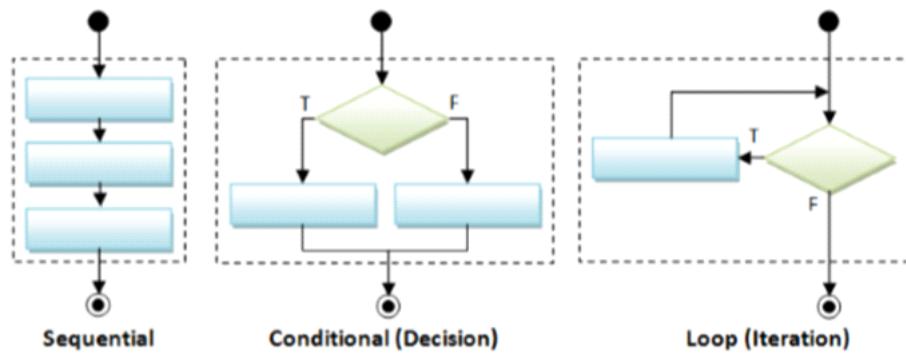
注意分号的表示按顺序进行，循环结构while的条件如果满足，就一直做下去不能停止，直到while不满足为止才进入if

End if 前面的括号是对应于else后面所有东西的反括号包含

三种基本算法的控制结构（重点）

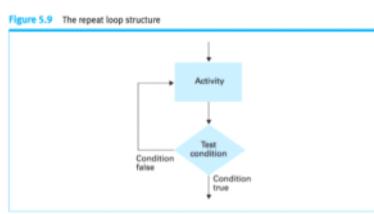
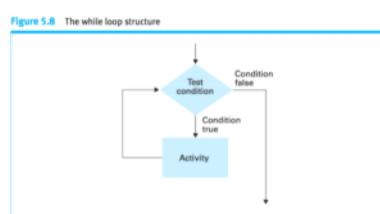
顺序执行，条件分支（判断），和循环（重复）

任何算法/计算可以被这三种基本控制结构的组合来表达



While先判断条件，条件成立才做do，如果不成立do就不做

还要一种until结构，如果不成立一直循环，直到满足条件为止，条件至少做一次



正确的算法

如果每一个输入的例子都可以停止并且给出正确的输出，我们说这个算法是正确的（能够停下来）

正确的算法能够解决可计算问题

不正确的算法

一个算法如果在默写输入的例子中不能停止，或者输出了不正确的答

案，我们说这个算法是不正确的  
与你设想的结果不同  
如果我们可以控制错误比率，有些不正确的算法可能是有用的

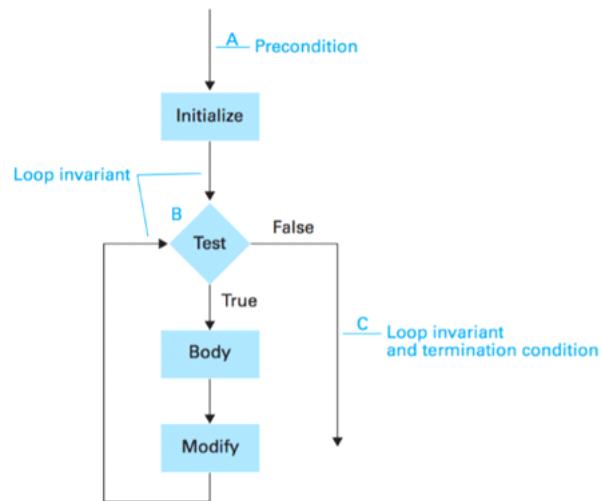
---

## 工程方法

### 测试

所有的工程方法不能保证百分百正确，但是可以确定它的正确程度  
正式的方法（用数学方法或者逻辑上的东西）  
代表了一个验证的对象（算法/程序）或者它规范的正式表达（逻辑上的公式）

Note: 给予一个正确的逻辑体系



# Data, information, and knowledge and their representations

2019年12月4日 20:09

## 数据，信息，知识和它们的表达

---

### 数据：它是什么？

给予或给予的事物;被认为是事实的东西，作为推理或计算的基础;推断，推理推断的假设或前提

由计算机和其他自动设备执行的操作，可以以电信号、磁带或穿孔卡片上的记录等形式存储或传送的数量、字符或符号。

2收集起来作为参考或信息的事实，特别是数字事实

计算机程序所运行的任何形式的信息。

从更有限的意义上说，数据与计算机操作的其他信息的对比形式是不同的，例如文本、图形、语音和图像。它的独特之处在于，它以一种有组织的、重复的、经常被压缩的方式组织起来。

以适合人类或自动方式进行交流、解释或处理的方式来表达事实、概念或指令。“任何在软件的文档或操作中观察到的，与先前验证过的软件产品或参考文档的期望相背离的内容。”

“有时用作文档documentation的同义词。”

### “数据”的概念

数据是记录事物的符号集合，只有经过解释才能代表某种意义(信息)。

#### 例

20080808:密码?学生账号吗?谁的生日?对某人来说特别的一天?(如果张艺谋看到这个数字?)

518055:这是什么数据?你大学的邮政编码吗?)

ytisrevinu nrehtuos:这是什么数据?(你用于课堂练习吗?)

---

### 信息：它是什么？

1.

“通知的动作(在动词的意义上);心智或性格的形成或塑造、训练、指导、教导;传播有益的知识。”

“训练项目;一条指令。”

“神圣的指令,灵感。”

“通知的能力;启发性。”

2.

“通知的动作(在动词的意义上);传播关于某一事实或事件的知识或“消息”;讲述或被告知某事的行为。

3

“关于某一特定事实、主题或事件的知识;被告知或告知的;情报,消息。与

## 数据对比"

与and连用的名词，例如:一项信息或情报;被告知的事实或情况在以前的用法中，指(某事物的)叙述、关系、叙述。

"与被告知者分离的，或没有被告知者的含义的:以两种或两种以上的顺序、安排等中的一种存在的，对某物产生不同反应的，并且能够被无生命的事物储存、传递和沟通的事物。”

"作为数学上定义的量(见引用);现在特别是一个代表的程度选择行使选择或形成的一个特殊符号,序列,消息,等等,一系列可能的,这是对数的定义的统计概率出现的符号或元素的消息。”

4

检举告发、控告或控告(某人)的行为。

"一般来说，信息是任何能够使人类大脑改变其对现实世界当前状态的看法的东西。在形式上，特别是在科学和工程领域，信息是任何有助于减少系统状态的不确定性的东西;在这种情况下，不确定性通常以客观可测量的形式表示。”

"人类通过应用于数据的已知约定赋予数据的意义。”

## "信息"的因素

"意义meaning", "人类humans", "分配assign to", "数据data"

## "信息"的概念

信息是一个概念/概念，只有在有接收者的情况下才成立，以接受者的存在为前提。

信息对接受者有一定的意义，因此，它是一个与接受者的价值观紧密联系和依赖的概念。

关于“信息”最古老的例子:烽火台(中国古代，周朝，大约公元前1000年)在高处或突出的地方燃起火或点燃灯，作为警告或信号(通过点火或以光速)通知下一个烽火台敌人快来了。

---

## 数据与信息的对比

### 数据

数据是基本的价值或事实。

数据可能是非结构化unstructured的，缺少上下文lack context。

### 信息

信息是经过组织organized和/或处理processed的，对解决某种问题有用的数据。

信息帮助我们回答问题(它“通知”。

以有用的方式组织或处理的信息数据

## 数据和信息的不同

接收者的存在the existence of recipient

信息：需要接收者的存在，并且与接收者的值有紧密的联系和依赖。

数据：不需要接收者的存在。

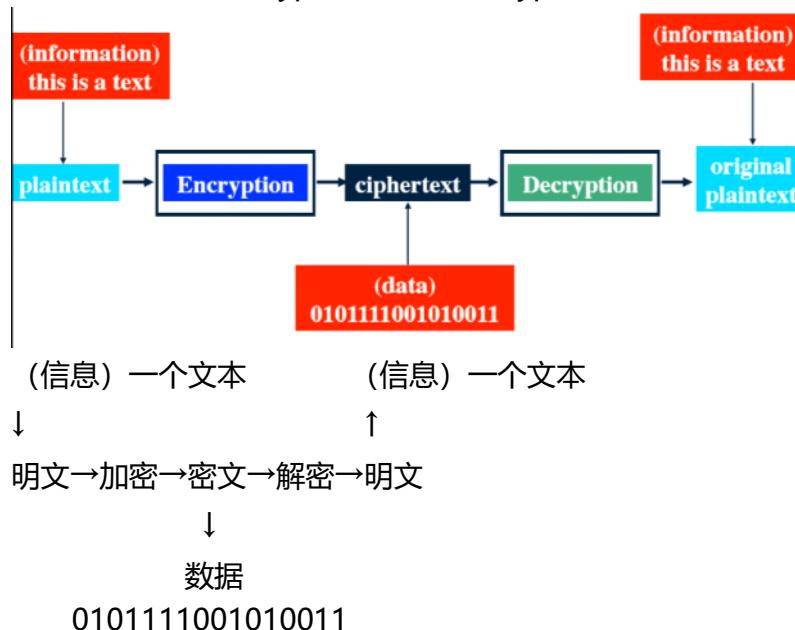
解释interpretation

信息：它的意义取决于接受者的价值观，如果接受者不能理解它，它就没有意义。

数据：如果不进行解释，数据就没有意义，并且通过不同的解释可能会有不同的意义。

---

信息和数据加密encryption和解密decryption



---

数据压缩data compression

数据压缩：减少了存储一块数据

带宽bandwidth：所需的空间，即在固定时间内从一个地方传输到另一个地方的比特或字节数

压缩比compression ratio：压缩数据的大小除以未压缩数据的大小

无损压缩lossless compression：一种不丢失信息的数据压缩技术

有损压缩lossy compression：一种有信息丢失的数据压缩技术

---

模拟数据和数字数据analog data and digital data

数据的表示：模拟数据和数字数据

可以用两种方式之一表示模拟或数字。

模拟数据

模拟数据是一种连续的表示，类似于它所表示的实际信息。

## 数字数据

数字数据是一种离散discrete representation的表示，它将信息分解成不同的元素。

---

## 数据的各种类型

多媒体multimedia几种不同的媒体类型

数字numbers

文本text

音频audio

图像和图形images and graphics

视频video

最后，所有这些数据都存储为二进制数字binary digits。

每个文档、图片和声音片段都以某种方式表示为1和0的字符串。

---

## 知识：它是什么？

“承认是已知的，或关于已知的事物的事实;认可。”

“知道一件事、一种状态等的事实，或(一般意义上)知道一个人;熟人;熟能生巧。”

“对事实的了解;关于事实或事物的知觉或信息;知晓或被告知的状态;意识的东西)。宾语通常是一个表达或暗示的命题。知道一个人是贫穷的，知道他是贫穷的。”

“了解事实，了解信息。”

通过信息或事实而非直接经验获得的对人、事物或感知的知识。

对事实或真理的认识或感知;清晰而确定的心理理解;理解，理解理解的事实、状态或条件从前，还有理解力、智力、才智。”

“对某一学科、语言或类似学科的熟悉;对艺术、科学、工业等的理论或实践的理解。

“被指示的事实或条件，或通过学习或研究获得的信息;熟悉已查明的事实。事实或原则;通过学习获得的信息;学习;博学。”

“已知的总和。”“一门学问;一门科学;一种艺术”。

可以表示为一组事实并为代理或程序所知的信息。知识可以通过其在代理中的体现与信息或数据区分开来;例如，一个代理可能会接收到增加其知识的信息。”

“知识是对某人或某事的熟悉、认识或理解，如事实、信息、描述或技能，这些都是通过感知、发现或学习而获得的经验或教育。”

“知识可以指对一个学科的理论或实践的理解。它可以是隐式的(如实践技能或专业知识)，也可以是显式的(如对一个主题的理论理解);它可以更正式，也可以更不正式，也可以更系统化。”

在哲学中，对知识的研究被称为认识论。

---

数字系统numeral system的例子

一些问题

110:这个数代表的自然数是什么？

ABCDEF:这个“数字”所代表的自然数是什么？

更多关于问题的问题

◆  $1 \times 10^2 + 1 \times 10^1 + 0 \times 10^0 = ?$  (denary, the decimal system)

◆  $1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = ?$  (binary, the binary system)

◆  $1 \times 16^2 + 1 \times 16^1 + 0 \times 16^0 = ?$  (hex, the hexadecimal system)

最后一个问题

从这些例子中你能想到什么？

---

数字和计算numbers and computing

数字是计算的关键

除了使用计算机执行数值计算之外，我们使用计算机存储和管理的所有类型的数据最终都存储为数字。

在最底层，计算机只使用数字0和1来存储所有数据。

数学中的数字和计算机中的数字

数字可以被分为各种类别：自然数、负数、有理数、无理数，以及许多其他在数学中很重要但对理解计算并不重要的数字。

---

数字：它是什么？

数字的定义

数字是一个属于抽象数学系统的单位，它遵循特定的演绎法、加法和乘法法则。

数字是一个值的表示，某些算术运算可以一致地应用于这些值。

抽象数学系统中服从算术法则的一个单位

---

按位记数法positional notation

基数

数字系统的基数指定系统中使用的位数。数字总是从0开始，一直到比基数小1。

基数也决定了数位位置的意义。

基数：一个数字系统的基本，它规定数字的数目和数位位置的值

例子

- ◆ There are 2 digits in base 2: 0 and 1.
- ◆ There are 8 digits in base 8: 0 through 7.
- ◆ There are 10 digits in base 10: 0 through 9.

最右边的数字代表它的值乘以基数的0次幂。它左边的数字代表它的值乘以底的一次方。下一位表示它的值乘以底的2次方，以此类推。

若基r数制中的一个数有n位，则表示为：表示数字中第i个位置的数字：

$$d_n * R^{n-1} + d_{n-1} * R^{n-2} + \dots + d_2 * R^1 + d_1 * R^0$$

### 零的重要性

按位记数法之所以可行，是因为零的概念。零是现代数学各分支交叉的基本概念。正如乔治•伊夫拉(Georges Ifrah)在他的《计算世界史》(The Universal History of Computing)一书中所指出的那样：“总而言之，‘零’的重大发现赋予了人类思维异常强大的潜力。”没有任何其他的人类创造对人类智力的发展产生过这样的影响

---

### 位置数系统

在位置数系统中，符号在数字中的位置决定了它所代表的值。

在这个系统中，一个数字表示为：

$$\pm (S_{k-1} \dots S_2 S_1 S_0 \cdot S_{-1} S_{-2} \dots S_{-l})_b$$

这个数的值为

$$n = \pm S_{k-1} \times b^{k-1} + \dots + S_1 \times b^1 + S_0 \times b^0 \\ + S_{-1} \times b^{-1} + S_{-2} \times b^{-2} + \dots + S_{-l} \times b^{-l}$$

其中S是符号的集合，b是基(或基)，它等于集合S中符号的总个数，S和S是数字的整数部分和小数部分的符号。请注意，我们使用的表达式可以从右或从左扩展。换句话说，b的幂在一个方向上可以是0到k-1或者从-1到-l，方向相反。b的非负次项与整数部分有关，而b的负次项与小数部分有关。这个符号表示这个数可以是正的，也可以是负的。在这一章中，我们将研究几个位置数系统。

但是为了简单起见，我们经常去掉括号、底数和加号(如果数字ber是正数)。例如，我们将数字

+ (552.23)base10

写成552.23——底数和加号是隐式的。

---

### 位置数系统：整数integers

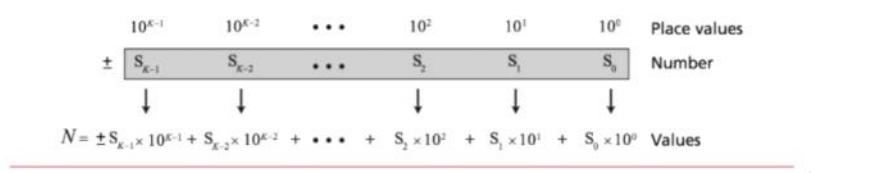
在十进制中，整数(没有小数部分的整数)是我们大家都熟悉的——我们在日常生活中使用整数。事实上，我们经常使用它们，所以它们很直观。我们用±S, ..., S1, S0, 表示整数。计算值为：

$$N = \pm S_{K-1} \times 10^{K-1} + S_{K-2} \times 10^{K-2} + \dots + S_2 \times 10^2 + S_1 \times 10^1 + S_0 \times 10^0$$

其中 $S_i$ 是位数， $b = 10$ 是基数， $K$ 是位数的数量。

在数字系统中显示整数的另一种方法是使用位值place values，它是10的幂乘一个数。图2.1显示了十进制系统中使用的一个整数。

Figure 2.1 Place value for an integer in decimal system



### 位置数系统：实数reals

实数十进制中的实数(带有小数部分的数字)也很常见。

例如，我们使用这个系统来显示美元和美分(\$23.40)。

我们可以用 $\pm S(k-1) \dots S_1 S_0 \cdot S_{-1} \dots S_{-L}$ 表示实数a。计算值为：

Integral part	Fractional part
$R = \pm S_{K-1} \times 10^{K-1} + \dots + S_1 \times 10^1 + S_0 \times 10^0 +$	$S_{-1} \times 10^{-1} + \dots + S_{-L} \times 10^{-L}$

其中 $S_i$ 为位数， $b = 10$ 为基数， $K$ 为整数部分的位数， $L$ 为小数部分的位数。我们在表示中使用的小数点将小数部分与整数部分分开。

## 二进制、八进制和十六进制binary octal and hexadecimal

### 二进制binary

二进制数字系统(有两个数字0和1)在计算中特别重要。

熟悉以2为幂的数字系统也很有帮助，比如以8为基数(八进制)和以16为基数(十六进制)。

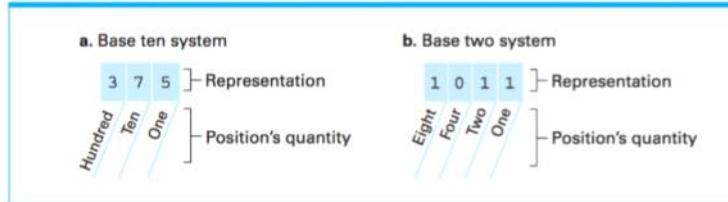
### 八进制octal

基数-8(八进制)数字系统有8位(0-7)。

### 十六进制hexadecimal

16进制数字系统有16位:0 1 2 3 4 5 6 7 8 9 A B C D E F。

**Figure 1.15** The base ten and binary systems



**Figure 1.16** Decoding the binary representation 100101

Binary pattern	1	0	0	1	0	1
	1	x one	= 1			
	0	x two	= 0			
	1	x four	= 4			
	0	x eight	= 0			
	0	x sixteen	= 0			
	1	x thirty-two	= 32			
Value of bit						37 Total
Position's quantity						

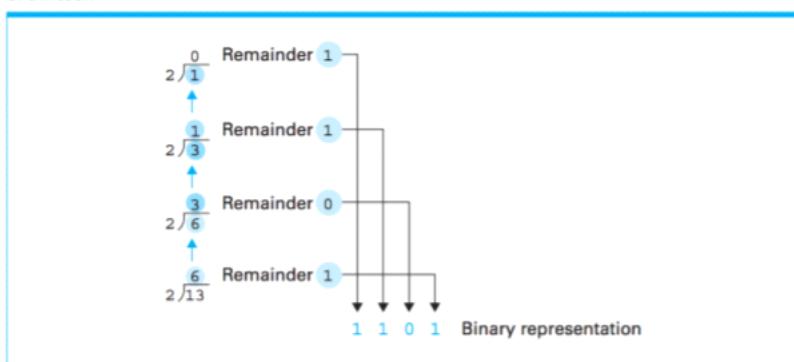
## 如何将十进制化成二进制

步骤1。将值除以2，然后记录其余的值。

步骤2。只要所得的商数不为零，就继续除把最新的商数除以2，再把余数记下来。

步骤3。现在已经得到了一个零商，原始值的二进制表示由从右到左列出的余数按照它们记录的顺序组成。

**Figure 1.18** Applying the algorithm in Figure 1.17 to obtain the binary representation of thirteen



## 二进制值和计算机

### 二进制表示

虽然早期的一些计算机是十进制的，但是现代的计算机是二进制的，也就是说。计算机中的数据和指令以二进制形式表示。

原因是计算机中的每个存储单元都包含一个低压信号或一个高压信号。

因为每个位置只能有两种状态中的一种，所以将这些状态等同于0和1是合乎逻辑的。

### 位、字节和字bit byte and word

二进制数字：二进制数字系统中的数字0或1

位：二进制数字

字节：八位二进制数字

字：一个或多个字节的一组，一个字的位bit就是计算机的字长

---

数据的二进制表示

二进制表示的固有性质

一个位bit只能代表两件东西

两个位bit可以表示四种东西，因为0和1的四种组合可以由两位构成：00、01、10和11

一般来说，n位可以表示 $2^n$ 个东西，因为0和1的 $2^n$ 个组合可以由n位组成。

每次将可用的位数增加1，表示的东西的数量就会增加一倍。

最小存储量

一个计算机架构可以同时处理和移动的最小的位的数量，通常是2的幂。

---

位的组合

1 Bit	2 Bits	3 Bits	4 Bits
0	00	000	0000
1	01	001	0001
	10	010	0010
	11	011	0011
		100	0100
		101	0101
		110	0110
		111	0111
			1000
			1001
			1010
			1011
			1100
			1101
			1110
			1111

---

二进制系统：整数

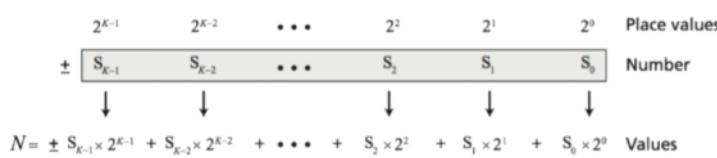
我们用 $\pm S_{K-1}, \dots, S_1, S_0$ 表示整数。计算值为：

$$N = \pm S_{K-1} \times 2^{K-1} + S_{K-2} \times 2^{K-2} + \dots + S_2 \times 2^2 + S_1 \times 2^1 + S_0 \times 2^0$$

其中 $S_i$ 是位数， $b = 2$ 是基数， $K$ 是位数的数量。

在数字系统中显示整数的另一种方法是使用位值place values，它是2的幂乘一个数。图2.1显示了二进制系统中使用的一个整数。

Figure 2.2 Place values in an integer in the binary system



---

二进制系统：实数

实数二进制中的实数(带有小数部分的数字)也很常见。

一个二进制系统表示师叔可以由K个在左边的字节和L个在右边的字节

我们可以用 $\pm S_{(k-1)} \dots S_1 S_0 \cdot S_{-1} \dots S_{-L}$ 表示实数a。计算值为:

$$R = \pm S_{(k-1)} \times 2^{k-1} + \dots + S_1 \times 2^1 + S_0 \times 2^0 + S_{-1} \times 2^{-1} + \dots + S_{-L} \times 2^{-L}$$

其中 $S_i$ 为位数,  $b = 2$ 为基数,  $K$ 为整数部分的位数,  $L$ 为小数部分的位数。我们在表示中使用的小数点将小数部分与整数部分分开。注意 $K$ 从0开始, 而 $L$ 从-1开始。最高的幂是 $K-1$ , 最低的幂是 $-L$

## 八进制与十六进制同理

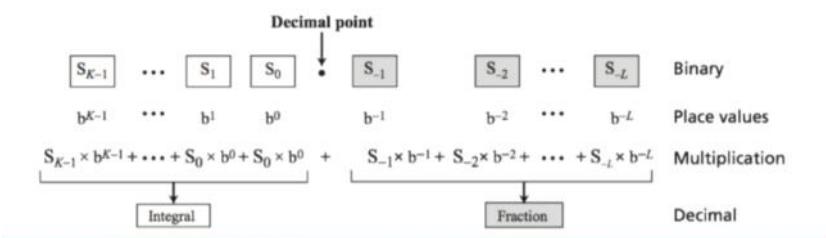
**Table 2.1** Summary of the four positional number systems

System	Base	Symbols	Examples
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	2345.56
Binary	2	0, 1	(1001.11) <sub>2</sub>
Octal	8	0, 1, 2, 3, 4, 5, 6, 7	(156.23) <sub>8</sub>
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	(A2C.A1) <sub>16</sub>

**Table 2.2** Comparison of numbers in the four systems

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

**Figure 2.5** Converting other bases to decimal



数字数据numeric data的表示:

符号幅度表示signed-magnitude representation

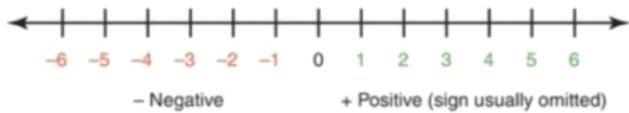
符号量值表示在传统的十进制系统中, 符号(+或-)放在数字的值之前, 尽

管通常假设是正数。符号表示顺序，数字表示数字的大小。

问题:0有两种表示——0和+0。

### 符号幅度表示法

数字表示的一种，其中符号表示数字(正负)的顺序，数值表示幅度

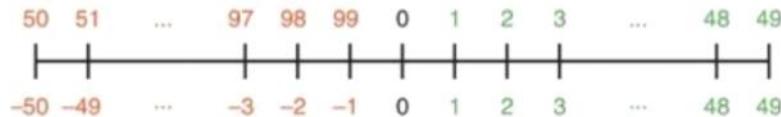


数字数据的表示:固定大小的数字fixd-sized numbers

### 固定大小的数字

如果我们只允许一个固定数量的值，我们可以将数字表示为整数值，其中一半表示负数

符号是由数字的大小决定的。例如，如果我们可以表示的小数位数的最大值是2，我们可以让1到49是正数1到49，让50到99是负数-50到-1。



### 添加固定大小的数字

我们只要把数字加在一起，然后放弃最左边的进位

Signed-Magnitude	New Scheme
$\begin{array}{r} 5 \\ + -6 \\ \hline -1 \end{array}$	$\begin{array}{r} 5 \\ + 94 \\ \hline 99 \end{array}$
$\begin{array}{r} -4 \\ + 6 \\ \hline 2 \end{array}$	$\begin{array}{r} 96 \\ + 6 \\ \hline 2 \end{array}$
$\begin{array}{r} -2 \\ + -4 \\ \hline -6 \end{array}$	$\begin{array}{r} 98 \\ + 96 \\ \hline 94 \end{array}$

### 固定大小的数字的减法

加减法的关系: $A-B=A+(-B)$

我们可以把一个数减去另一个数把第二个数的负数加到第一个数上。

Signed-Magnitude	New Scheme	Add Negative
$\begin{array}{r} -5 \\ - 3 \\ \hline -8 \end{array}$	$\begin{array}{r} 95 \\ - 3 \\ \hline \end{array}$	$\begin{array}{r} 95 \\ + 97 \\ \hline 92 \end{array}$

固定大小的数字:十的补码符号ten's complement notation

我们还可以使用一个公式来计算负的表示:负( $I$ ) =  $10^k - I$ ，其中 $I$ 是我们想要表示的小数， $k$ 是固定大小的数字系统中允许的位数。

**Example of two-digits system:**  $-(3) = 10^2 - 3 = 97$

**Example of three-digits system:**  $-(3) = 10^3 - 3 = 997$

## 二进制补码系统

Figure 1.21 Two's complement notation systems

a. Using patterns of length three		b. Using patterns of length four	
Bit pattern	Value represented	Bit pattern	Value represented
011	3	0111	7
010	2	0110	6
001	1	0101	5
000	0	0100	4
111	-1	0011	3
110	-2	0010	2
101	-3	0001	1
100	-4	0000	0
		1111	-1
		1110	-2
		1101	-3
		1100	-4
		1011	-5
		1010	-6
		1001	-7
		1000	-8

### 二进制补码系统:表示正的值

该系统由一串适当长度的操作系统开始, 然后以二进制进行计数, 直到得到一个0后面跟着1的模式。

这些模式表示值0, 1, 2, 3...

正数: 用二进制表示法写出该数, 在这串数前面再加0, 注意要求是几

位就加几个0保证到所有位数加和=要求位数

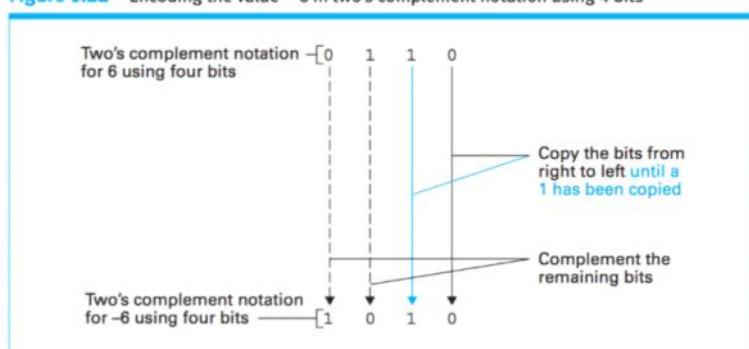
例如要求位数为4位, 表示6为0110

### 二进制补码系统:表示负值

表示负值的位模式是通过从适当长度的1字符串开始, 然后在二进制中向后计数, 直到到达由一个1后跟0组成的模式为止。这些模式代表的值是-1, -2, -3...

负数: 将表示好的正数开始转录, 从最左边开始抄写, 直到出现第一个1的时候, 把这个1抄成1, 然后把右边的数是0的抄为1, 是1的抄为0

Figure 1.22 Encoding the value -6 in two's complement notation using 4 bits



### 二进制补码系统的加法

Figure 1.23 Addition problems converted to two's complement notation

Problem in base ten	Problem in two's complement	Answer in base ten
$\begin{array}{r} 3 \\ + 2 \\ \hline \end{array}$	$\begin{array}{r} 0011 \\ + 0010 \\ \hline 0101 \end{array}$	5
$\begin{array}{r} -3 \\ + -2 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ + 1110 \\ \hline 1011 \end{array}$	-5
$\begin{array}{r} 7 \\ + -5 \\ \hline \end{array}$	$\begin{array}{r} 0111 \\ + 1011 \\ \hline 0010 \end{array}$	2

数字数据表示:表示实数

小数点 radix point

小数点在任何数字中，将实数的整个部分与小数部分分隔开的点

浮点表示法 floating point notation

任何实值都可以用三个属性来描述:符号sign(正或负), 尾数mantissa, 由数值中的数字组成, 假设小数点在右边;指数exponent, 它决定了小数点相对于尾数的移位方式。

浮点数: 一种实数的表示, 它跟踪符号、尾数和指数

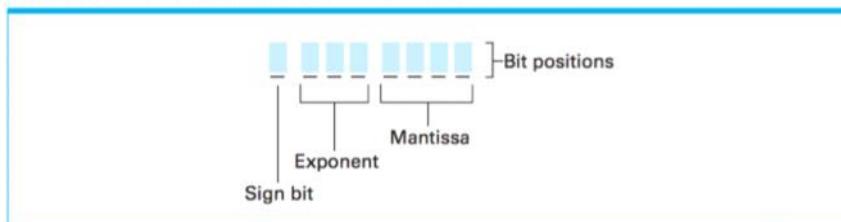
浮点表示法的例子

TABLE 3.1 Values in decimal notation and floating-point notation (five digits)

Real Value	Floating-Point Value
12001.00	$12001 * 10^0$
-120.01	$-12001 * 10^{-2}$
0.12000	$12000 * 10^{-5}$
-123.10	$-12310 * 10^{-2}$
155555000.00	$15555 * 10^4$

一个只是用一个字节存储的浮点表示法例子

Figure 1.26 Floating-point notation components



IEEE浮点表示法的标准

Figure 3.12 IEEE standards for floating-point representation

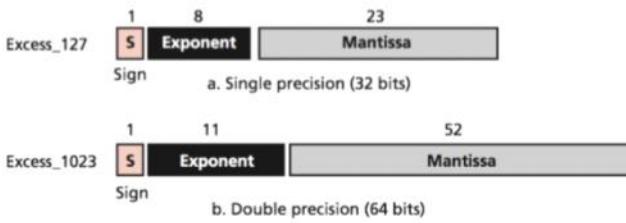


Table 3.2 Specifications of the two IEEE floating-point standards

Parameter	Single Precision	Double Precision
Memory location size (number of bits)	32	64
Sign size (number of bits)	1	1
Exponent size (number of bits)	8	11
Mantissa size (number of bits)	23	52
Bias (integer)	127	1023

两个IEEE浮点标准的规范

参数	双精度	单精度
内存位置大小(比特数)	32	64
符号大小(比特数)	1	1
指数大小(比特数)	8	11
尾数大小(比特数)	23	52
偏差(正数)	127	1023

文本数据表示

表示文本

文本文档可以分解为段落、句子、单词，最后是单个字符。

要以数字形式表示文本文档，只需能够表示可能出现的每个字符。

文档是连续的(模拟的)实体，不同的字符是离散的(数字的)元素，我们需要这些元素来表示和存储在计算机内存中。

字符集

用于表示每个字符的字符和代码的列表

Figure 3.14 Representing symbols using bit patterns



Table 3.3 Number of symbols and bit pattern length

Number of symbols	Bit pattern length	Number of symbols	Bit pattern length
2	1	128	7
4	2	256	8
8	3	65 536	16
16	4	4 294 967 296	32

ASCII字符集

ASCII(发音为AS-kee)代表美国标准信息交换代码。

美国国家标准协会(ANSI, 发音为“AN-see”)采用了ASCII。ASCII字符集最初使用7位数来表示每个字符，允许128个唯一字符。每个字符字节中的第8位最初用作检查位check bit，这有助于确保正确的数据传输。

### Latin-1 Extended ASCII字符集

后来ASCII的发展使得所有的8位都被用来表示一个字符。这个8位数正式称为Latin-1扩展ASCII字符集扩展。

ASCII字符集允许256个字符，包括重音字母letters和其他几个特殊符号special symbols。

Left Digit(s)	Right Digit	ASCII									
		0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3	
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	
3	RS	US	□	!	“	#	\$	%	&	'	
4	(	)	*	+	,	-	.	/	0	1	
5	2	3	4	5	6	7	8	9	:	;	
6	<	=	>	?	@	A	B	C	D	E	
7	F	G	H	I	J	K	L	M	N	O	
8	P	Q	R	S	T	U	V	W	X	Y	
9	Z	[	\	]	^	_	‘	a	b	c	
10	d	e	f	g	h	i	j	k	l	m	
11	n	o	p	q	r	s	t	u	v	w	
12	x	y	z	{		}	-	DEL			

### ASCII字符集

图中代码被表示为十进制数，但是这些值被转换为它们的二进制等价物，以便存储在计算机中。

### ASCII字符集中不同的顺序

根据用于存储ASCII字符的代码，ASCII字符有不同的顺序。每个字符在其他字符之前或之后都有一个相对位置。这个属性在几个不同的方面有帮助。

### 字符集中的特殊字符

ASCII字符表中的前32个字符没有可以打印到屏幕上的简单字符表示。这些字符保留用于特殊目的，如回车和制表符。

#### Example

- ◆ H: 072
- ◆ e: 101
- ◆ l: 108
- ◆ o: 111
- ◆ .: 046

Figure 1.13 The message “Hello.” in ASCII

01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

文本数据的表示:Unicode字符集

### Unicode字符集

ASCII字符集的扩展版本提供256个字符，这对于英语来说已经足够了，但是对于国际使用来说还不够。

Unicode字符集的目标就是用全世界使用的每种语言表示每种字符，包括所有的亚洲表意文字。它还代表许多特殊用途的字符，如科学符号。

Unicode使用16位的独特模式来表示每个符号。因此，Unicode由65,536个不同的位模式组成，足以表示用中文、日文等语言编写的文本。

ASCII字符集是Unicode字符集的子集

Unicode的一个方便的方面是它将ASCII字符作为具有相同数值的子集。

Code (Hex)	Character	Source
0041	A	English (Latin)
042F	Я	Russian (Cyrillic)
OE09	ڻ	Thai
13EA	ࡏ	Cherokee
211E	܂	Letterlike symbols
21CC	܃	Arrows
282F	܄	Braille
345F	ݔ	Chinese/Japanese/Korean (common)

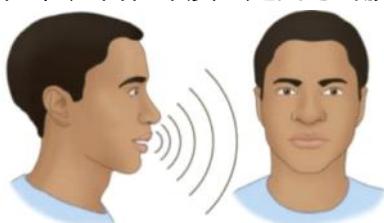
---

音频数据的表示audio data

### 声波

当一系列的空气压缩震动我们耳朵里的一层薄膜时，我们就会感知到声音，而 this 层薄膜会向我们的大脑发送信号。

因此，声音在本质上是由与耳膜相互作用的空气波来定义的。



声波抽样sound wave sampling

为了表示声音，我们必须以某种方式表示适当的声波。

为了在计算机上表示音频数据，我们必须将声波数字化，以某种方式将其分解成离散的、可管理的片段。

模拟信号的电压连续变化。为了使信号数字化，我们定期测量信号的电压并记录适当的数值。这个过程叫做采样sampling。

我们得到的不是一个连续的信号，而是一系列代表不同电压水平的数字。十年当诵，与志同行（与梦同行）

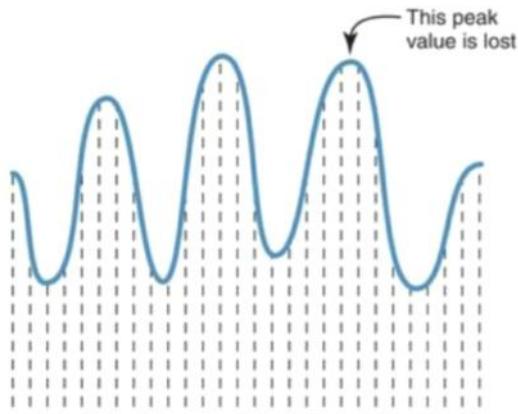


Figure 3.15 An audio signal

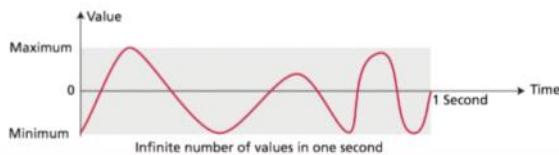


Figure 3.16 Sampling an audio signal

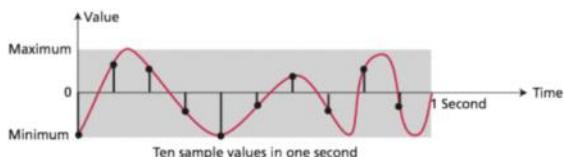
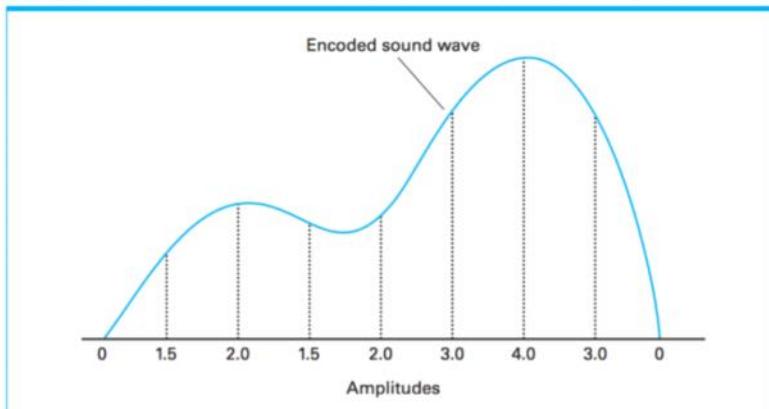


Figure 1.14 The sound wave represented by the sequence 0, 1.5, 2.0, 1.5, 2.0, 3.0, 4.0, 3.0, 0



## 编码音频数据

对计算机存储和处理的音频数据进行编码的最普遍的方法是对声波的振幅按一定的间隔采样，并记录所得到的一系列值。

该技术的采样速率为每秒8000个样本，多年来一直用于长途语音电话通信。

虽然每秒8000个样本似乎是一个快速的速度，但对于高保真的音乐录制来说是不够的。为了获得今天的音乐cd所获得的高质量的声音再现，使用了每秒44,100个样本的采样率

## 音频格式audio formats

例如:WAV, AU, AIFF, VQF和MP3。

所有这些格式都基于从模拟信号中采样的电压值的存储，但都以不同的方式识别数据的细节，并在某种程度上使用各种压缩技术。

## Mp3

MP3是MPEG-2的缩写，音频层3文件。MPEG是国际电影专家组的缩写。MP3首先分析了频率传播，并将其与人类心理声学的数学模型进行了比较。然后，它丢弃人类听不到的信息。最后，使用一种形式的哈夫曼编码对比特流进行压缩，以实现额外的压缩。

---

## 图像image和图形数据geaphic data的表示

### 代表颜色

颜色是我们对到达视网膜的各种频率的光的感知。

我们的视网膜有三种类型的彩色感光锥细胞，它们对不同的频率有反应。

这些光感受器类别对应于红色、绿色和蓝色(三原色)。所有其他肉眼可见的颜色都可以由这三种颜色的不同数量组合而成(三色理论)。

### RGB颜色模型

在计算机中，颜色通常表示为RGB(红绿蓝)值，RGB实际上是三个数字，表示这三种原色的相对贡献。

### 颜色深度color depth

用来表示颜色的数据量称为颜色深度。它通常用表示颜色的比特数来表示。

### 高的颜色high color

高颜色表示16位颜色深度16-bit color depth。在这个方案中，RGB值中的每个数字使用5位，额外的位有时用来表示透明性。

### 真实的颜色True Color

表示24位的颜色深度24-bit color depth。使用这种方案，RGB值中的每个数字都得到8位，每个数字的范围是0到255。这使得它能够代表超过1670万种独特的颜色。

RGB VALUE			
Red	Green	Blue	Color
0	0	0	black
255	255	255	white
255	255	0	yellow
255	130	255	pink
146	81	0	brown
157	95	82	purple
140	0	0	maroon

**Table 3.4** Some colors defined in True-Color

Color	Red	Green	Blue	Color	Red	Green	Blue
Black	0	0	0	Yellow	255	255	0
Red	255	0	0	Cyan	0	255	255
Green	0	255	0	Magenta	255	0	255
Blue	0	0	255	White	255	255	255

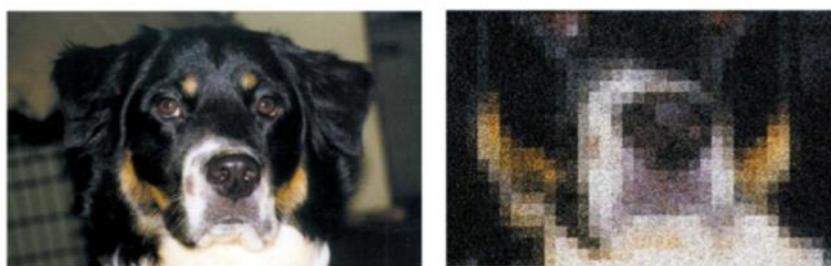
数字化的图像和图形

将一幅图像数字化就是将其表示为一个称为像素pixels的点的集合，像素是表示图像元素的术语。每个像素由单一颜色组成。

用来表示一幅图像的像素数称为分辨率resolution。如果使用了足够的像素(高分辨率)，然后按正确的顺序并排显示，人眼可能会误以为看到的是连续的图像。

像素：用来表示一幅画的单个点，代表行成图像的元素

分辨率：用于表示图像的像素的数量



**FIGURE 3.12** A digitized picture composed of many individual pixels

光栅图形格式raster-graphics format

在逐像素基础上存储图像数据称为栅格图形格式。

目前有几种流行的光栅图形文件格式，包括位图(BMP)、GIF、JPEG和PNG。

光栅格图形格式：逐像素存储图像信息

BMP

一个BMP (bitmap) 文件是最直接的图形表示形式之一。

除了一些管理细节外，BMP文件还包含图像从左到右和从上到下的像素颜色值。

BMP文件支持24位真彩，尽管通常可以指定颜色深度来减小文件大小。

GIF

图形交换格式(GIF)将图像中可用颜色的数量限制为256种。

一个GIF图像只能由256种颜色组成，但是每个GIF图像可以由不同的256种颜色组成。这种技术称为索引颜色indexed color，其结果是更小的文件大小，因为需要引用的颜色更少。

GIF格式的一个版本允许通过存储一系列的图像来定义小的动画，这些图

像是诸如浏览器之类的程序连续显示的。

## JPEG

JPEG格式是为了利用我们眼睛的本质而设计的。

人类对远处亮度和颜色的逐渐变化比对快速变化更敏感。因此，JPEG格式存储的数据平均了短距离

## PNG

PNG(发音为“ping”)代表便携式网络图形portable network graphics。

PNG图像通常可以实现比gif更大的压缩，同时提供更广泛的彩色细线。

然而，PNG图像不支持动画

## 图形的矢量表示vector representation of graphics

矢量图形格式用线条和几何形状来描述图像。

矢量图形是描述线条方向、厚度和颜色的一系列命令。

使用这些格式生成的文件大小往往很小，因为不必考虑每个像素。图像的复杂性，例如图像中的项数，决定了文件的大小。

## 调整resizing

栅格图形必须经过多次编码以适应不同的大小和比例。相比之下，矢量图形可以通过数学方法调整大小，并且可以根据需要动态计算这些变化。

## 画画

矢量图形图像不适合表示现实世界的图像，但矢量图形图像适合用于线条艺术和卡通风格的绘图。

## Flash

现在网络上使用的最流行的矢量图形格式是Flash。

Flash图像以二进制格式存储，需要一个特殊的编辑器来创建。

---

## 表示视频

视频数据是最复杂的数据类型之一，需要捕获、压缩并得到人眼能够理解的结果。

视频剪辑包含许多相当于静止图像的内容，每个图像都必须压缩。

## 视频编解码器video codecs

Codec代表压缩机Compressor/解压缩器Decompressor。

视频编解码器指的是缩小电影尺寸的方法，这样就可以在电脑或网络上播放几乎所有的视频，

编解码器都使用有损压缩来最小化与视频相关的大量数据。因此，目的是不丢失影响观众感官的信息。



# Data storage

2019年11月18日 10:33

## 数据存储

### 真实值true-values和位bits

为了理解单个位在计算机中是如何存储和操作的，可以方便地将其想象为0位代表真值“false”，而1位代表真值“true”，因为这允许我们将操纵位看作操纵真/假真值。

### 逻辑(布尔Boolean)操作

操作真/假真值的操作称为逻辑操作logical operations或布尔操作

Boolean operation(为了纪念逻辑学家/数学家乔治·布尔(1815-1864))。

三个基本的逻辑(布尔)操作是AND(连接)，OR(析取)和XOR(排他析取)。

#### The Boolean operations AND, OR, and XOR (exclusive or)

##### The AND operation

$$\begin{array}{r} \text{AND } 0 \\ \text{AND } 0 \\ \hline 0 \end{array} \quad \begin{array}{r} \text{AND } 0 \\ \text{AND } 1 \\ \hline 0 \end{array} \quad \begin{array}{r} \text{AND } 1 \\ \text{AND } 0 \\ \hline 0 \end{array} \quad \begin{array}{r} \text{AND } 1 \\ \text{AND } 1 \\ \hline 1 \end{array}$$

##### The OR operation

$$\begin{array}{r} \text{OR } 0 \\ \text{OR } 0 \\ \hline 0 \end{array} \quad \begin{array}{r} \text{OR } 0 \\ \text{OR } 1 \\ \hline 1 \end{array} \quad \begin{array}{r} \text{OR } 1 \\ \text{OR } 0 \\ \hline 1 \end{array} \quad \begin{array}{r} \text{OR } 1 \\ \text{OR } 1 \\ \hline 1 \end{array}$$

##### The XOR operation

$$\begin{array}{r} \text{XOR } 0 \\ \text{XOR } 0 \\ \hline 0 \end{array} \quad \begin{array}{r} \text{XOR } 0 \\ \text{XOR } 1 \\ \hline 1 \end{array} \quad \begin{array}{r} \text{XOR } 1 \\ \text{XOR } 0 \\ \hline 1 \end{array} \quad \begin{array}{r} \text{XOR } 1 \\ \text{XOR } 1 \\ \hline 0 \end{array}$$

## 逻辑门

当给定逻辑(布尔)操作的输入真值时，产生逻辑(布尔)操作输出的设备称为逻辑门logic gate。

逻辑门可以由多种技术构成，如齿轮、继电器和光学设备。

今天，逻辑门通常被实现为小型电子电路，其中数字0和1被表示为电压电平。

逻辑门是逻辑电路的基本构件basic building blocks

## 逻辑门的画法



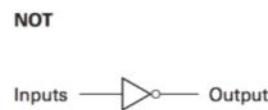
Inputs	Output
0 0	0
0 1	0
1 0	0
1 1	1



Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	1



Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	0



Inputs	Output
0	1
1	0

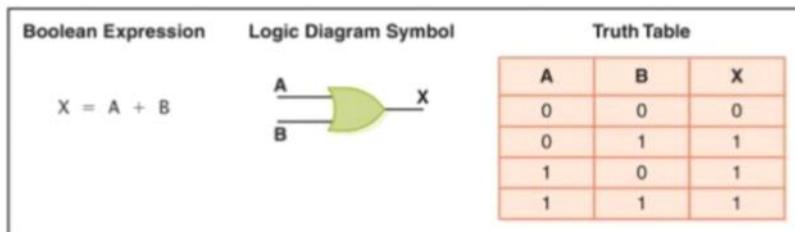
门gate: 一种对电信号进行基本操作的装置，接受一个或多个输入信号并产生单个输出信号

电路circuit: 一种相互作用的门的组合，用来完成特定的逻辑功能

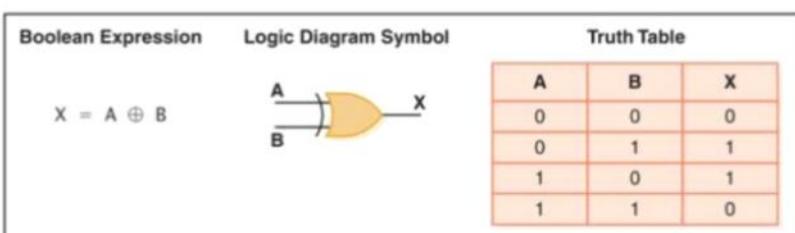
布尔代数Boolean algebra: 一种表示两值逻辑函数的数学符号

逻辑图logic diagram: 电路的图形表示;每种门都有自己的符号

真值表truth table: 显示所有可能的输入值和相关的输出值的表



**FIGURE 4.3** Representations of an OR gate



**FIGURE 4.4** Representations of an XOR gate

Boolean Expression	Logic Diagram Symbol	Truth Table						
$X = A'$		<table border="1"> <thead> <tr> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	X	0	1	1	0
A	X							
0	1							
1	0							

FIGURE 4.1 Representations of a NOT gate

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A \cdot B$		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	X	0	0	0	0	1	0	1	0	0	1	1	1
A	B	X															
0	0	0															
0	1	0															
1	0	0															
1	1	1															

FIGURE 4.2 Representations of an AND gate

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = (A + B)'$		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	X	0	0	1	0	1	1	1	0	1	1	1	0
A	B	X															
0	0	1															
0	1	1															
1	0	1															
1	1	0															

FIGURE 4.5 Representations of a NAND gate

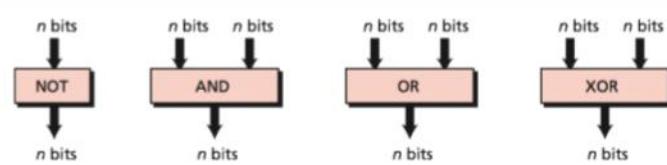
Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = (A + B)'$		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	X	0	0	1	0	1	0	1	0	0	1	1	0
A	B	X															
0	0	1															
0	1	0															
1	0	0															
1	1	0															

FIGURE 4.6 Representations of a NOR gate

Figure 4.1 Logic operations at the bit level

NOT																															
<table border="1"> <thead> <tr> <th>x</th> <th>NOT x</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	NOT x	0	1	1	0	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>x AND y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	x AND y	0	0	0	0	1	0	1	0	0	1	1	1									
x	NOT x																														
0	1																														
1	0																														
x	y	x AND y																													
0	0	0																													
0	1	0																													
1	0	0																													
1	1	1																													
OR																															
<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>x OR y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	x OR y	0	0	0	0	1	1	1	0	1	1	1	1	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>x XOR y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	x XOR y	0	0	0	0	1	1	1	0	1	1	1	0
x	y	x OR y																													
0	0	0																													
0	1	1																													
1	0	1																													
1	1	1																													
x	y	x XOR y																													
0	0	0																													
0	1	1																													
1	0	1																													
1	1	0																													

Figure 4.2 Logic operators applied to bit patterns



## 触发器flip-flops

触发器是一种逻辑电路，它产生一个输出值0或1作为两种可能的稳定状态，它保持不变，直到一个脉冲被应用到一个控制输入，并导致它转移到另一个值。

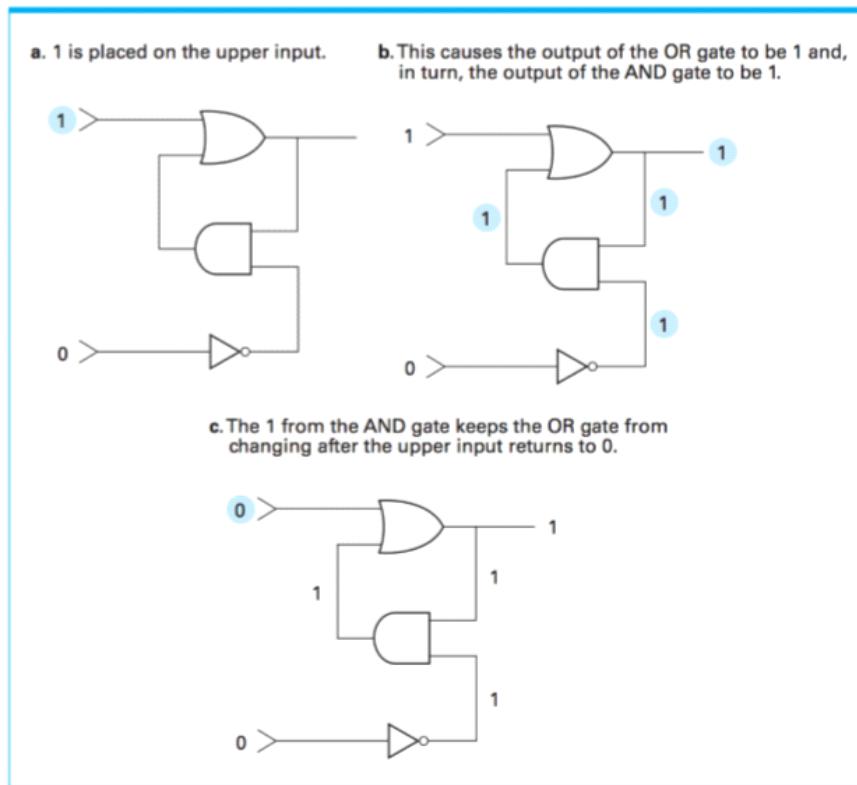
换句话说，在外界刺激的控制下，触发器的输出会在两个值之间来回切换。

触发器是计算机的基本构件

可以将触发器设置为输出值为0或1。其他电路可以通过向触发器的输入发送脉冲来调整这个值，还有一些电路可以使用触发器的输出作为输入来响应存储的值。

因此，许多作为电路构造的触发器可以在计算机内部作为一种记录数据的方法，这些数据被编码为0和1的模式。

Figure 1.4 Setting the output of a flip-flop to 1



## 表示位串

一长串bits经常被称为stream。

stream是人类难以理解的。

为了简化stream的表示，我们通常使用一种称为十六进制hexadecimal notation表示法的简写表示法，它利用了这样一个事实：机器中的位模式的长度往往是4的倍数。

特别是，十六进制表示法使用单个符号表示四位bit的模式。

## 例子

一个12位的字符串可以由3个十六进制符号表示。

16位模式1010010011001000可以简化为更合适的形式A4C8。

Bit pattern	Hexadecimal representation
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

主存main memory:内存组织memory organization

内存组织

为了存储数据，计算机包含大量的电路circuits(如触发器)，每一个都能存储一个比特。这个位存储器被称为机器的主存储器。

内存单元memory units(cell)

计算机的主存储器被组织成可管理的单元，称为单元，典型的单元大小为8位(1字节)。

虽然在计算机中没有左右之分，但我们通常认为存储单元中的位是按行排列的。

这一行的左端称为高阶端high-order end，右端称为低阶端low-order end。最左边的位称为高阶位high-order bit或最高有效位most significant bit，最右边的位称为低阶位low-order bit或最低有效位least significant bit

Figure 1.7 The organization of a byte-size memory cell



内存单元地址memory unit address

为了识别计算机主内存中的单个单元，每个单元都被分配一个唯一的“名称”，称为其地址address。

更准确地说，我们设想将所有单元格放在一行中，并按照从valu 0开始的顺序编号。

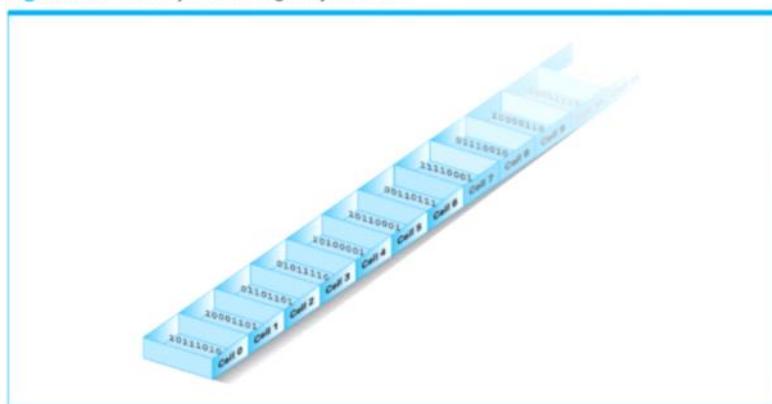
## 安排单元 ordering cells

这样的寻址系统不仅为我们提供了一种唯一地标识每个单元的方法，而且还将单元与单元关联起来，为我们提供诸如“下一个单元”或“前一个单元”之类的短语。

为主存中的单元和每个单元中的位分配顺序的一个重要结果是，计算机主存中的所有位的集合基本上是按长行顺序排列的。

因此，这一长行中的片段可以用来存储可能比单个单元长度还长的位模式。

Figure 1.8 Memory cells arranged by address



---

主存:测量存储器容量

KB

在早期的计算机中，内存的大小常常是通过测量来确定的1024(也就是 $2^{10}$ 次幂)个单元。

因为1024接近于1000，所以计算社区在引用这个单元时采用前缀kilo，术语kilobyte(缩写KB)用来指代1024字节。

MB, GB, TB

它们的误用随着内存变得更大，这个术语逐渐包括MB (MB)、GB (GB)和TB (TB)。

不幸的是，前缀kilo-、mega-等的应用是对术语的误用，因为它们已经在其他领域中用于表示一千的乘方。

一般来说，千次幂、兆次幂等术语在计算机内存环境中使用时指的是 $2^{10}$ 的幂，但在其他环境中使用时指的是千的幂。

Figure 5.3 Main memory

---

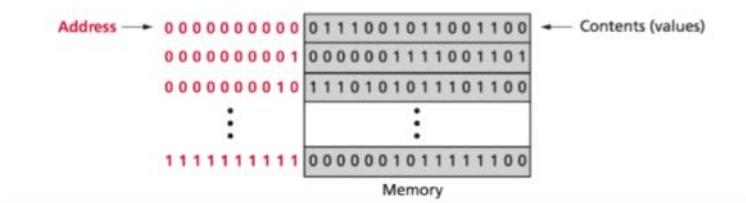


Table 5.1 Memory units

Unit	Exact Number of Bytes			Approximation	
Power of 10	Power of 2	Value of Power of 2	Prefix	Abbreviation	Derivation
$10^{-12}$			pico	p	Italian for <i>little</i>
$10^{-9}$			nano	n	Greek for <i>dwarf</i>
$10^{-6}$			micro	$\mu$	Greek for <i>small</i>
$10^{-3}$			milli	m	Latin for <i>thousandth</i>
$10^3$	$2^{10}$	1024	kilo	K	Greek for <i>thousand</i>
$10^6$	$2^{20}$	1,048,576	mega	M	Greek for <i>large</i>
$10^9$	$2^{30}$	1,073,741,824	giga	G	Greek for <i>giant</i>
$10^{12}$	$2^{40}$	not enough room	tera	T	Greek for <i>monster</i>
$10^{15}$	$2^{50}$	not enough room	peta	P	Greek prefix for <i>five</i>

主存：RAM和ROM

## RAM

随机存取存储器random access memory

(RAM)构成了计算机的大部分主存。由于计算机的主存储器被组织成独立的、可寻址的单元，这些单元可以根据需要独立地访问。

注意:术语“RAM”令人困惑，因为ROM也可以随机访问。RAM与ROM的区别在于RAM可以读写。

## ROM

除了RAM之外，大多数计算机还包含第二种存储器，称为ROM，ROM是只读存储器read-only memory的缩写。

## RAM

CPU可以将数据写入RAM，然后覆盖它。

RAM是易失性的volatile。例如：如果计算机断电，数据就丢失了。

RAM技术分为两大类:SRAM和DRAM。

### 静态存储器SRAM

静态RAM (SRAM)技术使用传统的触发器门来保存数据。这些门保持它们的状态(0或1)，这意味着只要电源是开着的，数据就会被存储，而且不需要刷新内存位置。SRAM速度快但价格昂贵。

### 动态随机存储器

动态随机存取存储器(DRAM)技术使用电容器，可以存储能量的电子设备，用于数据存储。如果电容充电，状态为1;如果它被释放，状态为0。由于电容器会随着时间的推移而失去部分电荷，因此DRAM存储单元需要周期性地重新放电。DRAM速度慢，但价格便宜。

## ROM

ROM中的内容是永久的，不能通过存储操作store operation进行更改。

ROM是非易失的non-volatile，也就是说。如果你关掉电脑，它的内容就不会丢失。

将位模式bit pattern放入ROM中称为刻录burning(在ROM制造时或在计算机部件组装时)。

---

## 大容量(二次)存储器mass (secondary) storage systems

### 大容量存储器mass storage

由于计算机主存储器的易变性和有限的容量，大多数计算机都有称为大容量存储(或二次存储)系统的附加存储设备，包括磁盘、磁带、cd、dvd和闪存驱动器。

#### 优势

更少的挥发性、更大的存储容量、更低的成本，以及将存储介质从机器中移除以用于存档的能力。

#### 缺点

需要机械运动，因此，需要更多的时间来存储和检索数据比一台机器的主存储器。

---

### 大容量存储器：磁光盘magnetic disks

#### 磁光盘:跟踪track

在磁盘中，一个薄的带磁性涂层的旋转磁盘用来存储数据。

读/写磁头被放置在磁盘的上面和/或下面，这样当磁盘旋转时，每个磁头就会穿过一个圆，称为磁道track。

通过重新定位读/写磁头，可以访问不同的同心磁道。

#### 磁光盘:圆筒cylinder

在许多情况下，磁盘存储系统由几个磁盘组成，这些磁盘安装在一个普通的主轴上，一个在另一个的上面，有足够的空间让读/写磁头在磁盘之间滑动

在这种情况下，读/写头会一致移动。每次读/写磁头重新定位时，就会有一组新的轨迹——即所谓的“圆筒cylinder”——可供使用。

#### 磁光盘:扇形sector

由于一个磁道包含的数据可能比我们在任何时候想要处理的数据要多，所以每条磁道都被划分成称为扇区的小弧，在扇区上数据被记录为连续

的位串。

磁盘上的所有扇区包含相同数量的位(典型的容量范围是512字节到几

KB), 在最简单的磁盘存储系统中, 每个磁道包含相同数量的扇区。

在大容量磁盘存储系统中, 靠近外缘的磁道能够包含比靠近中心的磁道多得多的扇区, 这种能力通常通过应用一种称为带位记录zoned-bit recording的技术来实现。

## 磁盘格式化formatting

磁道和扇区的位置不是磁盘物理结构的永久部分。相反, 它们通过一个称为磁盘格式化(或初始化initializing)的过程进行磁性标记。

Figure 1.9 A disk storage system

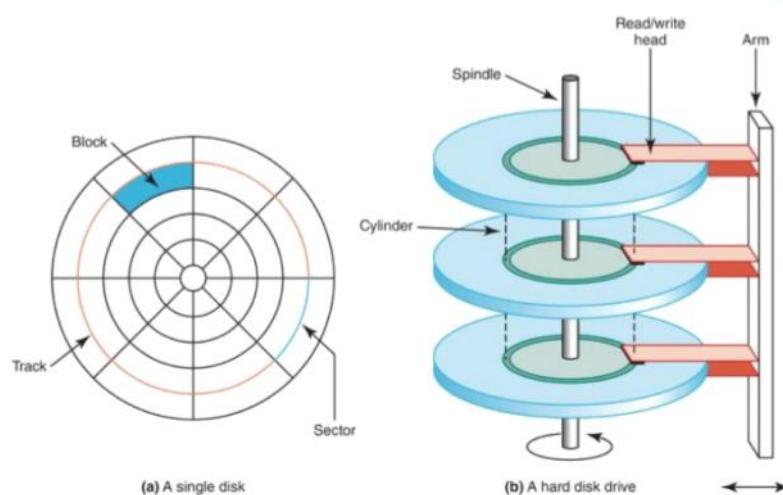
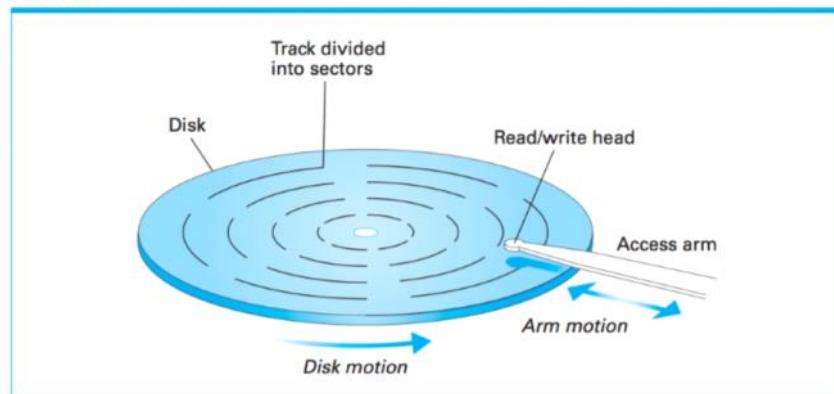
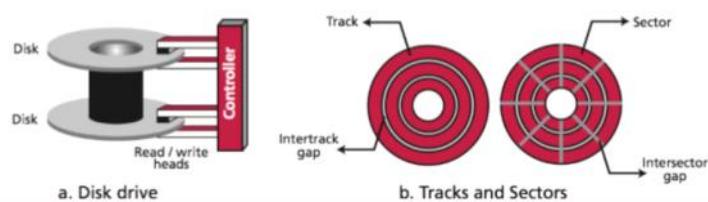


Figure 5.6 A magnetic disk



## 评估磁盘系统性能的度量

(1)寻道时间seek time(将读/写磁头从一个磁道移动到另一个磁道所需的时间)

(2)旋转延迟或延迟时间rotation delay or latency time(磁盘完成旋转所需时间的一半, 即所需要的数据在读写磁头被定位到所需要的磁道上后旋转到读写磁头所需的平均时间)

- (3)访问时间access time(寻道时间和旋转延迟之和)
  - (4)传输速率transfer rate(数据从磁盘到磁盘的传输速率)。
- 

大容量存储器：光盘存储系统compact disk storage system

光盘(CD)

光盘的直径是12厘米(约5英寸)，由反光材料覆盖着一层透明的保护涂层。

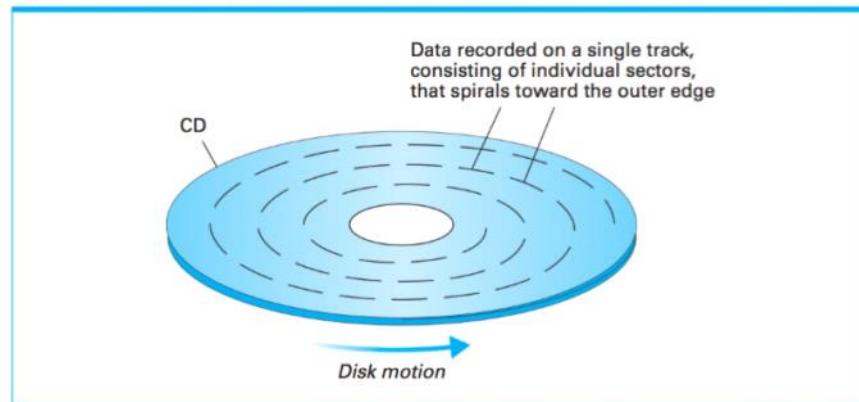
通过在它们的反射表面产生变化来记录数据。

当光盘旋转时，激光束可以探测到光盘反射表面的不均匀性，然后利用激光束获取这些数据。

CD-DA

光盘的CD技术最初应用于使用CD- DA(激光唱碟数字音频compact disk-digital audio)记录格式的音频记录，今天用于计算机数据存储的CD基本上使用相同的格式。

**Figure 1.11** CD storage format



---

大容量存储器:闪存flash memory

物理运动导致速度慢

基于磁或光学技术的大规模存储系统的一个共同特性是需要物理运动来存储和检索数据，例如旋转磁盘、移动读写头和瞄准激光束。这意味着数据存储和检索的速度比电子电路慢

闪存

在闪存系统中，比特是通过直接向存储介质发送电子信号来存储的，在存储介质中，电子被困在二氧化硅的小腔中，从而改变了小型电子电路的特性。

由于这些电子室能够保存它们的俘获电子很多年，因此这种技术适合离线存储数据。

---

文件存储和检索file storage and retrieval

## 文件file

文件是相关数据的命名集合。一个文件可以看作是位、字节、行或记录的序列，这取决于您如何看待它。从用户的角度来看，文件是可以写入辅助内存的最小数据量。将所有内容组织到文件中为数据存储提供了统一的视图。

## 文件系统file system

文件系统是操作系统提供的逻辑视图，以便用户可以将数据作为文件集合来管理。文件系统通常是通过将文件分组到目录中来组织的。

## 物理记录physical record

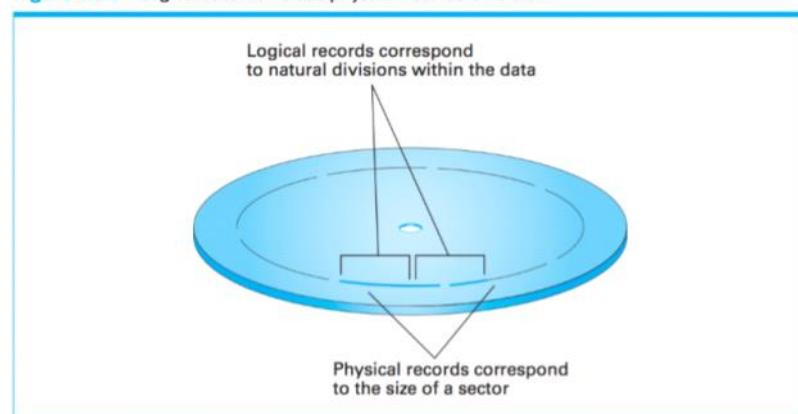
存储在磁盘上的文件必须由扇区操作，扇区的大小是预先确定的。符合存储设备特定特性的数据块称为物理记录。因此，存储在大容量存储器中的大文件通常包含许多物理记录。

## 逻辑记录logical record

与物理记录的这种划分不同，文件通常具有由所表示的数据决定的自然划分。

这些自然产生的数据块称为逻辑记录。

Figure 1.12 Logical records versus physical records on a disk



## 字段field

逻辑记录通常由称为字段field的较小单元组成。

## 键字段key field和键key

有时，通过记录中的特定字段，可以唯一地标识文件中的每个逻辑记录。这样的标识字段称为键字段。

键字段中保存的值称为键。

---

## 文件系统

一个文件可以看作是位、字节、行或记录的序列，这取决于您如何看待它。

文件的创建者决定文件中的数据如何组织，文件的任何用户都必须理解该组织。

文件：一个命名的数据集合，用于组织辅助内存

文件系统：操作系统管理的文件的逻辑视图

目录directory：一个命名的文件组

文件类型

所有文件可以大致分为文本文件或二进制文件

文本文件text files

在文本文件中，数据字节被组织为ASCII或Unicode字符集中的字符。

一个包含字符的文件

二进制文件binary files

在二进制文件中，数据字节按位模式组织。二进制文件需要基于文件中的数据对位进行特定的解释。

以特定格式包含数据的文件，需要对其位进行特殊解释

请注意术语“文本文件”和“二进制文件”有点误导人。它们似乎暗示文本文件中的信息不是以二进制数据的形式存储的。

一个重要的f

最终，计算机上的所有数据都存储为二进制数字。这些术语指的是这些位的格式:8位或16位bit的块chunk，解释为字符或其他特殊格式。

---

# Computer Architecture

2019年11月11日 10:21

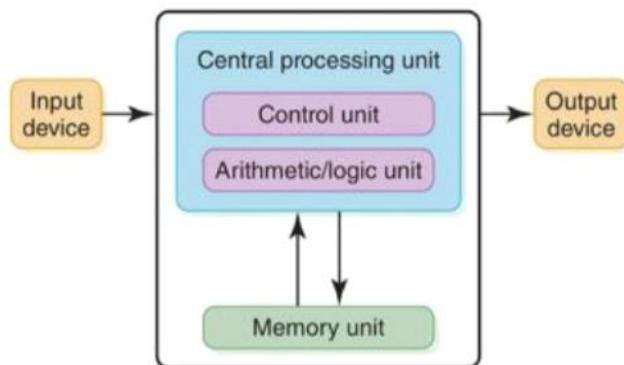
## 计算机功能结构

定义：

(数字) 计算机系统的一般规格，包括从编程(用户)的角度描述指令集和用户界面、内存组织和寻址、I/O操作和控制等。

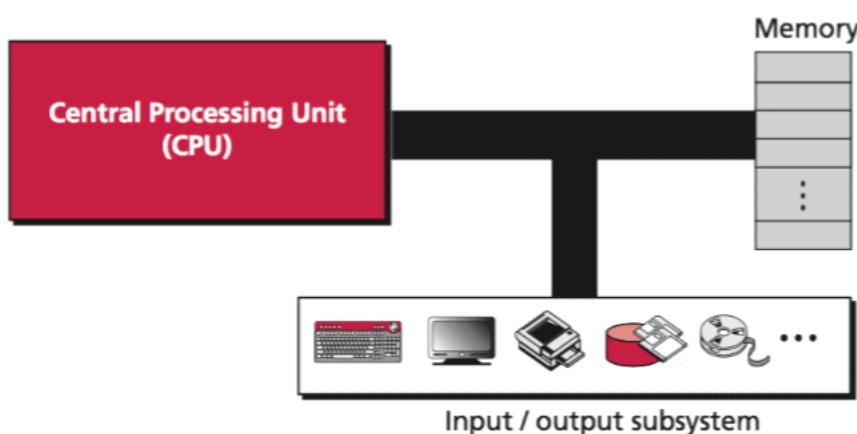
IEEE：一个系统或组件的组织结构。

Von Neumann architecture 冯诺依曼体系结构

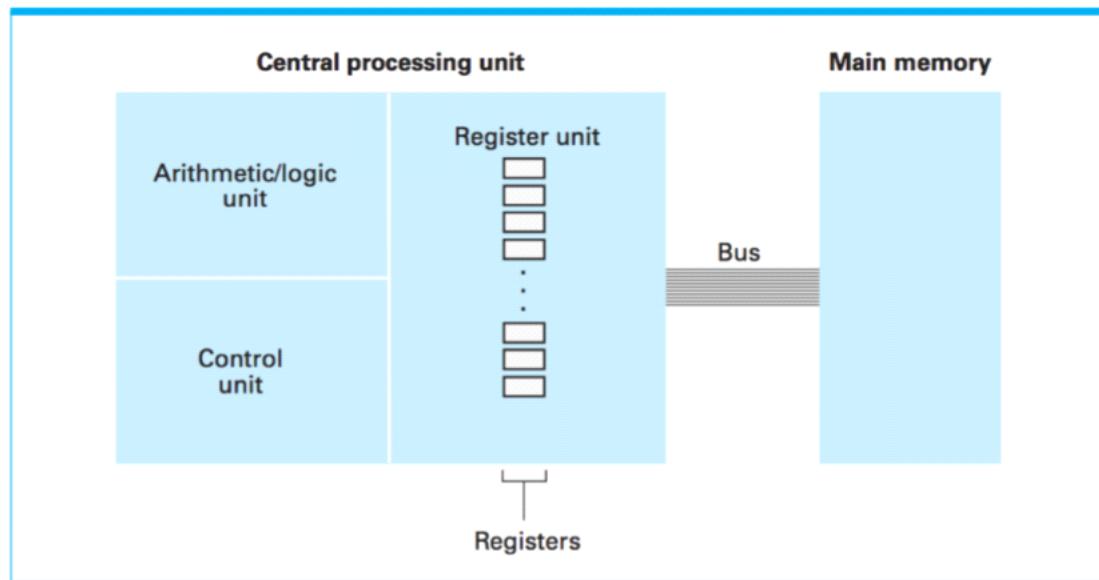


输入设备 → 控制单元 → 输出设备  
算法/逻辑单元  
↓↑  
存储单元

Central processing unit: 中心控制系统CPU: 包括了控制单元和逻辑/算法单元

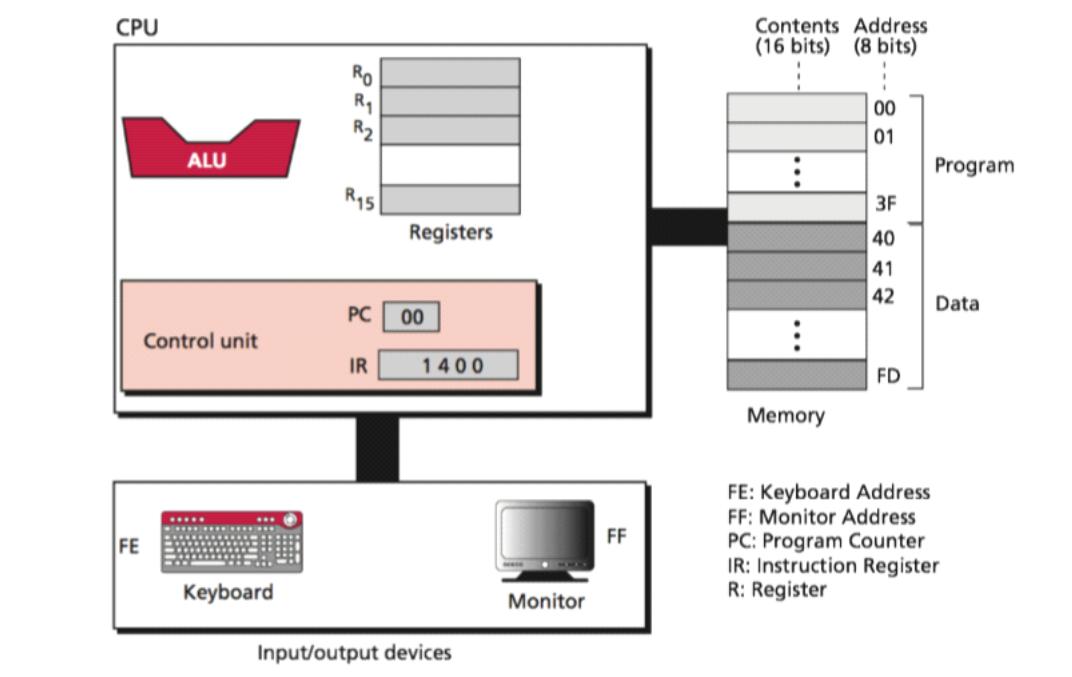


**Figure 2.1** CPU and main memory connected via a bus



导线: bus

**Figure 5.30** The components of a simple computer



重要的转折点：数据和指令在逻辑上是相同的，可以存储在相同的位置

冯诺依曼体系结构的另一个主要特征是处理数据的单元（包括控制单元和算法/逻辑单元）

与存储数据的单元是分离的

该体系的构建者并不是冯诺依曼

存储-程序概念 stored-program concept

程序 (program) 和数据一样可以编码并存在主存 (main memory) 中

如果控制单元的设计目的是从内存中提取程序，解码指令，并执行它们，那么只需改变计算机内存的内容，而不是重新连接CPU，就可以改变计算机所执行的程序。

“冯·诺伊曼”架构这种特性导致了冯·诺依曼架构的以下五个组成部分以确保所有其他部件协调工作:

存储数据和指令的存储单元

能够执行算术的算术/逻辑单元

以及对数据的逻辑操作将外界数据输入计算机的输入装置

将结果从计算机内部传送到外部世界的输出装置

作为舞台管理器的控制单元

---

存储单元memory unit

存储单元是每个小单元 (cells) 的集合，每个单元都有一个惟一的物理地址。

可寻址能力addressability: 每一个在存储中可寻址的位数的数量

这里我们使用的是通用单元 (cell) 格，而不是字节 (byte) 或字 (word)，因为每个可寻址位置的位数 (bits)。称为存储器的可寻址性，在不同的机器之间是不同的。

今天，大多数计算机是字节 (byte) 可寻址的

算术/逻辑单元ALU

该算术逻辑单元能够执行基本的算术运算，如加、减、乘、除两个数字

该单元还能够执行诸如AND、OR和NOT之类的逻辑操作。

单词长度

计算机的字长 (word length) 是ALU一次处理的位数 (bit) 。

算术/逻辑单元(ALU)执行算术运算(加、减、乘、除)和逻辑运算(两个值的比较)的计算机部件

输入单元input unit

输入单元是一种将外界的数据和程序输入计算机的装置。

第一个输入单元解释纸带或卡片上的穿孔。

现代的输入设备包括键盘、鼠标和超市里使用的扫描设备。

输入单元：接受存储在内存中的数据的设备

输出单元output unit

输出单元是一种将存储在计算机内存中的结果提供给外部世界的设备。

最常见的输出设备是打印机和显示器。

输出设备：打印或以其他方式显示存储在内存中的数据，或对存储在内存或其他设备中的信息进行永久复制的设备

控制单元control unit

控制单元是计算机中的组织力量，它负责取指-执行循环。

在控制单元中有两个特殊寄存器。

指令寄存器 (instruction register) (IR)包含正在执行的指令

程序计数器 (program counter) (PC)包含将要执行的下一条指令的地址

## 寄存器register

大多数现代ALU都有少量的特殊存储单元，称为寄存器。

这些寄存器包含一个单词，用于存储立即需要的数据。

### 通用或专用寄存器寄存器

寄存器单元中的一些寄存器被认为是通用寄存器 (general-purpose registers)，而另一些则是专用寄存器 (special-purpose registers)

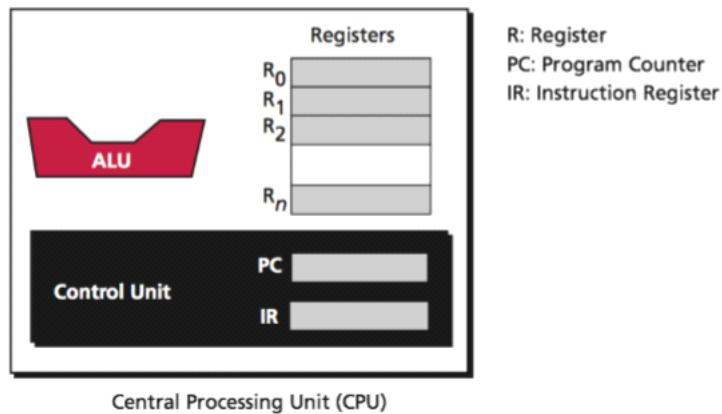
寄存器：在CPU中寄存的一个小的存储区域，用来存储中间值或特殊数据

## 中央处理器CPU：

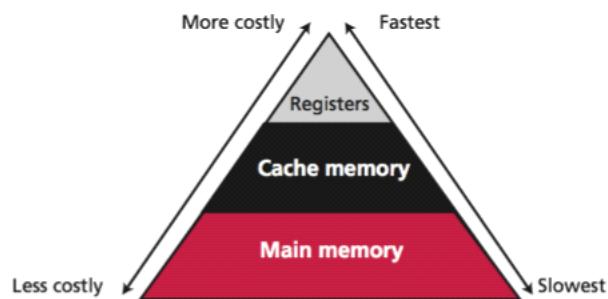
中央处理器由于ALU和控制单元紧密地工作在一起，它们通常被认为是一个称为中央处理单元的单元，或称为CPU(通常仅指处理器)。

中央处理器，算术/逻辑单元和控制单元的组合;解释和执行指令的计算机的“大脑”

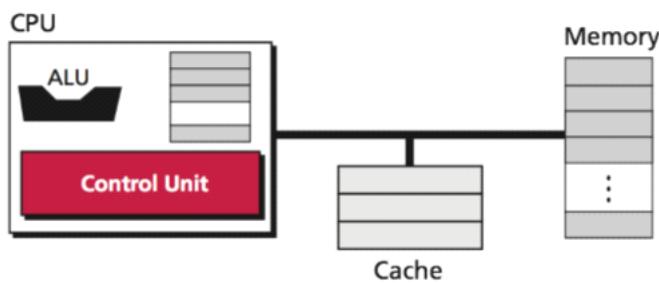
**Figure 5.2 Central processing unit (CPU)**



**Figure 5.4 Memory hierarchy**



**Figure 5.5 Cache memory**



从主存到寄存器，调取数据速度越来越快，花费越来越高

高速缓冲存储器cache memory

Figure 5.12 Connecting CPU and memory using three buses

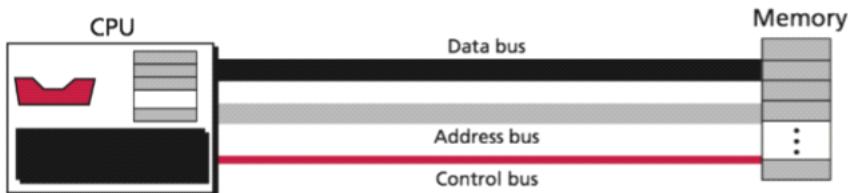


Figure 5.13 Connecting I/O devices to the buses

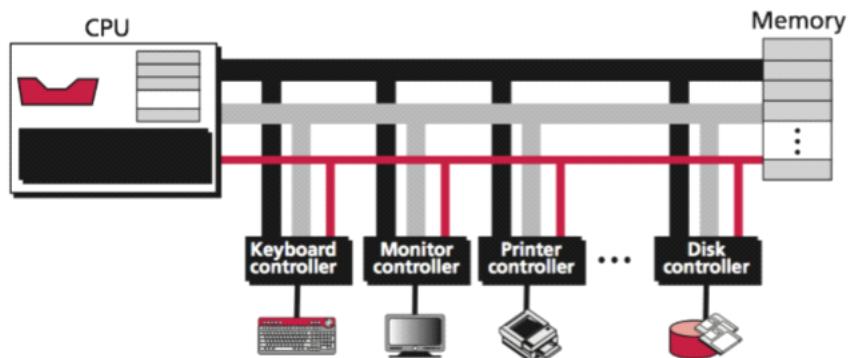


Figure 5.14 SCSI controller

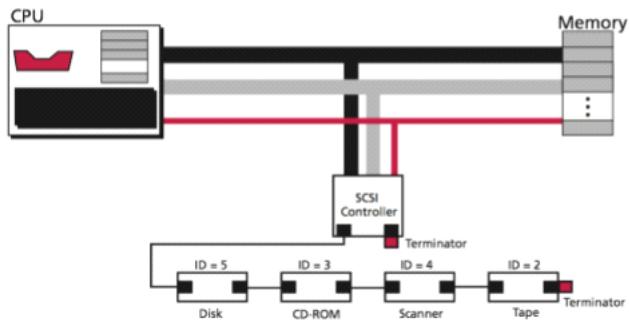


Figure 5.16 USB controller

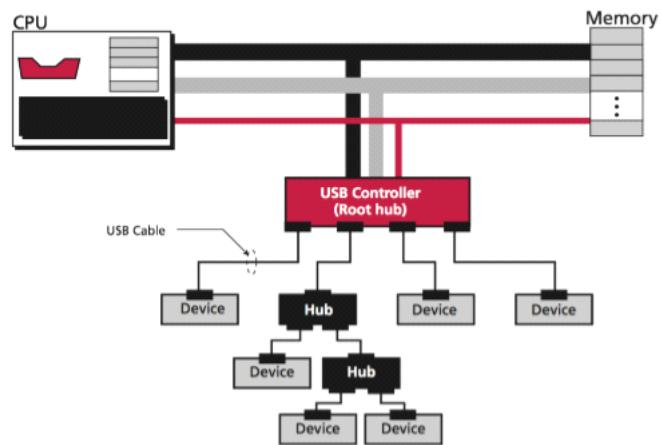
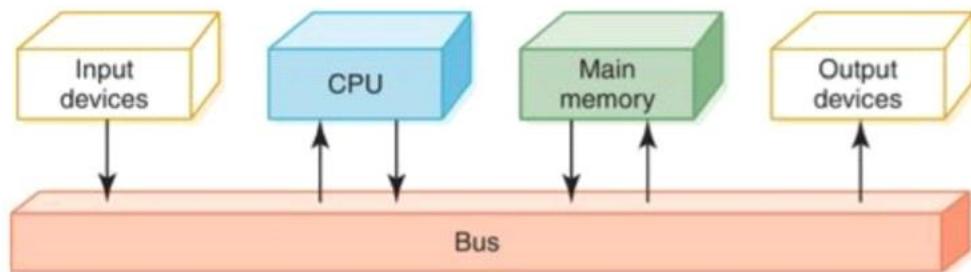
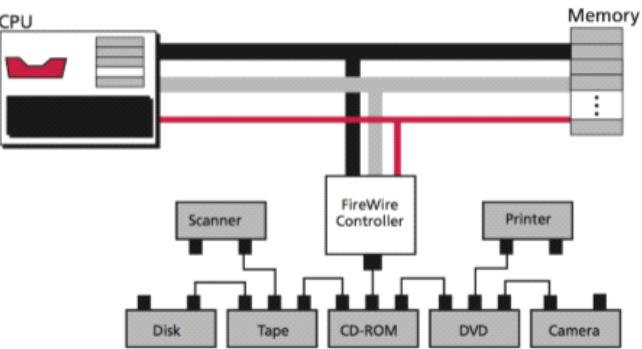


Figure 5.15 FireWire controller



**FIGURE 5.2** Data flow through a von Neumann machine

取指-执行fetch-execute cycle

周期指令和数据的可寻址性

计算机底层的存储程序原理意味着指令和数据都是可寻址 (addressable) 的。

指令存储在连续的存储器中;要操作的数据被存储在主存的不同部分。

fetch-execute cycle

为了开始取指-执行循环, 第一个指令的地址被加载到程序计数器中。

处理循环包括四个步骤:

在控制单元中

(1)从主存中获取下一条指令;

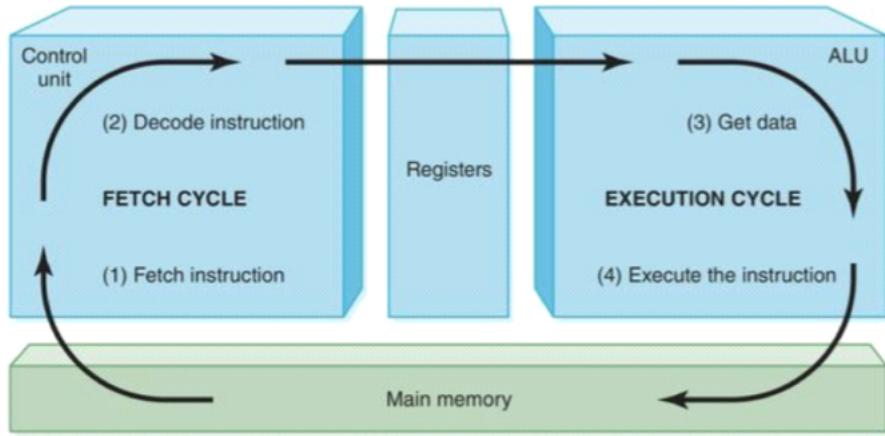
(2)解码指令;

放在寄存器在传上ALU

(3)根据需要获取数据 (如果需要修改指令, 则需要取数据, 否则不需要)

(4)执行指令。

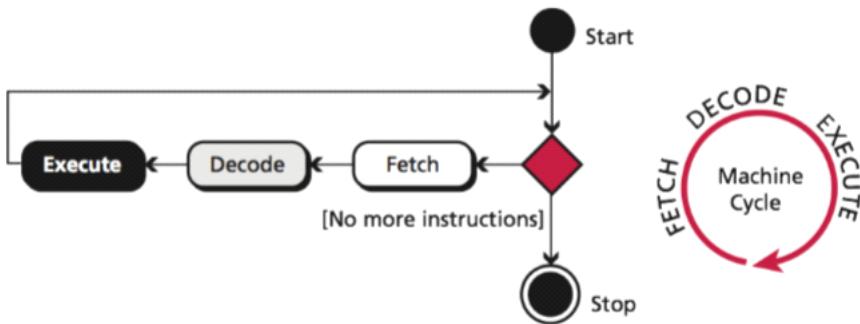
将通过指令完成的步骤存储回主存中



**FIGURE 5.3** The fetch–execute cycle

分为三个步骤：Fetch获取 Decode解码 Execute执行

**Figure 5.19** The steps of a cycle



### 获取下一条指令 Fetch

程序计数器(PC)包含要执行的下一条指令的地址，因此控制单元进入PC中指定的内存地址，复制内容，并将副本放入指令寄存器。

此时，IR包含要执行的指令。

在进入循环的下一个步骤之前，必须更新PC以保证当前指令完成时要执行的下一条指令的地址。

由于指令在内存中是连续存储的，所以将当前指令的字节数添加到程序计数器中应该会将下一条指令的地址放入PC中。

因此，控制单元增加PC。

这是可能的，PC可能会改变后，正在执行的指令。

### 执行指令 Execute

一旦一条指令被解码，任何操作数(数据)被获取，控制单元就准备好执行指令。

执行包括向算术/逻辑单元发送信号以进行处理。

在向寄存器添加数字的情况下，操作被发送到ALU并添加到寄存器的内容中。

下一个周期执行完成后，循环再次开始。

如果最后一条指令是向:register的内容添加一个值，那么下一条指令可能是将结果存储到内存中的某个位置。

然而，下一条指令可能是控制指令——也就是说，一条指令询问关于最后一条指令的结果的问题，并且可能改变程序计数器program counter的内容。

---

把一个值添加到主存中的步骤

1. 得到一个将要放在主存里的值，并将其放入寄存器中
  2. 得到另一个将要放在主存里的值，并将其放在另一个寄存器中。  
**Q. 这里的寄存器是在ALU还是在控制单元里面的？？？**
  3. 在ALU中激活加法指令使用步骤1和步骤2中使用的寄存器作为输入和另一个寄存器我要去获得结果。
  4. 将结果存储在内存中。
  5. 停止。q
- 

机器指令 Machine instruction

机每台计算机必须有一组定义好的机器指令。

为了应用存储程序的概念，CPU被设计成识别编码bit模式的机器指令。

CPU必须能够解码和执行的机器指令集非常短。

事实上，一旦一台机器可以执行某些基本的但经过精心挑选的任务，增加更多的特性并不会增加机器的理论能力。

---

机器语言 Machine language

一系列指令和编码系统被称为机器语言。

机器语言由计算机直接使用的二进制编码指令组成的语言

---

两种CPU架构的哲学观点

一个是CPU应该被设计来执行一组最少的机器指令

另一个是CPU应该被设计来执行大量复杂的指令，即使其中许多指令在技术上是冗余的。

RISC体系结构：如inter处理器，AMD处理器

第一种方法是所谓的简化指令集计算机(RISC)。

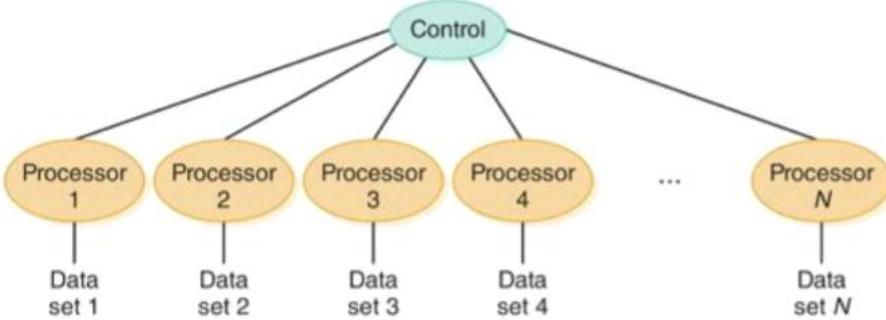
支持RISC架构的论点是，这样的机器效率高、速度快，而且制造成本更低。

CISC体系结构：如PowerPC处理器，ARM处理器

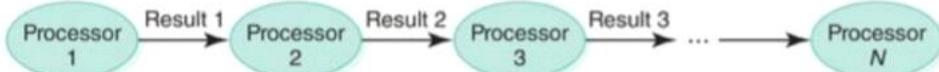
第二种方法是所谓的复杂指令集计算机(CISC)。

支持CISC体系结构的论据是，更复杂的CPU可以更好地处理当今软件不断增加的复杂性。

使用CISC，程序可以利用一组功能强大的指令，其中许多指令在RISC设计中需要多指令序列。

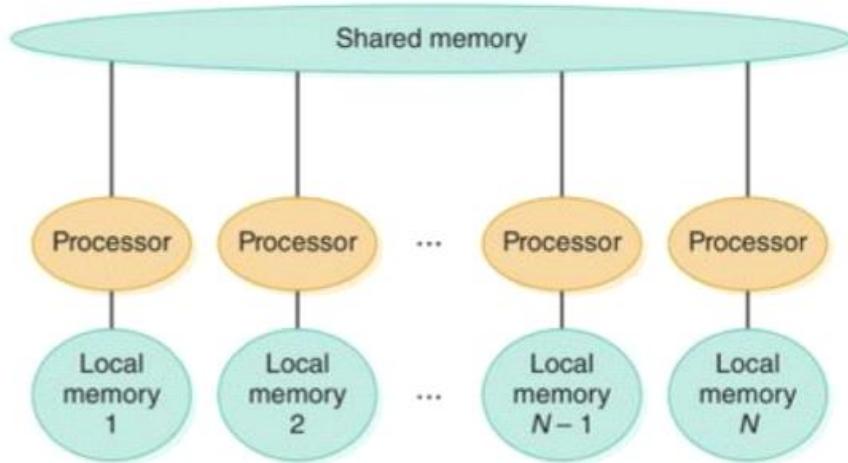


**FIGURE 5.8** Processors in a synchronous computing environment



**FIGURE 5.9** Processors in a pipeline

同时处理型和直线型



**FIGURE 5.10** A shared-memory parallel processor

---

计算机组织分类

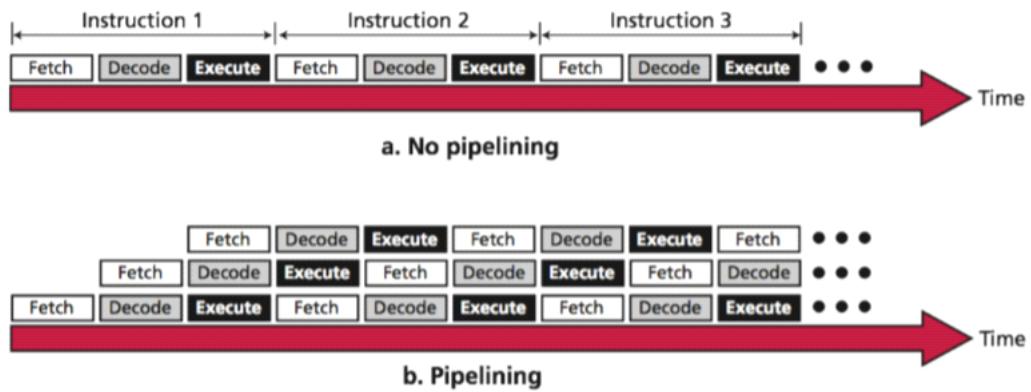
SISD单指令流，单数据流

SIMD单指令流，多个数据流

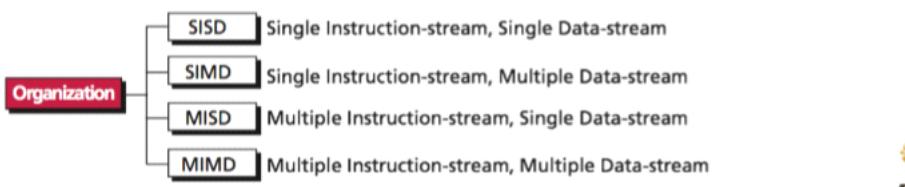
MISD多个指令流，单个数据流

MIMD多指令流，多数据流

**Figure 5.24 Pipelining**



**Figure 5.25 A taxonomy of computer organization**



# Data manipulation in Computer Systems

2019年11月14日 21:32

## 计算机系统中的数据处理

---

### 三组机器指令instruction

无论选择RISC还是CISC，机器指令都可以分为三组：

- (1) 数据传输组data transfer group
- (2) 算术/逻辑组arithmetic/logic group
- (3) 控制组control group

### 数据传送指令data transfer instruction

数据传输(实际上是一个误称)由要求将数据从一个位置移动到另一个位置的指令组成。

传输指令所涉及的过程更像是复制数据，而不是移动数据。因此，诸如复制或克隆之类的术语更好地描述了这组指令的操作。

---

### 数据传送指令

特殊术语special terms是指CPU和主存之间的数据传输。用存储单元的内容来填充通用寄存器的请求通常称为加载指令LOAD;相反，将寄存器的内容传输到内存单元的请求称为存储指令STORE。

数据传输类别中的一组重要指令包括与cpu主内存上下文之外的设备通信的命令(打印机、键盘、显示屏、磁盘驱动器等)。

由于这些指令处理机器的输入/输出(I/O)活动，因此它们被称为I/O指令(I/O instructions)，有时也被视为一个单独的类别。我们将把I/O指令视为数据传输组的一部分

---

### 算术/逻辑指令

算术/逻辑组由指令组成，指令告诉控制单元在算术/逻辑单元中请求一个活动。

算术/逻辑单元能够执行基本算术操作basic arithmetic之外的操作。这些附加操作中的一些是布尔操作Boolean operation, AND, OR 和 XOR。

在大多数算术/逻辑单元中可用的另一个操作集合允许寄存器的内容在寄存器内向左或向右移动。这些操作被称为移位SHIFT或旋转ROTATE操作。

---

### 控制指令

控制组由那些指导程序执行而不是操纵数据的指令组成。

这个组包含许多有趣的指令，例如跳转JUMP(或分支BRANCH)指令，用于指示CPU执行列表中的下一条指令以外的另一条指令。

电脑里的程序不一定都是按顺序进行的  
这些跳转指令有两种:无条件跳转和条件跳转unconditional jumps  
conditional jumps.

---

## 除法

- 1.从主存中加载LOAD数据进入寄存器
  - 2.从主存中加载另一个数据进入另一个寄存器
  - 3.如果第二个数据的值是0, 那么跳到第六步
  - 4.将第一个寄存器的内容除以第二个寄存器, 将结果保留在第三个寄存器中。
  - 5.把在第三个寄存器中的计算结果存储STORE到主存里
  - 6.结束运行
- 

## 体系结构

它有16个通用寄存器和256个主存储器单元cells, 每个单元的容量为8位bits。

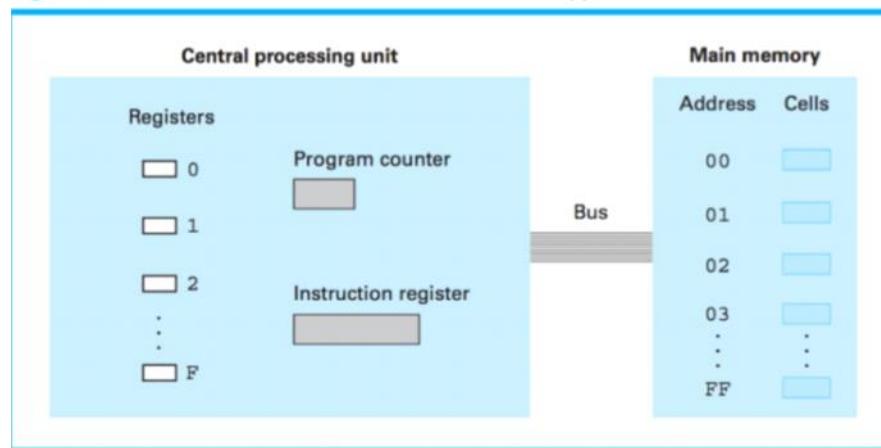
## 标签和地址labels and addresses

我们用值0到15来标记寄存器, 用值0到255来处理内存单元。

为了方便起见, 我们将这些标签和地址看作以2为基底的值, 并使用十六进制表示法压缩得到的位模式。

一个十六进制的数字代表了4个bit

Figure 2.4 The architecture of the machine described in Appendix C



---

## 机器指令的编码版本

机器指令的编码版本由两部分组成:op-code (operation code的缩写)字段和operand字段。

## 机器指令的位模式bit pattern

bit pattern出现在op-code字段中的位模式指示哪些基本操作, 如存储STORE, 移位SHIFT、异或和EOR、跳转JUMP是由指令请求的。

在operand字段中发现的位模式提供了关于op-code指定的操作的更详细的信息。

每条指令都使用16位编码。由四个十六进制数字表示。

一个指令有两个byte，十六个bit

## 机器指令

每条机器指令有两个字节bytes长。

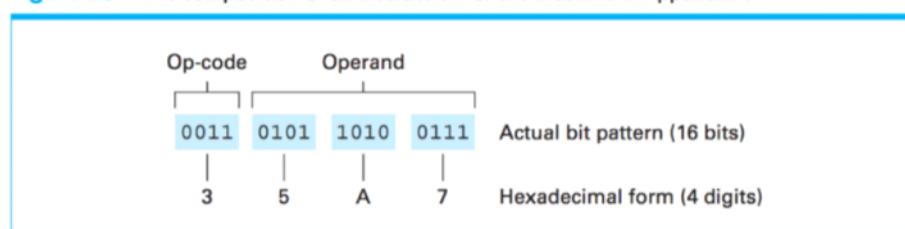
前4位bits提供操作码op-code;最后12位组成操作数operand。

下表以十六进制表示法列出了指令，并对每个指令进行了简短描述。

在表示寄存器标识符register identifier的字段中，用字母R、S和T代替十六进制数字，寄存器标识符根据指令的特定应用而变化。

在不表示寄存器的变量字段中，用X和Y代替十六进制数字

**Figure 2.5** The composition of an instruction for the machine in Appendix C



### Op-code    Operand

1            RXY

2            RXY

3            RXY

4            ORS

5            RST

6            RST

7 RST  
8 RST  
9 RST  
A ROX  
B RXY  
C 000

1.用在地址为XY的内存单元中找到的位模式加载寄存器R。

例如:14A3将导致位于地址A3的存储单元的内容被放到寄存器4中。

2.用位模式XY加载寄存器R。

例如:20A3将导致值A3被放在寄存器0中。

3.将在寄存器R中找到的位模式存储在地址为XY的存储单元中。

例如:35B1将导致寄存器5的内容被放置在地址为B1的内存单元中。

4.将在寄存器R中找到的位模式移动到寄存器S中。

例如:40A4会导致寄存器A的内容被复制到寄存器4中。

5.在寄存器S和T中添加位模式，就好像它们是两个补码表示一样，然后将结果留在寄存器R中。

例如:5726将导致寄存器2和寄存器6中的二进制值被添加，而寄存器7中的和被添加。

6.在寄存器S和T中添加位模式，就好像它们用浮点表示法表示值一样，而在寄存器R中保留浮点结果。

例如:634E将使寄存器4和E中的值作为浮点值添加，结果将放在寄存器3中。

7.或者是寄存器S和T中的位模式，并将结果放在寄存器R中。

例如:7CB4将导致将寄存器B和4的内容的ORing的结果放在寄存器C中。

8.以及S和T寄存器中的位模式，并将结果放在R寄存器中。

例如:8045将导致寄存器4和寄存器5的内容的结束结果放在寄存器0中。

9.互斥或寄存器S和T中的位模式，并将结果放在寄存器R中。

例如:95F3将导致独占寄存器F和3内容的结果被放到寄存器5中。

10.将寄存器R中的位模式向右旋转一位x次。每次将从低阶端开始的位放到高阶端。

例如:A403将导致寄存器4的内容以循环方式向右旋转3位。

11.如果寄存器R中的位模式等于寄存器0中的位模式，则跳转到位于地址

XY的内存单元中的指令。否则，继续按正常顺序操作。(跳转是通过在执行阶段将XY复制到程序计数器来实现的。)

例如:B43C将首先比较寄存器4的内容和寄存器0的内容。如果这两个相等，那么模式3C将被放置在程序计数器中，以便下一个执行的指令将是位于该内存地址的指令。否则，什么也不会做，程序执行将按正常顺序继续。

## 12.停止执行

例如:C000会导致程序执行停止。

这些不同的指令也可以表明，存储和寄存器中既可以存储数据，又可以存储指令，同时还可以对指令进行修改

### 译码指令的操作数域

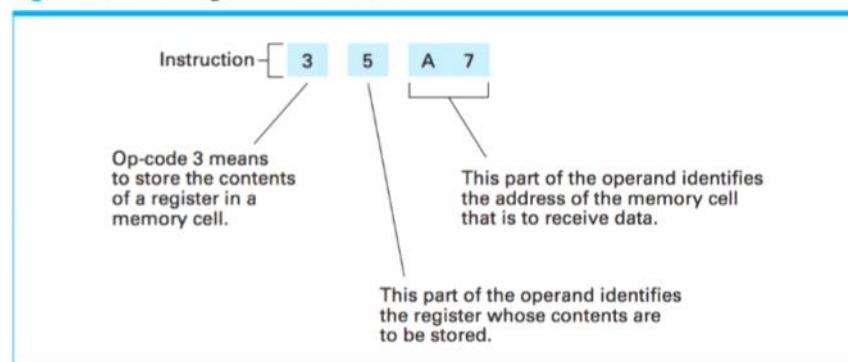
每条指令的操作数字段由三个十六进制数字(12位bits)组成，在每种情况下都阐明了运算码op-code给出的一般指令。

如图2.6

如果一条指令的第一个十六进制数字是3(存储寄存器内容的运算码)，则指令的下一个十六进制数字将指示要存储哪个寄存器，最后两个十六进制数字将指示要接收数据的存储单元。

因此，指令0011010110100111(位模式)/35A7(十六进制)转换成语句“将寄存器5中找到的位模式存储在地址为A7的内存单元中”。

Figure 2.6 Decoding the instruction 35A7



注意图片中的定冠词

3对应的解释是放在a register 和 a memory cell，意思是某一个寄存器或者某一个存储单元

而5和A7对应的解释是the register, the memory cell，意思是特定的一个寄存器或者存储单元

### 示例程序(图2.7)

该程序从主存储器中检索两个值，计算它们的和，并将它们存储在主存储器单元中。

把程序存入存储器我们首先需要把程序放在内存中的某个地方。

我们假设程序存储在从地址A0(十六进制)开始的连续地址中。使用这种方式存储的程序，我们可以通过在程序计数器中放置第一条指令的地址(A0)并启动机器来执行它。

Figure 2.7 An encoded version of the instructions in Figure 2.2

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

Figure 2.8 The machine cycle

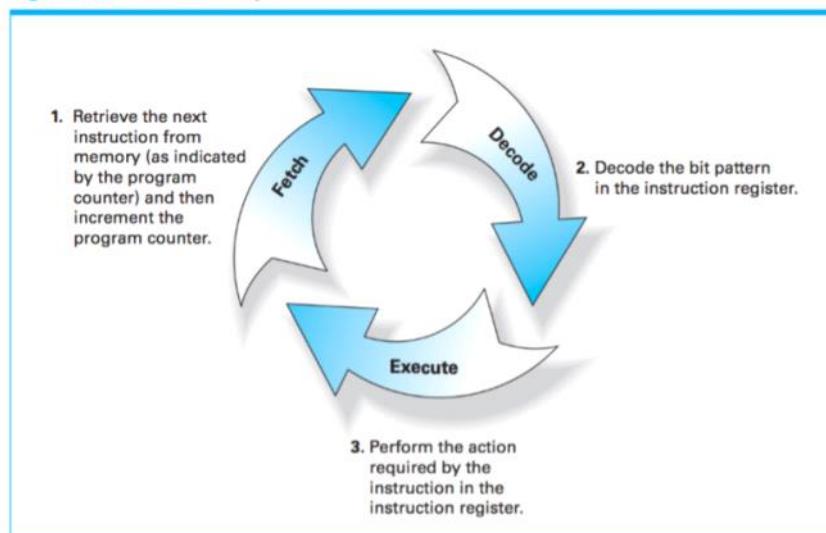
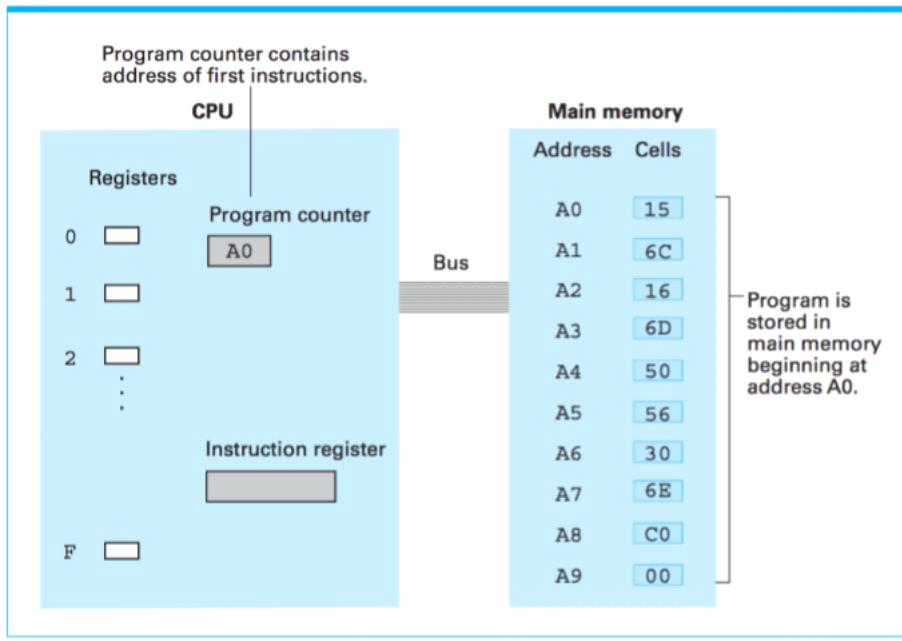


Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution



### 执行第一个机器周期的获取步骤

CPU通过提取存储在主存位置A0上的指令，并将这条指令(156C)放在它的Instruction寄存器中，开始机器周期的获取步骤。

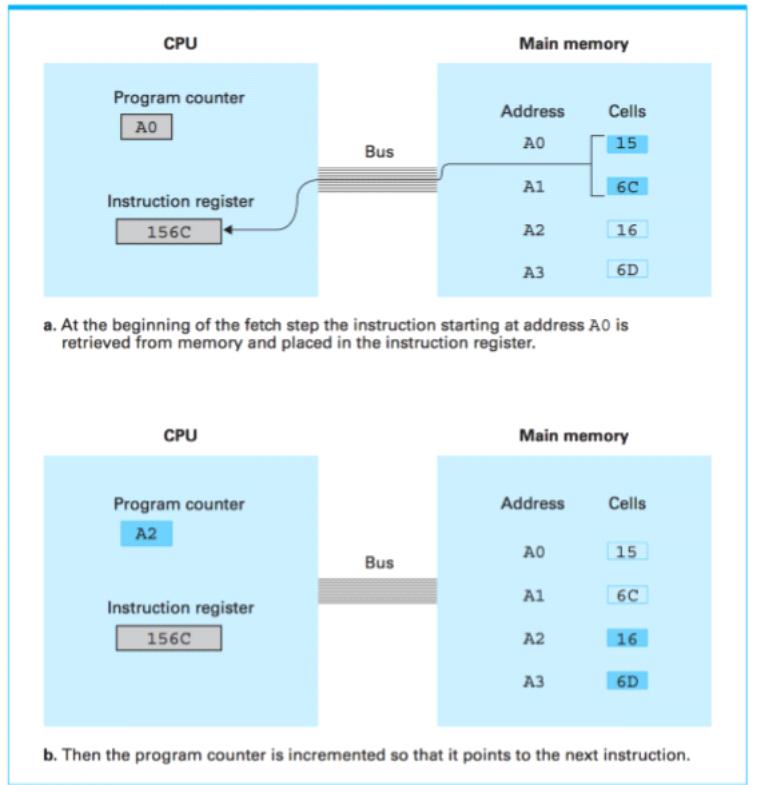
因此，要获取的整个指令占用了地址A0和A1的存储单元。

CPU的设计考虑了这一点，因此它检索两个单元的内容，并将接收到的位模式放入16位长的指令寄存器中。

然后CPU向程序计数器中添加两个，以便该注册表包含下一条指令的地址。

程序计数器主要要看加载的指令的长度，比如这里加载的是16个bit，用十六进制表示为4个数字，每个cell里面有两个数字，所以以此加载要加载两个，即A0, A1是同时加载进去的，但是计数器里面显示的是A0，如果执行第二条指令，该从A2开始，加载A2, A3，以此类推  
不同指令的长度导致程序计数器的递增数是不一样的

Figure 2.11 Performing the fetch step of the machine cycle



### 执行第一个机器周期的解码步骤

CPU分析其指令寄存器中的指令，并得出结论，它将在地址6C处加载寄存器5的内存单元内容。

### 执行第一个机器周期的执行步骤

load活动在机器周期的执行步骤中执行，然后CPU开始下一个周期。

### 执行第二个机器周期

第二个周期从从地址A2开始的两个内存单元获取指令166D开始。CPU将这条指令放入指令寄存器，并将程序计数器增加到A4。

### 执行第二个机器周期

在执行第二个周期的取出步骤后，程序计数器和指令寄存器中的值就变成了下面的值:程序计数器:A4指令寄存器:166D

现在CPU对指令166D进行解码，并决定用内存地址6D的内容加载寄存器6。

然后CPU执行指令。此时，寄存器6实际上是被加载的。

### 执行第三个机器周期

因为程序计数器现在包含了A4，所以CPU从这个地址开始提取下一条指令。

结果是5056被放在指令寄存器中，程序计数器增加到A6。

CPU现在对其指令寄存器的内容进行解码，并通过激活寄存器5和寄存器6的两个补码加法电路来执行它。

在这个执行步骤中，算术/逻辑单元执行请求的加法，将结果保留在寄存

器0中(根据控制单元的请求), 并向控制单元报告它已经完成。

然后CPU开始下一个机器周期。

#### 执行第四次机器循环

CPU从从内存位置A6开始的两个内存单元获取下一条指令(306E), 并将程序计数器增加到A8。

然后解码并执行该指令。此时, sum位于内存位置6E。

#### 执行第五次机器循环

从内存位置A8开始获取下一条指令, 并将程序计数器增加到AA。

指令寄存器(C000)的内容现在被解码为halt指令。

因此, 机器在执行步骤期间停止。的机器循环, 程序就完成了。

---

#### 位串的逻辑运算logic operations on bit strings

将两个输入位组合起来产生单个输出位的逻辑操作AND、OR、和XOR  
(exclusive OR)可以扩展到将两个位串组合起来产生单个输出串的操作, 方法是将基本操作应用到单个列上。

例如

$$\begin{array}{r} \begin{array}{r} 10011010 \\ \underline{\text{AND}} \quad 11001001 \\ 10001000 \end{array} & \begin{array}{r} 10011010 \\ \underline{\text{OR}} \quad 11001001 \\ 11011011 \end{array} & \begin{array}{r} 10011010 \\ \underline{\text{XOR}} \quad 11001001 \\ 01010011 \end{array} \end{array}$$

---

#### 屏蔽Masking:一个例子

在不知道第二个操作数的内容的情况下, 我们仍然可以得出这样的结论: 结果中最重要的四位是0。此外, 结果中最不重要的四位将是第二个操作数的那部分的副本。

#### 屏蔽

AND操作的主要用途之一是将操作系统放置在位模式的一部分中, 而不会干扰另一部分。

和操作的使用是一个称为掩码masking的过程的示例:一个操作数(称为掩码mask)确定另一个操作数的哪一部分将影响结果。

---

#### 在操作位图中使用和AND操作

位图是一个位串, 每个位代表一个特定对象的存在或不存在。

假设一个存储单元中的8位被用作一个位图, 我们想要找出与来自高阶末端的第三位相关联的对象是否存在。

我们只需要用掩码00100000和整个字节, 当且仅当位图的高阶末端的第

三位自身为0时，掩码00100000生成所有操作系统的一个字节。  
此外，如果第三位从高阶的位图1，我们想将其更改为0没有扰乱其他部分，  
我们可以和一些地图面具11011111，然后将结果存在原始位图中  
AND两个都是1的才是1，除此之外都是0

使用或OR操作复制位字符串的一部分

在将操作系统放置在非复制部分时，可以使用AND操作复制位串的一部分，而在将1放置在非复制部分时，可以使用OR操作复制位串的一部分。为此，我们再次使用掩码，但这一次我们使用0s来指示要复制的位位置，并使用1s来指示非复制的位置。

例如，对于任何带有11110000的字节，其结果中最有效的4位是1，而其余的4位是另一个操作数的最低有效的4位的副本。

```
11110000
OR 10101010
11111010
```

OR两个都是0的才是0，除此之外都是1

使用XOR操作复制位串的一部分

XOR操作的一个主要用途是形成位串的补码。

将所有1的掩码都赋给任何字节都会产生字节的补码。

例如，请注意下面示例中的第二个操作数和结果之间的关系。

```
11111111
XOR 10101010
01010101
```

两个1变成0，除此之外都是1

旋转和移位操作

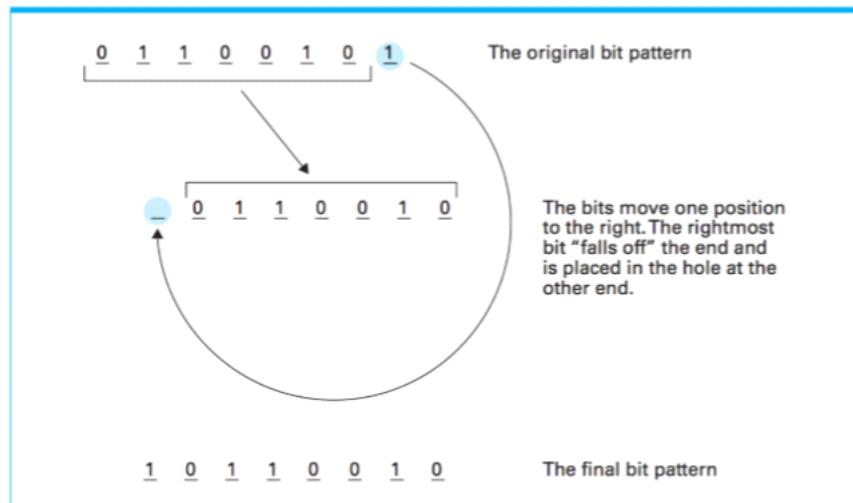
旋转和移位操作类中的操作提供了在寄存器内移动位的方法，通常用于解决对齐问题。

这些操作按照运动方向(右或左)和过程是否为圆形进行分类。考虑一个包含位字节的寄存器。

如果我们把它的内容向右移动1位，我们可以想象最右边的部分从边上掉下来，最左边的一端出现一个洞。

在不同的移位操作中，这个额外的钻头和孔的区别是什么。

Figure 2.12 Rotating the bit pattern 65 (hexadecimal) one bit to the right



### 旋转移位(圆周移动)

一种方法是把从右边掉下来的钻头放在左边的孔里。

其结果是一个循环移位，也称为旋转。

因此，如果我们对字节大小的位模式执行8次右循环移位，我们将获得与开始时相同的位模式。

### 逻辑移位

另一种方法是去掉掉边缘的部分，用0来填充。

术语“逻辑移位”通常用来指这些操作。

这种向左的移位可用于将2的补码表示乘以2。

毕竟，将二进制数字向左移动相当于乘以2，就像将十进制数字向左移动相当于乘以10一样

### 算术移位

此外，可以通过将二进制字符串右移来完成二除。

在任何一种移位中，在使用某些符号系统时，必须注意保留符号位。

因此，我们经常发现右移位总是用它的原始值填充孔(发生在符号位的位置)。

保留符号位不变的移位有时称为算术移位。

### 右圆移位

附录C中描述的机器语言只包含一个由op-code a指定的右循环移位。

在这种情况下，操作数中的第一个十六进制数字指定要旋转的寄存器，其余的操作数指定要旋转的位数。

因此A501指令的意思是“将寄存器5的内容向右旋转1位”。

特别是，如果寄存器5最初包含位模式65(十六进制)，那么在执行这条指令之后它将包含B2(图2.12)。

Figure 4.3 Simple shift operations



Figure 4.4 Circular shift operations



Figure 4.5 Arithmetic shift operations



## 减法，乘法和除法

减法可以通过加减来模拟。

乘法只不过是重复的加法。除法就是反复减法。

有些小型cpu只设计了加法或减法指令。

Note:

每个算术运算都有许多变化。

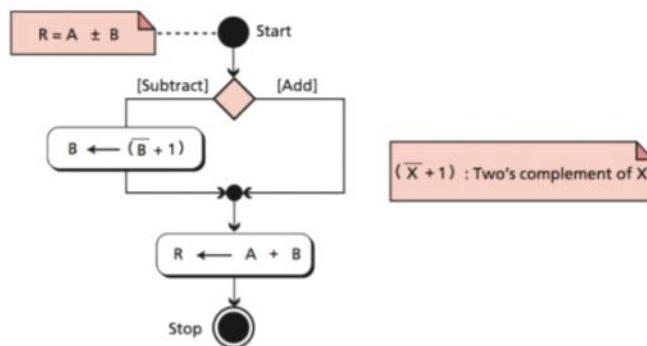
## 不同的添加（不考）

如果要添加的值存储在2的补数表示法中，则添加过程必须以逐位直接添加的方式执行。

如果操作数存储为浮点值，加法过程必须提取每个操作数的尾数，根据指数组合将它们右移或左移，检查符号位，执行加法，并将结果转换为浮点符号。

因此，尽管这两种操作都被认为是加法，但是机器的动作是不一样的。

Figure 4.6 Addition and subtraction of integers in two's complement format



The procedure is as follows:

1. If the operation is subtraction, we take the two's complement of the second integer.  
Otherwise, we move to the next step.
2. We add the two integers.

# Programming language and compilers

2019年11月18日 10:26

## 编程语言和编译器

---

### 程序的定义：

一组语句，(从编程语言形式转换成可执行形式后)可由计算机执行，以便从计算机产生所需的行为。

"一种计算机指令和数据定义的组合，使计算机硬件能够执行计算或控制功能。"

### 编程的定义：

"广义上讲，所有涉及到程序生产的技术活动，包括需求分析和所有阶段的设计和实现。狭义上讲，它是对某个给定设计的程序进行编码和测试。"

"一种计算机指令和数据定义的组合，使计算机硬件能够执行计算或控制功能。"

### 语言的定义

一个国家、民族或种族使用的全部单词和组合单词的方法。

"能表达思想的文字和方法的结合。"

指某一特定民族、国家所使用和理解的词语、发音及其组合方法等。

人类对声音的使用，通常是书写代表这些声音的符号，以有组织的组合和模式来表达和交流思想和感情。

由一个特定国家的人或一群有共同历史或一套传统的人所使用的这种组合和模式形成的一套词汇系统。

### 编程语言的定义：

用于精确描述计算机程序或算法的符号。编程语言是人工语言，语法和语义都是严格定义的。因此，当它们为它们的目的服务时，它们不允许自然语言所特有的表达自由。"

用来表示计算机程序的语言。

### 程序的定义：

程序是一组指定计算的符号序列。

### 编程语言的定义：

编程语言是一套规则，它规定了组成程序的符号序列，以及程序所描述的计算过程。编程语言是一种抽象机制。它使程序员能够抽象地指定计

算，并让程序(通常称为汇编器assembler、编译器complier或解释器interpreter)以在计算机上执行所需的详细形式实现规范。

编程语言的定义：

“编程语言是影响软件系统最终质量的最重要的因素之一，而不是最不重要的因素之一。”

编程语言的存在只是为了弥补硬件和现实世界之间的抽象层次的差距。

更高层次的抽象更容易理解，使用起来也更安全，而较低层次的抽象更灵活，通常可以更有效地实现，两者之间不可避免地存在紧张关系。”

编程语言是符号。它们用于指定、组织和推理计算。”

抽象的定义：

抽象概念有一个普遍的真理：抽象越高，丢失的细节越多

编程语言是由规则来指定的，这些规则用于编写正确的语句，将它们组织成模块，将它们提交给编译器，编译器将代码翻译成特定机器可以理解的语言，最后运行程序。, 将输入输入计算机，计算机根据程序中的指令将其转换成输出

“任何一种编程语言都可以被认为是与其他任何一种编程语言相同的，因为每一种编程语言都会改变存储的值，但是它们在概念和实现层面上都是非常不同的。”语言是围绕一个特定的概念模型组织起来的。”

“编程语言，就像我们的‘自然’语言一样，是为了促进人们之间思想的表达和交流而设计的然而，编程语言与自然语言有两种不同之处。

第一，它们的表达域更窄，因为它们只促进人们之间算法思想的交流。

其次，编程语言也使人们和计算机之间的算法思想交流成为可能。”

编程语言提供了程序员用来编写好程序的抽象、组织原则和控制结构。

编程语言是计算机编程艺术的表达媒介。一种理想的编程语言可以使程序员很容易地简洁而清晰地编写程序。因为程序是要在其一生中被理解、修改和维护的，所以一门好的编程语言将帮助其他人阅读程序并理解它们是如何工作的。”

“一种好的大规模编程语言将帮助程序员有效地管理软件组件之间的交互。”

---

程序语言的世代

第一代语言：机器语言 machine languages

二进制语言

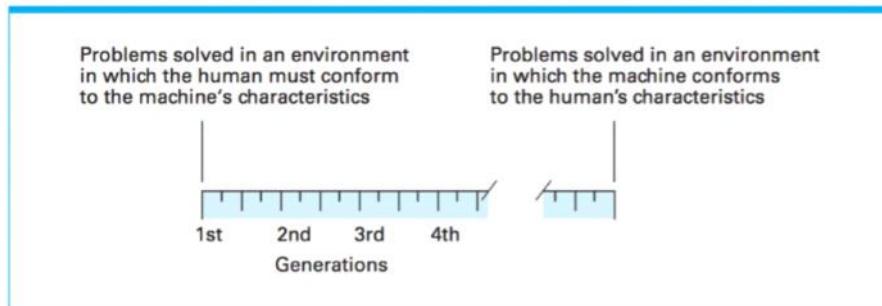
第二代语言：汇编语言 assembly languages

助记符，用一些自然语言的词语帮助记住指令

第三代语言：高级语言 high-level languages

目前的编程用语

**Figure 6.1** Generations of programming languages



机器语言machine language是计算机硬件所能理解的唯一the only语言，它是由具有两种状态的电子开关构成的:off(表示0)和on(表示1)。

计算机唯一能理解的语言是机器语言！！计算机只能按照机器语言的指令来做！！！

## 二进制的指令

**Table 9.1** Code in machine language to add two integers

Hexadecimal	Code in machine language			
(1FEF) <sub>16</sub>	0001	1111	1110	1111
(240F) <sub>16</sub>	0010	0100	0000	1111
(1FEF) <sub>16</sub>	0001	1111	1110	1111
(241F) <sub>16</sub>	0010	0100	0001	1111
(1040) <sub>16</sub>	0001	0000	0100	0000
(1141) <sub>16</sub>	0001	0001	0100	0001
(3201) <sub>16</sub>	0011	0010	0000	0001
(2422) <sub>16</sub>	0010	0100	0010	0010
(1F42) <sub>16</sub>	0001	1111	0100	0010
(2FFF) <sub>16</sub>	0010	1111	1111	1111
(0000) <sub>16</sub>	0000	0000	0000	0000

作为一个更广泛的例子，机器语言通常是

156C  
166D  
5056  
306E  
C000

添加了6C和6D的存储单元memory cell内容，并将结果存储在6E位置可以表示为，这种表示方法叫汇编语言

```
LD R5,Price
LD R6,ShippingCharge
ADDI R0,R5 R6
ST R0,TotalCost
HLT
```

通过助记符号（这里我们用LD,ADDI,ST和HLT来代表load,add,store,和halt），此外，我们用了一些描述性的文字如Price，

ShippingCharge, 和Totalcost来指位于6C, 6D, 6E的存储单元。

这样的描述性文字我们把它叫做标识符identifiers

注意到, 虽然还缺少助记形式, 但它在表示通常的机器语言的意义方面比数字形式做得更好。

---

编写一个汇编语言 (加和两个整数)

LOAD RF Keyboard 从键盘控制器中加载到寄存器F里

STORE Number1 RF 把寄存器F的内容存储到Number1里

LOAD RF Keyboard 从键盘控制器中加载到寄存器F里

STORE Number2 RF 把寄存器F的内容存储到Number2里

LOAD R0 Number1 把Number1里的内容加载到寄存器0里

LOAD R1 Number2 把Number2里的内容加载到寄存器1里

ADDI R2 R0 R1 把寄存器0和1的内容加和取结果放在寄存器2里

STORE Result R2 储存寄存器2的结果

LOAD RF Result 把结果加载进寄存器F里

STORE Monitor RF 把寄存器F里的结果储存在monitor controller里

HALT 停止

---

使用机器语言的问题

不可靠的:用任何机器语言编程都是不可靠的。

低效:任何机器语言的编程都是低效的。

不可读和难以记忆:任何机器程序都是不可读和难以记忆的。

---

汇编语言和汇编程序

汇编语言为每条机器语言指令分配助记字母代码。程序员使用这些字母代码来代替二进制数字。

因为在计算机上执行的每一个程序最终都必须是计算机的机器语言, 所以一个称为汇编程序assembler的程序以助记符的形式读取每一个结构并将其翻译成机器语言的等价物。

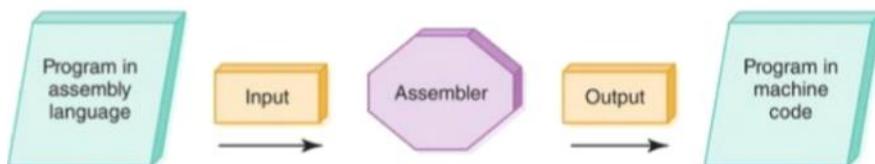
一个叫做汇编程序的特殊程序被用来把汇编语言中的代码翻译成“机器语言”。

汇编语言assembly language

一种低级的程序设计语言, 其中助记符mnemonic代表特定计算机的每条机器指令

汇编程序/汇编器assembler

一个能把汇编语言形式的程序转化为机器代码的程序



### 汇编语言的优势

汇编语言在易错性、效率、可读性和可记性方面优于机器语言。

### 汇编语言的缺点

用汇编语言编写程序仍然容易出错，效率低下，而且汇编程序仍然难以阅读和记忆。

每种汇编语言都特定于特定的计算机体系结构，但通常不能跨多个计算机体系结构移植。

任何汇编程序都必须是与机器相关的(不可移植的)。虽然程序员不需要以数字形式编写指令，但他们仍然被迫从机器语言的小的、递增的步骤来考虑问题。

### 结构中的原语和设计中的原语primitives in construction and primitives in design

产品最终必须使用的基本原语不一定是产品设计中应该使用的原语。

### 设计过程中的原语

设计过程更适合使用高级原语，每个原语代表一个与产品的主要特性相关联的概念。

### 在实现原语primitives in implementation

一旦设计完成，就可以将这些原语转换为与实现细节相关的低层概念。

### 为什么需要高级编程语言?

为了提供高水平的抽象(高级原语)

为了提供程序的可移植性portability(机器的独立性)，在不同的计算机上执行相同的程序

### 第三代语言：

FORTRAN由IBM公司开发科学及工程应用(1957)。

COBOL(面向商业的通用语言):由美国海军开发，用于商业应用(1959)。

ALGOL, APL, PL/I, BASIC, C, Pascal, Ada(83, 95, 2005年, 2012年), ...

### 第三代语言

第三代编程语言与前几代的不同之处在于，它们的原语是更高级别的(以更大的增量large increments表示指令)和与机器无关的(不依赖于特定

机器的特性)。

一般来说，第三代语言的方法是识别一组高级原语，在这些原语中可以开发软件。

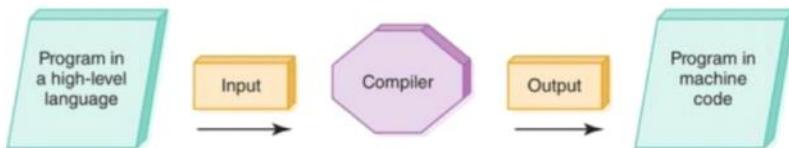
其他高级编程语言

函数式编程语言:LISP, ML, Scheme, ...

逻辑编程语言:Prolog, ...

---

编译器complier: 一个能把高级语言转化为机器语言的程序



---

解释器interpreter

因为编译器的难度系数还是有点大，要生成一整个机器语言程序，人们引用了解释器

一行一行的解释高级语言的语句，然后解释一行，执行一行

一种程序，它用高级语言输入一个程序，并指示计算机根据每条语句中特定的说明执行特殊的行动

一个按顺序翻译和执行语句的程序。解释器可以看作是“理解”程序编写语言的模拟器或虚拟机。

解释器interpreter与转换器translater(汇编器assembler或编译器compiler)的区别

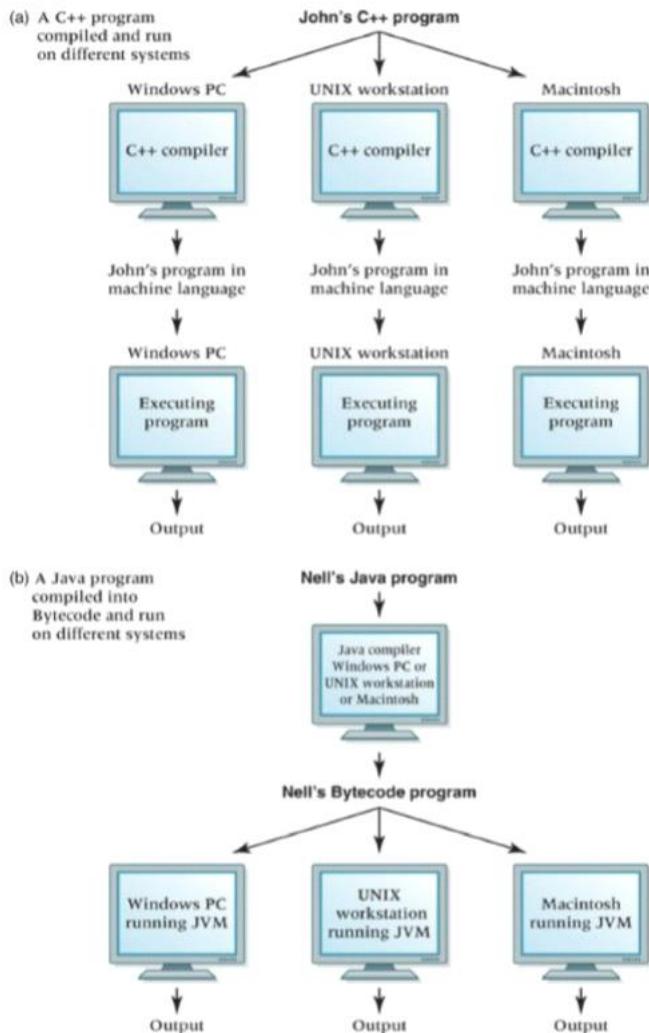
与将机器码作为输出(然后在单独的步骤中执行)的汇编器或编译器不同，解释器翻译语句，然后立即执行execute语句。

翻译器(汇编器或编译器)只是用适当的机器语言生成一个等价的程序，然后再立即运行它。

模拟器simulator直接执行输入程序。

C++编译器不是只有一种，在WindowsPC, MAC, UNIX中的编译器是不一样的，不一样的编译器对应同一个C++语言在不同的操作系统中才能执行

JAVA存在一个中间程序nells JAVA program,能够把JAVA语言转换较为低级的语言被不同的操作系统中的JVM模拟器执行



## 解释interpretation

解释是将源程序的每一行翻译成目标程序相应的行并执行的过程。

逐行翻译，逐行执行

在解释方面有两种趋势:Java之前的一些语言使用的解释(1996年)和Java使用的解释。

## 第一种解释方法

源程序的每一行都被翻译成正在使用和立即执行的计算机的机器语言。

如果在转换和执行过程中出现任何错误，流程将显示一条错误消息，其余的流程将中止。

## 第二解释方法

为了实现可移植性，源程序到目标程序的转换分为两个步骤

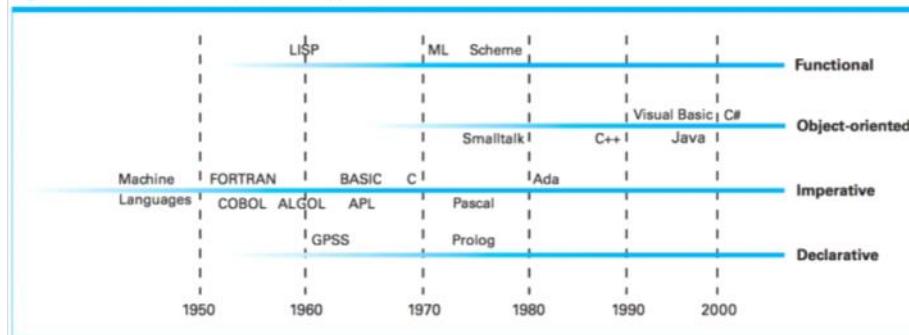
编译comililation和解释interpretation。

首先编译Java源程序来创建Java字节码，它看起来像机器语言中的代码，但不是任何特定计算机的目标代码:它是虚拟机(称为Java虚拟机或JVM)的目标代码。

然后，任何运行JVM模拟器的计算机都可以编译或解释字节码。运行字

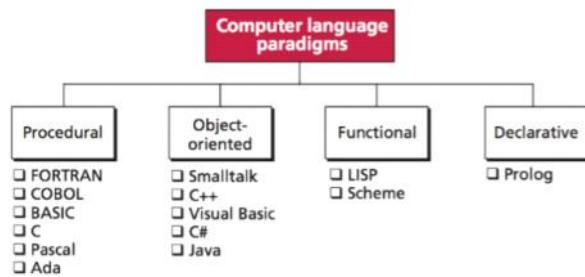
节码的计算机只需要JVM模拟器emulator，而不需要Java编译器。

Figure 6.2 The evolution of programming paradigms



## 计算机语言范例

Figure 9.2 Categories of programming languages



1.程序上的语言

2.面向对象上的语言

3.功能性的

4.说明性的

### 声明式(程序性)范式imperative (procedural) paradigm

声明式范式将编程过程定义为开发一系列命令，这些命令在执行时操作数据以产生所需的结果。

因此，命令式范例告诉我们，通过找到一个算法来解决手头的问题，然后将该算法表示为一个命令序列，从而接近编程过程。

### 定义式范式declarative paradigm

不需要准备算法，只需要提出要解决的是什么问题，会有一系列已经设计出来的基本的算法来解决这个问题

定义式编程系统应用预先建立的通用问题解决算法来解决呈现给它的问题。

在这样的环境中，程序员的任务变成了开发问题的精确描述，而不是描述解决问题的算法。

定义式范式：函数程序设计（基于 $\lambda$ 演算）（不考不考！）

函数编程functional program

在函数式编程范式下，函数式程序被视为接受输入并产生输出的函数。

## 函数式编程作为构建函数

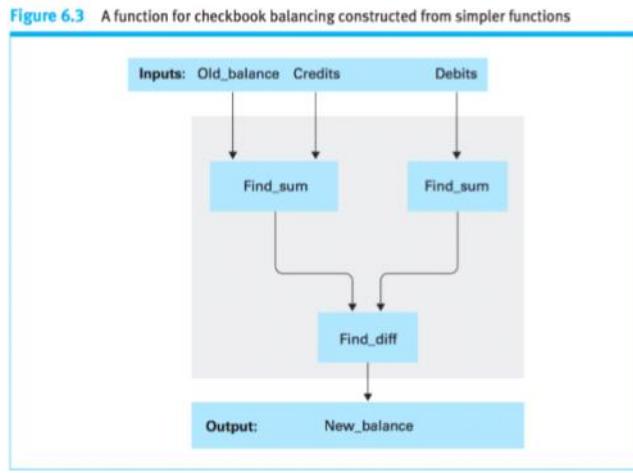
函数程序是通过连接较小的预先定义的程序单元(预先定义的函数)来构造的，以便每个单元的输出被用作另一个单元的输入，从而获得所需的整体输入-输出关系。

简而言之，函数范型下的编程过程是将函数构建为简单函数的嵌套复合体。

---

## LISP程序例子 (不考)

- ◆ (**(Find\_diff (Find\_sum Old\_balance Credits) (Find\_sum Debits))**)



---

## 定义式范式：逻辑程序设计logic programming (不考)

作为逻辑公式的程序

在逻辑编程范式下，逻辑程序是由一系列逻辑公式组成的，这些逻辑公式表达了问题域的事实和推理规则。这样的程序通过向它提问来使用，它试图通过询问事实和规则来回答。

逻辑程序并不确切地说明如何计算结果，而是描述结果的形式。因此，逻辑程序是声明性的，而不是过程性的，因为它只声明所需结果的规范，而不是详细的过程。

## 逻辑编程作为知识表示和推理deduction

逻辑程序设计为计算所需结果提供了相关信息和推理方法。

---

## Prolog程序示例 (不考)

乌龟比蜗牛快的事实可以用Prolog语句来表示

快(乌龟、蜗牛),

兔子比乌龟跑得快这一事实可以用下列数字来表示

faster(兔子,乌龟)。

推理规则(faster (X, Y) AND faster (Y, Z)) → faster (X, Z)

以Prolog形式表示

更快(X, Z):-更快(X, Y), 更快(Y, Z)。

对于程序，问问题“快(兔子，蜗牛)？”

会得到“是”的结果；

问“更快(W, 蜗牛)？”

会得到“更快(乌龟，蜗牛)”和“更快(兔子，蜗牛)”的结果

---

(声明式程序) 面向对象范型object-oriented paradigm

定义清楚对象，和针对对象的操作，针对每一个具体的实例应该怎么做

本身还是要先设计算法的

按照这种范例，软件系统被看作是单元的集合，称为对象，每个对象都能够执行与自身直接相关的操作以及请求其他对象的操作。

这些对象相互作用以解决问题。

对象和方法objects and methods

这些对象中的每一个都包含一组过程(称为方法method)，描述该对象如何响应各种事件的发生。

因此，整个系统将被构造成为一个对象集合，每个对象都知道如何响应与之相关的活动

---

类和实例classes and instance

一个对象可以由数据和一组对数据执行活动的方法组成。这些特性必须用书面程序中的语句来描述。

对象属性的这种描述称为类class

一旦构建了一个类它就可以在任何需要这些特征的对象的时候被应用。

因此，多个对象可以基于同一个类

基于特定类的对象被称为该类的实例the instance of the class。

---

面向对象范式object-oriented paradigm和声明式范式imperative paradigm

对象中的方法的本质上是小型命令式程序单元。

这意味着大多数基于面向对象范式的编程语言都包含命令式语言中的许多特性。

---

定义性语句declarative statements

定义性语句定义了以后在程序中使用的自定义术语customized terminology，例如用于引用数据项的名称。

告诉编译器定义了哪些对象，对象具有哪些性质，让编译器翻译成机器程序

声明式语句imperative statement

声明式语句(赋值语句和控制语句)描述了底层算法中的计算执行步骤。

## 注释comments

注释通过以更适合人类的形式解释其深奥的特性来增强程序的可读性。

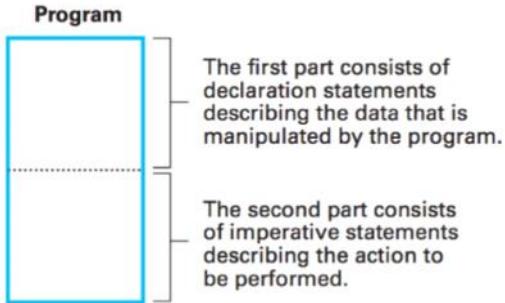
---

典型声明式程序或程序单元的组成

程序

第一部分由描述被程序操作的数据的声明性语句组成。

第二部分由描述要执行的操作的命令式语句组成。



声明declaration：语言中的变量、动作或其他实体相关联的语句，可以为这些实体指定名称，以便程序员可以通过名称引用这些实体

---

## 变量variables

高级编程语言允许通过描述性名称而不是数字地址来引用主内存中的位置。

对应主存里程序/数据具体的位置，这里可以用一个真正定义的名字来表示这个位置，而不用真正使用地址上的编码来表示这个位置

如：我们给地址A309取一个名字“学号”，以后输入变量“学号”都是从这个地址来取数据

这样的名称称为一个变量variable(名称name)，它识别这样一个事实，即通过更改存储在该位置的值，与名称关联的值在程序执行时发生更改(通过赋值语句)。

Note:

程序中的变量与数学中的变量有本质的区别。

在程序中，变量是内存“盒子”的名字。

---

## 数据类型

在程序的其他地方使用变量之前，通过声明性语句declarative statement进行标识。

这些声明性语句还要求程序员描述存储在与变量关联的内存位置的数据的类型。

这种类型称为数据类型data type，它包含对数据项进行编码的方式和可以对该数据执行的操作。

Note:

准确定义数据类型是提高程序质量的重要手段。

---

### 常数constants

高级编程语言允许描述性名称:指定特定的、不可更改的值。这样的名字叫做常数constant。

### 文字literals

值的显式外观(表示形式)称为文字。

Note:

使用常数constant而不是字面值是一种很好的编程实践。

文字的使用不是好的编程实践，因为文字可能会掩盖它们出现的语句的含义。

---

### 算术表达式arithmetic expression得到一个数值

算术表达式是标识符序列sequence of identifiers。由算术运算符分隔，得到一个数值。

### 逻辑(布尔)表达式logic (Boolean) expressions得到一个真/假结果

逻辑(布尔)表达式是一组标识符，由兼容的操作符分隔，其计算结果为真或假。

布尔表达式可以是以下任意一种:

一个布尔变量，

一个算术表达式，后跟一个关系运算符，后跟一个算术表达式，

一个布尔表达式后面跟一个逻辑(布尔)运算符，后面跟一个布尔表达式。

---

### 关系运算符relational operators

关系运算符是比较两个数值的运算符。

两个算术表达式之间的关系运算符询问两个表达式之间是否存在关系。

Symbol	Meaning	Example	Evaluation
<	Less than	Number1 < Number2	True if Number1 is less than Number2; false otherwise
<=	Less than or equal	Number1 <= Number2	True if Number1 is less than or equal to Number2; false otherwise
>	Greater than	Number1 > Number2	True if Number1 is greater than Number2; false otherwise
>=	Greater than or equal	Number1 >= Number2	True if Number1 is greater than or equal to Number2; false otherwise
!= or <> or /=	Not equal	Number1 != Number2	True if Number1 is not equal to Number2; false otherwise
= or ==	Equal	Number1 == Number2	True if Number1 is equal to Number2; false otherwise

## 赋值语句assignment statement

最基本the most basic的命令式语句是赋值语句，它请求将一个值分配给一个变量(或者更精确地说，存储在变量标识的内存区域中)。

左边是变量名字，右边是一个算术表达式/逻辑表达式

意思是把表达式计算出的值放在变量里面去，替换变量里原有的数值

## 赋值语句的语法syntax和语义semantics

赋值语句通常语法形式的由变量构成，后跟表示赋值操作的符号，然后是表示要赋值的表达式。

这种语句的语义是要计算表达式并将结果存储为变量的值。

例如,Z:= X + Y。

---

## 条件语句if statement

在C, C++, C#和JAVA中使用

if (条件) 语句A

else 语句B;

在Ada中使用

IF 条件 THEN

语句A;

ELSE 语句B;

END IF;

## 循环语句while statement

在C, C++, C#和JAVA中使用

while (条件)

{循环体}

在Ada中使用

WHILE 条件 LOOP

## 循环体

END LOOP;

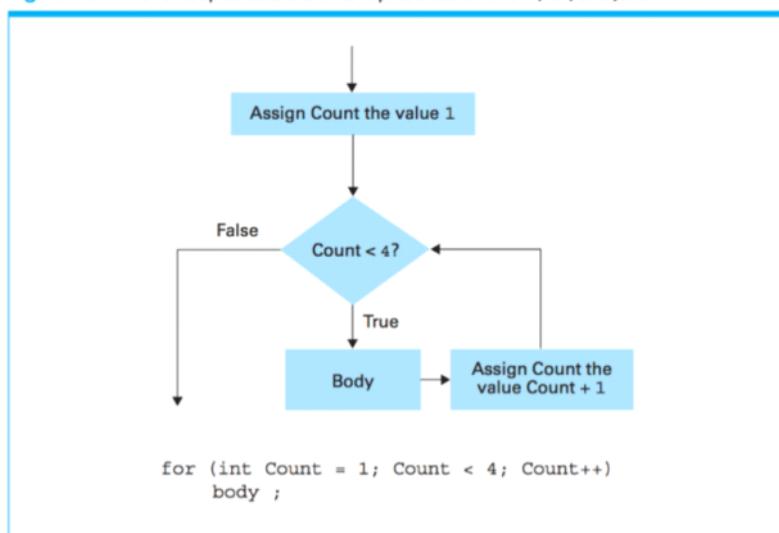
### ♣ If statement

- ♦ if (condition) statementA  
    else statementB; (in C, C++, C#, and Java)
- ♦ IF condition THEN  
    statementA;  
ELSE statementB;  
END IF; (in Ada)

### ♣ While statement

- ♦ while (condition)  
    {loop body} (in C, C++, C#, and Java)
- ♦ WHILE condition LOOP  
    loop body  
END LOOP; (in Ada)

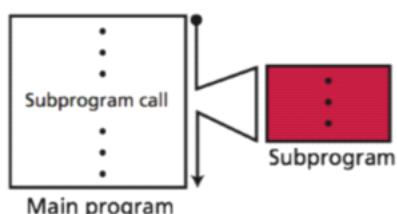
Figure 6.7 The for loop structure and its representation in C++, C#, and Java



## 子程序subprogram

我们可以给一段代码(称为子程序)一个名称，然后在程序的另一部分(称为calling unit)中使用这个名称作为语句。

当该子程序的名称出现时，调用单元中的处理将挂起，主程序暂停，先执行子程序，当指定的代码完成执行后，处理将继续，主程序继续执行，语句位于名称出现的地方。



## 子程序的两种基本形式

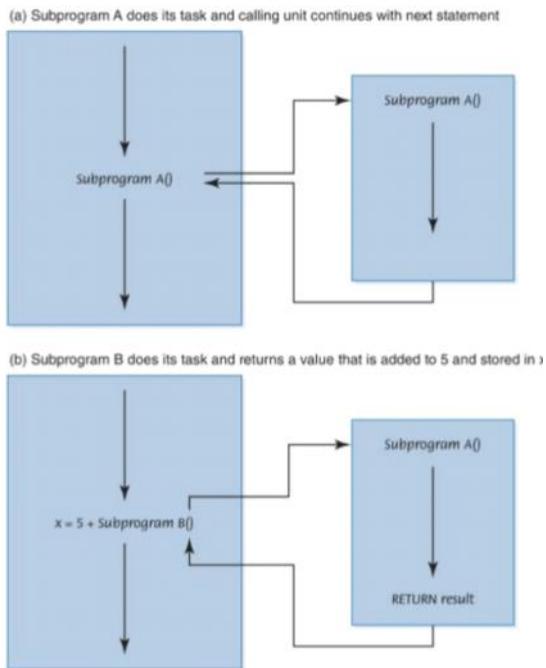
Void子程序：执行特定任务的指定代码;在呼叫单元中用作语句返回值的子程序：

已命名的子程序：它也执行一项任务，但只向单元调用返回一个值；在调用单元中的表达式中使用，其中返回的值将用于表达式的求值。

### 子程序作为强大的抽象工具

已命名子程序的清单允许程序的读者查看正在执行的任务，而不必担心任务实现的细节。

如果子程序需要数据来执行其任务，我们将数据值的名称放在子程序标题的括号中



---

### 两种子程序的表示

过程procedure上图a

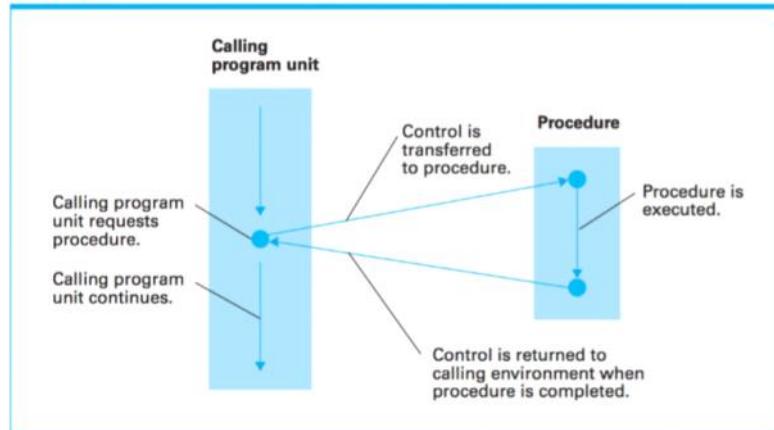
过程是一组用于执行任务的指令，可以作为其他程序单元的抽象工具使用。

控制在程序需要其服务时转移到程序，然后在程序完成后返回到原始程序单元

将控制权转移到过程的过程通常称为调用calling

或调用过程invoking。请求执行过程的程序单元称为调用单元calling unit。

Figure 6.8 The flow of control involving a procedure



### 函数function上图b

函数是类似于过程的程序单元，只是一个值作为“函数的值”被传送回调用的程序单元。

### 参数parameters

过程通常使用在应用过程时指定的通用术语来编写。过程中的这种通用术语称为参数。

### 形式和实际参数

过程procedure中使用的术语称为形式参数formal parameters，应用过程时分配给这些形式参数的精确含义称为实际参数actual parameters。

当涉及到多个参数时，实际的参数将一个条目一个条目地与过程标头中列出的形式参数相关联。

然后，将实际参数的值有效地转换为相应的形式参数，并执行该过程

### 在实际参数和形式参数之间传输数据（不考）

在实际参数和形式参数之间传输数据的任务由不同的编程语言以各种方式处理。

### 调用(传递)数值

由实际参数表示的数据的副本被生成并提供给过程。

使用这种方法，程序对数据的任何更改都只反映在副本中，调用程序单元中的数据不会更改。

按值传递参数可以保护调用单元中的数据不被设计糟糕的过程错误地更改。

### 通过引用调用(传递)（不考）

通过告诉过程调用程序单元中实际参数的地址，使过程直接访问实际参数。

通过引用传递参数允许过程修改驻留在调用环境中的数据。

## 例子

正式:=正式+1;

假设变量Actual赋值为5，我们使用语句Demo (Actual)调用Demo。

## 通过值调用(传递)

如果参数是通过值传递的，则过程中的形式更改不会反映在变量中实际(图6.10)。

## 通过引用调用(传递)

如果通过引用传递参数，那么Actual的值将增加1(图6.11)。

Figure 6.10 Executing the procedure Demo and passing parameters by value

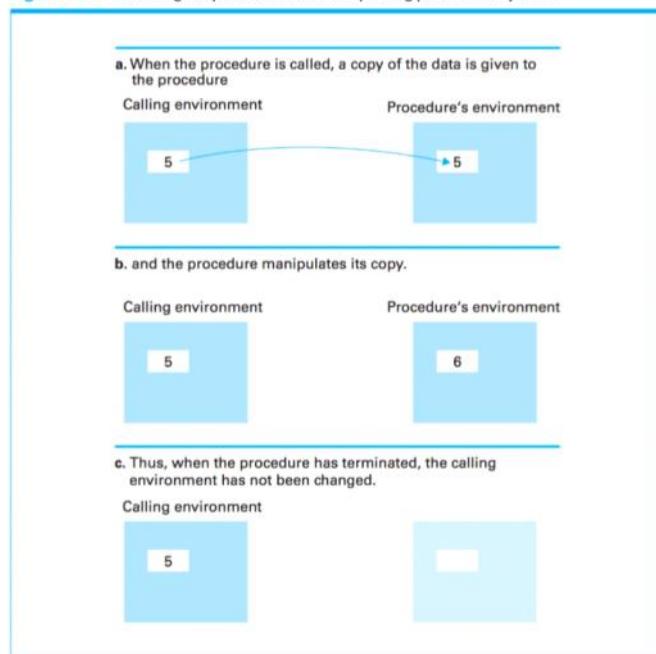


Figure 6.11 Executing the procedure Demo and passing parameters by reference

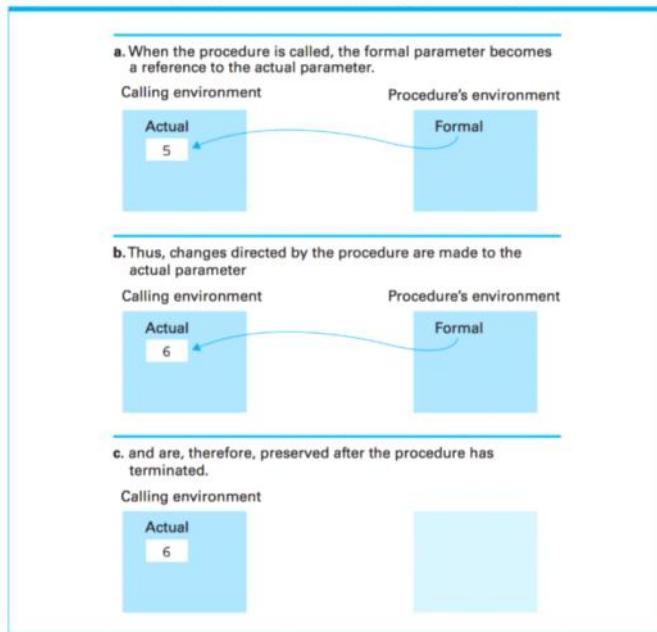


Figure 9.12 An example of pass by value

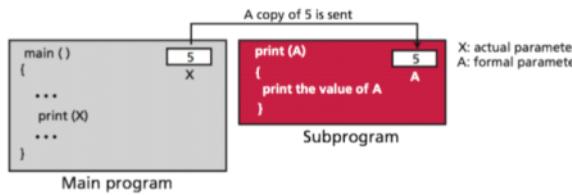


Figure 9.13 An example in which pass by value does not work

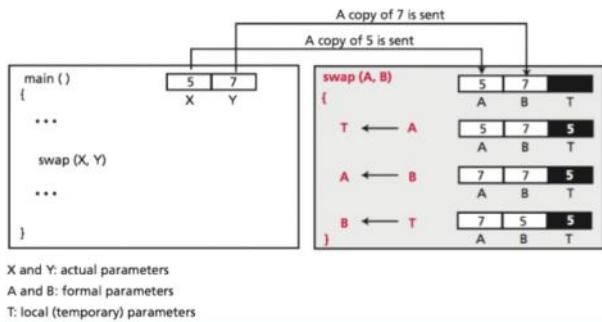
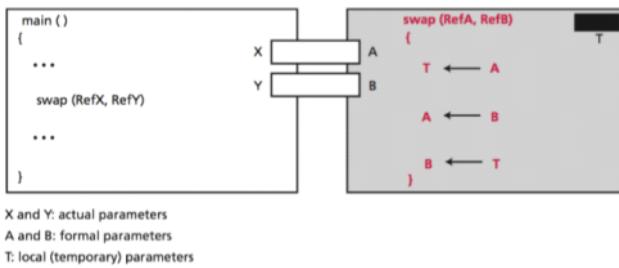


Figure 9.14 An example of pass by reference



## 程序编译(翻译)program compilation translation

将程序从一种编译compilation(翻译translation)语言转换为另一种描述(表示)语言的过程称为编译(翻译)。

原始形式的程序称为源程序source program, 翻译后的版本称为目标程序object program。

翻译过程由三个活动组成:词法分析lexical analysis、解析parsing和代码生成code generation, 这些活动由编译器(转换器)中的三个单元执行, 即词法分析器lexical analyzer、解析器parser和代码生成器code generator。

Figure 6.13 The translation process

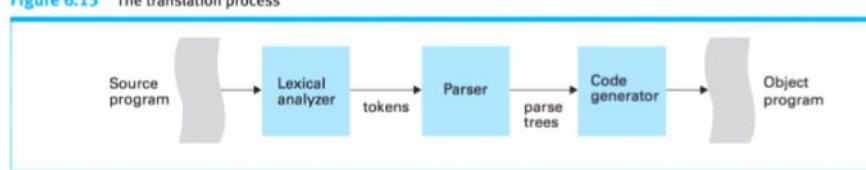
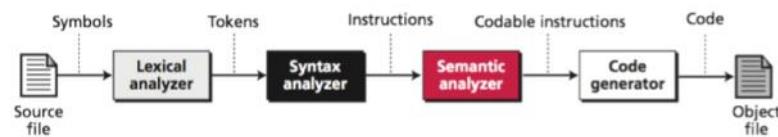


Figure 9.1 Source code translation process



## 词法分析器lexical analyzer

词法分析器通过读取源代码和符号，并在源语言中创建标记列表。

把每一个字符串的词分析出来

解析器parser中包含了语法分析器和语义分析器

## 语法分析器syntax analyzer

语法分析器解析一组标记来查找指令。

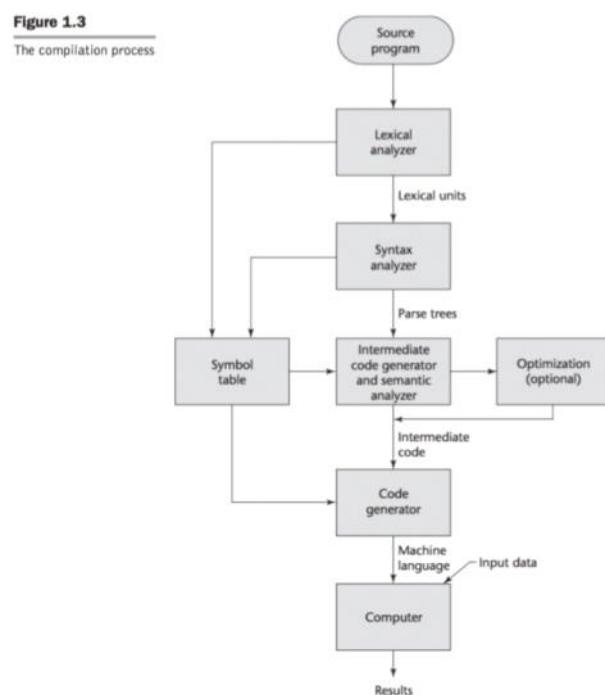
词是否按照程序设计语言规定的排列顺序写的

## 语义分析器semantic analyzer

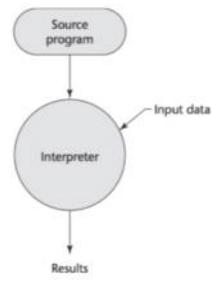
语义分析器检查由语法分析器创建的句子，以确保它们不包含歧义。

## 代码生成器code generator

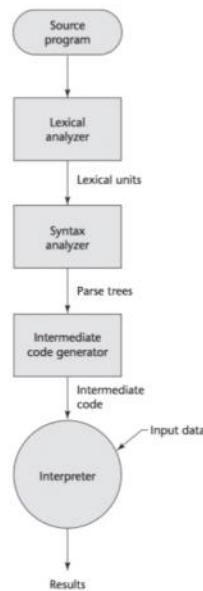
在语义分析器创建了明确的指令之后，每条指令都被转换成一组机器语言指令，供程序在其上运行。这是由代码生成器完成的



**Figure 1.4**  
Pure interpretation



**Figure 1.5**  
Hybrid implementation system



### 词法分析lexical analysis

词法分析是识别源程序中哪些字符串表示单个实体(称为标记token)的过程。

词法分析器通过符号来读取源程序的符号，识别哪些符号组表示令牌，并根据这些令牌是否是数值、单词、算术运算符等对它们进行分类。

词法分析器用其分类对每个标记进行编码，并将它们交给解析器。

### 解析器parsing

解析器根据词法单位lexical units(标记tokens)而不是单个符号来查看程序。

解析器的工作是将这些单元分组到语句中。

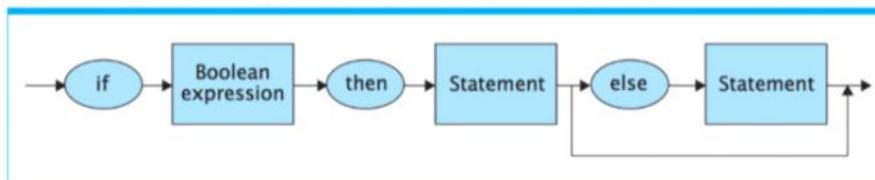
实际上，解析就是识别程序的语法结构和识别每个组件的作用的过程。

### 语法和句法图

解析过程基于一组定义编程语言语法的规则。这些规则被称为语法 grammar。

表达这些规则的一种方式是通过语法图syntax diagrams，语法图是语言语法结构的图示。

**Figure 6.14** A syntax diagram of our if-then-else pseudocode statement



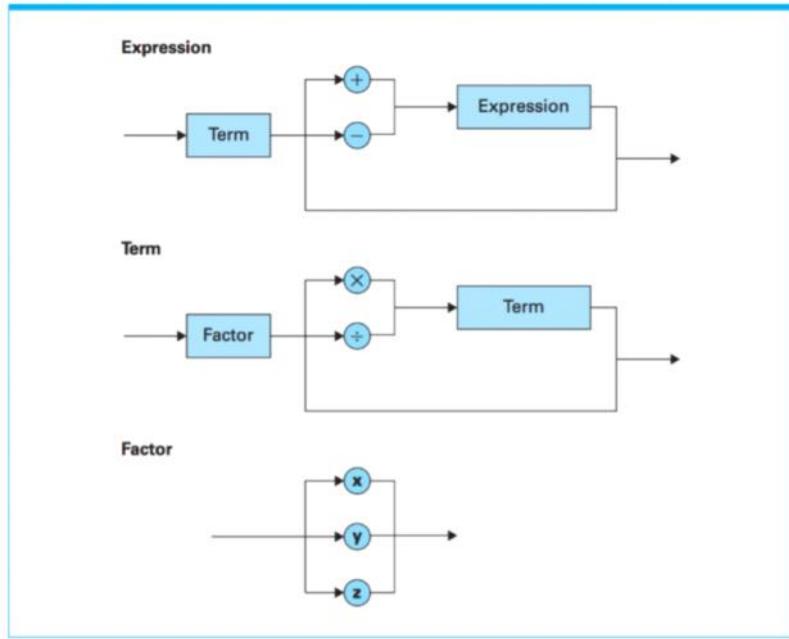
出现在椭圆中的术语称为终端符terminals

需要进一步描述的术语(用矩形表示的)称为非终端符non-terminals

非终端符：还需要进一步展开的

在对语言语法的完整描述中，非终结符由附加的图来描述。

Figure 6.15 Syntax diagrams describing the structure of a simple algebraic expression



### 解析树 parse tree

一个特定的字符串符合一组语法图，可以用解析树的图形形式表示。

解析一个程序的过程实质上就是为源程序构造唯一一个解析树的过程。

实际上，解析树表示解析器对程序语法组成的解释。

因此，描述程序语法结构的语法规则不能允许一个字符串有两个不同的解析树，因为这会导致解析器内的歧义。

Figure 6.16 The parse tree for the string  $x + y + z$  based on the syntax diagrams in Figure 6.15

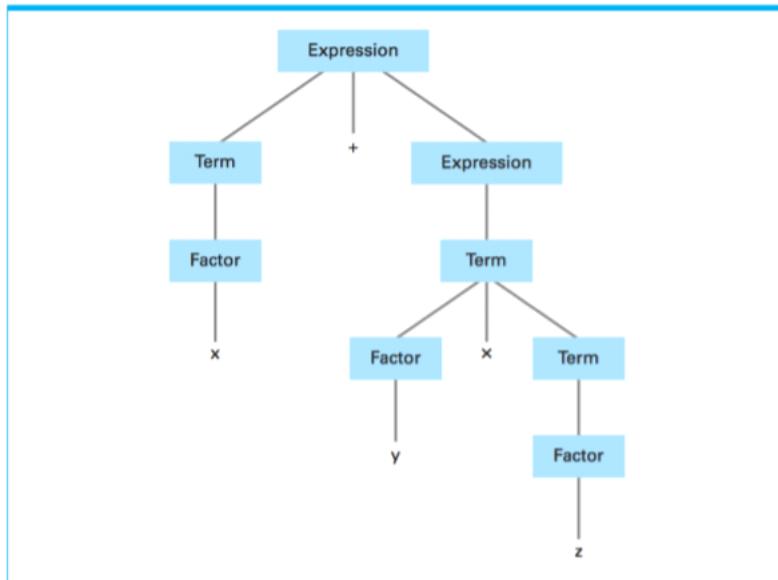
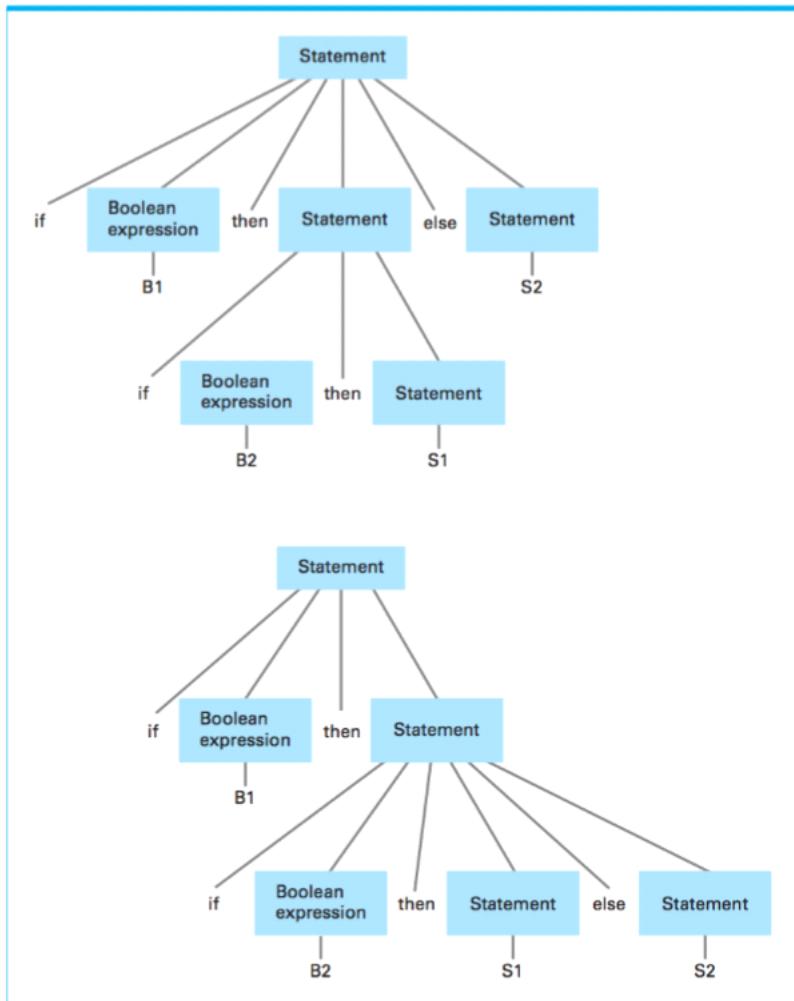


Figure 6.17 Two distinct parse trees for the statement if B1 then if B2 then S1 else S2



### 符号表symbol table

当解析器分析程序的语法结构时，它能够识别单个语句并区分声明性语句和定义性语句。

当它识别声明性语句时，它记录在一个称为符号表的表中声明的信息。

因此，符号表包含诸如程序中出现的变量的名称以及与这些变量相关联的数据类型和数据结构等信息。

### 代码code generation

生成编译(翻译)过程的最后一个活动是代码生成，这是构造机器语言指令来实现解析器识别的语句的过程，这样机器指令程序(目标程序)才能真正在目标计算机上执行。

这个过程涉及许多问题;当然，最重要的是生成正确的代码;另一个重要的任务是生成高效的机器语言版本的程序。

### C程序和UNIX操作系统的迭代

- ◆ CPL --> BCPL(M. Richards, Cambridge U, 1967) --> B(K. Thompson, AT&T Bell Lab, 1970) --> C(D.M. Richie, AT&T Bell Lab, 1972).
- ◆ UNIX in B(K. Thompson, AT&T Bell Lab, 1960s-1970) --> UNIX in C(K. Thompson & D.M. Richie, AT&T Bell Lab, 1973)

C—最危险的程序语言

第一系统软件编程语言。

最高效的编程语言。

最广泛使用的编程语言。

最简单的编程语言。

“最不可靠”的编程语言。

Ada编程语言—最安全的程序语言

Ada 83是第一种完全独立的编程语言，它定义了要设计的编程语言的需求。

Ada 83是第一种国际标准化的编程语言，它将并发性作为该语言的一个固有特性。

Ada 95是世界上第一个面向对象的编程语言。

Ada 2012是世界上首屈一指的工程安全、可靠的软件编程语言。

Ada 2012是最强大的编程语言。

波音777计划：与Ada一起工作

波音777中99%的程序都是用Ada编写的

“合作”是波音公司在1990年选择的项目名称，当时该公司首次萌生了生产777的想法。

商业航空

空客320,330,340;空客380;Beechjet 400A(美国商务机);山毛榉星舰一号(美国商用涡轮螺旋桨飞机);Beriev BE-200(俄罗斯森林火灾巡逻机);波音737 - 200,-400,-500,-600,-700,-800;波音747 - 400;波音757,767;波音777;波音787;波音787空调控制单元;Canadair支线客机;巴西航空工业公司CBA-123和CBA-145(巴西制造的支线客机);Fokker F-100(荷兰dc -9型客机-美国航空公司);Ilyushin 96米(俄罗斯喷气客机);洛克希德马丁大力神“飓风追逐者”;萨博2000年;图波列夫TU-204(俄罗斯喷气客机)

- ◆ **Airbus 320, 330, 340; Airbus 380; Beechjet 400A (US business jet); Beech Starship I (US business turboprop); Beriev BE-200 (Russian forest fire patrol plane); Boeing 737-200, -400, -500, -600, -700, -800; Boeing 747-400; Boeing 757, 767; Boeing 777; Boeing 787; Boeing 787 Air Conditioning Control Unit; Canadair Regional Jet; Embraer CBA-123 and CBA-145 (Brazilian-made regional airliners); Fokker F-100 (Dutch DC-9-size airliner - American Airlines flies these); Ilyushin 96M (Russian jetliner); Lockheed-Martin Hercules “hurricane chaser”; Saab 2000; Tupolev TU-204 (Russian jetliner)**

航天交通管理系统使用Ada的国家

**Australia,**  
**Belgium, Brazil,**  
**Canada, China, Czech Republic,**  
**Denmark,**  
**Finland, France,**  
**Germany, Greece,**  
**Hong Kong, Hungary,**  
**India, Ireland,**  
**Kenya,**  
**Netherlands, New Zealand,**  
**Pakistan,**  
**Scotland, Singapore, South Africa, Spain, Sweden,**  
**Taiwan,**  
**United Kingdom, UK New En-Route Center (NERC),**  
**United States FAA En Route Automation Modernization (ERAM), United States FAA Conflict Detection Tool (URET), United States FAA Air Route Traffic Control Centers Display System Replacement (DSR),**  
**Vietnam,**  
**European Air Traffic Flow Management**

#### C++

c++语言是由贝尔实验室的Bjarne Stroustrup开发的，是对C语言的改进。它使用类来定义类似对象的一般特征以及可以应用于它们的操作。在c++语言的设计中使用了三个原则:封装encapsulation、继承inheritance和多态性polymorphism。

#### Java

Java是由Sun Microsystems公司开发的。它基于C和c++，但是为了使语言更健壮，删除了c++的一些特性;它是面向类的。

#### LISP

LISP(列表编程)是20世纪60年代早期由麻省理工学院的一组研究人员设计的。

它是一种函数式编程语言，在这种语言中，一切都被看作是一个列表。

#### Scheme

Scheme语言(在20世纪70年代早期由MIT开发)定义了一组解决问题的原始函数。

函数名和函数的输入列表用括号括起来。

结果是一个输出列表，它可以用作另一个函数的输入列表。

## SPACK

SPARK是一种正式定义的编程语言，基于Ada语言，旨在开发高完整性的软件，用于可预测和高可靠predictable and high reliable operation的操作是必不可少的系统;它促进需要安全性、安全性和业务完整性的应用程序的开发。

SPARK 2014，基于Ada 2012，于2014年4月30日发布，

SPARK语言由定义良好的Ada语言子集组成，Ada语言使用契约以适合于静态和动态验证的形式描述组件的规范。

## Python

Python是一种编程语言，由Guido van Rossum在20世纪80年代后期创建。今天，它在开发Web应用程序、科学计算和作为学生入门语言方面很流行。Python强调可读性，并包含命令式、面向对象和函数式编程范例的元素。Python也是使用固定格式的现代语言的一个例子。它使用缩进来表示程序块，而不是使用标点符号或保留文字

---

# Operating system

2019年11月26日 11:07

## 操作系统OS

---

### 操作系统

联合控制系统资源以及在计算机系统上使用这些资源的过程的软件产品集  
控制计算机程序执行并在计算机系统中提供诸如计算机资源分配、作业控  
制、输入/输出控制和文件管理等服务的软件、固件和硬件部件的集合。

---

### 操作系统operating system

操作系统是控制计算机整体操作的软件。

它提供了用户存储和检索文件的方法，提供了用户请求程序执行的接口，并  
提供了执行所请求的程序所需的环境。

### 操作系统的必要性

CPU的高效率vs I/U设备的低效率。

为了自动为各种用户提供各种服务，一个控制和管理系统(软件)是必不可少  
的。

如果没有操作系统，一台“裸机”很难使用，如果不是不可能的话。

---

### 操作系统

操作系统是复杂的，因此很难给出一个简单的通用定义。相反，以下是一些  
常见的定义：

操作系统是计算机硬件和用户(程序或人)之间的接口。

一个操作系统是一个程序(或一系列的程序)，促进了其他项目的执行。

操作系统充当总经理的角色，监督计算机系统中每个用户的活动。作为一个  
总经理，操作系统检查硬件和软件资源是否得到了有效利用，当使用资源出  
现问题时，操作系统会进行协调来解决问题。

操作系统是计算机硬件和用户(程序或人)之间的接口，它促进其他程序的执  
行以及对硬件和软件资源的访问。

操作系统的两个主要设计目标是：

有效地使用硬件。

易于使用资源。

操作系统：管理计算机资源并为系统交互提供接口的系统软件操作系统。

虽然我们可以给不同的定义根据操作系统的不同视图，下面的非正式的定义  
是一个好的起点：

操作系统是一组一个或多个软件模块,管理和控制计算机的资源或其它计算机或电子设备,让用户和程序接口利用这些资源。托管资源包括内存、处理器、文件、输入或输出设备等。

早期的计算机系统:批处理batch processing和交互处理intertive processing

批处理:把程序数据等要做的东西排队,按顺序来

交互处理:把程序数据等要做的东西一起输入,在不同的地方计算,再输出

Figure 3.1 Batch processing

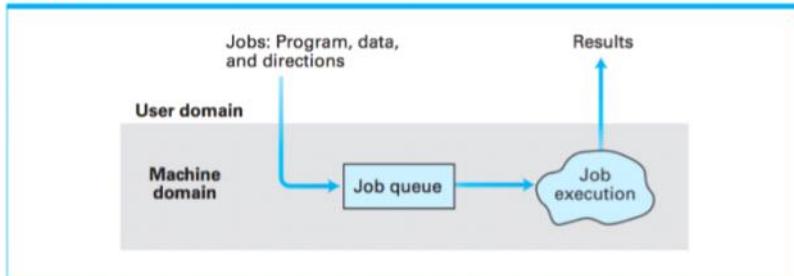
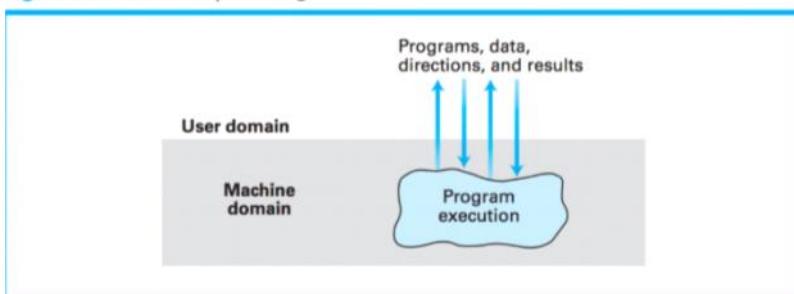
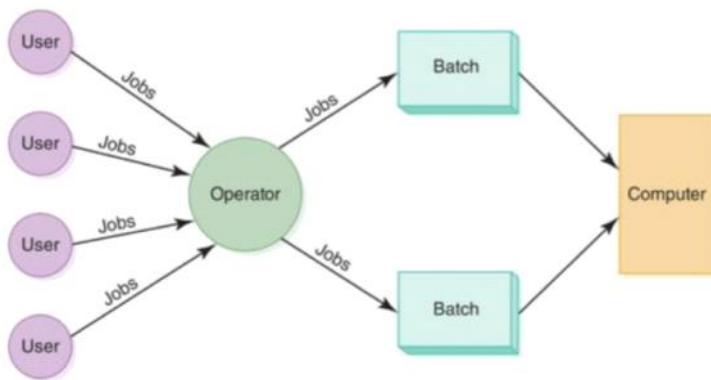


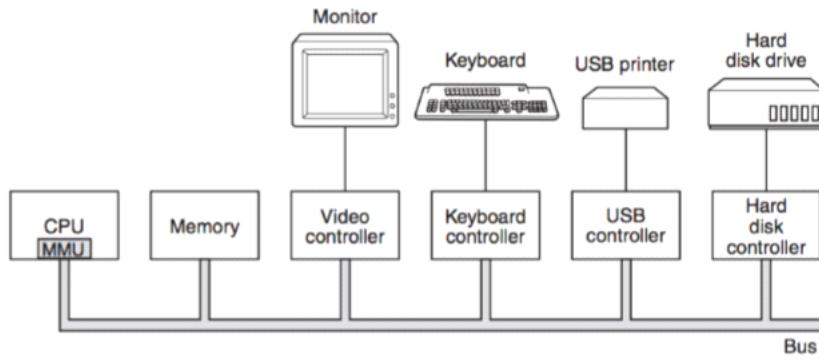
Figure 3.2 Interactive processing



在早期的系统中,人工操作员会分批组织作业

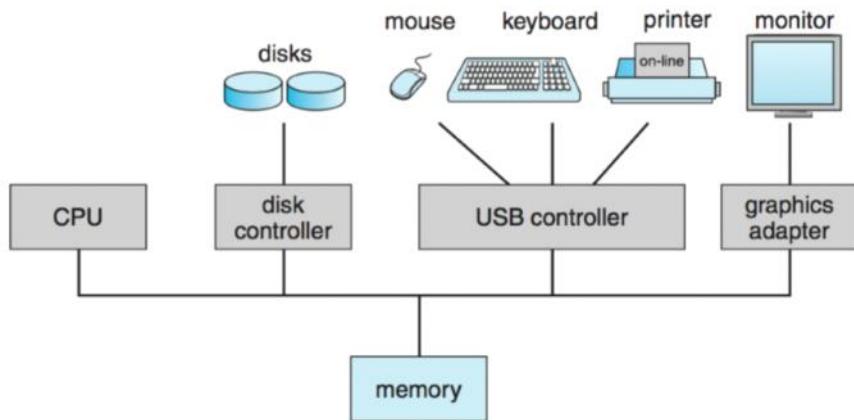


现代操作系统——一个简单的个人电脑的具备的设备



一个总导线，连着CPU，主存，显示控制器，键盘控制器，USB控制器，硬盘控制器

### 现代的电脑系统

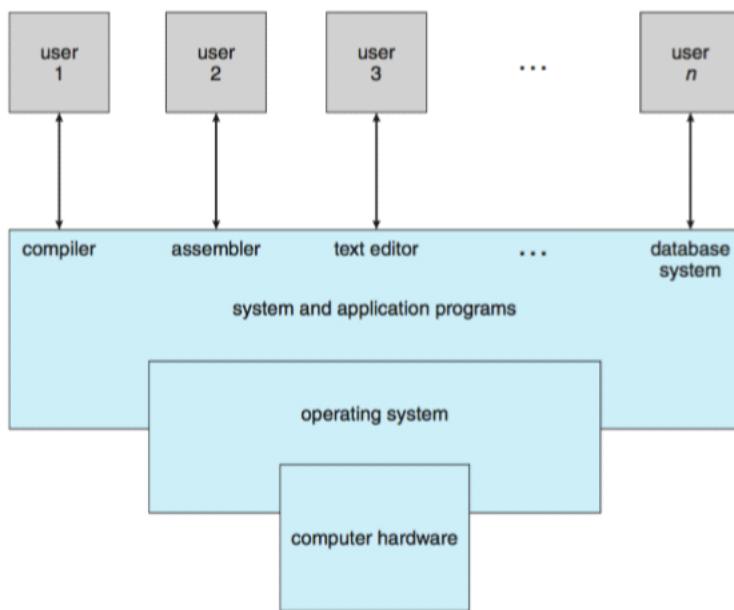


### 抽象概念

只有硬件hardware的电脑叫“裸机”

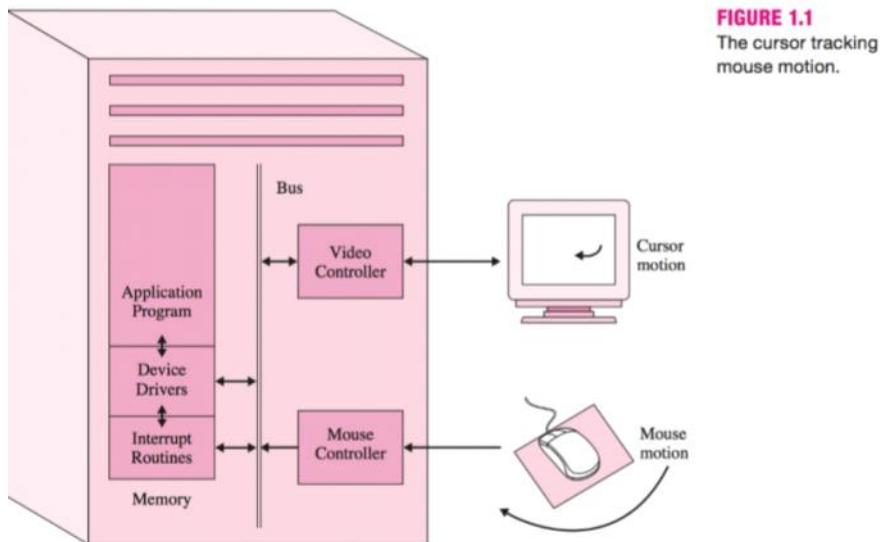
如果有操作系统OS套在外面，则计算机会便于使用

如果再外面有其他的系统或者应用软件，就基本形成了当前使用的计算机，那么不同的user（可以理解为需求）在不同的系统或者应用软件执行



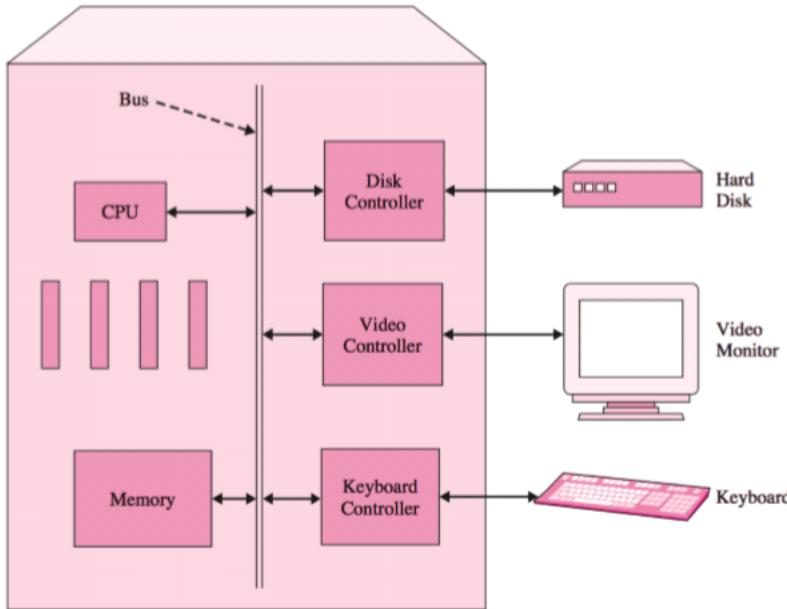
**Figure 1.1** Abstract view of the components of a computer system.

各种设备（硬件）通过操作系统与电脑（软件）进行联系  
光标跟踪鼠标的移动



硬件hardware:非常简单的小型个人电脑。

(注:此图过于简单。实际上，在视频和内存之间通常有多个总线。我们将在附录中看到更详细的图片。)




---

一个简化的关于操作系统中软件与硬件的联系的模型

其他壳牌程序(命令解释程序)

实用程序

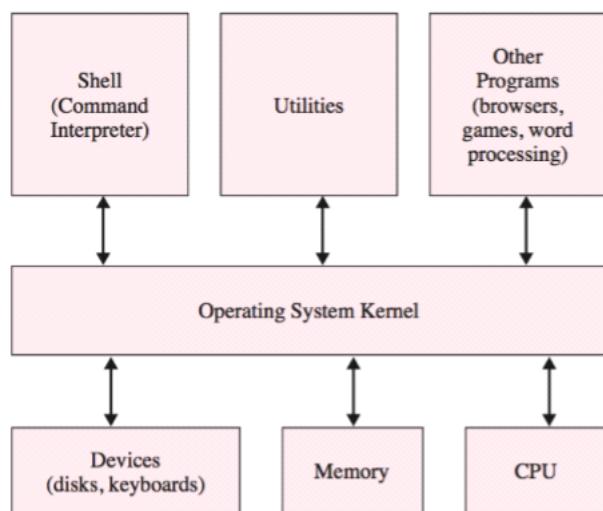
其他程序(浏览器、游戏、文字处理)

操作系统内核

设备(磁盘、键盘)

内存

CPU

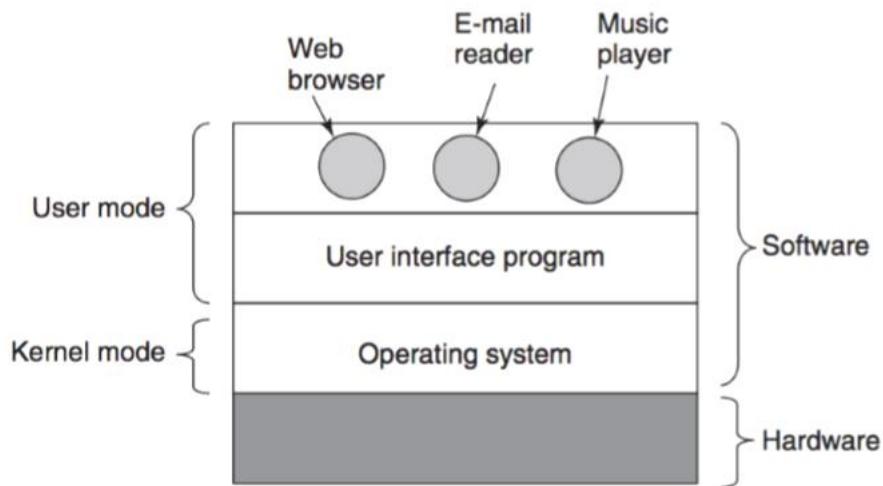



---

操作系统OS应用于哪里

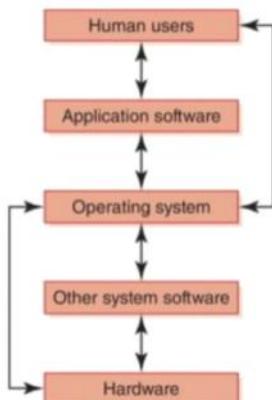
直接接在硬件上面的，一定是操作系统

操作系统管理各种硬件，同时为软件提供服务  
用户给出命令→对应的内部软件→连结到操作系统→与硬件关联



**Figure 1-1.** Where the operating system fits in.

操作系统:与计算机系统的许多方面进行交互



**FIGURE 10.1** An operating system interacts with many aspects of a computer system.

软件分类

软件software 应用application

系统system 实用性utility

操作系统OS 用户界面user interface

核心kernel

系统软件：所有用户要使用该操作系统，都要使用的软件  
(操作系统在系统软件里)

应用软件：部分用户有特殊的需求，需要使用的软件

OS：启动过程Booting process

步骤1：

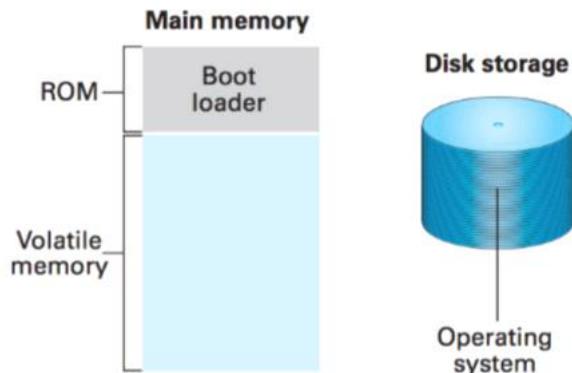
机器通过执行引导加载程序启动程序已经在内存中。

操作系统存储在大容量存储器中。

ROM：一种主存储，它不会因为断电而不被保存，所以启动所加载boot loader都存储在ROM里面

Boot loader启动后，就会从磁盘里把OS提出来在一种挥发性存储器 volatile memory里面加载OS

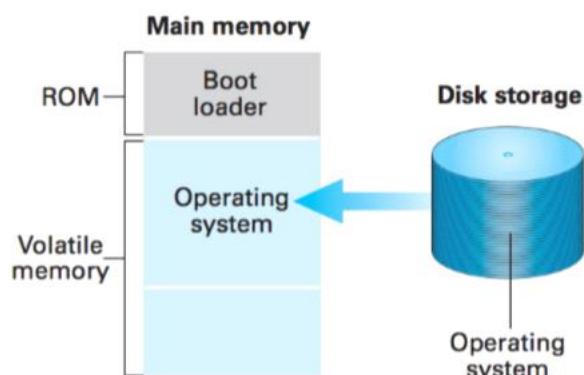
操作系统开始执行后，就开始做各种各样的准备，把各种使用于用户指令的一系列东西准备好，电脑就开机完了



步骤2：

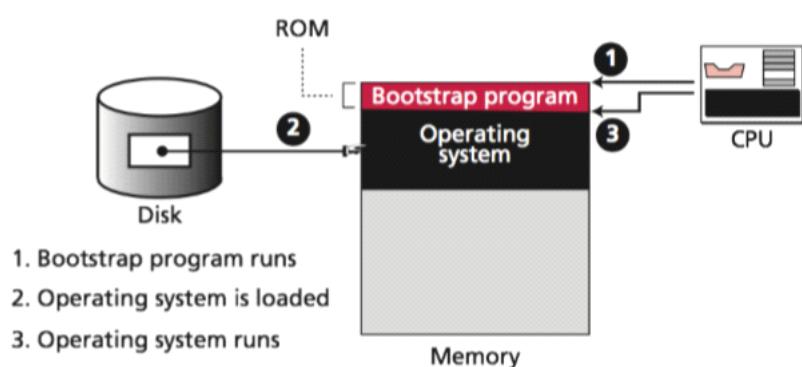
引导加载程序指导操作系统进入主存。

然后将控制权转移给它。



---

OS:引导过程



---

OS:构成

四大管理模块

## 1. 内存管理

CPU本身只会从内存里面拿指令数据，但是需要操作系统来管理

## 2. 进程管理

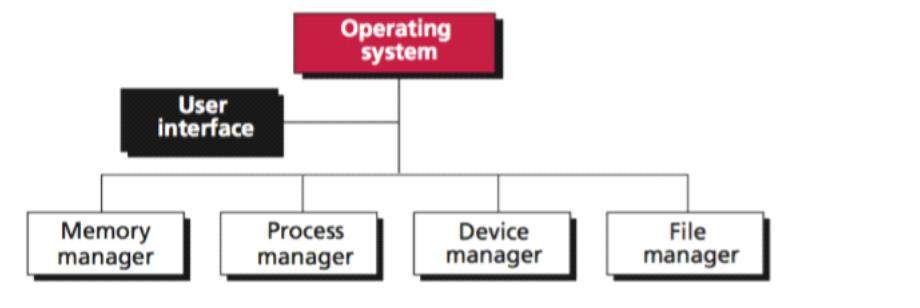
程序编写，存储，执行一系列管理

## 3. 驱动管理

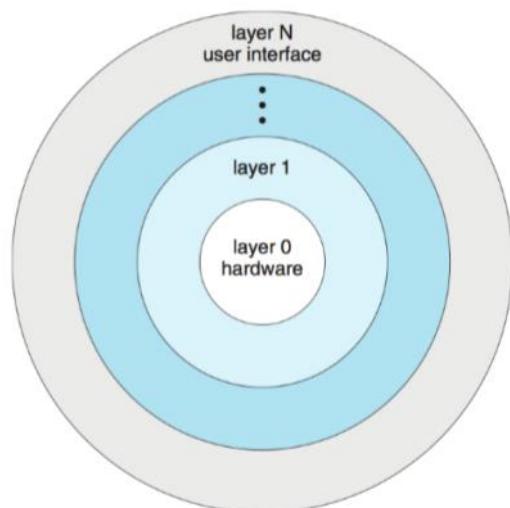
对各种各样的设备进行管理

## 4. 文件管理

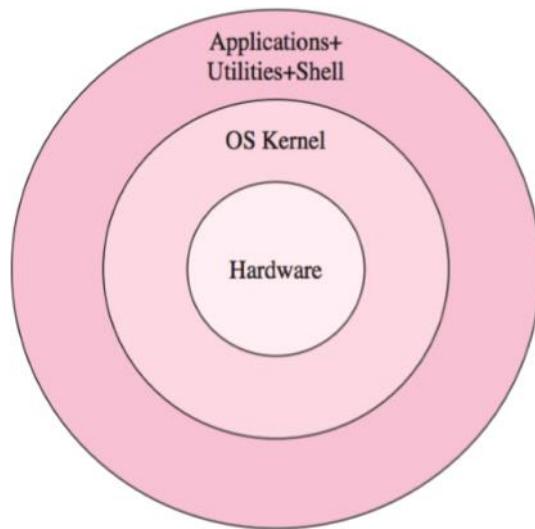
任何指令数据在磁盘里都是以文件的形式存储，操作系统负责管理这些文件



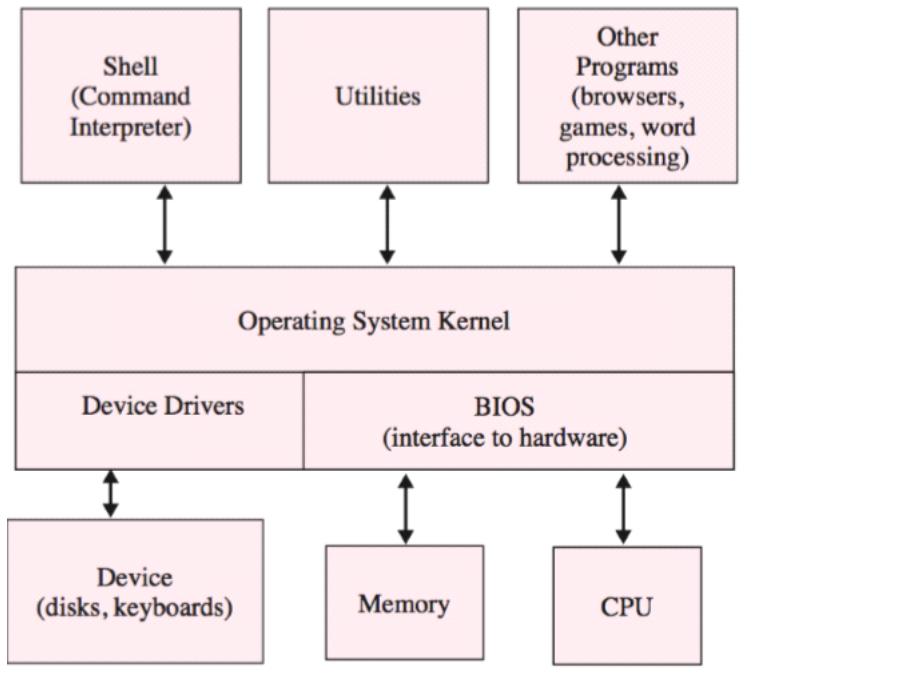
一个分层的操作系统



分层的方式看操作系统



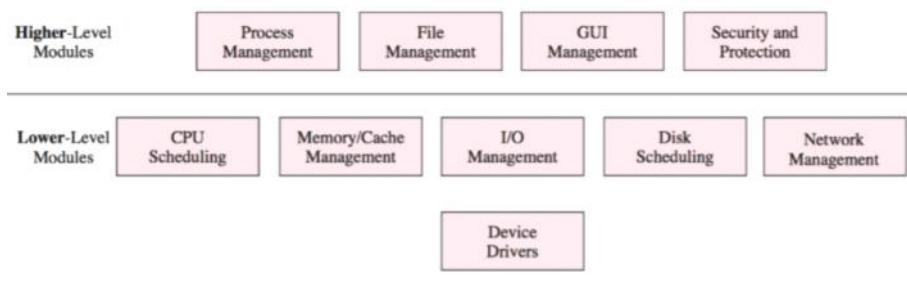
OS的PC模型



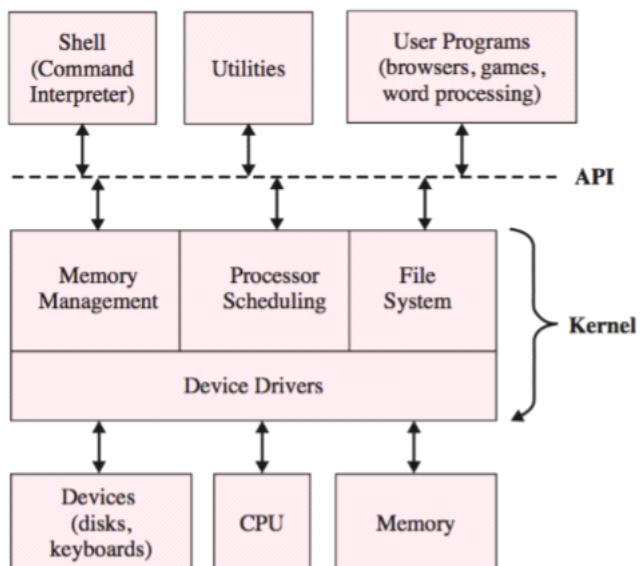
## 主要OS模型

Higher level——操作系统中和软件关系密切的

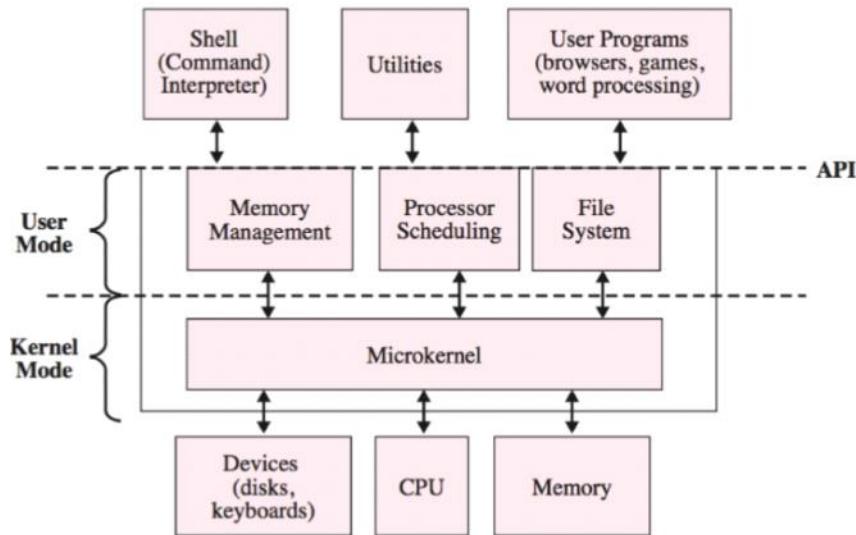
Lower lever——操作系统中和硬件关系密切的



## OS的分层模型

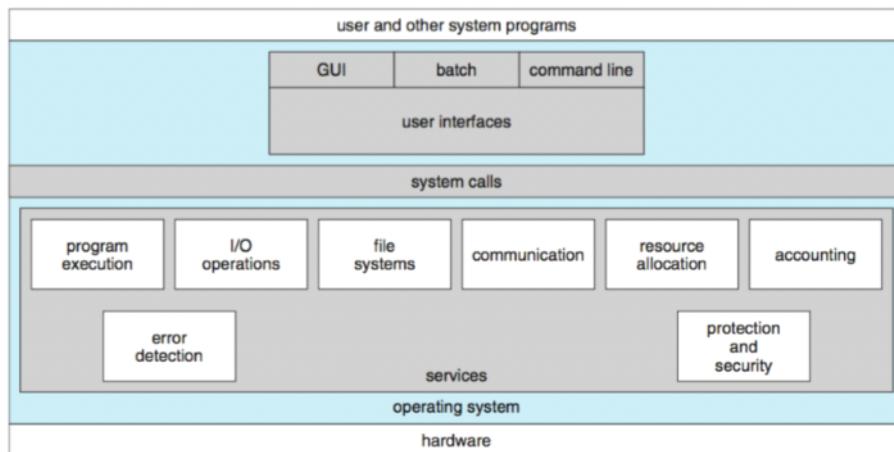


## OS的微核microkernel模型



## 操作系统提供的各种服务

上面的软件想使用操作系统中的软件时，一定要使用系统调用system calls命令，从而使操作系统执行各种各样的操作



## 不同的使用者所关注的不同的点

对于终端用户：最后真正使用计算机的人

易于使用和学习

适应用户的做事风格

输入动态响应

提供大量的视觉线索

没有不愉快的意外(例如，删除一个文件没有警告)

统一的方式做同样的事情(例如，移动图标或向下滚动窗口-在不同的地方)

做一件事的其他方法(例如，一些用户喜欢使用鼠标，另一些用户喜欢使用键盘)

对于应用程序员：目前所谓的码农和程序员

易于通过程序访问低级操作系统调用(例如，读取击键、在屏幕上绘图、获取鼠标位置)

提供系统的一致程序员视图

易于使用高级操作系统工具和服务(例如，创建新窗口，或从网络读写)

可移植到其他平台

对于系统程序员：对计算机系统程序进行修改更新的人

容易创建正确的程序

容易调试错误的程序

容易维护程序

系统经理和管理员

易于添加或删除设备，如磁盘、扫描仪、多媒体附件和网络连接

提供操作系统安全服务，以保护用户、系统和数据文件

易于升级到新的操作系统版本

易于创建和管理用户帐户

平均响应良好且可预测的

可以支付购买系统的费用

---

操作系统中与软件和用户接触最紧密的部分

用户界面the user interface

为了执行计算机用户请求的操作，操作系统必须能够与这些用户通信。操作系统中处理这种通信的部分通常称为用户界面。

壳Shell——命令行程序 (现在一般都不用这个)

shell通过文本消息(通过键盘和监视器屏幕)与用户通信。

用命令行打复合操作系统的程序语言来执行操作系统的命令

GUI

在ag graphical用户界面(GUI)中，对象以图标的形式显示在显示器上。

GUI允许用户通过使用几种常见的输入设备之一来发出命令。

单击什么图标就可以执行操作系统的相关命令



---

用户界面起着OS和用户中间的中介的作用

用户界面充当中介操作系统的用户界面仅仅充当计算机用户和操作系统的真正核心之间的中介。

用户界面和操作系统内部部分之间的这种区别得到了强调，因为一些OS允许用户在不同的界面之间进行选择，从而为特定的用户获得最舒适的交互。

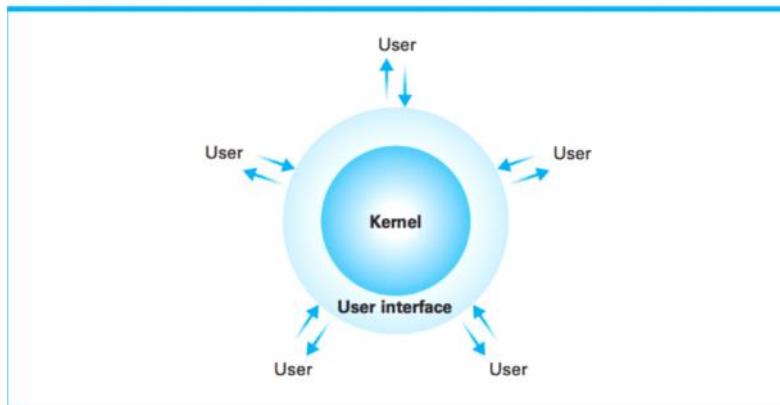
例子

UNIX操作系统的用户可以在各种shell中进行选择，包括Bourne shell、C shell、Korn shell，以及一个名为的GUIX11

在最新版本的Windows操作系统中，DOS cmd.exe shell仍然可以作为一个实用程序使用

苹果的os x保留了一个终端实用程序外壳，这让人想起了该系统的UNIX祖先。

**Figure 3.4** The user interface acts as an intermediary between users and the operating system's kernel



---

操作系统的核心：

### 内核the kernel

操作系统的内核包含那些执行计算机安装所需的基本功能的软件组件。

### 文件管理器file manager

文件管理器协调使用机器的大量存储设施。

更准确地说，文件管理器维护所有存储在大容量存储器中的文件的记录，包括每个文件的位置、哪些用户可以访问各种文件、大容量存储器的哪些部分可以用于新文件或现有文件的扩展。

大多数文件管理器允许将文件分组到一个称为目录directory或文件夹 folder的包中。

这种方法允许用户根据其目的组织文件，方法是将相关文件放在相同的文件夹中。

通过允许目录包含其他目录(称为子目录subdirectories或子文件夹 subfolder)，可以构造层次结构的组织。

目录中的目录链称为目录路径directory path。

其他软件单元对文件的任何访问都是由文件管理器决定的。

### 设备驱动程序device driver

设备驱动程序是与控制器通信的软件单元，用于在附加到机器上的外围端口上执行操作。

每个设备驱动程序都针对其特定类型的设备进行了独特的设计，并将通用请求转换为分配给该驱动程序的设备所需的更技术性的步骤。

## 内存管理器memory manager

内存管理器协调机器的主内存使用。

在要求计算机一次只执行一个任务的环境中，执行当前任务的程序被放置在主存储器中预定的位置上执行，然后由执行下一个任务的程序代替。

在多用户或多任务环境中，要求计算机同时处理许多需求，许多程序和数据块必须同时驻留在主内存中。

因此，内存管理器必须为这些需求查找和分配内存空间，并确保每个程序的操作都限制在程序的分配空间内。

此外，随着不同活动需求的变化，内存管理器必须跟踪那些不再占用的内存区域。

## 进程管理器process management

进程管理典型的分时/多任务计算机运行许多进程，所有这些进程都在争夺计算机的资源。

操作系统的任务是管理这些进程，每个进程的资源(外围设备，在主存空间，访问文件，以及获得一个CPU)，它需要独立的过程不干扰对方，这进程需要交换数据能够这样做。

### 进程管理器下分

调度器scheduler和分配器dispatcher

#### 调度器scheduler

在多用户或多任务系统中，调度器确定要考虑哪些进程来执行。

哪个进程执行，哪个不执行

#### 分配器dispatcher

在多用户或多任务系统中，分配器控制这些进程的时间分配。

### 调度器和分配器

与协调进程执行相关的任务由os内核中的调度器和分配器处理。

#### 调度器的任务

调度器维护计算机系统中出现的过程的记录，将新过程引入这个池pool，并从池中删除已完成的过程。

当用户请求执行应用程序时，将该应用程序的执行添加到当前进程池的是调度器。

为了跟踪所有进程，调度器在主内存中维护一个名为process table的信息块

#### 调度器的任务

每次请求程序执行时，调度器都会在进程表中为该进程创建一个新条目。

这个条目包含分配给进程的内存区域(从内存管理器获得)、进程的优先级以

及进程是否已经准备好或正在等待等数据/信息。  
如果它处于可以继续前进的状态，则过程以及准备就绪;如果它的进程当前被延迟，则它将等待，直到某些外部事件发生，  
例如完成大规模存储操作、按键盘上的键或来自另一个过程的消息到达。

## 分配器的任务

分配器管理调度好的进程的执行。

在分时/多任务系统中，这个任务是通过多程序设计来完成的;也就是说，将时间分成短段，每段称为一个时间片time slice(通常以毫秒或微秒为单位)，然后在允许每个过程执行一个时间片时，在进程之间切换CPU的注意力。

从一个进程切换到另一个进程的过程称为进程切换process switch(或上下文切换context switch)。

---

## 进程process的概念:

进程一个程序和它的执行之间的区别

现代os最基本的概念之一就是程序和执行程序的活动之间的区别。

前者是一组静态的方向(指令)，而后者是一个动态的活动，其性质随着时间的推移而变化。

## 进程process

在操作系统控制下执行程序的活动称为进程process

与进程关联的是活动的当前状态，称为进程状态process state

## 进程状态process state

进程状态包括正在执行的程序的当前位置(即程序计数器PC的值)以及其他CPU寄存器和相关内存单元中的值。

过程状态是机器在特定时间的快照。

在程序执行的不同时间(在过程的不同时间)，将观察到不同的快照(不同的过程状态)。

---

## 进程的概念

进程process：在执行过程进程中动态表示程序

进程管理process management：活动的进程跟踪信息的行为

进程状态process states：在操作系统管理的过程中，陈述进程移动的概念阶段

进程控制块process control block(PCB)：操作系统用来管理进程信息的数据结构

上下文切换context switch：当一个进程从CPU中移除而另一个进程取而代之时发生的寄存器信息交换

---

## PCB进程控制块用途

唯一的进程标识符 (给各种进程起名字)

处理优先级信息

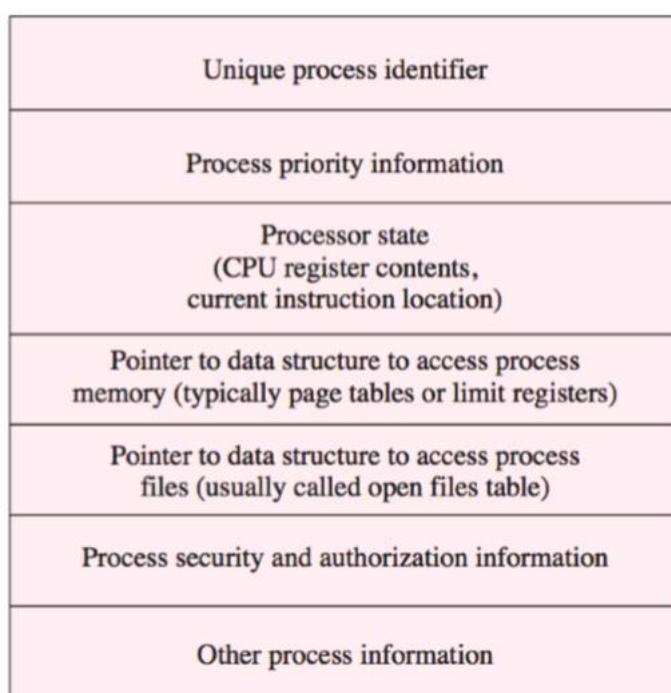
进程状态(CPU寄存器内容, 当前指令位置)

指向访问进程内存的数据结构的指针(通常是页表或限制寄存器)

访问进程文件的数据结构指针(通常称为打开文件表)

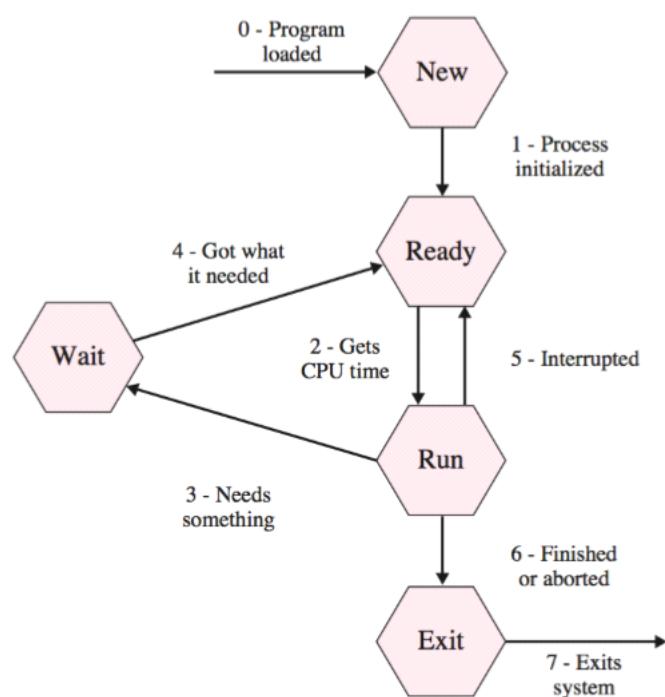
处理安全性和授权信息

其他进程信息

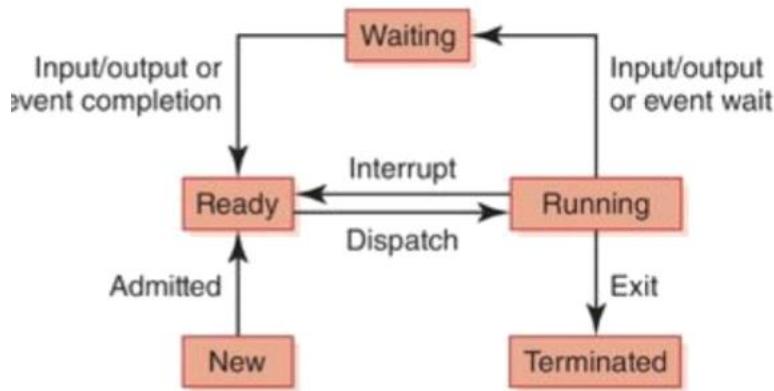


---

## 进程说明和进程转换



## 过程循环process life cycle



## 中断处理interrupt handling

每次分配器将一个时间片分配给一个进程时，它都会启动一个计时器电路，通过生成一个称为中断的信号来指示这个时间片的结束。

分配器给该进程分配的时间到了，就会自动启动中断处理程序  
中断处理器把上一个进程的东西该保存的保存，该吊起的吊起  
有中断寄存器把中断时的状态保存下来，以便下次执行的时候找到中断前该  
程序的状态

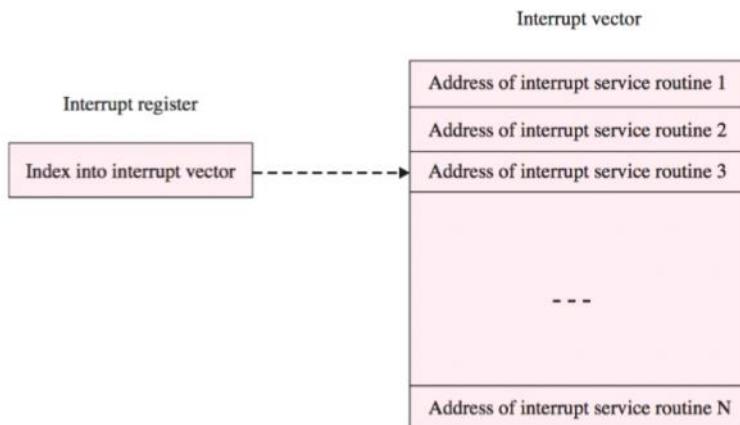
当CPU接收到中断信号时，它完成当前的机器周期，保存当前进程中的位置，并开始执行一个称为中断处理程序interrupt handler的程序，该程序存储在主内存中预定的位置。

这个中断处理程序是分配器的一部分，它描述了调度器应该如何响应破坏信号。

因此，中断信号的作用是抢占当前进程并将控制权转移回调度程序。

此时，调度程序从进程表中选择就绪进程中具有最高优先级的进程进程(由调度器决定)，重新启动计时器电路，并允许所选过程开始它的时间片。

多编程系统成功的关键是能够停止进程，然后重新启动进程。

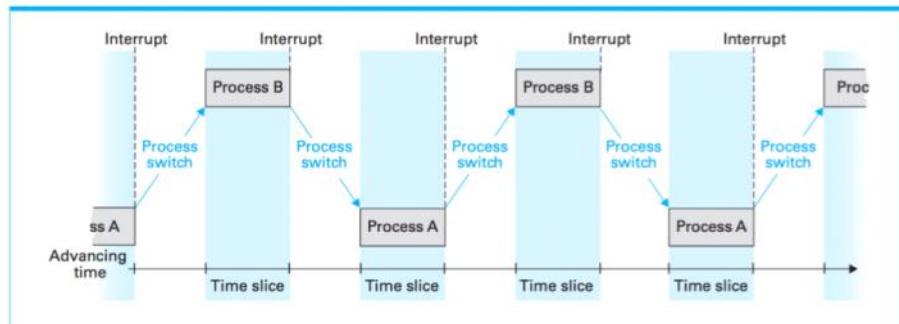


进程状态和进程转换的简化图

分配器给程序A和B分配好时间

谁的时间到了中断谁，然后转化进程

Figure 3.6 Multiprogramming between process A and process B



### 处理进程之间的竞争

计算机中，CPU和硬盘只有一个，而进程有非常多，进程需要的资源远远超过与计算机中能提供的资源

要对资源进行合理的分配

资源分配resource allocation

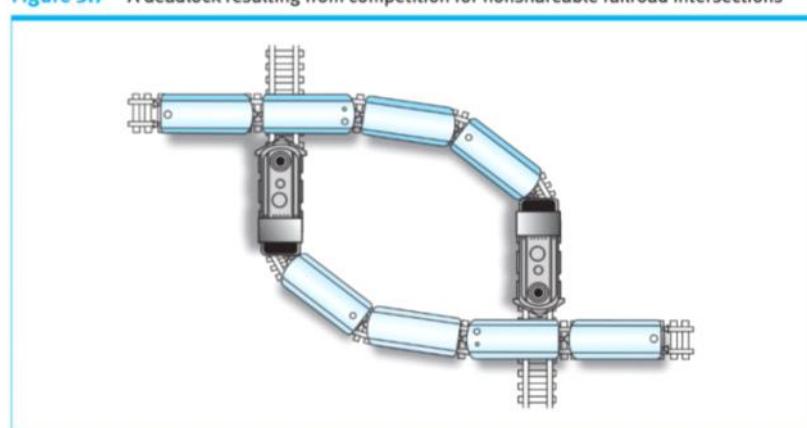
操作系统的一个重要任务是将机器的资源分配给系统中的进程。

这里我们使用广义的术语资源，包括机器的外围设备以及机器本身的特性

资源竞争例如:死锁deadlock(两个或多个进程被阻止前进，因为每个进程都在等待分配给另一个进程的资源)

为了构建可靠的操作系统，我们必须开发出涵盖所有可能的偶发事件的算法，不管这些偶发事件看起来有多么微不足道。

Figure 3.7 A deadlock resulting from competition for nonshareable railroad intersections



死锁我们没有给出死锁的正式定义，而是给出了一个示例。

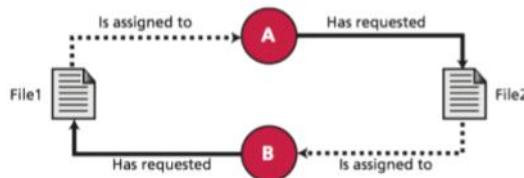
在进程A往一个文件里写东西的时候，另外一个进程B是不能读取它的，因为当前文件的东西正在改变，第二个进程是无法读取无法改变的  
这也就是打开一个word文档后，想更改名字，扔进回收站都不行的原因

假设有两个过程，A和B。进程A持有一个文件(即，文件1被分配给A)，在它获得另一个文件(即，A请求了文件2)之前不能释放它。

进程B持有文件2(即，将文件2分配给B)，在它拥有文件1(即，B请求了文件1)之前不能释放它。

大多数系统中的文件是不可共享的——当一个进程使用一个文件时，另一个进程也不能使用它。如果在这种情况下没有规定强制进程释放文件，就会创建死锁(图7.16)。

Figure 7.16 Deadlock

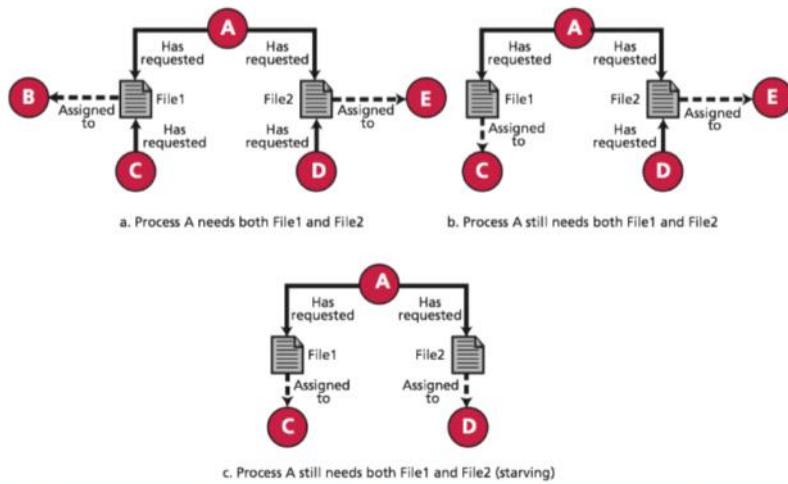


### 饥饿Starvation

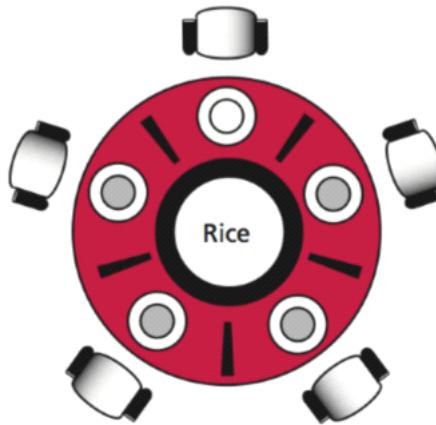
饥饿是僵局的反面。

当操作系统对一个进程施加过多的资源限制时，就会发生这种情况。例如，设想一个操作系统，它指定一个进程在运行之前必须拥有所需的资源。在图7.18中，假设进程A需要两个文件File1和File2。进程B使用File1，进程E使用File2。进程B首先终止并释放File1。无法启动进程A，因为File2仍然不可用。此时，只需要File1的进程C被允许运行。现在进程E终止并释放File2，但是进程A仍然不能运行，因为File1不可用。

Figure 7.18 Starvation



### 饮食哲学家的问题·僵局、生命冻结和饥饿



Edsger Dijkstra提出了一个经典的饥饿问题。五位哲学家坐在圆桌旁，如图。每个哲学家需要两根筷子来吃一碗米饭。然而，一个或两个筷子可以被邻居使用。如果两根筷子不能同时使用，哲学家可能会饿死。

---

多道程序设计multiprogramming：在主存中同时保存多个程序以争夺CPU的技术

内存管理memory management：跟踪程序在主存中是如何以及在何处加载的行为

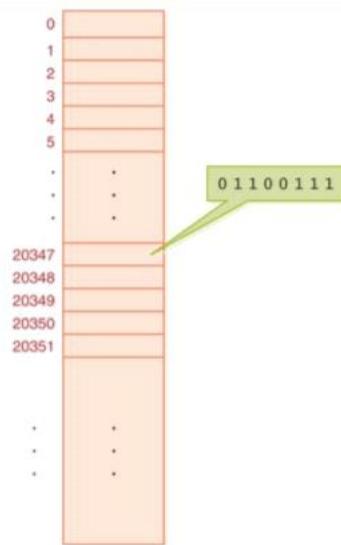
逻辑地址logical address：是对存储值的引用，该值相对于进行引用的程序

物理地址physical address：主存储器设备中的实际地址

地址绑定address binding：从逻辑地址到物理地址的映射

---

内存管理memory management (操作系统中最复杂的管理)



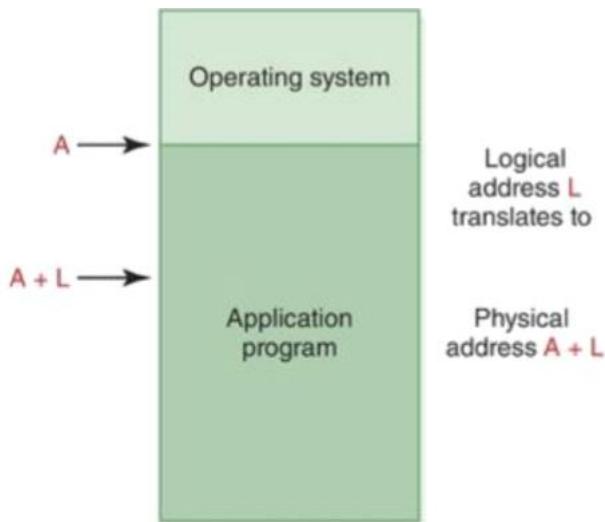
**FIGURE 10.4** Memory is a continuous set of bits referenced by specific addresses

主存分为两个部分



---

将逻辑地址（抽象的）绑定到物理地址



---

单一连续内存管理single contiguous memory management: 将程序装入一个连续内存区域的内存管理方法

固定分区技术fixed-partition technique: 一种内存管理技术，在这种技术中，内存被划分成特定数量的分区，并将程序装入这些分区中

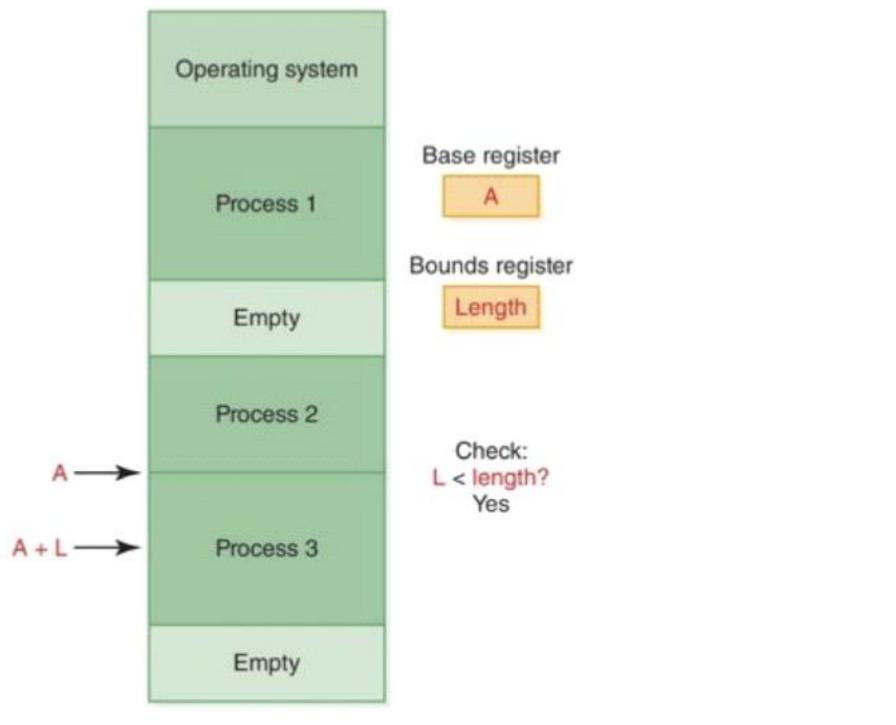
动态分区技术dynamic-partition technique: 一种内存管理技术，在这种技术中，内存根据需要划分成分区以容纳程序

基寄存器base register: 保存当前分区起始地址的寄存器

界限寄存器bounds register: 保存当前分区长度的寄存器

---

分区内存管理中的地址解析



## 文件系统管理

文件file: 一个命名的数据集合，用于组织辅助内存

文件系统file system: 操作系统管理的文件的逻辑视图

目录directory: 一个命名的文件组

文本文件text file: 包含字符的文件

当然还有音频文件，视频文件，图像文件等等

最终文件都是会以二进制文件存储在电脑里

二进制文件binary file: 以特定格式包含数据的文件，需要对其位进行特殊解释

文件类型file type: 文件中包含的特定类型的信息，如Java程序或Microsoft Word文档

文件拓展名file extension: 表示文件类型的一部分

Extensions	File type
.txt	text data file
.mp3, .au, .wav	audio file
.gif, .tiff, .jpg	image file
.doc, .wp3	word processing document
.java, .c, .cpp	program source files

**FIGURE 11.1** Some common file types and their extensions

## 文件系统具体介绍

文件:

从用户的角度来看，文件是可以写入辅助内存secondary memory的最小数据量。

将所有内容组织到文件中为数据存储提供了统一的视图。

文件的内容:

一个文件可以看作是位、字节、行或记录的序列，这取决于您如何看待它。与内存中的任何数据一样，必须先对存储在文件中的位进行解释，然后才有意义。

文件的创建者决定文件中的数据如何组织，文件的任何用户都必须理解该组织。

文本文件和二进制文件

所有文件可以大致分为文本文件或二进制文件。

在文本文件中，数据字节被组织为ASCII或Unicode字符集中的字符。

二进制文件需要基于文件中的数据对位进行特定的解释。

“文本文件”和“二进制文件”

术语“文本文件”和“二进制文件”有点误导人。它们似乎暗示文本文件中的数据不是作为二进制数据存储的。

然而，最终，计算机上的所有数据都是以二进制数字存储的。这些术语指的是这些位的格式：8位或16位的块，解释为字符或其他特殊格式。

文件类型

大多数文件，无论是文本格式还是二进制格式，都包含特定类型的数据。

文档中包含的数据类型称为文件类型。

大多数操作系统识别特定文件类型的列表。

文件扩展名

指定文件类型的常见机制是将类型指定为文件名称的一部分。

文件名通常由一个句点分隔成两个部分：主名main name和文件扩展名file extension。扩展名指示文件的类型。

文件类型允许操作系统以对该文件有意义的方式对该文件进行操作。它们通常也使用户的生活更容易。

---

文件操作file operation

在操作系统的帮助下，可以对文件执行以下操作之一：

创建一个文件

删除一个文件

打开一个文件

关闭一个文件

从文件中读取数据  
将数据写入文件  
在文件中重新定位  
当前文件指针将数据追加到文件末尾  
截断文件(删除其内容)  
重命名一个文件  
复制一个文件

---

### 文件访问file access

文件中的数据可以通过几种不同的方式访问。

#### 顺序文件访问sequential file access

最常见的访问技术，也是最容易实现的，是顺序文件访问，它将文件视为线性结构。

它要求按顺序处理文件中的数据。

读写操作根据读取或写入的数据量移动当前文件指针。

有些系统允许将文件指针重置为文件开头，并/或通过一定数量的记录向前或向后跳过。

#### 直接文件访问direct file access

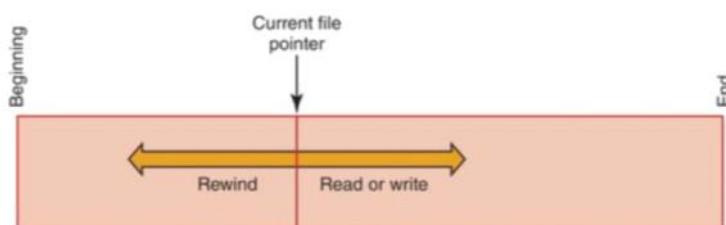
具有直接文件访问权限的文件在概念上划分为编号的逻辑记录。

直接访问允许用户通过指定记录号来将文件指针设置为任何特定的记录。

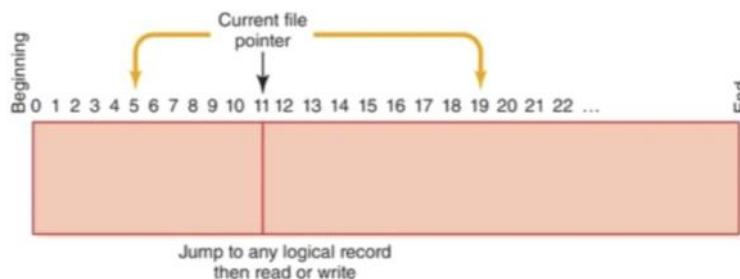
因此，用户可以按需要的任何特定顺序读写记录。

顺序文件访问：以线性方式访问文件中的数据的技术

直接文件访问：通过指定逻辑记录号直接访问文件中的数据的技术



**FIGURE 11.2** Sequential file access



**FIGURE 11.3** Direct file access

## 文件保护file protection

在多用户系统中，文件保护是最重要的。也就是说，我们不希望一个用户能够访问另一个用户的文件，除非这种访问是特别允许的。

确保有效的文件访问是操作系统的责任。不同的操作系统以不同的方式管理它们的文件保护。

## 文件保护机制file protection mechanism

文件保护机制决定了谁可以使用文件，以及文件的一般用途。

### UNIX中的文件保护

UNIX操作系统中的文件保护设置分为三类:所有者owner、组group和世界world。

在每个类别下，用户可以确定文件是否可以读取、写入和/或执行。

在这种机制下，如果可以写入文件，也可以删除文件。

	Read	Write/Delete	Execute
Owner	Yes	Yes	No
Group	Yes	No	No
World	No	No	No

## 目录directories

目录是一个命名的文件集合。

它是一种对文件进行分组的方法，以便您能够以逻辑方式组织它们。

操作系统必须小心地跟踪目录及其包含的文件。

## 目录文件directory files

在大多数操作系统中，目录以文件的形式表示。目录文件包含关于目录中所有文件的数据。

对于任何给定的文件，目录文件包含文件名、文件类型、存储文件的磁盘上的地址、文件的当前大小、保护数据以及描述文件何时创建和最后一次修改的数据。

## 目录树directory trees

文件目录可以包含在另一个目录中。

包含另一个目录的目录通常称为父目录parent directory，其中的目录称为子目录subdirectory。

为了可视化层次结构，通常将文件系统视为目录树，显示其他目录中的目录和文件。

最高级别的目录称为根目录root directory。

目录树：显示文件系统的嵌套目录组织的结构

根目录：最上层的目录，其中包含所有其他目录

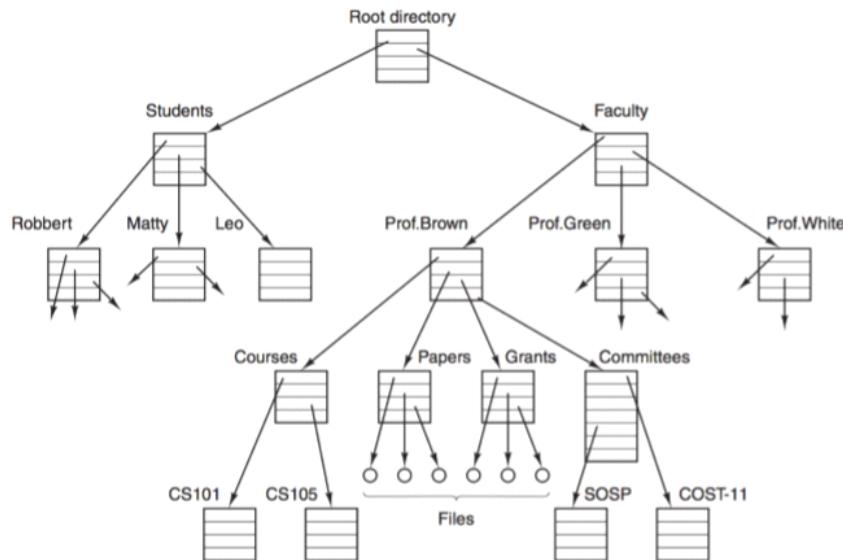


Figure 1-14. A file system for a university department.

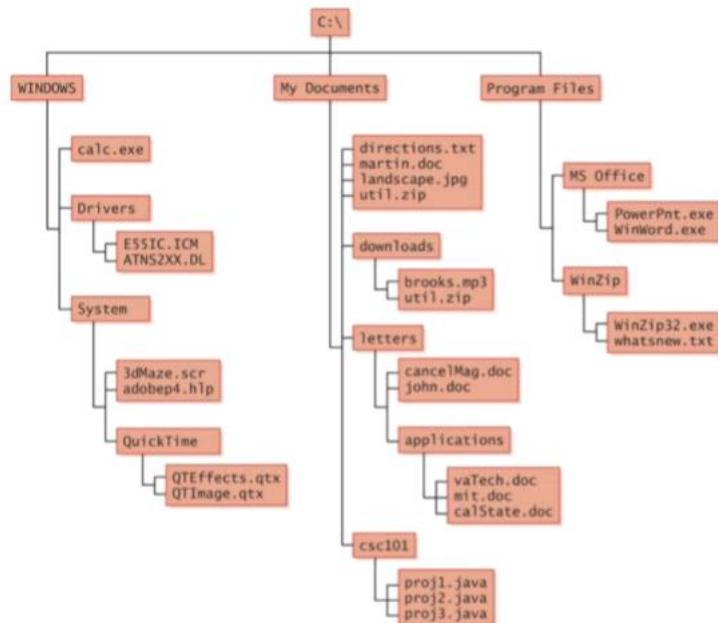


FIGURE 11.4. A Windows directory tree

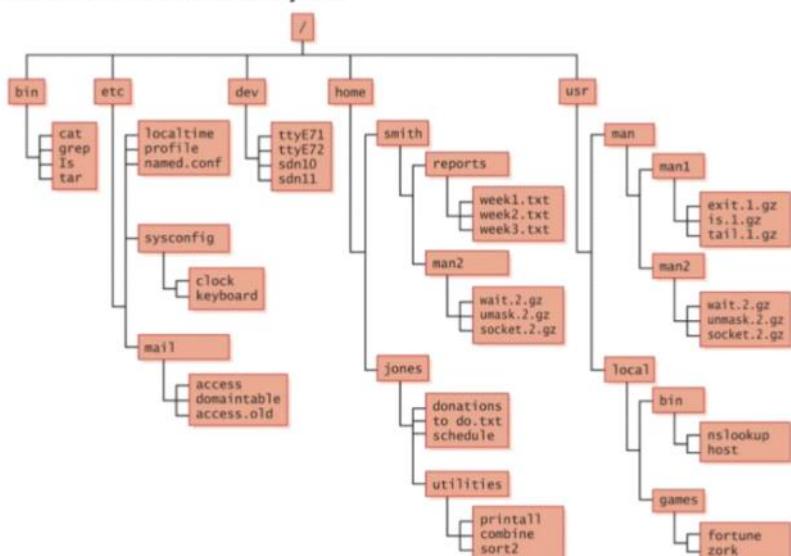
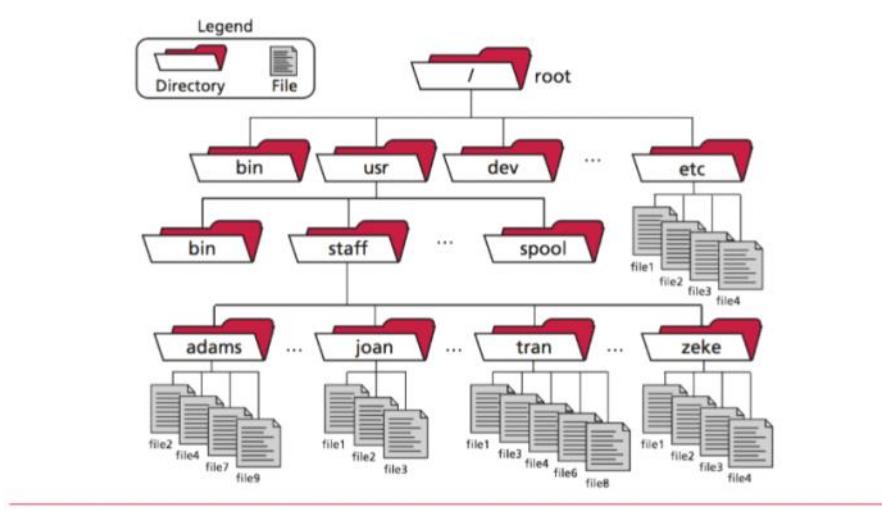


FIGURE 11.5 A UNIX directory tree

Figure 13.14 An example of directory system in UNIX



操作系统（民用）基本就三类

UNIX操作系统

Windows操作系统

MAX操作系统

UNIX操作系统

UNIX最初是由贝尔实验室计算机科学研究中心的汤姆森和里奇在1969年开发的。此后，UNIX经历了许多版本。然后。它已经在计算机程序员和计算机科学家中成为一种流行的操作系统。它是一个非常强大的操作系统，具有三个突出的特性。首先，UNIX是一种可移植的操作系统，可以从一个平台转移到另一个平台，而不需要做很多更改。原因是它主要是用C语言编写的，而不是专门针对特定计算机系统的机器语言。其次，UNIX有一组功能强大的实用工具(命令)，可以将它们组合在一起(在称为脚本的可执行文件中)来解决许多需要在其他操作系统中进行编程的问题。第三,它是与设备无关的,因为它包括操作系统中的设备驱动程序,这意味着它可以很容易地配置为运行任何设备

UNIX是一种多用户,多处理、可移植操作系统为了便于编程,文本处理、沟通、和许多其他任务,预计从一个操作系统。它包含数百个简单的、单一用途的函数，可以组合起来执行几乎所有可以想象得到的处理任务。它的灵活性被它在三种不同的计算环境中使用的事实在证明:独立的个人环境、分时系统和客户机-服务器系统。

UNIX是一个多用户、多处理的系统。可移植的操作系统。它的设计目的是方便编程、文本处理和通信。

Figure 7.20 Components of the UNIX operating system

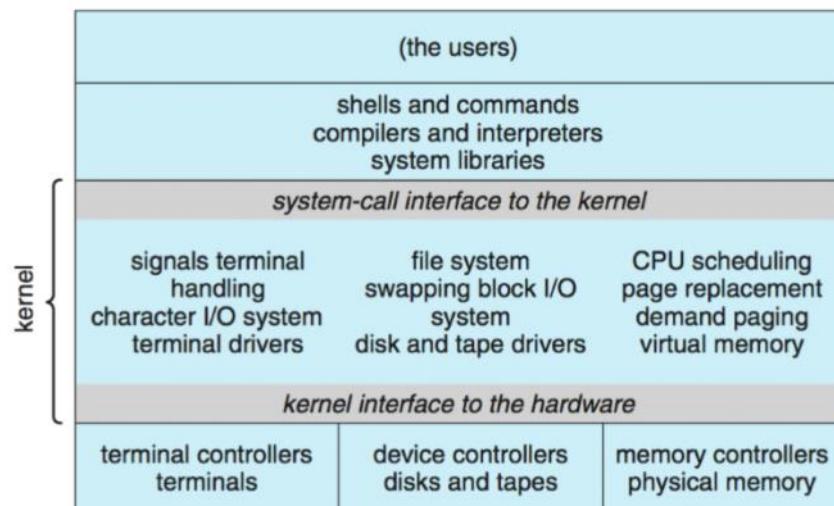
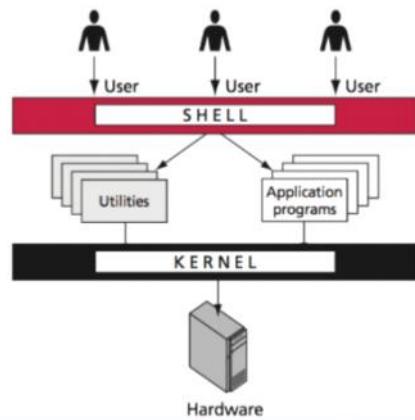
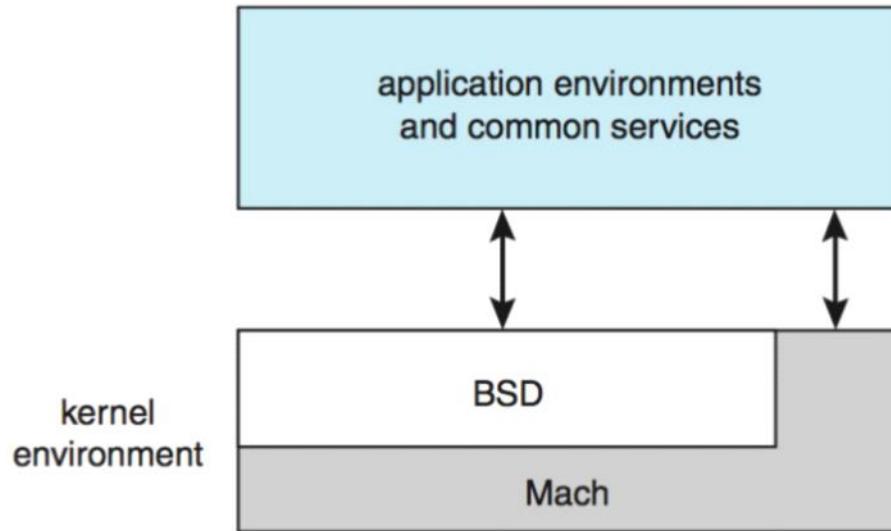


Figure 2.13 Traditional UNIX system structure.

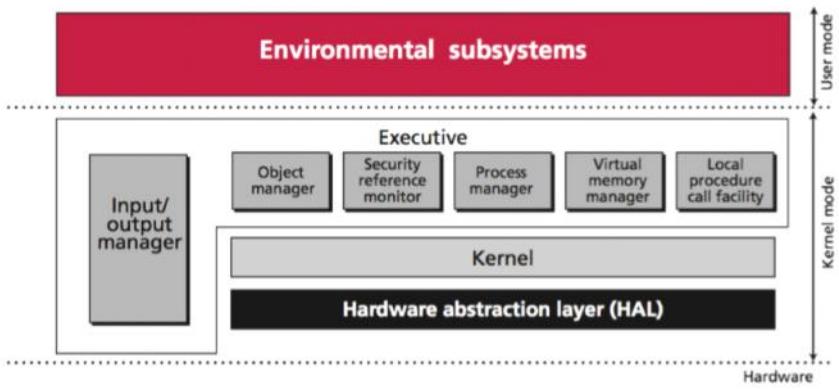
内核kernel

信号终端	文件系统	CPU调度
处理	交换块I/O	分页替换
I/O系统	系统	请求分页
终端驱动程序	磁盘和磁带驱动器	虚拟内存



**Figure 2.16** The Mac OS X structure.

**Figure 7.21** The architecture of Windows



# System software and application software

2019年12月4日 13:22

系统软件和应用软件 (数据库必考)

---

系统软件system software

产生最终用户可接受的系统所需的全部软件。

为便于操作和维护计算机系统及其相关程序而设计的软件;例如，操作系统、汇编程序、实用程序。”

管理计算机系统并与硬件交互的程序

---

应用软件application software

应用程序的总称。

为满足用户特定需要而设计的软件;例如，用于导航、工资单或过程控制的软件。

帮助我们解决现实问题的程序

应用程序application program

“任何特定于某一特定角色并对该角色的执行做出直接贡献的程序。”

---

软件分类

软件 应用

    系统 实用

        操作系统 用户界面

            内核

---

系统软件和应用软件的区别

系统软件和应用软件之间的区别不是那么明确和严格。

例如:微软IE浏览器 微软IE是应用软件还是Windows操作系统的一部分?

如果是前者，那么微软应该把IE作为一个单独的产品销售。

如果是后者，微软可能会在Windows操作系统中出售IE。

这一区别对于判断微软是否违反反垄断法具有重要意义。

---

系统软件的特点

控制和管理资源。

与硬件交互。

为所有用户和应用软件提供公共服务。

我们已经学习了系统软件  
汇编器assemblers、编译器compilers、操作系统OS(包括加载器  
loaders、设备驱动程序device drivers等)。

其他系统软件  
编辑器editor、链接器linkers、调试器debuggers、存储媒体格式器  
storage-media formatters、网络软件net work software和各种实用工  
具。

---

应用软件的特点  
每一个应用软件都有特殊的目的  
每个应用软件都是解决一个特殊的问题  
不为所有用户服务。

应用软件  
科学计算软件，工程计算软件，商业计算软件，CAD软件。数据库管理  
系统(DBMS)等。  
办公软件(日历、调度器、记事本、计算器、阅读器、文字处理器、演示  
文稿、表格计算、绘图、字典等)。  
互联网软件(聊天，电子邮件，浏览器，下载。终端等)。  
多媒体软件(图片处理，音乐播放器，视频播放器，录音，语音编辑，视  
频编辑等)。  
娱乐软件(游戏等)。

---

数据库系统:是什么?

数据库系统  
通常严格地说，是指利用数据库管理系统的设施，在计算机系统中保存  
的大量信息。所有信息的访问和更新都将通过该软件提供的设施进行，  
记录日志文件、数据库恢复和多访问控制也将通过该软件提供的设施进  
行。  
一种软件系统，它为组织和管理某些特定应用程序或相关应用程序组所  
需的大量信息提供全面的设施。

---

数据库database(DB)  
数据库是在结构上、组织上和一起存储在计算系统中的逻辑相关数据的  
集合。  
数据库是一堆数据，这些数据是有逻辑有组织的存储在电脑里的  
Note:“数据”而不是“信息”，“逻辑相关”，“在结构上、组织上和一起存  
储”，以及“在计算系统中”

## 数据库管理系统database management system(DBMS)

数据库管理系统(DBMS)是运行在计算系统上的软件系统，它提供了一种系统的方法来管理数据库中存储的数据并控制对数据库的访问。

DBMS是一种通用的软件系统，它简化了在各种用户和应用程序之间定义、构造、操作和共享数据库的过程。

它和应用不相关，独立于应用的，各种各样的数据库可以用同一种数据库管理系统来管理，但是本质上它还是会被程序调用引用，是一种应用软件

## 数据库系统data base system(DBS)

数据库系统由数据库和管理数据库的数据库管理系统组成。

数据管理包括定义数据存储结构和提供数据操作机制。

此外，数据库系统必须确保存储的数据的安全性和安全性，尽管系统崩溃或尝试未经授权的访问。

Note: 使用数据库管理系统来管理不同的数据库可能会创建和维护不同的数据库系统。

## 数据库应用(DB application)

数据库应用程序就是在执行过程中与数据库系统交互的程序。

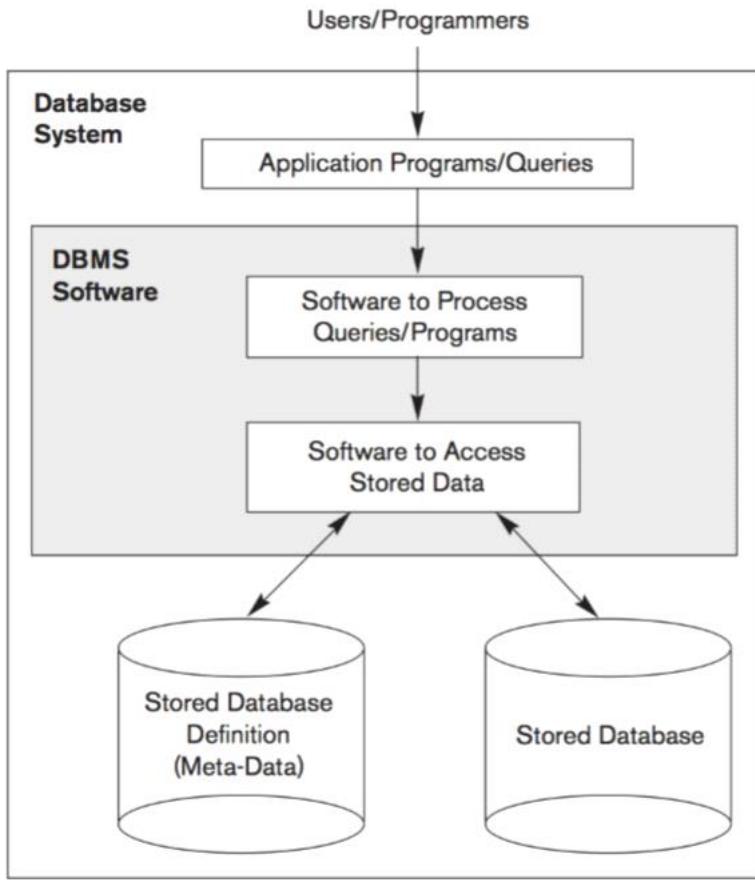
如果一个应用背后有数据库支持，那么它就是一个数据库应用。

## 必考重点

数据库系统DBS=数据库管理系统DBMS（管理数据库的应用软件）+数据库DB

---

一个简化的数据库系统环境（重点）



这里有两种数据库

Meta-Data是管理数据库的数据库，与存储的数据库是不一样的

---

一个数据库的例子

**STUDENT**

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

**COURSE**

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

**SECTION**

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

**GRADE\_REPORT**

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

**PREREQUISITE**

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

文件组织file organization与数据库组织database organization

文件与数据库组织的对比

a.面向文件的信息系统———对应

客户记录--客户部门

工资记录--工资部门

员工记录--人事部门

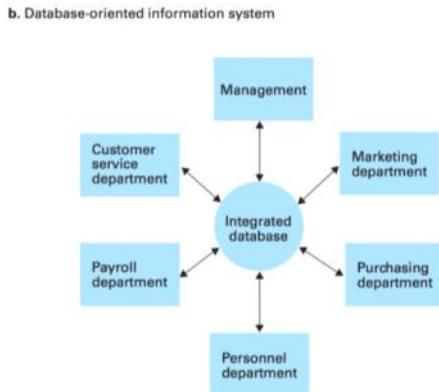
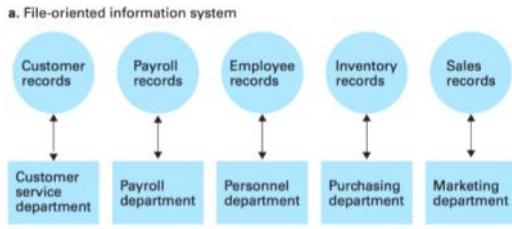
库存记录--采购部门

销售记录--市场营销部门

数据是孤立的，每一个系统对应一个文件

b.面向数据库的信息系统

所有部门都从数据库内提取信息



## 基于文件的系统

不用数据库管理系统来管理数据的系统就叫基于文件的系统

为最终用户执行服务的应用程序的集合。

每个程序定义和管理自己的数据。

## 用文件处理数据的缺点

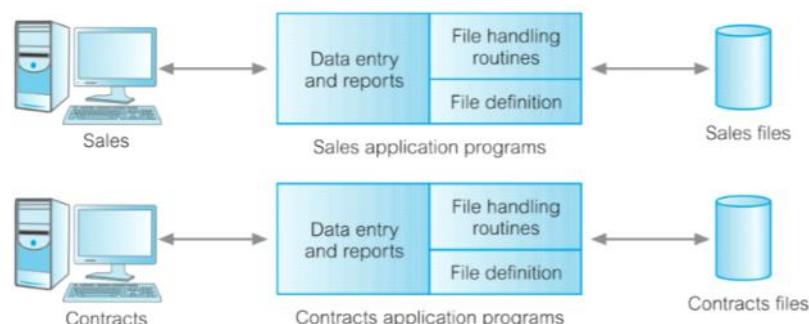
数据冗余和不一致;查阅资料有困难;数据隔离;诚信问题;原子性问题;并发访问异常;

## 数据库系统存在的必要性

文件处理数据的缺点促使数据库系统的发展。

数据库系统的概念和算法使它们能够解决文件处理系统的问题。

## 例子



销售-----资料输入及报告，文件处理例程，文件定义-----销售文件  
销售应用程序

合同-----资料输入及报告，合同处理例程，合同定义-----合同文件  
合同应用程序

#### Sales Files

**PropertyForRent** (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)

**PrivateOwner** (ownerNo, fName, lName, address, telNo)

**Client** (clientNo, fName, lName, address, telNo, prefType, maxRent)

#### Contracts Files

**Lease** (leaseNo, propertyNo, clientNo, rent, paymentMethod, deposit, paid, rentStart, rentFinish, duration)

**PropertyForRent** (propertyNo, street, city, postcode, rent)

**Client** (clientNo, fName, lName, address, telNo)

---

### 文件处理数据的缺点

#### 数据冗余(重复)和不一致性data redundancy (duplication) and inconsistency

由于不同的程序员在很长一段时间内创建文件和应用程序，不同的文件可能具有不同的结构，程序可能用几种编程语言编写。

而且，相同的数据可能在几个地方(文件)重复。

这种冗余导致更高的存储和访问成本。

此外，可能会导致数据不一致;也就是说，相同数据的不同副本可能不再一致。

#### 数据分离和隔离data separation and isolation

传统的文件处理环境不允许以方便和高效的方式检索所需的数据的方式。

由于数据分散在不同的文件中，文件可能采用不同的格式，因此编写新的应用程序来检索适当的数据非常困难。

当数据被隔离在单独的文件中时，访问应该可用的数据就更加困难了。

应用程序开发人员必须同步处理两个或多个文件，以确保提取了正确的数据。

#### 数据相关data dependence

数据文件和记录的物理结构和存储在应用程序代码中定义。

这意味着很难对现有结构进行更改。

程序员需要识别所有受影响的程序，修改它们，然后重新测试它们。

显然，这个过程可能非常耗时并容易出错。

数据出问题了可能直接影响到管理使用该数据的程序

基于文件的系统的这种特性称为程序-数据依赖program-data dependence。

#### 完整性问题integerity problems

存储在数据库中的数据值必须满足某些类型的一致性约束。

开发人员通过在各种应用程序中添加适当的代码来在系统中实施这些约束。

但是，当添加新的约束时，很难更改程序来强制执行它们。

当约束涉及来自不同文件的多个数据项时，问题就变得复杂了。

## 原子性问题atomicity problems

一个计算机系统，就像任何其他设备一样，容易出故障。在许多应用程序中，如果发生故障，必须将数据恢复到故障之前的一致状态。

原子性：对一个数据的操作，要么就成功，要么就失败，没办法保存在中间的状态

也就是说，失败的操作必须是原子性的，即。它必须全部发生，否则就根本不会发生。

在传统的文件处理系统中很难保证原子性。

## 并发访问异常concurrent-access anomalies

为了提高系统的整体性能和更快的响应速度，许多系统允许多个用户同时更新数据。

在这样的环境中，并发更新的交互是可能的，并且可能导致不一致的数据。

为了防止这种可能性，系统必须保持某种形式的监督。

但是提供监督是困难的，因为数据可能被许多以前没有被协调的不同的应用程序访问。

## 安全问题security problem

并不是数据库系统的每个用户都能够访问所有数据。

每一种用户只能访问数据库系统中的指定数据

但是，由于应用程序是以一种特定的方式添加到文件处理系统中的，因此很难实施这种安全约束。

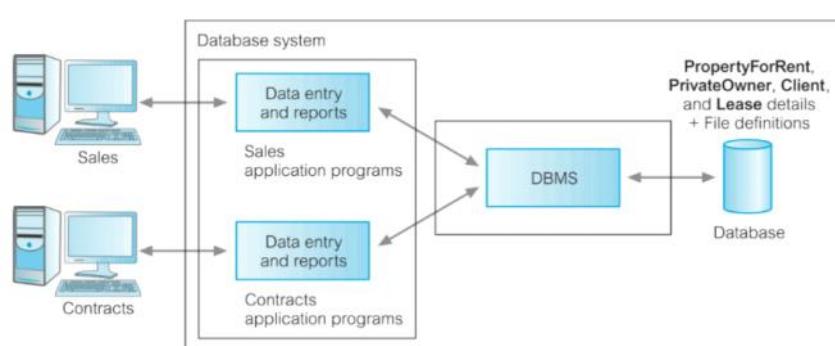
## 基于文件的方法的两个内在因素

数据的定义嵌入到应用程序中，而不是抽象定义，单独存储。

除了应用程序强加的数据访问和操作之外，没有其他控制。

---

## 例子：数据处理数据库系统



## 图9.2数据库实现的概念层（多层抽象）

用户

↓从应用程序的角度来看数据库

应用软件

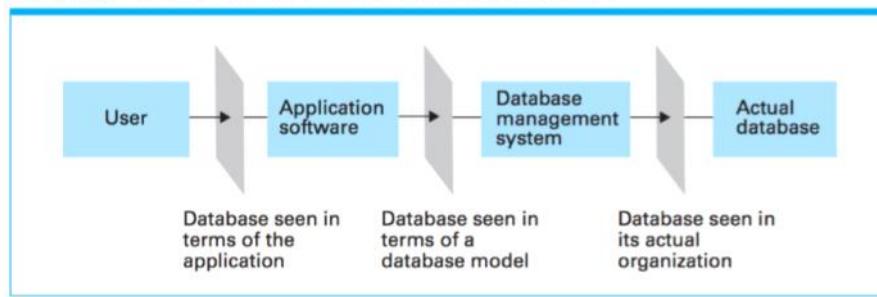
↓从数据库模型的角度来看数据库

数据库管理系统

↓从其实际组织中看到的数据库

实际数据库

Figure 9.2 The conceptual layers of a database implementation



数据库管理系统（DBMS）作为数据库与用户的接口

DBMS作为接口

用户和应用程序不直接操作数据库。

数据库的实际操作是由DBMS执行的。

应用程序和DBMS之间的二分法的好处

它允许构造和使用抽象工具。如果数据库实际上是如何存储的细节在

DBMS中是独立的，那么应用程序软件的设计可以大大简化。

这样的组织为控制对数据库的访问提供了一种方法。

DBMS的功能

为用户服务

允许用户创建/定义新的数据库并指定他们的模式(数据的逻辑结构)，通常通过专门的数据定义语言data-definition language(DDL)实现。

允许用户从数据库中插入、删除、更新和检索数据，通常是通过数据操作语言data manipulation language(DML)。

拥有一个用于所有数据和数据描述的中央存储库，DML就可以为这些数据提供一个通用的查询工具，称为查询语言query language。

## 数据库管理DB management

支持在很长一段时间内存储大量数据，允许对数据进行高效访问和数据库修改。

使数据库在面对故障、多种错误或故意误用时能够持久、恢复。

同时控制对来自多个用户的数据的访问，不允许用户之间发生意外的交互(称为隔离)，也不允许对数据执行部分但不完全的操作(称为原子性)。

例如，它可以提供：

保安系统，防止未经授权的用户进入数据库;维护存储数据一致性的完整性系统;允许数据库共享访问的并发控制系统;恢复控制系统，在硬件或软件故障后将数据库恢复到以前的一致状态;用户可访问的目录，其中包含数据库中数据的描述。

表 1-2 DBMS 的优点

受控的数据冗余	经济合算的规模
数据一致性	平衡各种需求冲突
相同数据量表示更多信息	增强的数据可访问性和响应性
数据共享	提高的生产率
增强的数据完整性	通过数据的独立性增强可维护性
增强的安全性	提高的并发性
强制执行标准	增强的备份和恢复服务

表 1-1 基于文件系统的局限性

数据被分离和孤立
数据存在冗余
数据存在依赖性
文件格式不相容
查询一成不变 / 应用程序需不断翻新

表 1-3 DBMS 的缺点

复杂性高
规模大
DBMS 的费用高
需要附加的硬件费用
转化费用大
性能相对较低
故障带来的影响较大



## 在企业中运用数据库

### 企业信息

销售sales:提供客户、产品和购买信息。

会计accounting:用于支付、收款、账户余额、资产等会计信息。

人力资源human resources:提供有关雇员、工资、工资税和福利的信息，并生成工资支票。

制造manufacturing:管理供应链，跟踪工厂的生产，仓库和商店的库存，以及订单。

在线零售商online retailers:针对上述销售数据，加上在线订单跟踪，生成推荐列表，并维护在线产品评估。

### 银行及金融

银行:用于客户信息、帐户、贷款和银行交易。

信用卡交易:用于信用卡购物和生成月度报表。

**金融:**用于存储有关股票和债券等金融工具的持有、销售和购买的信息;还可以存储实时的市场数据,以支持客户的在线交易和公司的自动交易。

### 大学

学生信息、课程注册、成绩(人力资源、会计等标准企业信息)。

### 航空公司

查询预约及时间表资料。航空公司是第一批以地理分布方式使用数据库的公司。

### 电信

保存通话记录,生成每月账单,保持预付费电话卡的余额,并存储有关通信网络的信息。

### 数据库应用的普遍性

如列表所示,数据库是当今每个企业的重要组成部分,不仅存储大多数企业常见的数据类型,还存储特定于企业类别的数据。

---

## 在日常生活中运用数据库

### 在超市买东西

当您从本地超市购买商品时,可能会访问数据库。

结账助理使用条形码阅读器扫描你的每件商品。

该阅读器链接到一个数据库应用程序,该应用程序使用条形码从产品数据库中查找商品的价格。

然后,应用程序减少库存中此类商品的数量,并在收银机上显示价格。

如果重新排序级别低于指定的阈值,数据库系统可能会自动下订单以获取该项目的更多信息。

### 用信用卡购物

#### 有若干个数据库在支持这个行为

当你用信用卡购物时,助理通常会检查你是否有足够的信用来购物。

这种检查可以通过电话进行,也可以通过与计算机系统相连的卡片阅读器进行。

无论哪种情况,在某个地方都有一个数据库,其中包含有关您使用信用卡购物的信息为了检查您的信用,有一个数据库应用程序使用您的信用卡号来检查您希望购买的商品的价格,以及您本月已经购买的商品的金额是否在您的信用限额内。

确认购买后，购买的详细信息将添加到此数据库中。  
在授权购买之前，数据库应用程序还访问数据库以确认信用卡不在被盗或丢失的信用卡列表中。

### 向旅行社预订假期

当你询问假期时，你的旅行社可能会访问几个包含假期和轻松细节的数据库。

当您预订假期时，数据库系统必须进行所有必要的预订安排。

在这种情况下，系统必须确保两个不同的代理不会预订同一个假期或超额预订航班座位。

旅行社可能有另一个单独的数据库用于开发票。

### 使用本地图书馆

您所在的本地图书馆可能有一个数据库，其中包含图书馆中图书的详细信息、读者的详细信息、预订信息等等。

将有一个计算机化的索引，允许读者根据书名、作者或主题领域找到一本书。

数据库系统处理预订，允许读者预订一本书，并在有书时通过邮件或电子邮件得到通知。

该系统还会向逾期还书的借书者发出提醒。

通常，该系统将有一个条形码阅读器，类似于前面描述的超市使用的阅读器，用于跟踪图书进出图书馆。

### 投保

无论何时您希望投保，您的代理人都可以访问包含各种保险机构数据的几个数据库。

您提供的个人信息，如姓名、地址、年龄以及您是否饮酒或吸烟，将被数据库系统用于确定保险费用。

保险代理人可以搜索几个数据库来找到给你最优惠的保险公司。

### 使用互联网

互联网上的许多站点都是由数据库应用程序驱动的。

谷歌 Facebook Twitter 亚马逊 百度 QQ Weixin(微信)

---

### 数据库例子

包含雇员信息的关系

Figure 9.3 A relation containing employee information

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555
•	•	•	•
•	•	•	•
•	•	•	•

由三个关系组成的雇员数据库

Figure 9.5 An employee database consisting of three relations

EMPLOYEE relation			
Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555

JOB relation			
Job Id	Job Title	Skill Code	Dept
S25X	Secretary	T5	Personnel
S26Z	Secretary	T6	Accounting
F5	Floor manager	FM3	Sales
•	•	•	•
•	•	•	•
•	•	•	•

ASSIGNMENT relation			
Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2010
34Y70	F5	10-1-2009	*
23Y34	S26Z	5-1-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

找到雇员23Y34工作的部门

Figure 9.6 Finding the departments in which employee 23Y34 has worked

EMPLOYEE relation			
Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555
•	•	•	•
•	•	•	•
•	•	•	•

JOB relation			
Job Id	Job Title	Skill Code	Dept
S25X	Secretary	T5	Personnel
S26Z	Secretary	T6	Accounting
F5	Floor manager	FM3	Sales
•	•	•	•
•	•	•	•
•	•	•	•

ASSIGNMENT relation			
Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2010
34Y70	F5	10-1-2009	*
23Y34	S26Z	5-1-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

后面都不考！！！！！

### 数据抽象data abstraction

DBS的主要用途是:提供数据的抽象视图

DBS的主要目的是为用户提供数据的抽象视图。

也就是说，系统隐藏了数据存储和维护的某些细节。

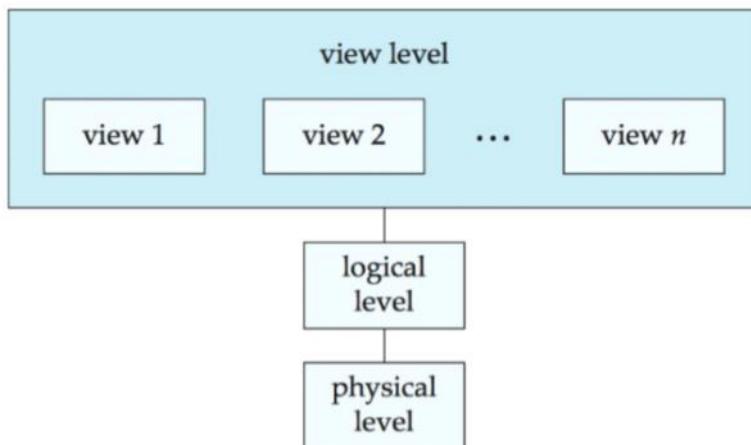
在多个级别上进行数据抽象

为了使DBS可用，它必须有效地检索数据。

对效率的需求促使设计者使用复杂的数据结构来表示数据库中的数据。

由于许多DBS用户没有经过计算机培训，开发人员通过几个抽象级别向用户隐藏复杂性，以简化用户与系统的交互。

### 三种数据抽象的级别



#### 物理级别physics

最低抽象级别描述了数据的实际存储方式。

物理层详细描述了复杂的底层数据结构。

#### 逻辑级别logical

下一个更高级别的抽象描述了数据库中存储的数据，以及这些数据之间存在的关系。

因此，逻辑层用少量相对简单的结构描述整个数据库。

#### 视图级别view

最高抽象级别只描述整个数据库的一部分。

即使逻辑层使用更简单的结构，复杂性仍然存在，因为大型数据库中存储的数据种类繁多。

视图抽象级别的存在是为了简化它们与系统的交互。

系统可以为同一数据库提供多个视图。

---

#### ANSI-SPARC三层架构

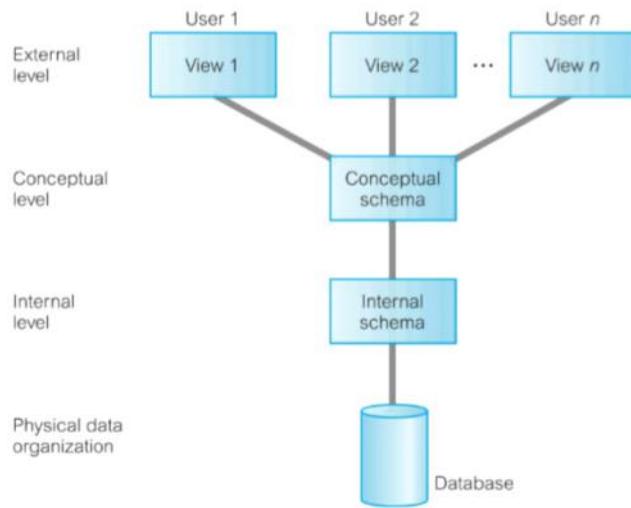
#### DBTG两级架构

CODASYL在1971年任命的DBTG在1971年提出了关于数据库系统的标准术语和通用体系结构的早期建议。

DBTG认识到需要一种两级方法，系统视图system view称为模式 schema，用户视图user view称为子模式subschema。

### ANSI-SPARC三层架构

美国国家标准协会(ANSI)标准规划和需求委员会(SPARC)，或  
ANSI/X3/SPARC，在1975年提出了类似的术语和体系结构(ANSI, 1975)。  
ANSI-SPARC体系结构认识到需要使用系统目录的三层方法



### ANSI-SPARC三层架构

这些层次构成了一个三层的体系结构，包括外部层次external level、概念层次conceptual level和内部层次internal level。

用户感知数据的方式称为外部层次。

DBMS和操作系统感知数据的方式是在内部级别上的，其中数据实际上是使用数据结构和文件组织存储的。

概念层次提供了外部层次和内部层次之间的映射和所需的独立性。

#### 外部层次

用户对数据库的视图。

此级别描述与每个用户相关的数据库部分。

外部层由许多不同的数据库外部视图组成。

每个用户都有一个以该用户熟悉的形式表示的“真实世界”视图。

外部视图只包含用户感兴趣的“真实世界”中的那些实体、属性和关系。

其他不感兴趣的实体、属性或关系可能在数据库中表示，但是用户并不知道它们。

#### 概念层次

数据库的社区视图。

此级别描述数据库中存储的数据以及数据之间的关系。

此级别包含数据库应用程序所看到的整个数据库的逻辑结构。

它是组织的数据需求的完整视图，独立于任何存储考虑。

概念层表示:所有实体、它们的属性和它们的关系;数据的限制;数据的语

义信息;安全性和完整性信息。

## 内部层次

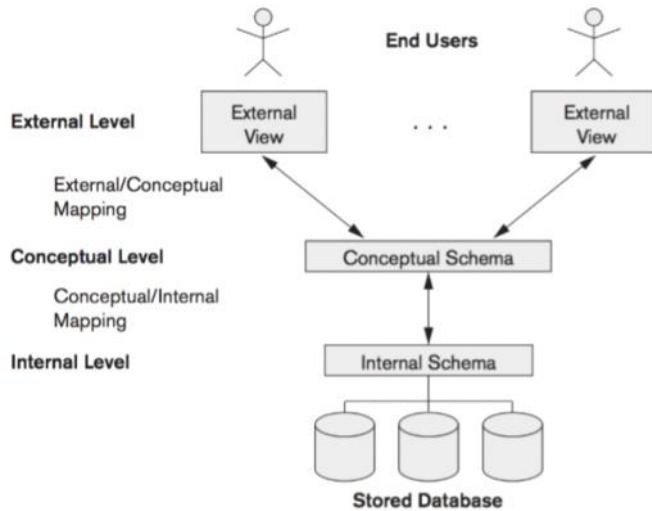
数据库在计算机上的物理表示形式。

此级别描述如何在数据库中存储数据。

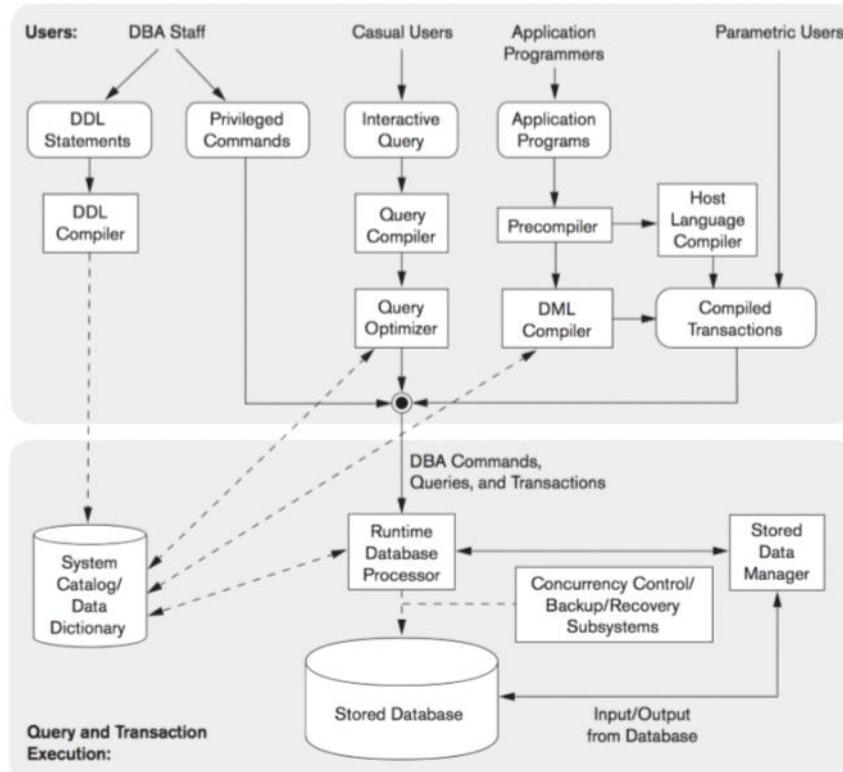
内部级别包括数据库的物理实现，以实现最佳的运行时性能和存储空间利用。

它包括用于在存储设备上存储数据的数据结构和文件组织。

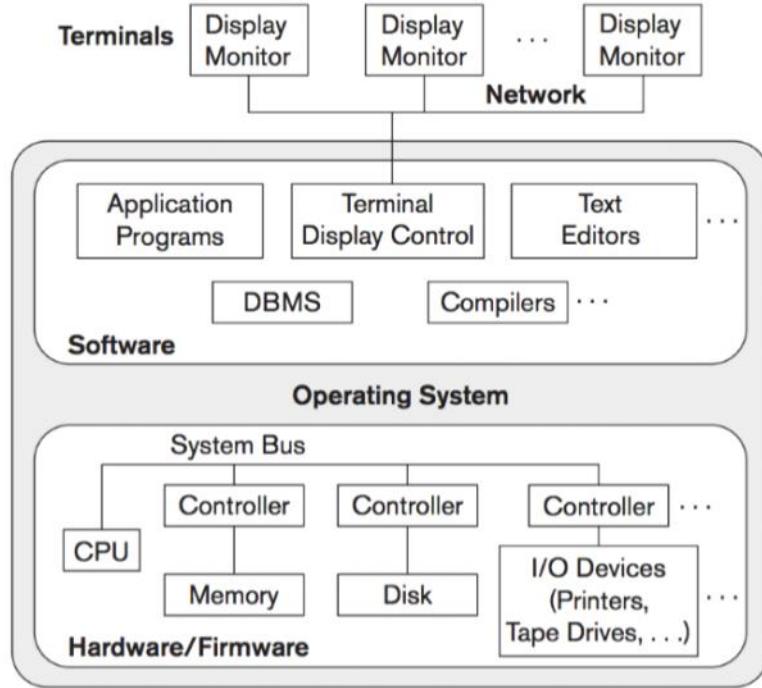
它与操作系统访问方法交互，将数据放在存储设备上，构建索引，检索数据，等等。



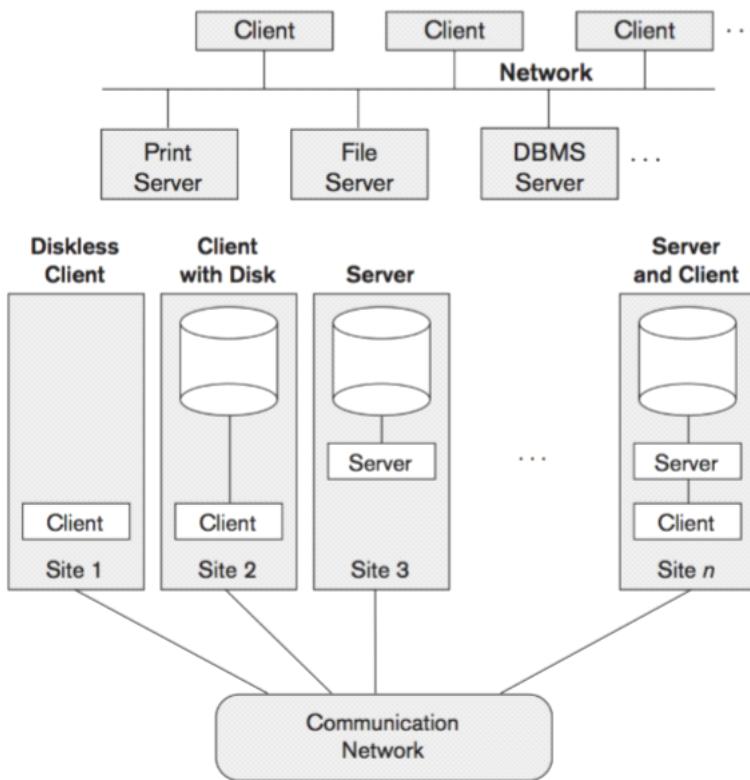
## DBMS的组件及其交互interaction



## 物理集中式体系结构physical centralized architecture



### 逻辑两层架构和物理两层架构



### 系统架构

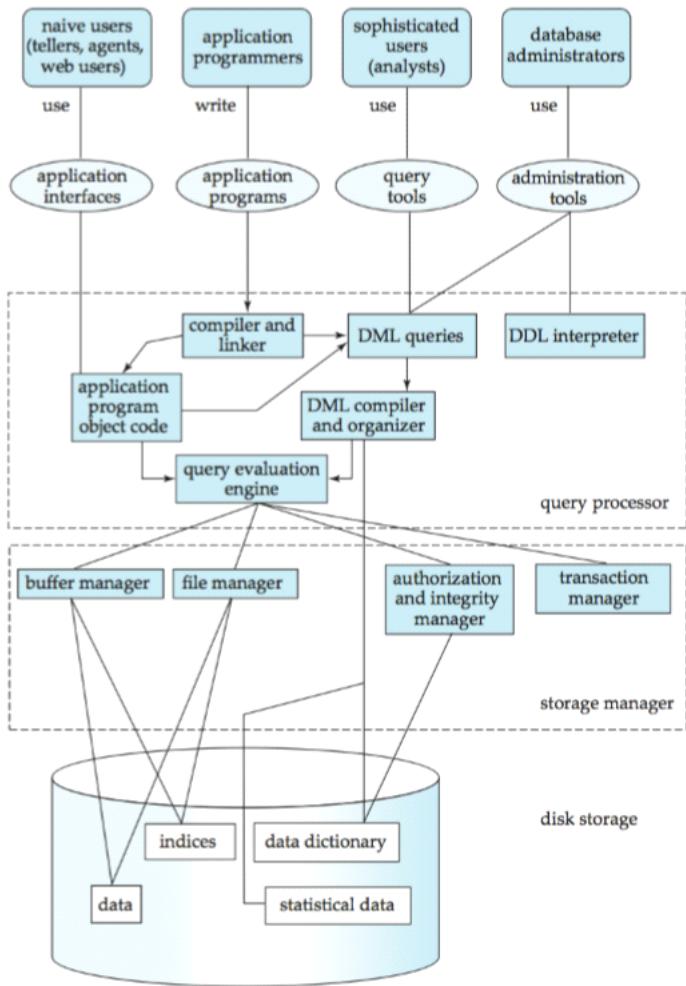


Figure 1.5 System structure.

## 数据库应用程序的两层two-tier和三层three-tier架构

### 客户端clients和服务器servers

目前，数据库系统的大多数用户都不在数据库系统的站点上，而是通过网络连接到数据库系统

因此，我们可以区分客户机(远程数据库用户在其上工作)和服务器(数据库系统在其上运行)

### 两层架构

应用程序驻留在客户机上，通过查询语言语句调用服务器上的数据库系统功能。

### 三层架构

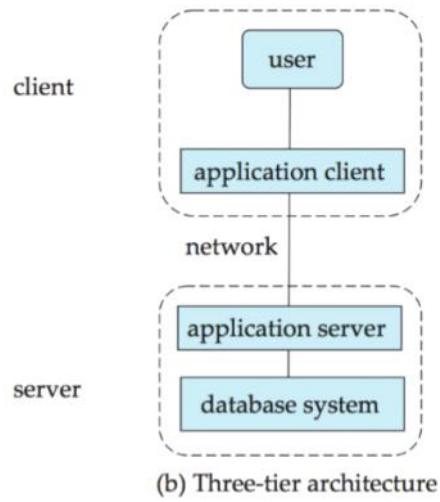
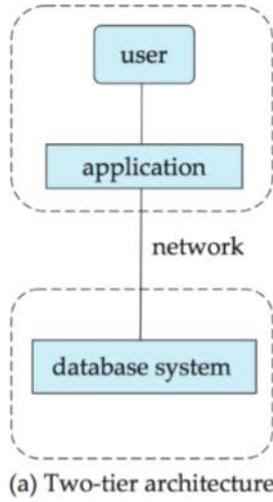
客户机仅充当前端，不包含任何直接数据库调用。

相反，客户端通常通过表单接口与应用服务器通信。

应用服务器与数据库系统通信以访问数据。

应用程序的业务逻辑(表示在什么条件下执行什么操作)嵌入到应用服务器中，而不是分布在多个客户机上。

三层应用程序更适合大型应用程序和在www上运行的应用程序



# Software engineering

2019年12月12日 23:48

## 软件工程

---

软件工程：它是什么？

软件工程：

建立和使用健全的工程原理，以获得经济可靠的软件，并在真实的机器上有效地工作。

用于设计和开发软件的整个活动范围，带有一些“良好实践”的内涵。所包含的主题包括用户需求引出、软件需求定义、体系结构和详细设计、程序规范、使用一些公认的方法(如……)，以及软件工程环境的开发和使用。此外，软件工程通常被期望解决软件开发的实际问题，包括那些在大型或复杂系统中遇到的问题。

软件工程是计算机科学的一个分支，它寻求指导大型复杂软件系统开发的原则。开发这样的支持系统所面临的问题不仅仅是编写小程序所面临问题的放大版本。

(1) 对软件的开发、操作和维护采用系统的、学科化的、可计量的方法；也就是把工程应用到软件上。

(2) 研究这些工程技术的一门学问

软件：

一个通用术语，指的是计算机系统中那些无形的而非物理的部件。它最常用来指计算机系统所执行的程序与该计算机系统的物理硬件不同，并包括此类程序的符号形式和可执行形式。

与计算机系统操作有关的计算机程序、程序以及可能相关的文件和数据。

---

## 软件可靠性software reliability

可靠性：

可靠、可靠的品质。

系统或部件在规定的条件下，在规定的时间内执行其所需功能的能力。

计算机系统在一定时间内执行其所需功能的能力。它经常被引用为正常运行时间的百分比，但更有用的表达方式可能是MTBF(平均故障间隔时间)。

软件可靠性：

在需要软件系统提供可用服务时，软件系统能够提供可用服务的程度的

量度。”软件的可靠性与程序的“正确性”有很大的不同。正确性是指程序与规范保持一致的静态特性，而可靠性则是指对系统提出的动态要求以及对这些要求作出满意响应的能力。

软件可靠性工程(SRE)：

提供设计、开发、操作和维护(功能)可靠软件系统的原则、方法和工具。

软件工程是解决软件可靠性的问题，信息安全是解决信息可靠性的问题

---

软件既是产品又是提供产品的载体

软件扮演着双重角色。它是一个产品，同时也是提供产品的载体。

软件即产品product

软件提供由计算机硬件或更广泛地由本地硬件可访问的计算机网络所体现的计算潜力。

软件即载体vehicle

作为交付产品的载体，软件是控制计算机(操作系统)、信息通信(网络)以及创建和控制其他程序(软件工具和环境)的基础。

软件

(1)程序(计算机程序)在执行时提供所需的特性、功能和性能

(2)使程序能够充分操纵信息的数据结构

(3)描述程序的操作和使用的文档。

软件=程序programs+数据结构data structures+文本documentation

软件的特殊特性(软件是逻辑的而不是物理的系统元素)

软件是被开发或设计的，而不是传统意义上的制造。

软件不会“磨损”，但会(由于变化)恶化。

尽管该行业正在转向基于组件的构建，但大多数软件仍然是定制构建的

---

软件不是什么

软件不仅仅是程序programs!

软件工程(开发)不仅仅是编程programming!

软件工程师不仅仅是程序员programmers!

---

关于软件的问题

旧的问题

今天，那些被单独的程序员问到的问题和现代计算机系统构建时问的问题是一样的。

问题

为什么完成软件需要这么长时间?

为什么开发成本如此之高?

为什么我们不能在把软件交给客户之前找出所有的错误呢?

为什么我们要花这么多时间和精力来维护现有的程序?

为什么在软件开发和维护的过程中，我们仍然难以度量进度?

软件是什么?

计算机程序和相关文档。软件产品可以针对特定客户开发，也可以针对一般市场开发。

好的软件有哪些特性?

好的软件应该交付所需的对用户来说，功能和性能应该是可维护的、可靠的和可用的。

什么是软件工程?

软件工程是一门涉及软件生产各个方面的工程学科。

什么是基本的软件工程活动?

软件规范、软件开发、软件验证和软件演化。

软件工程和计算机科学有什么区别?

计算机科学侧重于理论和基础:软件工程关注开发和交付有用软件的实用性。

软件工程和系统工程的区别是什么?

系统工程涉及计算机系统开发的各个方面，包括硬件、软件和过程工程。软件工程是这个更一般的过程的一部分。

软件工程面临的主要挑战是什么?

应对日益增加的多样性，减少交付时间的需求，以及开发可靠的软件.

软件工程的成本是什么?

大约60%的软件成本是开发成本:40%是测试成本。对于定制软件，演进成本通常超过开发成本。

最好的软件工程技术和方法是什么?

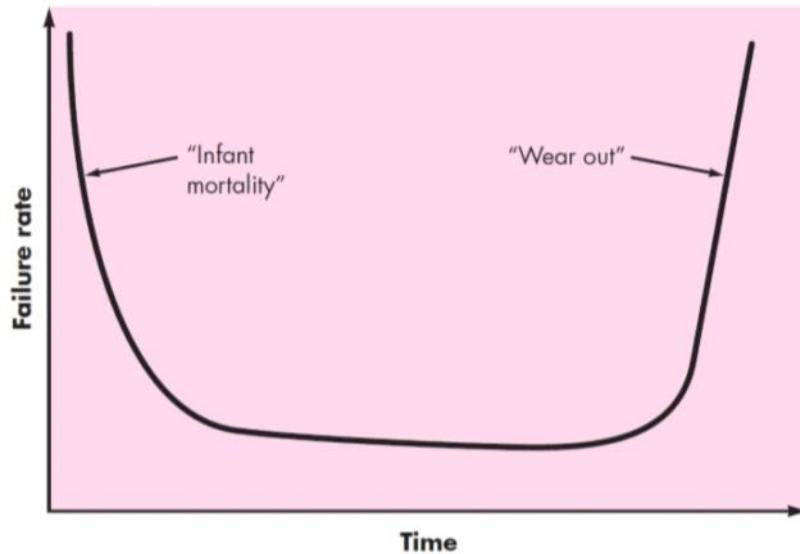
虽然所有的软件项目都必须经过专业的管理和开发，但是不同的技术适用于不同类型的系统。例如，游戏应该总是使用一系列原型开发，而安全关键控制系统需要一个完整的和可分析的规范来开发。因此，你不能说一种方法比另一种好。

网络对软件工程有什么不同?

网络已经导致了软件服务的可用性和开发基于服务的高度分布式系统的可能性。基于网络的系统开发导致了编程语言和软件重用方面的重要进步。

---

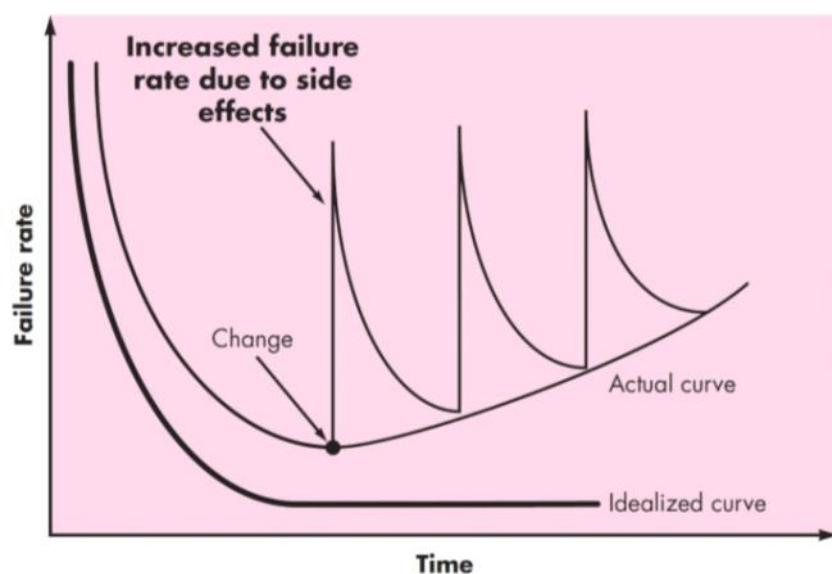
### 硬件故障曲线



开始错误很多—修改到错误比较少—物理损耗导致出现更多问题

---

### 软件故障曲线



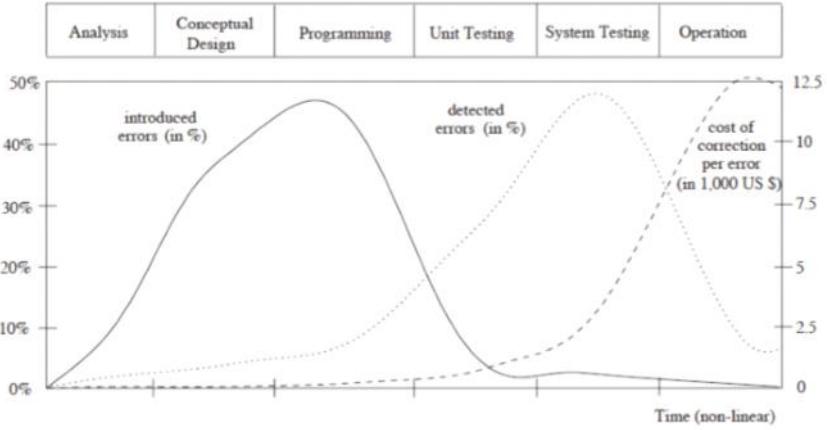
设计好的软件—用户产生新的建议—修改—循环

修改中会产生新的错误

---

### 软件错误

分析 概念设计 程序 单元测试 系统测试 操作



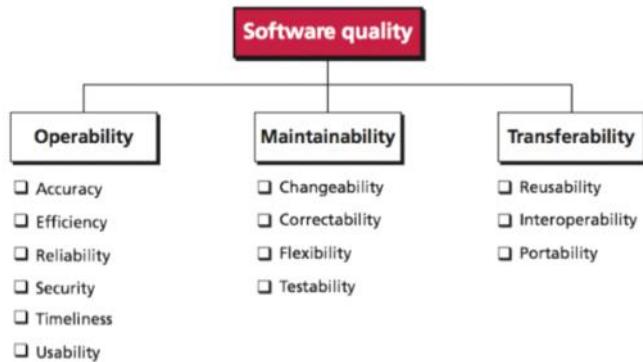
软件生命周期和错误引入、检测和修复成本

### 软件工程：一个分层的技术（重要）



质量的关注a quality focus

支持软件工程的基础是关注质量。



进程process model

软件工程的基础是进程层。

进程定义了一个框架，它必须为软件工程技术的有效交付而建立。

软件工程进程形成了软件项目管理控制的基础，并建立了应用技术方法、生产工作产品、建立里程碑、确保质量和适当管理变更的环境。

方法methods

软件工程方法提供了构建软件的技术方法。

软件工程方法依赖于一组控制技术每个领域的基本原则，包括建模活动和其他描述技术。

## 工具tools

软件工程工具为过程和方法提供自动化或半自动化的支持。

---

## 软件开发工序：它是什么

### 进程process

进程是在创建工作产品时执行的活动、操作和任务的集合。

“一个进程定义了谁在什么时候做什么，以及如何达到某个目标。”

5W1H who why what where when how

### 活动activity

一个活动努力实现一个广泛的目标，并且不考虑应用程序领域、项目的大小、工作的复杂性或应用软件工程的严格程度。

例如：与利益相关者的沟通。

### 行动action

一个行动包含一组生成主要工作产品的任务。

行动范例：建筑设计。

工作产品示例：一个架构设计模型。

### 任务task

一个任务的重点是一个小的，但定义明确的会产生一个有形的结果。

任务示例：执行单元测试。

有形的结果示例：单元测试报告。

## 过程框架process framework

一个过程框架建立了一个基础，通过识别少量的框架活动来关注一个完整的软件工程过程，这些活动适用于所有的软件项目，而不考虑它们的大小和复杂性。

## 过程框架活动process framework activites

软件工程的通用过程框架包含五个活动：通信communication、计划

planning、建模modeling、构建construction和部署deployment。

这五个通用的框架活动可用于开发小型、简单的程序、创建大型网络应用程序，以及大型、复杂的基于计算机的系统工程。在每种情况下，软件过程的细节会有很大的不同，但是框架活动保持不变。

## 过程框架活动

沟通：客户协作和需求收集

计划：建立工程工作计划，描述技术风险，列出所需的资源，生产的工作

产品，并定义工作时间表

建模:创建模型来帮助开发人员和客户理解需求和软件设计

构造:代码生成和测试

部署:交付给客户评估和反馈的软件

### 进程伞活动process umbrella activities

软件工程活动由一些伞形活动补充。

通常，伞形活动应用于整个软件项目，并帮助软件团队管理和控制进度、质量、变更和风险

典型的伞形活动包括:软件项目跟踪和控制、风险管理、软件质量保证、技术审查、度量、软件配置管理、可重用性管理、工作产品准备和生产。

#### 进程伞活动

软件项目跟踪和控制:允许团队评估进度并采取纠正措施来维护进度

风险管理:评估可能影响项目结果或质量的风险

软件质量保证:维护软件质量所需的活动

技术评审:评估工程工作产品，在它们传播到下一个活动度量之前发现和消除错误

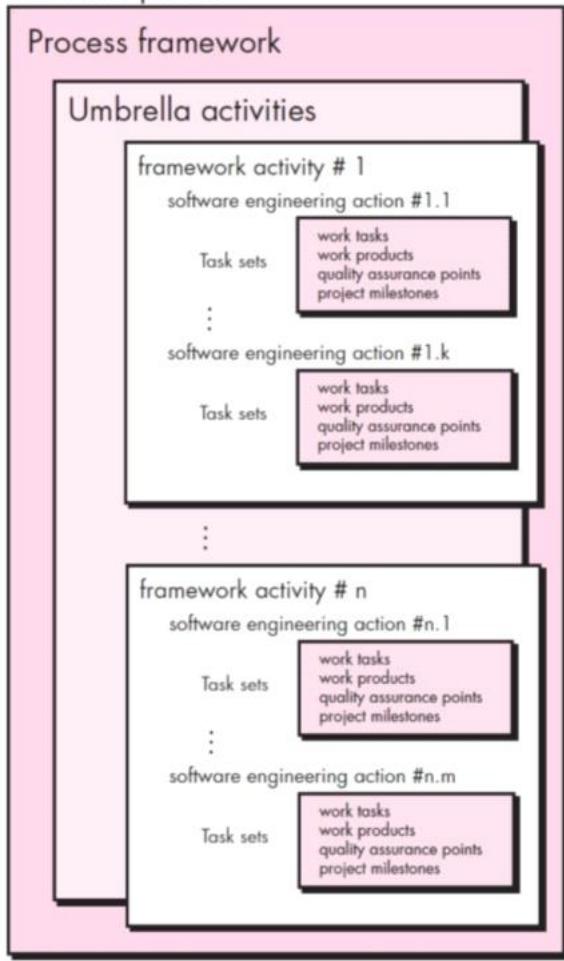
测量:定义和收集过程、项目和产品度量，以帮助团队交付满足客户需求的软件

软件配置管理:管理变更的影响

可重用性管理:定义工作产品重用的标准，并建立实现组件重用的机制

工作产品准备和生产:创建模型、文档、日志、表单、列表等的活动

## Software process



## 软件工程实践

### 软件工程实践的本质

- 1.理解问题(沟通和分析)。
- 2.计划一个解决方案(建模和软件设计)。
- 3.执行计划(代码生成)。
- 4.检查结果的准确性(检测和质量保证)。

### 软件工程实践中的基本问题

#### 理解问题

所在谁与这个问题的解决方案有利害关系?也就是说, 谁是利益相关者?

什么是未知数?需要什么数据、功能和特性才能正确地解决问题?

这个问题可以划分开来吗?有没有可能表示更容易理解的小问题?

这个问题能用图形表示吗?可以创建分析模型吗?

#### 计划一个解决方案

你以前见过类似的问题吗?在潜在的解决方案中是否存在可识别的模式?

是否存在实现所需数据、功能和特性的现有软件?

类似的问题解决了吗?如果是, 解决方案的元素是可重用的吗?

可以定义子问题吗?如果是,那么子问题的解决方案是否很明显?  
你能以一种有效实现的方式来表示解决方案吗?可以创建设计模型吗?

### 执行计划

解决方案是否符合计划?源代码可以追溯到设计模型吗?  
解决方案的每个组成部分都是可证明正确的吗?是否对设计和代码进行了评审,或者更好的是,是否将正确性证明应用于算法?

### 检查结果

有可能测试解决方案的每个组成部分吗?是否实施了合理的测试策略?  
解决方案是否产生符合所需的数据、功能和特性的结果?软件是否根据所有涉众的需求进行了验证?

---

### 软件开发的一般原则的作用

原则:是什么?

原则:

广义上:事物产生的基本来源

产生或决定特定结果的主要因素、力量或法律

某物存在所超越的最终基础;因为,从最广泛的意义上说。

### 软件开发通用原则的作用

原则会帮助你建立一套坚实的软件工程实践思维。

### 软件开发的通常原则

第一条原则:它存在的原因

软件系统存在的原因只有一个:为用户提供价值。所有的决定都应该牢记这一点。

在开始一个软件项目之前,确保软件有一个商业目的,并且用户能够从中感知价值。

软件是给别人用的,开发的时候应该从别人的角度来考虑  
所有其他原则都支持这一点。

第二条原则:接吻(保持简单,笨蛋!)

所有的设计都应该尽可能的简单,而不是更简单。

爱因斯坦的名言“每件事都应该尽可能地简单,但不能更简单。”

第三个原则:保持远见

清晰的远景对于软件项目的成功是至关重要的。

考虑软件的更新问题,如何能快速修改,快速匹配新的环境,新的功能

第四条原则:你生产什么,别人就会消费什么。

一定要指定、设计和实现，要知道其他人必须理解你在做什么。

#### 第五条:面向未来

永远不要把自己设计成一个角落。

总是问“如果”，并通过创建解决一般问题而不是特定问题的系统来准备所有可能的答案。

#### 第六个原则:提前计划重用plan ahead for reuse

提前规划重用可以降低成本并增加可重用组件和系统的价值。

#### 第七个原则:思考!

把清晰、完整的思想置于行动之前，几乎总会产生更好的结果。

---

原则1质量第一

原则2质量在旁观者的眼中

原则3生产力和质量是密不可分的

原则4高质量的软件是可能的

原则5不要试图改进质量

原则6可靠性差比效率差更糟糕

原则7尽早将产品交付给客户

原则8与客户/用户沟通

原则9调整对开发者和客户的激励

原则10计划扔掉一个

原则11建立正确的原型

原则12将正确的功能构建到原型中

原则13快速构建一次性原型

原则14以增量方式增长系统

原则15人越多，需要越多

原则16开发过程中的变化是不可避免的

原则17:如果可能，购买而不是建造

原则18构建软件，使它需要一个短用户手册

原则19每一个复杂的问题都有一个解决的

原则20记录你的假设

原则21不同阶段使用不同语言

原则22技术先于工具

原则23使用工具，但要现实

原则24给优秀的工程师提供软件工具

原则25案例工具是昂贵的

原则26“知道什么时候”和“知道如何做”一样重要

原则27当你达到目标时就停止

原则28知道正式的方法  
原则29将声誉与组织相结合  
原则30小心地跟踪旅鼠  
原则31不要忽视技术  
原则32使用文档标准  
原则33每个文件都需要一个词汇表  
原则34每个软件文档都需要一个索引  
原则35对相同的概念使用相同的名称  
原则36研究-然后转移不工作  
原则37承担责任

---

### 软件工程的巨大挑战

软件工程SE的基本原理  
定义软件系统的原始的、明确的、可测量的、一致的和完整的质量标准。  
建立软件系统和SE活动的逻辑/数学基础。  
为SE活动建立有效的形式化方法。特别是针对大型、复杂、高可靠、安全的软件系统的SE活动。

SE的自动化和智能化  
为SE活动开发自动化技术。  
为SE活动开发智能技术。

---

# Knowledge engineering and artificial intelligence

2019年12月14日 10:51

## 知识工程和人工智能

知识工程：它是什么？

知识工程

“与构建专家系统有关的人工智能的一个分支。”

知识体系knowledge-based system(KBS)

使用知识库来支持推理过程以解决应用问题的计算机系统。专家系统 expert systems就是例子，但是基于知识的系统可以采取许多其他形式，可以在人工智能的许多领域找到。”

知识库knowledge base

“一组知识，通常与特定的应用领域相关，已在适当的模式中形式化，以支持推理过程。通常使用基于规则的形式主义，但也有其他的知识表示方法。知识库(a)他们不同于数据库不仅存储数据，便于修改，修改和其他形式的内部操作的知识，(b)他们也适合于处理的知识是不完整的，不一致的，不确定的，(c)他们可能使用命令式和陈述性知识的形式。”

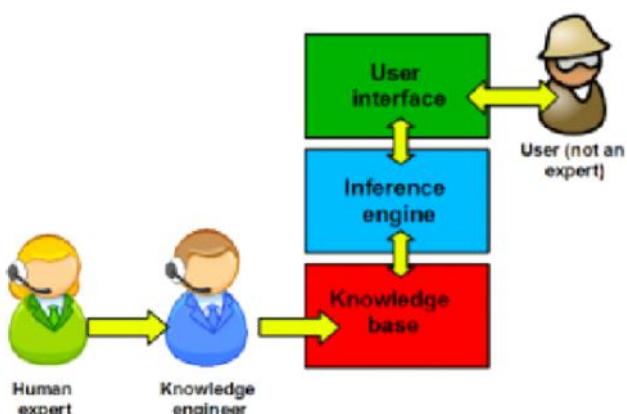
相关信息、事实或陈述的集合。

与数据库不一样的是，知识库里面存着一些推理规则，通过推理机是可以推出一个没有存在于知识库里面的内容

专家系统expert systems

利用人工智能的编程技术，特别是那些为解决问题而开发的技术，为商业应用程序编写的计算机程序。建立专家系统的多种多样，包括医疗诊断。电子故障查找、矿物勘探和计算机系统配置。

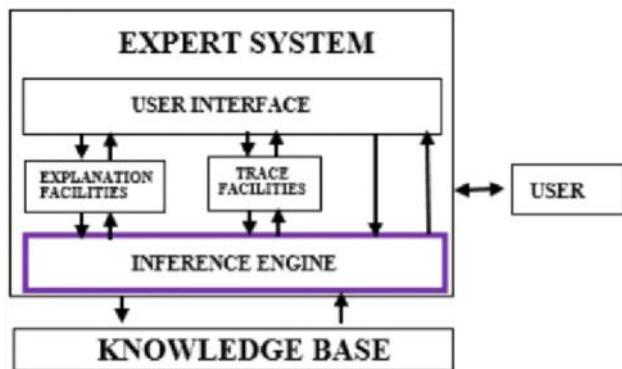
专家系统的示意图



人类专家：不一定需要掌握计算机的使用，提供某方面的专业知识

知识工程师：把专家的知识翻译成电脑可以读懂的语言，构造知识库

推理机：如何由现在的知识库的事实与一些规则结合后，得到知识库里不存在的新的结果



---

知识工程：比较困难的部分

知识获取（最难做的部分）knowledge acquisition

很难将人类专家的知识形式化（用公式，01等）表达

人类专家通常不能明确而正式地描述他们的知识。

从不同人类专家那里获得的知识可能是不一致的。

知识表示knowledge representation

在不同的应用领域有不同的知识。

知识表示的各种方法和/或工具。

自动推理automated deduction/reasoning

自动演绎/推理的有效性标准。

自动推理/推理引擎的效率。

前提正确，推理逻辑正确，一定有一个标准保证推论正确

---

KR&R知识表示和推理 knowledge representation and reasoning

KR&R是人工智能巨大挑战的核心

知识表达和推理是人工智能面临的巨大挑战的核心：深入理解智能和认知的本质，从而使计算机能够表现出类似人类的能力。

KR&R中的一般方法和/或工具

各种逻辑、各种推理、可满足性(SAT)求解器、约束规划、概念图、答案集、信念修正、定性建模、基于模型的问题求解、贝叶斯网络、情景演算、事件演算.....

---

逻辑：它是什么？

前提（声明证据）

↓

什么意味着什么？

从什么得出什么？

为什么?

评估标准是什么?

如何建立/定义评价标准?

如何评估参数/推理吗?

这些怎么确立就是逻辑的研究范围

↓

结论

从证据中可以得出什么结论

如何制定一个标准，让前提可以推导到结论

---

推理reasoning：一个例子

推理的例子

(1)所有有理数都可以表示为整数的比率。

(2)  $\pi$ 不能表示为整数之比。因此

(3)  $\pi$ 不是有理数。

(4)  $\pi$ 是一个数。因此，

(5)至少存在一个无理数。

---

推理能力是人类智力最基本的属性

伟大的推理能力，尤其是概念化的推理能力，是人类智慧最基本的属性，因此也是人类与动物之间最本质的区别。

推理能力的重要性

概念推理的能力在我们的日常生活中是极其重要的，因为它是获取大部分知识的途径。

我们的大部分知识是推理的；它不是通过直接观察而获得的，而是通过从一件事推断出另一件事而获得的。

概念推理能力是我们预测各种危险以避免自然或人为灾难的唯一方法。

---

推理：它是什么

推理是得出新结论的一个过程

推理是指从已知的事实或先前假设的前提下得出新结论，为结论提供证据的过程。

Notes

“过程”、“新结论”、“前提”、“证据”

推理是一个有序的过程

一般来说，推理是由一定顺序的若干个论证(推论)组成的。在美国，推理

是一个有序的过程。

Note “对象arguments”、“推论inferences”。“顺序order”

推理是获得新知识的一种方式

推理是一个从已知或假设(前提)到未知(新结论)的过程。

推理是扩展我们知识的一种方式

推理在本质上是扩张性的ampliative。它的功能是扩大或延伸某些事物，或在已知或假定的基础上加以补充。

---

推理的特征

前提与结论之间的证据关系

推理的前提应该是为推理的结论提供证据。

虽然推理的前提旨在为推理的结论提供一些证据，但实际上并不需要这样做。

Note：提供好证据=>好的推论，不提供好证据=>不好的推论

新的结论

推理的结论对于推理的前提来说应该是新的。如何正式而又令人满意地定义“新”的概念，一直是一个困难的哲学问题。

推理的正确性和/或有效性

好推理和坏推理。

正确和不正确的推理。

有效的和无效的推理(指推理形式)。

推理的基本问题

什么是好的，正确的，有效的推理？

判断一个推理的前提是否真的为推理的结论提供了证据的标准是什么？

判断一个推理的结论是否真的是新的推理前提的标准是什么？

什么是论证(或推理)？

---

推理和逻辑

我们在哪里可以找到关于推理的基本问题的解决方案？

它是处理一般理论中推理的正确性和/或有效性的逻辑。

Note：有效性validity、通用性generality

推理和逻辑

逻辑主要是推理，推理；特别是，它是关于什么构成正确推理的研究

逻辑学是研究用来区分好的(正确的)推理和坏的(错误的)推理的方法和原则。

证明：它是什么

定义：

证明真实、真实或有效的行为，示范

建立声明的行为。

证明

证明是在已知的事实或预先假设的前提下，为一个明确规定了的命题  
specified statement 寻找一个正当理由 justification 的过程。

证明是对已找到的正当理由的描述 description。

Note：“过程 process”、“理由 justification”、“具体命题 specified statement”，“前提 premises”

证明和逻辑

建立古典数学逻辑(CML)是为了提供形式语言来描述数学家工作的结构，  
以及他们可用的证明方法；它的主要目的是精确而充分地理解数学证明的  
概念。

逻辑上有效的证明 logically valid proving

一个逻辑有效的证明是这样一种证明：为了获得正确的证明，它是基于某  
种逻辑有效性标准而被证明的。

Note：任何“正确性”都必须依赖于一个确定定义的标准。

---

推理与证明：内在差异？

本质

推理与证明最本质的区别在于前者具有内在的规定性和预测性，而后者  
具有内在的描述性和非预测性。

目的

推理的目的是发现一些以前不知道或不认识的新命题，而证明的目的是  
为一些以前知道的或假定的寻找理由

目标(具体命题)

证明有一个明确指定的目标作为它的目标，而推理没有。

典型推理模式

从 A, B, C …… 我们能说什么？

在推理之前，我们不知道我们能从前提中得出什么结论。

典型证明模式

从A, B, C...我们可以说D吗?

在证明之前, 我们知道从前提中我们需要证明什么。

---

知识工程KE的巨大挑战

KE的基本原理

定义基于知识的系统的原始的、明确的、可测量的、一致的和完整的质量标准。

建立推理/推理的一般逻辑有效性标准。

建立有效的KE活动形式化方法。

KE的自动化

为KE活动提供开发自动化技术。

开发基于知识的系统的自主进化机制。

KE能否自动识别自然语言

---

“计算机与智能”(图灵测试)

种子文件

图灵, “计算机与智能”, 《心智》, 第236卷, 第433-460页, 1950年。

原始问题

我建议考虑这样一个问题:‘机器会思考吗?这应该从‘机器’和‘思考’这两个词的定义开始。’的定义可能会陷害所以尽可能反映到目前为止的正常使用的话,但是这种态度是很危险的,如果他的意思的词“机”和“认为”是通过检查发现通常如何使用这些工具很难逃脱的结论的意义和问题的答案,“机器能思考吗?在盖洛普(Gallup)等统计调查中可以找到。但这是荒谬的。

我不打算下这样的定义,而是用另一个问题来代替这个问题,这个问题与这个问题密切相关,并用相对明确的词语来表达。

模仿游戏

“这个问题的新形式可以用一个游戏来描述,我们称之为‘模仿游戏’。这个游戏有三个人,一个男人(a),一个女人(B)和一个审讯者(C),他们可能是男女。审问者和另外两个人分开待在一个房间里。对审讯者来说,游戏的目的是确定另外两人中哪一个是男的,哪一个是女的。他通过标记X和Y来识别它们,在游戏结束时,他知道要么是“X是A,Y是B”,要么是“X是B,Y是A”。审讯者可以向A和B提出以下问题:请告诉我他或她的头发有多长?假设X是A,那么A是必须的。A在游戏中的目标是试图让C做出错误的识别。因此,他的回答可能是:“我的头发是带状的,最长的有9英寸长。为了使语气对审讯者没有帮助,答案应该写下来,或者最好是用打字机打出来。理想的安排是让一台电传打字机在两个房间之间通信。或者,问题和答案可以由中介重复。游戏的目的是帮助审问者。对她来

说，最好的策略可能是如实回答。她可以这样回答：“我是女人，别听他的！”但这没有用，因为男人也会说类似的话。

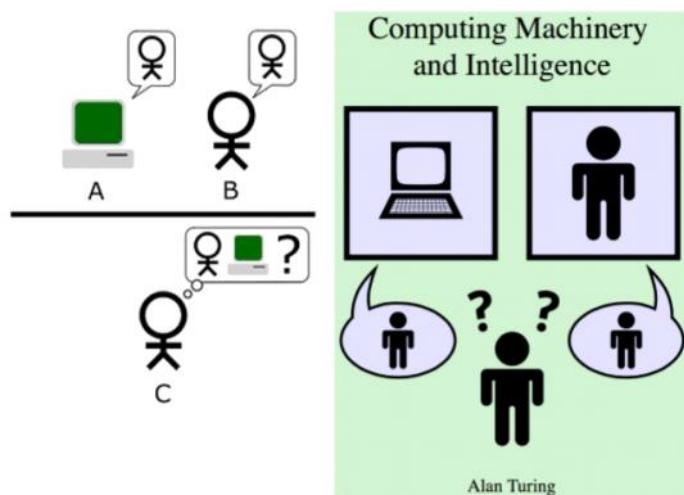
### 图灵测试问题

“我们现在问的问题是，‘如果一台机器在游戏中扮演a的角色，会发生什么？当游戏是这样进行的时候，审问者会像在一男一女之间进行的时候一样经常做出错误的决定吗？这些问题取代了我们原来的“机器会思考吗？”

“我们现在可以再次考虑在3美元时提出的观点。有人试探性地建议这个问题。“机器能思考吗？应该被“有没有可能在模仿游戏中表现出色的数字电脑？”如果我们希望把这个表面上更一般化，然后问‘是否有状态离散化的机器会做得很好？’

### 第二版《模仿游戏》

但从普遍性的性质来看，这两个问题都是等价的。让我们把注意力集中在一台特定的数字计算机上c .通过修改这个电脑真的有足够的存储,适当增加行动的速度,并提供一个适当的计划,c可以满意地玩模仿游戏的一部分,B的一部分被被一个男人吗?



### 图灵的信仰

如果我先解释一下我自己对这件事的看法，读者就会觉得事情简单多了。首先考虑问题的更准确的形式。我相信可以在大约五十年”时间,计划计算机,存储容量约10的九次方,让他们玩模仿游戏,平均审问者不会有超过70%的机会做出正确的识别五分钟后询问。原来的问题。“机器能思考吗?”认为太没有意义而不值得讨论。然而，我相信，在本世纪末，词汇的使用和普遍的有教养的观点将发生巨大的变化，人们将能够在谈论机器思维时不期望被反驳。我还相信，隐瞒这些信念不会达到任何有用的目的。流行的观点认为科学家从不受任何改进的建筑的影响，而是坚持不懈地从一个确定的事实发展到另一个确定的事实，这种观点是错误的。只要弄清楚哪些是经过证实的事实，哪些是猜测，就不会有什么损害。针叶树是非常重要的，因为它们提出了有用的研究方向。

## 图灵测试的意义

第一个提议，定义一个标准来衡量“机器智能”。

### 一些基本的问题

程序的“智能行为”真的是“机器的智能(思维)”吗?

图灵测试真的测试了人类智力(思维)的所有方面吗?

“五分钟”、“70%”或“30%”真的有意义吗?

---

## 人工智能：它是什么？

### 开创性的定义

它是制造智能机器的科学和工程，尤其是智能计算机程序。这与使用计算机来理解人类智能的类似任务有关，但人工智能不必局限于生物学上可以观察到的方法。

图灵不是提出人工智能的人，是McCarthy

“一门有关计算机程序的建造的学科，这些程序在由人完成时需要智力来完成任务。然而，已知决策过程的智能任务(例如反矩阵)通常被排除在外，而似乎不涉及智能的感知任务(例如视觉)通常被包括在内。因此，AI最好通过指示其范围来定义。人工智能处理的任务包括:游戏操作、自动推理、机器学习、自然语言理解、规划、语音理解和定理证明。”

“人们发现，感知任务(例如视觉和听觉)涉及的计算量比内省明显要多得多。这种计算在人类中是无意识的。这使得它很难模拟。AI在智力任务(如游戏操作和定理证明)上比感性任务取得了更多的成功。有时，这些计算机程序旨在模拟人类行为，以帮助心理学家和神经科学家。有时它们是用来解决技术应用方面的问题的。”

“理论研究和应用研究都对计算机科学做出了非常重要的贡献。源于AI的计算技术包括增广过渡网络、均值/末端分析。生产规则系统，解决方案，语义网络。和启发式搜索。”“哲学家们一直对‘电脑会思考吗?’有两种思想流派:弱人工智能，即电脑至少可以模拟思维和智能;和强大的。认为可以执行认知任务的人实际上是在思考。这是一个复杂的话题，人们对意识产生了新的兴趣。”

---

### 强人工智能与弱人工智能

J. Searle的开创性论文J. 塞尔，“思想、大脑和程序”，《行为与脑科学》，第3卷，第7-424,198页

#### 种子论文的摘要

这篇文章可以被看作是试图探索两个命题的结果。(1)人类(和动物)的意向性是大脑因果特征的产物。我认为这是一个关于心理过程和大脑之间

的实际因果关系的经验事实。它简单地说，某些大脑过程足以产生意向性。(2)实例化一个计算机程序本身从来不是意图性的充分条件。

种子论文的摘要这个论证的形式是为了说明一个人类代理如何能够实例化程序而仍然没有相关的意图。这两个命题有如下的结果:(3)关于大脑如何产生意向性的解释不能是通过实例化一个计算机程序来实现的。这是1和2的严格逻辑推论。(4)任何能够产生意图的机制都必须具有与大脑相同的因果能力。这是1的平凡结果。任何人为地创造意图的尝试都不能仅仅通过设计程序而成功，而必须复制人脑的因果能力。这是由2和4推出的

种子论文的摘要“机器会思考吗?”根据这里提出的论点，只有机器能思考，而且只有非常特殊的机器，即大脑和具有与大脑相同的内在因果能力的机器。这就是为什么强大的人工智能几乎不能告诉我们什么是思考，因为它不是关于机器，而是关于程序，没有一个程序本身就足以思考。”

### j·塞尔的定义

“对于最近计算机模拟人类认知能力的努力，我们应该给予什么样的心理学和哲学意义?”在回答这个问题时，我发现区分我称之为“强”的AI和“弱”或“谨慎”的AI(人工智能)是很有用的。根据weak AI的说法，电脑在心智研究中的主要价值是它给了我们一个非常强大的工具。例如，它使我们能够以更严格和精确的方式来阐述和测试假设。但是根据斯特朗·艾尔的观点，电脑不仅仅是研究思维的工具;相反，经过适当编程的计算机实际上是一种思维，从这个意义上说，经过适当编程的计算机可以说能够理解并具有其他的认知状态。在斯特朗看来，由于程序控制的计算机具有认知状态，程序不仅仅是使我们能够测试心理解释的工具;相反，程序本身就是一种解释。”

### 塞尔的中国房间思想实验

“检验任何心智理论的一种方法是问问自己，如果我的心智真的按照该理论所说的所有心智所遵循的原则工作，那会是什么样子。”“假设我被锁在一个房间里，收到一大堆汉字。再进一步假设(事实确实如此)，我不懂中文，无论是书面的还是口头的，而且我甚至不确定我是否能认出中国文字是中国文字，而不是日本文字或毫无意义的潦草字迹。对我来说，汉字不过是一堆毫无意义的潦草字迹。”

“现在再进一步假设，在这第一批汉字的基础上，我得到了第二批汉字以及一套与第二批汉字相关的规则。这些规则是用英语写的，我和其他说英语的人一样理解这些规则。它们使我能够将一组形式符号与另一组形式符号联系起来，而“形式”在这里的意思是，我可以完全根据符号的形状来识别它们。

“现在再假设我收到了第三批中文符号和一些说明，同样是英文的。这使

我能够将第三批的元素与前两批的元素联系起来，这些规则指导我如何根据第三批给我的某些形状，将某些中国符号与某些形状联系起来。”“我不知道，给我这些符号的人把第一批叫做‘脚本’，把第二批叫做‘故事’。他们称之为第三批问题。“此外，他们把我给他们的符号叫回来，作为对第三批问题的回答。”他们给了我一套英语规则，他们称之为“程序”。

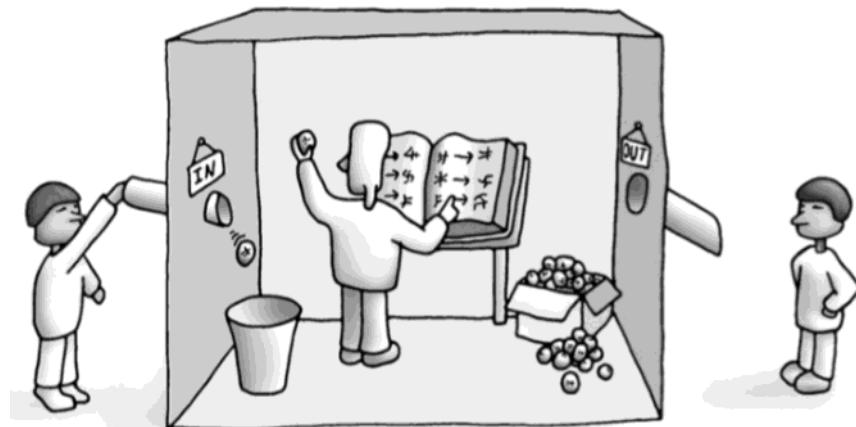
“现在为了让故事更复杂一点，想象一下这些人也用英语给我讲故事，我能听懂，然后他们用英语问我关于这些故事的问题，我用英语回答他们。”

“一段时间后也假设指令后我变得如此擅长操纵中国符号和程序员如此擅长编写程序罗外部的角度来看，从外面的人的角度来看我的房间锁——我的问题的答案绝对是区别于母语为汉语。没人能从我的回答看出我一句中文都不会说。”

“让我们也假设，我对英语问题的回答，毫无疑问，与其他母语为英语的人是无法区分的，因为我的母语是英语。”

从外部的角度来看——从阅读我的“答案”的人的角度来看——中文问题的答案和英文问题的答案一样好。但在中国的情况下，不像在英国的情况，我通过操纵未经解释的正式符号来产生答案。就中国人而言，我的行为就像一台电脑；我对正式指定的元素执行计算操作。对于中国人来说，我只是一个计算机程序的实例。”

“现在斯特朗·艾尔声称，程序控制的电脑能够理解这些故事，而且程序在某种程度上解释了人类的理解。”





### j·塞尔的说法

塞尔声称在中文房间实验中，电脑和他自己的角色并没有本质的区别。每一个都简单地遵循一个程序，一步一步地产生一个行为，然后被用户解释为演示智能对话。

然而，塞尔自己却无法理解这段对话。因此，他认为，计算机也将无法理解对话。

塞尔声称，如果没有“理解”，我们就不能将机器的行为描述为“思考”。因此，他得出结论，“强AI”是不可能的。

### 人工智能(AI)

“哲学家们一直对‘电脑会思考吗?’有两种思想流派:弱人工智能，即电脑至少可以模拟思维和智能;强人工智能认为，能够执行认知任务的实际上是思考。这是一个复杂的话题，人们对意识产生了新的兴趣。”

#### 强AI vs 弱AI

强AI：机器真的会思考吗?

弱AI：机器能智能地行动吗?”

### 人工智能的领域

人工神经网络

自动推理

自动化计划

自动定理证明/寻找

电脑游戏

决策进化计算(遗传算法)演化计算

智能代理系统

知识表示

以知识为基础的系统

机器学习  
机器视觉  
自然语言理解  
图像识别  
机器人  
言语理解

---

智能科学:它是什么?为什么要研究它?

基本事实1:AI仍然是一项没有科学基础的工程技术  
目前的人工智能没有被广泛接受的基本假设、第一原则和统一的基本理论。  
目前主要集中在寻找应用的新技术,而不是发现新的科学事实,创造新的概念,提出新的原则。

基本事实2:AI没有把“智能”作为科学的研究和工程实践的中心目标  
目前的人工智能并没有把“智能”作为科学的研究和工程实践的中心目标。  
但只要努力找到那些被认为是由“智力”产生的目标的实现方法。

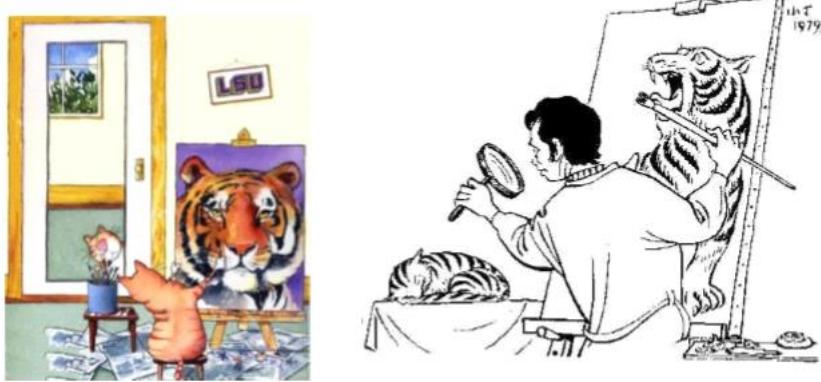
基本事实3:没有一门科学把“智力”作为科学的研究的中心目标  
无论是当前的人工智能,还是其他任何科学,都没有把“智能”作为科学的研究的中心目标。

基本事实4:计算机技术的科学(逻辑)基础是计算科学(数理逻辑),而不是电子学正是计算科学(数理逻辑)  
为计算机技术奠定了坚实的科学基础(基本假设、第一原理、统一的基本理论)。  
原则上,计算机是逻辑机器而不是电子机器。

科学与技术  
只有科学的创造(新概念、新原则、新理论和创新新方法、新方法)可以带来技术的跨越式发展。

人类智能vs动物智能  
人类的智能远比动物的智能强大。  
关于人类的智力,甚至动物的智力,还有许多未知的东西。  
有人声称计算机可以拥有“智能”。

没有一门科学把“智力”作为中心目标  
真的有老虎吗?真正的老虎在哪里?老虎不是龙!



所以?

找到真正的老鼠

最开始就研究老虎

---

# Information security engineering

2019年12月14日 10:56

# 数据类型，运算符

2020年1月14日 9:42

# Java语言概述

2020年1月15日 14:02

---

命令行

cmd：命令行窗口调出

打开命令行默认在c盘用户下

打开d盘：

【C:\>d:】

在d盘下新建一个叫java的目录

【D:\>md java】

进入d盘，java目录

【D:\>cd d:\java】

在d盘，java目录下，在新建一个子目录class1

【d:\java>md class1】

在子目录class1下查找所有文件

【d:\java\class1>dir】

从子目录class1下退回到目录java里

【d:\java\class1>cd..】

从d盘直接进入class1子目录中

【d:\>cd java\class1】

从class1子目录直接回到d盘

【d:\java\class1>cd/】

在class1下创建一个叫做【1】的doc文件，并且在文档里面写下name: Tom, age=12

【d:\java\class1>echo name:Tom,age=12>1.doc】

在class1目录下删除一个叫做【1】的doc文件

【d:\java\class1>del 1.doc】

在class1目录下删除所有的txt类文件

【d:\java\class1>del \*.txt】

删除class1空子目录

【d:\java>rd class1】

rd操作删文件必须保证删除的目录里面是空的

删除class1的所有内容

【d:\java\del class1】

会弹出是否删除class1里面的所有东西，确认按【Y】，不确认按【N】

---

Java-EE企业版

开发企业环境下的应用程序提供一套解决方案，包含技术如servlet, jsp等，针对web应用程序开发

Java-ME小型版

支持java程序在移动终端的平台

Java-SE标准版

支持面向桌面级应用的java平台，提供了完整的java核心API

现在基本与EE合并属于EE的一个部分了

---

java语言的特点

面向对象

基本概念：类，对象

特性：封装，继承，多态

跨平台性：java编写的应用程序在不同的系统平台（windows, mac, linux等）上都可以运行

原理：提供了不同操作系统对应的JVM虚拟机

---

java程序会出现内存泄漏和内存溢出的问题吗？

内存满了放不下的情况

会出现这种情况

有的时候java的垃圾回收器无法回收

---

# Java 编程

2020年1月15日 15:04

---

不会的名词可以查看API文档，里面对JAVA的语言语法做了详细的解释

---

# 《Matlab工程应用》自学

2020年2月21日 11:59

---

## 基础操作

### 默认变量与重复使用命令

如果在提示符处定义一个表达式，并且没有给该表达式分配变量，MATLAB 则使用名为 ans 的默认变量。例如，表达式  $6 + 3$  的结果存储在变量 ans 中：

```
>> 6 + 3  
ans =  
    9
```

只要在提示符处定义一个无变量的表达式，就会使用默认变量。

重复使用命令的快捷方式是按向上的方向键“↑”，该按键会显示前面已经使用过的命令。例如，如果想将表达式  $6 + 3$  的结果分配给变量 res 而不是使用默认变量 ans，可以按向上的方向键，然后使用向左的方向键来修改命令，而不是重新输入如下的整条语句：

```
>> res = 6 + 3  
res =  
    9
```

这个快捷键非常有用，特别是在长表达式输入有错误的时候，返回去纠正该表达式时非常方便。

### 相关信息的帮助指导

- info 展示产品的关联信息
- demo 给出 MATLAB 中的多项演示
- help 用来解释任何一个命令；help help 解释 help 本身是如何工作的
- helpbrowser 用来打开一个帮助窗口
- lookfor 在帮助中搜索一个特定的字符串（值得注意的是这个命令将会花很长一段时间）

### 与变量相关的命令

- who：显示在命令窗口中已经定义了的命令（仅显示变量的名称）
- whos：显示在命令窗口中已经定义了的命令（这一命令显示变量的更多信息，和工作区窗口中所显示的类似）
- clear：清除变量，这些变量将不再存在
- clear 变量名：清除指定的变量

### format表达式的输出格式和命令的输出格式

如前所示，在 MATLAB 中，default 表示数字小数部分占 4 位。format 命令可以指定表达式输出的格式。有很多选项可供选择，包括 format short(在默认状态下)和 format long。例如，变为 long 格式后数字小数部分占 15 位。这一影响将持续到变为 short 格式，可以用表达式和内置的 pi 值来说明这一点。

```
>> format long  
>> 2 * sin(1.4)  
ans =  
1.970899459976920  
  
>> pi  
ans =  
3.141592653589793  
  
>> format short  
>> 2 * sin(1.4)  
ans =  
1.9709  
  
>> pi  
ans =  
3.1416
```

format 命令还可以用来控制 MATLAB 命令或者表达式和结果之间的空格，格式为 loose(默认状态)或者 compact。

```
>> format loose  
>> 2^7  
  
ans =  
  
128  
  
>> format compact  
>> 2^7  
ans =  
128
```

## 分号和空格的作用

在语句的末尾输入一个分号可以抑制结果的输出。例如，

```
>> res = 9 - 2;  
>>
```

这条语句将右边表达式的结果 7 赋值给变量 res，但是这个结果并不显示出来，而是立即显示另一个提示符。然而，在工作区窗口处可以看见变量 mynum 和 res 的值。

注意：在本书的其余部分，结果后面的提示符将不再给出。

在语句和表达式中的空格不会影响结果，但可以使它们更具有可读性。下面没有空格的语句和前面的语句得到的结果是一样的。

---

## 基础数学运算

### 查看基本数学函数

查看一个特定的帮助主题所包含的一系列函数，先输入 help，然后输入主题名称。例如，

```
>> help elfun
```

### 常量

MATLAB 中有一些返回常量的函数，这些常量值包括：

pi 3.14159.....  
i  $\sqrt{-1}$   
j  $\sqrt{-1}$   
inf 无穷大  $\infty$   
NaN 代表“不是一个数”，例如，0/0 的结果

---

## § 3 随机变量的函数

# 第二章 随机变量

§ 1 离散随机变量

§ 2 连续随机变量

§ 3 随机变量的函数

(functions of a random variable)

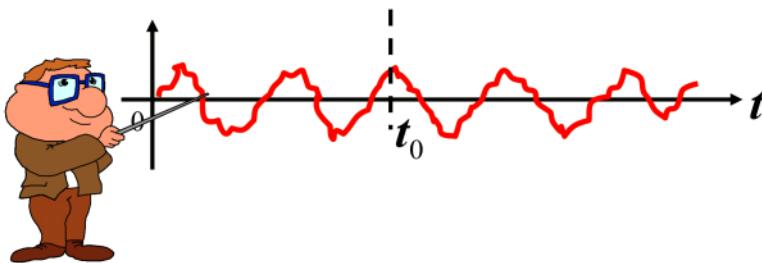
## 实际背景

**例** 在加工机件时, 只能测得工件的直径  $d$ , 然而我们关心的是工件的截面面积  $A$ . 如果知道 r.v  $d$  的分布, 问如何求 r.v  $A$  的分布?

**例** 在某电路中, 电流  $I$  是一个 r.v. 当电流通过一个  $10\Omega$  的电阻时, 问在该电阻上消耗的功率是多少?



一般地, 若  $X$  是 r.v,  $g(x)$  是一个函数,  
则  $Y = g(X)$  也是 r.v. 问怎样求  $Y$  的分布?



### § 3 随机变量的函数

3

#### (一) 离散型随机变量函数的频率函数

**例** 求  $Y = (X - 1)^2$  的频率函数, 其中 r.v  $X$  的频率函数为

$X$	-1	0	1	2
$p_k$	0.2	0.3	0.1	0.4

离散型—离散型

**解**  $Y = (X - 1)^2$  的频率函数为

$Y$	4	1	0	1
$p_k$	0.2	0.3	0.1	0.4

设 r.v  $X$  的频率函数为

$X$	$x_1$	$x_2$	...	$x_n$	...
$p_k$	$p_1$	$p_2$	...	$p_n$	...

则  $Y = g(X)$  的频率函数为

$Y$	$g(x_1)$	$g(x_2)$	...	$g(x_n)$	...
$p_k$	$p_1$	$p_2$	...	$p_n$	...

有些  $g(x_k)$  值相同  
相应的概率值合并相加

**例** 设 r.v  $X \sim U(0,1)$ , 定义

$$Y = \begin{cases} 0, & 0 < X \leq 0.25 \\ 1, & 0.25 < X \leq 0.75 \\ 2, & 0.75 < X < 1 \end{cases}$$

求 r.v  $Y$  的频率函数.

连续型—离散型

**解**  $\because X \sim U(0,1) \therefore Y$  的频率函数为

$$\begin{aligned} P\{Y = 0\} &= P\{0 < X \leq 0.25\} \\ &= \int_0^{0.25} 1 \cdot dx = 0.25 \end{aligned}$$

$$P\{Y = 1\} = P\{0.25 < X \leq 0.75\} = 0.50$$

$$\begin{aligned} P\{Y = 2\} &= P\{0.75 < X < 1\} \\ &= P\{0.75 < X \leq 1\} = 0.25 \end{aligned}$$

即  $Y$  的频率函数为

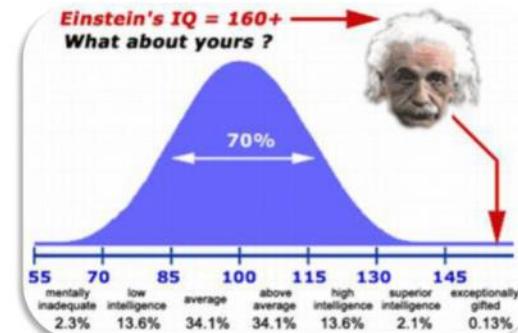
$Y$	0	1	2
$P_k$	0.25	0.50	0.25

**例** (儿童智商)

设儿童智商  $X \sim N(100, 100)$ , 将儿童按智商分为3类, 类标号  $Y$  规定如下:

$$Y = \begin{cases} 1, & X > 110 \\ 0, & 90 < X \leq 110 \\ -1, & X \leq 90 \end{cases}$$

求  $Y$  的频率函数.



$Y$	-1	0	1
$p_k$	0.16	0.68	0.16

## (二) 连续型随机变量函数的分布

**例** 设 r.v  $X$  的密度函数为

$$f_X(x) = \begin{cases} \frac{x}{8}, & 0 < x < 4 \\ 0, & \text{其它} \end{cases}$$

求 r.v  $Y = 2X + 8$  的密度函数.

**解**  $Y$  的分布函数为

$$F_Y(y) = P\{Y \leq y\} = P\{2X + 8 \leq y\} \quad F_X(g^{-1}(y))$$

$$= P\left\{X \leq \frac{y-8}{2}\right\} = F_X\left(\frac{y-8}{2}\right)$$

$$\therefore f_Y(y) = \begin{cases} \frac{1}{2} \cdot \frac{1}{8} \cdot \frac{y-8}{2}, & 0 < \frac{y-8}{2} < 4 \\ 0, & \text{其它} \end{cases}$$

$$= \begin{cases} \frac{y-8}{32}, & 8 < y < 16 \\ 0, & \text{其它} \end{cases}$$

$$f_Y(y) = F'_X(g^{-1}(y)) \\ = (g^{-1}(y))' f_X(g^{-1}(y))$$

**(二) 连续型随机变量函数的分布**

**问题** 设 r.v  $X$  的概率密度函数为  $f_X(x)$ , r.v  $Y = a + bX$ , 求 r.v  $Y$  的概率密度函数.

**分析**

$$F_Y(y) = P\{Y \leq y\} = P\{a + bX \leq y\}$$

①  $b > 0$  时:

$$F_Y(y) = P\left\{X \leq \frac{y-a}{b}\right\} = \int_{-\infty}^{\frac{y-a}{b}} f_X(x)dx$$

从而有

$$f_Y(y) = F'_Y(y) = \frac{1}{b} \cdot f_X\left(\frac{y-a}{b}\right)$$

**(二) 连续型随机变量函数的分布**

**问题** 设 r.v  $X$  的概率密度函数为  $f_X(x)$ , r.v  $Y = a + bX$ , 求 r.v  $Y$  的概率密度函数.

**分析**

$$F_Y(y) = P\{Y \leq y\} = P\{a + bX \leq y\}$$

②  $b < 0$  时:

$$F_Y(y) = P\left\{X \geq \frac{y-a}{b}\right\} = 1 - \int_{-\infty}^{\frac{y-a}{b}} f_X(x)dx$$

从而有

$$f_Y(y) = F'_Y(y) = -\frac{1}{b} \cdot f_X\left(\frac{y-a}{b}\right)$$

## § 3 随机变量的函数

### (二) 连续型随机变量函数的分布

基本流程: 求 r.v  $Y = g(X)$  的概率密度函数.

① 求 r.v  $Y$  的分布函数  $F_Y(y) = P\{Y \leq y\}$

② 转化为关于 r.v  $X$  的概率计算问题

需用到函数  $y = g(x)$  的性质!

③ 求导  $f_Y(y) = F'_Y(y)$

**问题** 设 r.v  $X$  的概率密度函数为  $f_X(x)$ ,  $y = g(x)$  单调递增且处处可导, 求 r.v  $Y = g(X)$  的概率密度函数.

**分析**

$$F_Y(y) = P\{Y \leq y\} = P\{g(X) \leq y\}$$

$y = g(x)$  单调递增

$$\begin{aligned} F_Y(y) &= P\{X \leq g^{-1}(y)\} \\ &= \int_{-\infty}^{g^{-1}(y)} f_X(x) dx \end{aligned}$$

$y = g(x)$  处处可导

$$f_Y(y) = F'_Y(y) = f_X(g^{-1}(y)) \cdot [g^{-1}(y)]'$$

## 讨论

① 若 r.v  $X$  有取值范围  $(a, b)$ , 则  $f_Y(y)$  有定义域  
$$g(a) < y < g(b)$$

② 为何要求  $y = g(x)$  严格递增?

若不然, 如何求  $g^{-1}(y)$ ?

③ 若  $y = g(x)$  严格单调递减, 有什么结论?

$$\begin{aligned}f_Y(y) &= F'_Y(y) = -f_X(g^{-1}(y)) [g^{-1}(y)]' \\&= f_X(g^{-1}(y)) \left| [g^{-1}(y)]' \right|\end{aligned}$$

### § 3 随机变量的函数

12

**定理** 设 r.v  $X$  的密度函数为  $f(x)$ , 又  $y = g(x)$  是严格单调函数, 其反函数  $h(y) = g^{-1}(y)$  连续可导, 则  $Y = g(X)$  的密度函数为

严格单调增  
(或单调减)

$$f_Y(y) = \begin{cases} |h'(y)| \cdot f(h(y)), & h(y) \text{ 有意义} \\ 0, & \text{其它} \end{cases}$$

**例** 设  $X \sim U(-\frac{\pi}{2}, \frac{\pi}{2})$ , 求  $Y = \tan X$  的密度函数.

**解** 记  $y = \tan x$ , 则

$$h(y) = \arctan y, \quad h'(y) = \frac{1}{1+y^2} \quad (-\infty < y < \infty)$$

$\therefore Y$  的密度函数为

$$f_Y(y) = |h'(y)| \cdot f_X(h(y))$$

$$f_X(x) = \begin{cases} \frac{1}{\pi}, & -\frac{\pi}{2} < x < \frac{\pi}{2} \\ 0, & \text{其它} \end{cases}$$

Cauchy 分布

$$= \frac{1}{\pi} \cdot \frac{1}{1+y^2} \quad (-\infty < y < \infty)$$

### § 3 随机变量的函数

13

**例** 设  $X \sim N(\mu, \sigma^2)$ , 求  $Y = aX + b$  的密度函数, 其中  $a (\neq 0), b$  为常数.

**解** 记  $y = ax + b$ , 则

$$h(y) = \frac{y-b}{a}, \quad h'(y) = \frac{1}{a} \quad (-\infty < y < \infty)$$

$\therefore Y$  的密度函数为

$$\begin{aligned} f_Y(y) &= |h'(y)| \cdot f_X(h(y)) = \frac{1}{|a|} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(h(y)-\mu)^2}{2\sigma^2}} \\ &= \frac{1}{|a|} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\frac{y-b}{a}-\mu)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi(|a|\sigma)^2}} e^{-\frac{[y-(a\mu+b)]^2}{2(|a|\sigma)^2}} \end{aligned}$$

$$\therefore aX + b \sim N(a\mu + b, (a\sigma)^2)$$

重要结论

正态 r. v. 的线性函数仍是正态 r. v.

**例** (股票价格) 考虑时间  $u$  后股票价格  $S_u$ , 已知  $S_u = S_0 e^{X_u}$ , 而  $X_u \sim N(u\mu, u\sigma^2)$ ,  $S_0$  为常数, 求  $S_u$  的密度函数  $f_S(s)$ .

**解**  $h(S) = \ln \frac{S}{S_0}$ ,  $h'(S) = \frac{1}{S}$ , 故

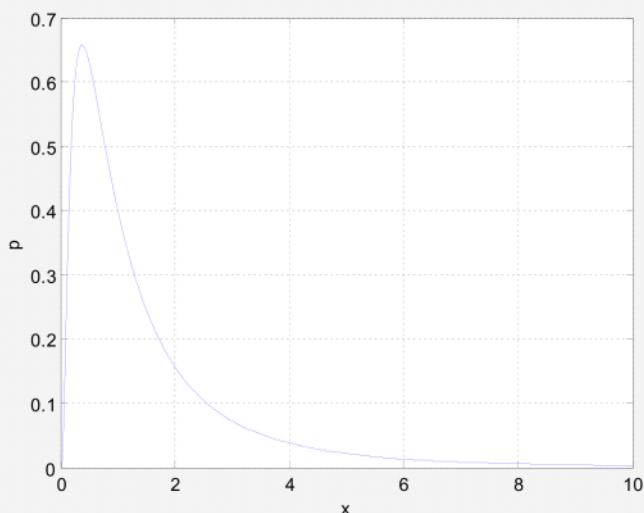
$$f_S(s) = \frac{1}{\sqrt{2\pi u\sigma^2} s} e^{-\frac{(\ln \frac{s}{S_0} - u\mu)^2}{2u\sigma^2}}$$

**注**  $\frac{S_u}{S_0}$  称服从参数为  $(u\mu, u\sigma^2)$  的对数正态分布, 记为  $LN(u\mu, u\sigma^2)$ .

## § 3 随机变量的函数

15

```
x = (0:0.02:10);  
y = lognpdf(x,0,1);  
plot(x,y); grid;  
xlabel('x'); ylabel('p')
```



**例** 设  $X \sim U(0,1)$ , 求  $Y = e^X$  的概率密度.

**问** 记  $y = e^x$ , 怎样确定其反函数 ?

**分析** 当  $y > 0$  时,  $y = e^x$  的反函数为

$$h(y) = \ln y \quad (y > 0) \quad \text{X}$$

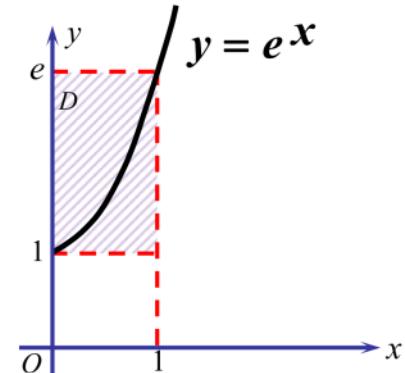
**正确的分析** ∵  $X \sim U(0,1)$ , 表明 r.v  $X$  几乎只在  $(0,1)$  上取值,

故  $y = e^x$  的反函数存在的区域是

$$D : 0 < x < 1, 1 < y < e$$

其反函数为

$$h(y) = \ln y \quad (1 < y < e)$$



**例** 设  $X \sim U(0,1)$ , 求  $Y = e^X$  的概率密度.

**解** 记  $y = e^x$ , 则当  $1 < y < e$  时, 反函数是

$$h(y) = \ln y \quad (1 < y < e)$$

$\therefore Y$  的密度函数为

$$\begin{aligned} f_Y(y) &= \begin{cases} |h'(y)| \cdot f_X(h(y)), & 1 < y < e \\ 0, & \text{其它} \end{cases} \\ &= \begin{cases} \frac{1}{y} \cdot \frac{1}{1}, & 1 < y < e \\ 0, & \text{其它} \end{cases} \\ &= \begin{cases} \frac{1}{y}, & 1 < y < e \\ 0, & \text{其它} \end{cases} \end{aligned}$$

 下面讨论直接计算法

**例** 设  $X \sim U(0,1)$ , 求  $Y = e^X$  的概率密度.

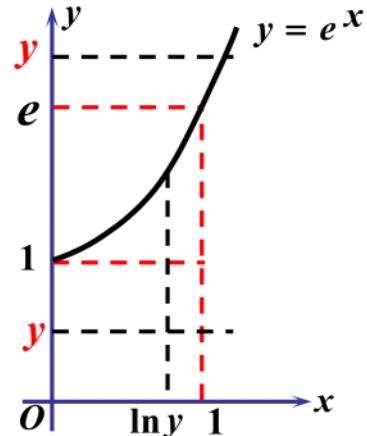
**解**  $Y$  的分布函数为

$$F_Y(y) = P\{Y \leq y\} = P\{e^X \leq y\}$$

$$= \begin{cases} 1, & y \geq e \\ \int_0^{\ln y} 1 \cdot dx, & 1 < y < e \\ 0, & y \leq 1 \end{cases}$$

$$\therefore f_Y(y) = \begin{cases} F'_Y(y), & 1 < y < e \\ 0, & \text{其它} \end{cases}$$

$$= \begin{cases} \frac{1}{y}, & 1 < y < e \\ 0, & \text{其它} \end{cases}$$



**问题** 若  $y = g(x)$  没有单调性,有什么结论?

**问题** 设 r.v  $X$  的概率密度函数为  $f_X(x)$ , r.v  $Y = X^2$  求 r.v  $Y$  的概率密度函数.

**分析**  $F_Y(y) = P\{Y \leq y\} = P\{X^2 \leq y\}$

$$= P\{-\sqrt{y} \leq X \leq \sqrt{y}\}$$

$$= \int_{-\infty}^{\sqrt{y}} f_X(x) dx - \int_{-\infty}^{-\sqrt{y}} f_X(x) dx$$

$$f_Y(y) = F'_Y(y) = f_X(\sqrt{y})[\sqrt{y}]' - f_X(-\sqrt{y})[-\sqrt{y}]'$$

讨论 函数  $y = g(x) = x^2$  是分段严格单调的

$$\begin{cases} y = g_1(x) = x^2, x > 0 \text{ 严格递增} \\ y = g_2(x) = x^2, x < 0 \text{ 严格递减} \end{cases}$$

→  $g_1^{-1}(y) = \sqrt{y}, \quad g_2^{-1}(y) = -\sqrt{y}$

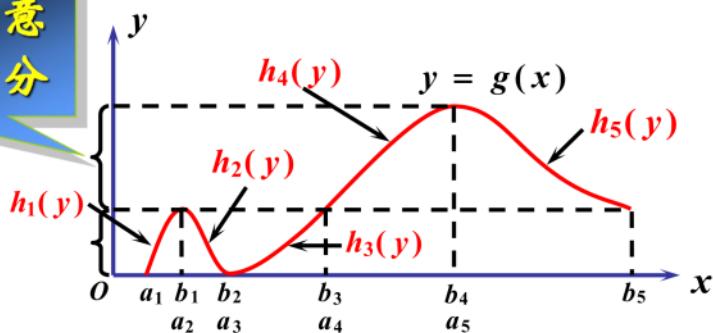
$$\begin{aligned} f_Y(y) &= F'_Y(y) \\ &= f_X(g_1^{-1}(y)) [g_1^{-1}(y)]' - f_X(g_2^{-1}(y)) [g_2^{-1}(y)]' \\ &= \sum_{k=1}^2 f_X(g_k^{-1}(y)) [g_k^{-1}(y)]' \end{aligned}$$

## 推广的定理

**定理** 设 r.v  $X$  的密度函数为  $f(x)$ , 又函数  $g(x)$  在互不相交的区间  $(a_1, b_1), (a_2, b_2), \dots$  上逐段严格单调, 且其反函数  $h_1(y), h_2(y), \dots$  均连续可导, 则  $Y = g(X)$  的密度函数为

$$f_Y(y) = \begin{cases} \sum_{i=1}^n |h'_i(y)| \cdot f(h_i(y)), & h_1(y), h_2(y), \dots \text{有意义} \\ 0, & \text{其它} \end{cases}$$

使得反函数有意义的  $y$  有两部分



### § 3 随机变量的函数

22

**例** 设  $X \sim N(0,1)$ , 求  $Y = X^2$  的密度函数.

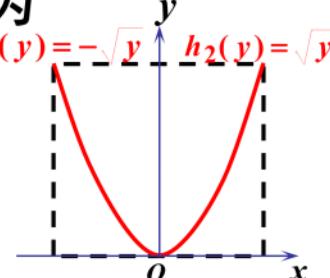
**解** 记  $g(x) = x^2$ , 则  $g(x)$  在  $(-\infty, 0)$  上严格单调减少, 而在  $(0, \infty)$  上严格单调增加, 其反函数分别为

$$h_1(y) = -\sqrt{y}, \quad h_2(y) = \sqrt{y} \quad (y > 0)$$

$$\text{且} \quad h'_1(y) = -\frac{1}{2\sqrt{y}}, \quad h'_2(y) = \frac{1}{2\sqrt{y}} \quad (y > 0)$$

$\therefore Y$  的密度函数为

$$\begin{aligned} f_Y(y) &= \begin{cases} |h'_1(y)| \varphi(h_1(y)) + |h'_2(y)| \varphi(h_2(y)), & y > 0 \\ 0, & y \leq 0 \end{cases} \\ &= \begin{cases} \left| -\frac{1}{2\sqrt{y}} \right| \frac{1}{\sqrt{2\pi}} e^{-\frac{(-\sqrt{y})^2}{2}} + \left| \frac{1}{2\sqrt{y}} \right| \frac{1}{\sqrt{2\pi}} e^{-\frac{(\sqrt{y})^2}{2}}, & y > 0 \\ 0, & y \leq 0 \end{cases} \\ &= \begin{cases} \frac{1}{\sqrt{2\pi}} y^{-\frac{1}{2}} e^{-\frac{y}{2}}, & y > 0 \\ 0, & y \leq 0 \end{cases} \end{aligned}$$



## 均匀分布与其它连续分布的关系

设 r.v.  $X$  的密度为  $f(x)$ , 分布函数为  $F(x)$ .

其中  $F(x)$  在某区间  $I$  上严格递增,  $I$  的左端点处  $F=0$ , 右端点处  $F=1$ .  $I$  可以是有界区间, 也可以是无界区间. 因此,  $F^{-1}(x)$  在  $I$  上都有定义.

① 令  $Z = F(X)$ , 那么  $Z \sim U(0,1)$ .

$$\begin{aligned} P\{Z \leq z\} &= P\{F(X) \leq z\} \\ &= P\{X \leq F^{-1}(z)\} \\ &= F(F^{-1}(z)) = z \end{aligned}$$

## 均匀分布与其它连续分布的关系

② 令  $U \sim U(0,1)$ ,  $X = F^{-1}(U)$ , 那么  $X$  的分布函数是  $F(x)$ .

$$P\{X \leq x\} = P\{F^{-1}(U) \leq x\} = P\{U \leq F(x)\} = F(x)$$

例：生成给定分布的伪随机数

要生成分布函数为  $F(x)$  的r.v., 只需将  $F^{-1}$  作用在均匀分布的随机数上即可.

例 为生成来自于指数分布的r.v., 可以取

$$T = -\ln V / \lambda, \quad \text{其中 } V \sim U(0,1).$$



## 课后作业

P49:54、59、64、补充题1, 2

**补充题1** 设随机变量  $X$  的频率函数为

$X$	-2	-1	0	1	2
$P$	1/5	1/6	1/5	1/15	11/30

求  $Y = X^2$  的频率函数.

**补充题2** 设随机变量  $X$  的概率密度为

$$f(x) = \begin{cases} \frac{2x}{\pi^2} & 0 < x < \pi \\ 0 & \text{其它} \end{cases}$$

求  $Y = \sin X$  的概率密度.

### 补充题3

设  $P\{X = k\} = \left(\frac{1}{2}\right)^k, k = 1, 2, \dots$ , 令  $\leftarrow$

$$Y = \begin{cases} 1, & \text{当 } X \text{ 取偶数时} \\ -1, & \text{当 } X \text{ 取奇数时.} \end{cases} \leftarrow$$

求随机变量  $X$  的函数  $Y$  的分布律.  $\leftarrow$

### 补充题4

设随机变量  $X$  在区间  $(1, 2)$  上服从均匀分布, 试求随机变量  $Y = e^{2x}$  的概率密度  $f_Y(y)$ .  $\leftarrow$



## 第二章 随机变量

### §1 离散随机变量

### §2 连续随机变量

### §3 随机变量的函数

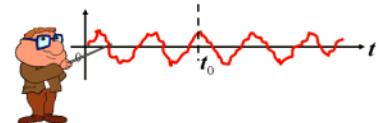
(functions of a random variable)

### 实际背景

**例** 在加工机件时, 只能测得工件的直径  $d$ , 然而我们关心的是工件的截面面积  $A$ . 如果知道 r.v.  $d$  的分布, 问如何求 r.v.  $A$  的分布?

**例** 在某电路中, 电流  $I$  是一个 r.v. 当电流通过一个  $10\Omega$  的电阻时, 问在该电阻上消耗的功率是多少?

**问题** **question** 一般地, 若  $X$  是 r.v.,  $g(x)$  是一个函数, 则  $Y = g(X)$  也是 r.v. 问怎样求  $Y$  的分布?



#### (一) 离散型随机变量函数的频率函数

**例** 求  $Y = (X-1)^2$  的频率函数, 其中 r.v.  $X$  的频率函数为

$X$	-1	0	1	2
$p_k$	0.2	0.3	0.1	0.4

离散型—离散型

解  $Y = (X-1)^2$  的频率函数为

$Y$	-4	1	0	1
$p_k$	0.2	0.3	0.1	0.4

设 r.v.  $X$  的频率函数为

$X$	$x_1$	$x_2$	...	$x_n$	...
$p_k$	$p_1$	$p_2$	...	$p_n$	...

则  $Y = g(X)$  的频率函数为

$Y$	$g(x_1)$	$g(x_2)$	...	$g(x_n)$	...
$p_k$	$p_1$	$p_2$	...	$p_n$	...

有些  $g(x_i)$  值相同

相应的概率值合并相加

**例** 设 r.v.  $X \sim U(0,1)$ , 定义

$$Y = \begin{cases} 0, & 0 < X \leq 0.25 \\ 1, & 0.25 < X \leq 0.75 \\ 2, & 0.75 < X < 1 \end{cases}$$

连续型—离散型

求 r.v.  $Y$  的频率函数.

解  $\because X \sim U(0,1) \therefore Y$  的频率函数为

$$\begin{aligned} P\{Y=0\} &= P\{0 < X \leq 0.25\} \\ &= \int_0^{0.25} 1 \cdot dx = 0.25 \\ P\{Y=1\} &= P\{0.25 < X \leq 0.75\} = 0.50 \\ P\{Y=2\} &= P\{0.75 < X < 1\} \\ &= P\{0.75 < X \leq 1\} = 0.25 \end{aligned}$$

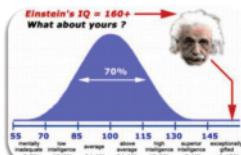
即  $Y$  的频率函数为

$Y$	0	1	2
$p_k$	0.25	0.50	0.25

#### (二) 儿童智商

设儿童智商  $X \sim N(100, 100)$ , 将儿童按智商分为3类, 类标号  $Y$  规定如下:

$$Y = \begin{cases} 1, & X > 110 \\ 0, & 90 < X \leq 110 \\ -1, & X \leq 90 \end{cases}$$



求  $Y$  的频率函数.

$Y$	-1	0	1
$p_k$	0.16	0.68	0.16

#### (二) 连续型随机变量函数的分布

**例** 设 r.v.  $X$  的密度函数为

$$f_X(x) = \begin{cases} \frac{x}{8}, & 0 < x < 4 \\ 0, & \text{其它} \end{cases}$$

求 r.v.  $Y = 2X + 8$  的密度函数.

解  $Y$  的分布函数为

$$F_Y(y) = P\{Y \leq y\} = P\{2X + 8 \leq y\} = P\{X \leq \frac{y-8}{2}\} = F_X(\frac{y-8}{2})$$

$$\therefore f_Y(y) = \begin{cases} \frac{1}{2} \cdot \frac{1}{8} \cdot \frac{y-8}{2}, & 0 < \frac{y-8}{2} < 4 \\ 0, & \text{其它} \end{cases}$$

$$= \begin{cases} \frac{y-8}{32}, & 8 < y < 16 \\ 0, & \text{其它} \end{cases}$$

$$f_Y(y) = F'_X(g^{-1}(y)) = (g^{-1}(y))' f_X(g^{-1}(y))$$

**(二) 连续型随机变量函数的分布**

**问题** 设 r.v  $X$  的概率密度函数为  $f_X(x)$ , r.v  $Y = a + bX$ , 求 r.v  $Y$  的概率密度函数.

**分析**

$$F_Y(y) = P\{Y \leq y\} = P\{a + bX \leq y\}$$

**①**  $b > 0$  时:

$$F_Y(y) = P\left\{X \leq \frac{y-a}{b}\right\} = \int_{-\infty}^{\frac{y-a}{b}} f_X(x)dx$$

从而有

$$f_Y(y) = F'_Y(y) = \frac{1}{b} \cdot f_X\left(\frac{y-a}{b}\right)$$

**(二) 连续型随机变量函数的分布**

**问题** 设 r.v  $X$  的概率密度函数为  $f_X(x)$ , r.v  $Y = a + bX$ , 求 r.v  $Y$  的概率密度函数.

**分析**

$$F_Y(y) = P\{Y \leq y\} = P\{a + bX \leq y\}$$

**②**  $b < 0$  时:

$$F_Y(y) = P\left\{X \geq \frac{y-a}{b}\right\} = 1 - \int_{-\infty}^{\frac{y-a}{b}} f_X(x)dx$$

从而有

$$f_Y(y) = F'_Y(y) = -\frac{1}{b} \cdot f_X\left(\frac{y-a}{b}\right)$$

**(二) 连续型随机变量函数的分布**

**基本流程:** 求 r.v  $Y = g(X)$  的概率密度函数.

**①** 求 r.v  $Y$  的分布函数  $F_Y(y) = P\{Y \leq y\}$ **②** 转化为关于 r.v  $X$  的概率计算问题需用到函数  $y = g(x)$  的性质!**③** 求导  $f_Y(y) = F'_Y(y)$ 

**问题** 设 r.v  $X$  的概率密度函数为  $f_X(x)$ ,  $y = g(x)$  单调递增且处处可导, 求 r.v  $Y = g(X)$  的概率密度函数.

**分析**

$$F_Y(y) = P\{Y \leq y\} = P\{g(X) \leq y\}$$

 $y = g(x)$  单调递增

$$F_Y(y) = P\{X \leq g^{-1}(y)\} = \int_{-\infty}^{g^{-1}(y)} f_X(x)dx$$

 $y = g(x)$  处处可导

$$f_Y(y) = F'_Y(y) = f_X(g^{-1}(y)) \cdot [g^{-1}(y)]'$$

**讨论**

**①** 若 r.v  $X$  有取值范围  $(a, b)$ , 则  $f_Y(y)$  有定义域  $g(a) < y < g(b)$

**②** 为何要求  $y = g(x)$  严格递增?若不然, 如何求  $g^{-1}(y)$ ?**③** 若  $y = g(x)$  严格单调递减, 有什么结论?

$$\begin{aligned} f_Y(y) &= F'_Y(y) = -f_X(g^{-1}(y)) [g^{-1}(y)]' \\ &= f_X(g^{-1}(y)) [g^{-1}(y)]' \end{aligned}$$

**定理** 设 r.v  $X$  的密度函数为  $f(x)$ , 又  $y = g(x)$  是严格单调函数, 其反函数  $h(y) = g^{-1}(y)$  连续可导, 则  $Y = g(X)$  的密度函数为

$$f_Y(y) = \begin{cases} |h'(y)| \cdot f(h(y)), & h(y) \text{ 有意义} \\ 0, & \text{其它} \end{cases}$$

**例** 设  $X \sim U(-\frac{\pi}{2}, \frac{\pi}{2})$ , 求  $Y = \tan X$  的密度函数.解 记  $y = \tan x$ , 则

$$h(y) = \arctan y, h'(y) = \frac{1}{1+y^2} \quad (-\infty < y < \infty)$$

 $\therefore Y$  的密度函数为

$$f_Y(y) = |h'(y)| \cdot f_X(h(y)) \quad f_X(x) = \begin{cases} \frac{1}{\pi}, & -\frac{\pi}{2} < x < \frac{\pi}{2} \\ 0, & \text{其它} \end{cases}$$

$$\text{Cauchy 分布} \quad = \frac{1}{\pi} \cdot \frac{1}{1+y^2} \quad (-\infty < y < \infty)$$

### §3 随机变量的函数

**例** 设  $X \sim N(\mu, \sigma^2)$ , 求  $Y = aX + b$  的密度函数, 其中  $a \neq 0, b$  为常数.

**解** 记  $y = ax + b$ , 则

$$h(y) = \frac{y-b}{a}, \quad h'(y) = \frac{1}{a} \quad (-\infty < y < \infty)$$

$\therefore Y$  的密度函数为

$$\begin{aligned} f_Y(y) &= |h'(y)| \cdot f_X(h(y)) = \frac{1}{|a|} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(h(y)-\mu)^2}{2\sigma^2}} \\ &= \frac{1}{|a|} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\frac{y-b}{a}-\mu)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi(|a|\sigma)^2}} e^{-\frac{|y-(a\mu+b)|^2}{2(|a|\sigma)^2}} \end{aligned}$$

$$\therefore aX + b \sim N(a\mu + b, (a\sigma)^2)$$

**重要结论**  
正态 r.v 的线性函数仍是正态 r.v.

### §3 随机变量的函数

**例** (股票价格) 考虑时间  $u$  后股票价格  $S_u$ , 已知

$S_u = S_0 e^{X_u}$ , 而  $X_u \sim N(u\mu, u\sigma^2)$ ,  $S_0$  为常数,

求  $S_u$  的密度函数  $f_S(s)$ .

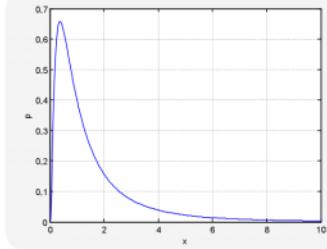
**解**  $h(S) = \ln \frac{S}{S_0}$ ,  $h'(S) = \frac{1}{S}$ , 故

$$f_S(s) = \frac{1}{\sqrt{2\pi u\sigma^2} s} e^{-\frac{(\ln \frac{s}{S_0} - u\mu)^2}{2u\sigma^2}}$$

**注**  $\frac{S_u}{S_0}$  称服从参数为  $(u\mu, u\sigma^2)$  的对数正态分布,  
记为  $LN(u\mu, u\sigma^2)$ .

### §3 随机变量的函数

```
x = (0:0.02:10);
y = lognpdf(x,0,1);
plot(x,y); grid;
xlabel('x'); ylabel('p')
```



### §3 随机变量的函数

**例** 设  $X \sim U(0,1)$ , 求  $Y = e^X$  的概率密度.

**问** 记  $y = e^x$ , 怎样确定其反函数?

**分析** 当  $y > 0$  时,  $y = e^x$  的反函数为

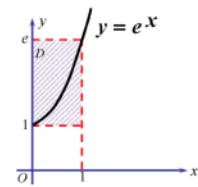
$$h(y) = \ln y \quad (y > 0) \quad \text{X}$$

**正确的分析**:  $\because X \sim U(0,1)$ , 表明 r.v  $X$  几乎只在  $(0,1)$  上取值, 故  $y = e^x$  的反函数存在的区域是

$$D : 0 < x < 1, 1 < y < e$$

其反函数为

$$h(y) = \ln y \quad (1 < y < e)$$



### §3 随机变量的函数

**例** 设  $X \sim U(0,1)$ , 求  $Y = e^X$  的概率密度.

**解** 记  $y = e^x$ , 则当  $1 < y < e$  时, 反函数是

$$h(y) = \ln y \quad (1 < y < e)$$

$\therefore Y$  的密度函数为

$$\begin{aligned} f_Y(y) &= \begin{cases} |h'(y)| \cdot f_X(h(y)), & 1 < y < e \\ 0, & \text{其它} \end{cases} \\ &= \begin{cases} \frac{1}{y} \cdot \frac{1}{1}, & 1 < y < e \\ 0, & \text{其它} \end{cases} \\ &= \begin{cases} \frac{1}{y}, & 1 < y < e \\ 0, & \text{其它} \end{cases} \end{aligned}$$

**下面讨论直接计算法**

### §3 随机变量的函数

**例** 设  $X \sim U(0,1)$ , 求  $Y = e^X$  的概率密度.

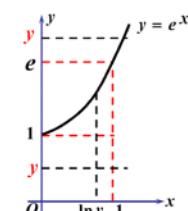
**解**  $Y$  的分布函数为

$$F_Y(y) = P\{Y \leq y\} = P\{e^X \leq y\}$$

$$= \begin{cases} 1, & y \geq e \\ \int_0^{\ln y} 1 \cdot dx, & 1 < y < e \\ 0, & y \leq 1 \end{cases}$$

$$\therefore f_Y(y) = \begin{cases} F'_Y(y), & 1 < y < e \\ 0, & \text{其它} \end{cases}$$

$$= \begin{cases} \frac{1}{y}, & 1 < y < e \\ 0, & \text{其它} \end{cases}$$



**问题** 若  $y = g(x)$  没有单调性, 有什么结论?

**问题** 设 r.v  $X$  的概率密度函数为  $f_X(x)$ , r.v  $Y = X^2$  求 r.v  $Y$  的概率密度函数.

$$\text{分析} \quad F_Y(y) = P\{Y \leq y\} = P\{X^2 \leq y\}$$

$$= P\{-\sqrt{y} \leq X \leq \sqrt{y}\}$$

$$= \int_{-\infty}^{\sqrt{y}} f_X(x) dx - \int_{-\infty}^{-\sqrt{y}} f_X(x) dx$$

$$f_Y(y) = F'_Y(y) = f_X(\sqrt{y})[\sqrt{y}]' - f_X(-\sqrt{y})[-\sqrt{y}]'$$

**讨论** 函数  $y = g(x) = x^2$  是分段严格单调的

$$\begin{cases} y = g_1(x) = x^2, x > 0 \text{ 严格递增} \\ y = g_2(x) = x^2, x < 0 \text{ 严格递减} \end{cases}$$

$$\Rightarrow g_1^{-1}(y) = \sqrt{y}, \quad g_2^{-1}(y) = -\sqrt{y}$$

$$f_Y(y) = F'_Y(y)$$

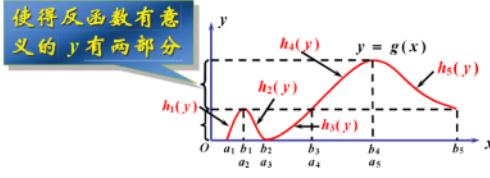
$$= f_X(g_1^{-1}(y)) [g_1^{-1}(y)]' - f_X(g_2^{-1}(y)) [g_2^{-1}(y)]'$$

$$= \sum_{k=1}^2 f_X(g_k^{-1}(y)) [g_k^{-1}(y)]'$$

### 推广的定理

**真理** 设 r.v  $X$  的密度函数为  $f(x)$ , 又函数  $g(x)$  在互不相交的区间  $(a_1, b_1), (a_2, b_2), \dots$  上逐段严格单调, 且其反函数  $h_1(y), h_2(y), \dots$  均连续可导, 则  $Y = g(X)$  的密度函数为

$$f_Y(y) = \begin{cases} \sum_{i=1}^n |h_i'(y)| \cdot f(h_i(y)), h_1(y), h_2(y), \dots \text{有意义} \\ 0, \quad \text{其它} \end{cases}$$



### 均匀分布与其它连续分布的关系

设 r.v.  $X$  的密度为  $f(x)$ , 分布函数为  $F(x)$ .

其中  $F(x)$  在某区间  $I$  上严格递增,  $I$  的左端点处  $F=0$ , 右端点处  $F=1$ .  $I$  可以是有界区间, 也可以是无界区间. 因此,  $F^{-1}(x)$  在  $I$  上都有定义.

**①** 令  $Z = F(X)$ , 那么  $Z \sim U(0,1)$ .

$$\begin{aligned} P\{Z \leq z\} &= P\{F(X) \leq z\} \\ &= P\{X \leq F^{-1}(z)\} \\ &= F(F^{-1}(z)) = z \end{aligned}$$

**例** 设  $X \sim N(0,1)$ , 求  $Y = X^2$  的密度函数.

**解** 记  $g(x) = x^2$ , 则  $g(x)$  在  $(-\infty, 0)$  上严格单调减少, 而在  $(0, \infty)$  上严格单调增加, 其反函数分别为  $h_1(y) = -\sqrt{y}$ ,  $h_2(y) = \sqrt{y}$  ( $y > 0$ )

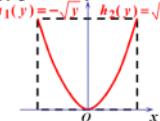
$$\text{且 } h_1'(y) = -\frac{1}{2\sqrt{y}}, \quad h_2'(y) = \frac{1}{2\sqrt{y}} \quad (y > 0)$$

$\therefore Y$  的密度函数为

$$f_Y(y) = \begin{cases} |h_1'(y)| \varphi(h_1(y)) + |h_2'(y)| \varphi(h_2(y)), y > 0 \\ 0, \quad y \leq 0 \end{cases}$$

$$= \begin{cases} \left| -\frac{1}{2\sqrt{y}} \right| \frac{1}{\sqrt{2\pi}} e^{-\frac{(-\sqrt{y})^2}{2}} + \left| \frac{1}{2\sqrt{y}} \right| \frac{1}{\sqrt{2\pi}} e^{-\frac{(\sqrt{y})^2}{2}}, y > 0 \\ 0, \quad y \leq 0 \end{cases}$$

$$= \begin{cases} \frac{1}{\sqrt{2\pi}} y^{-\frac{1}{2}} e^{-\frac{y}{2}}, y > 0 \\ 0, \quad y \leq 0 \end{cases}$$



### 均匀分布与其它连续分布的关系

**②** 令  $U \sim U(0,1)$ ,  $X = F^{-1}(U)$ , 那么  $X$  的分布函数是  $F(x)$ .

$$P\{X \leq x\} = P\{F^{-1}(U) \leq x\} = P\{U \leq F(x)\} = F(x)$$

**例：** 生成给定分布的伪随机数

要生成分布函数为  $F(x)$  的 r.v., 只需将  $F^{-1}$  作用在均匀分布的随机数上即可.

**例** 为生成来自于指数分布的 r.v., 可以取

$$T = -\ln V / \lambda, \quad \text{其中 } V \sim U(0,1).$$



## 课后作业

P49: 54、59、64、补充题1, 2

补充题1 设随机变量  $X$  的频率函数为

$X$	-2	-1	0	1	2
$P$	1/5	1/6	1/5	1/15	11/30

求  $Y = X^2$  的频率函数.补充题2 设随机变量  $X$  的概率密度为

$$f(x) = \begin{cases} \frac{2x}{\pi^2} & 0 < x < \pi \\ 0 & \text{其它} \end{cases}$$

求  $Y = \sin X$  的概率密度.

## 补充题3

设  $P\{X = k\} = \left(\frac{1}{2}\right)^k$ ,  $k = 1, 2, \dots$ , 令

$$Y = \begin{cases} 1, & \text{当 } X \text{ 取偶数时} \\ -1, & \text{当 } X \text{ 取奇数时.} \end{cases}$$

求随机变量  $X$  的函数  $Y$  的分布律.

## 补充题4

设随机变量  $X$  在区间  $(1, 2)$  上服从均匀分布, 试求随机变量  $Y = e^{2x}$  的概率密度  $f_Y(y)$ .