程序设计与计算思维

第 28 讲：计算技术的历史与展望

韩文弢

清华大学

2026 年 6 月

本讲内容

问题与目标

为什么学历史?

计算技术不是凭空出现的。了解它的历史，能帮助我们：

1. **理解现状**：为什么今天的计算机长这样？为什么 Python 是这样设计的?
2. **把握趋势**：哪些技术变革是周期性的，哪些是根本性的?
3. **做出选择**：作为未来的工程师和研究者，应该朝哪个方向投入精力?

本讲目标

1. 回顾计算技术发展的关键里程碑
2. 梳理编程语言的演化脉络及其设计哲学
3. 了解超级计算、类脑计算、量子计算等前沿方向
4. 思考计算技术的未来趋势与个人定位

计算技术的历史回顾

机械计算时代（17–19 世纪）

在电子计算机出现之前，人类已经发明了多种机械计算装置：

年代	人物	贡献
1642	帕斯卡（Blaise Pascal）	发明机械加法器——Pascaline
1671	莱布尼茨（G. W. Leibniz）	发明可执行四则运算的步进计算器
1822	巴贝奇（Charles Babbage）	设计差分机（Difference Engine），用于自动计算多项式值
1834	巴贝奇	提出分析机（Analytical Engine）

机械计算时代（17–19 世纪）

年代	人物	贡献
		概念，引入“存储程序”思想
1843	阿达·洛芙莱斯（Ada Lovelace）	为分析机编写算法，被认为是世界上第一位程序员

- 巴贝奇的分析机已具备现代计算机雏形：输入、处理、存储、输出
- 阿达·洛芙莱斯预言：这台机器不仅能处理数字，还能创作音乐、绘画

电子计算机的诞生（1940–1950 年代）

计算机	特点	意义
ENIAC (1945)	1.8 万电子管，30 吨，每秒 5000 次加法	第一台通用电子数字计算机
EDVAC (1949)	采用二进制和“存储程序”结构	冯·诺依曼体系结构的实践
UNIVAC I (1951)	第一台商用计算机	计算机从实验室走向商业应用

电子计算机的诞生（1940–1950 年代）

冯·诺依曼体系结构的核心思想：

1. 五大部件：**运算器、控制器、存储器、输入设备、输出设备**
2. **二进制**表示数据和指令；**存储程序**：指令与数据同等存放于存储器

这一结构至今仍是绝大多数计算机的设计基础。

集成电路与微处理器（1960–1970 年代）

里程碑	影响
集成电路（IC, 1958）	将多个晶体管集成到单一芯片，体积缩小、可靠性提升
摩尔定律（1965）	英特尔联合创始人戈登·摩尔预测：集成电路上晶体管数量每 18–24 个月翻一番
Intel 4004 (1971)	世界上第一款微处理器，4 位，2300 个晶体管
Intel 8086 (1978)	16 位处理器，x86 架构的开端

- 摩尔定律持续了半个多世纪，成为半导体行业发展的“自我实现的预言”
- 微处理器的出现让计算机从机房走向桌面成为可能

个人计算机与互联网（1980–2000 年代）

年代	事件	意义
1981	IBM PC 发布	确立个人计算机的开放标准
1984	Macintosh	图形用户界面（GUI）的普及
1989	万维网（WWW）	HTTP/HTML/URL，互联网进入大众视野
1991	Linux 内核	开源运动的旗帜，免费的操作系统
1998	Google 成立	搜索引擎重塑信息获取方式

计算技术完成了从“机构专属”→“个人工具”→“全球互联”的转变。

移动与云计算（2000–2010 年代）

年代	事件	意义
2006	Amazon EC2 / S3	云计算服务化，按需获取算力
2007	iPhone 发布	开启智能手机时代，计算进入口袋
2008	Android 发布	开源移动操作系统，智能手机普及
2012	AlexNet 夺冠 ImageNet	深度学习革命，AI 进入实用阶段

- **移动化**：计算无处不在，随时在线
- **云化**：算力成为像水电一样的公共服务
- **数据化**：海量数据驱动新的商业模式和科学发现

编程语言的发展

机器语言与汇编（1940–1950 年代）

机器语言：直接用二进制指令控制硬件

```
10110000 01100001     ; 将 97 存入寄存器 AL
```

- 优点：执行效率最高
- 缺点：难以编写、阅读和维护，高度依赖特定硬件

汇编语言：用助记符代替机器码

```
MOV AL, 61h           ; 将 0x61 存入 AL
```

- 引入标签、宏等抽象，但仍与硬件架构紧密绑定
- 直到今天，操作系统内核、嵌入式系统等场景仍在使用汇编

高级语言的诞生（1950–1960 年代）

语言	年份	设计目标与特点
Fortran	1957	科学计算，第一个广泛使用的高级语言
Lisp	1958	符号计算与人工智能，引入递归、垃圾回收、函数式编程
COBOL	1959	商业数据处理，接近自然语言的语法
ALGOL	1960	算法描述，引入块结构、递归、BNF 语法定义
BASIC	1964	易学易用，推动计算机教育普及

- 高级语言的核心思想：**让程序员用更接近人类思维的方式表达计算**
- 编译器负责将高级语言翻译为机器码，实现“一次编写，多处编译和运行”

结构化编程（1970 年代）

随着软件规模扩大，**goto 语句的滥用**导致代码难以维护。

语言	年份	贡献
Pascal	1970	强调结构化编程和数据结构，广泛用于教学
C	1972	简洁高效，兼具高级抽象和底层控制；操作系统和系统软件首选
Smalltalk	1972	纯面向对象语言，图形界面和 IDE 的先驱

结构化编程核心原则：用**顺序、选择、循环**三种基本结构组织代码，限制 goto。C 语言的设计哲学：**信任程序员**。

面向对象与托管时代（1980–1990 年代）

语言	年份	贡献
C++	1985	在 C 基础上引入类和继承；零开销抽象
Python	1991	强调可读性，动态类型，“一种明显的方式”
Java	1995	“一次编写，到处运行”；垃圾回收、虚拟机
JavaScript	1995	浏览器脚本语言，后发展为全栈语言

OOP 核心概念：**封装**（隐藏实现）、**继承**（复用代码）、**多态**（同一接口，不同实现）。

现代语言的多元化（2000 年代至今）

语言	年份	特点
C#	2000	微软托管语言，融合多种范式
Scala	2004	JVM 上的函数式与面向对象融合
Go	2009	强调并发编程和工程简洁性
Rust	2010	内存安全 without GC，系统编程新选择
Swift	2014	Apple 生态，安全与性能并重
Julia	2012	科学计算，兼具动态灵活和接近 C 的性能
Zig	2016	C 的现代化替代，显式控制与编译期计算
Mojo	2023	Python 超集 + 系统编程，为 AI 基础设施设计

趋势：**多范式融合**、**安全性优先**、**性能与效率**、**工具链生态**（包管理、LSP、AI 辅助编程）。

Python 的定位与成功

Python 并非性能最强，但在教育和科学计算领域占据主导：

因素	说明
可读性	强制缩进、简洁语法，代码即文档
生态	NumPy、Pandas、SciPy、PyTorch 等顶级科学计算库
胶水语言	轻松调用 C/C++/Fortran 底层库，兼顾开发效率和运行效率
AI 浪潮	深度学习框架首选接口语言

Python 的成功说明：**开发效率、生态系统和社区**有时比纯语言性能更重要。“慢”的 Python 通过调用快的 C 库实现高性能，这是**分层设计**的智慧。

计算技术的展望

超级计算与异构计算

特征	说明
规模	数十万节点，数百万 CPU/GPU 核心
性能	每秒百亿亿次浮点运算（Exascale，10^{18} FLOPS）
应用	气候模拟、核聚变、药物设计、天体物理、材料科学
挑战	功耗（兆瓦级）、可靠性、并行算法、数据移动瓶颈

异构计算：CPU + GPU + FPGA + 专用加速器协同

- CPU 擅长复杂逻辑和串行任务；GPU 擅长大规模并行（矩阵运算、渲染）
- **TPU/NPU**：专为神经网络设计的张量处理单元
- 编程模型：CUDA、OpenMP、MPI、oneAPI

人工智能与专用芯片

深度学习革命催生了 **AI 专用硬件**的爆发：

硬件	代表产品	特点
GPU	NVIDIA H100/B200	通用并行计算，AI 训练主力
TPU	Google TPU v5p	专为 TensorFlow 和神经网络优化
NPU	Apple Neural Engine	消费设备中推理加速
类脑芯片	Intel Loihi	脉冲神经网络，超低功耗

大语言模型（LLM）的算力需求：GPT-4 级训练需数万 GPU 运行数月，推动**模型压缩、量化、蒸馏**发展；**边缘 AI** 让模型在手机、IoT 设备上运行。

类脑计算（Neuromorphic Computing）

特征	传统计算机	生物大脑
架构	存储与计算分离	存储与计算融合
处理单元	少量高速核心	大量低速神经元（10^{11} 个）
连接	总线/网络互联	高度并行突触（10^{14} 个）
功耗	百瓦至兆瓦级	仅 20 瓦
学习	静态编程	动态自适应（突触可塑性）

类脑计算目标：构建像大脑一样高效、自适应的系统

1. **脉冲神经网络（SNN）**：用离散“脉冲”代替连续值传递信息
2. **忆阻器（Memristor）**：模拟突触的可变电阻器件
3. **神经形态芯片**：Intel Loihi、IBM TrueNorth，功耗仅为传统芯片的千分之一

量子计算（Quantum Computing）

量子比特（qubit）可处于叠加态：$\psi = \alpha|0\rangle + \beta|1\rangle$，其中 $|\alpha|^2 + |\beta|^2 = 1$。

特性	含义
叠加	n 个量子比特可同时表示 2^n 个状态
纠缠	多个量子比特之间存在非经典关联，改变一个瞬间影响另一个
干涉	通过量子门放大正确答案概率，抑制错误答案

量子计算（Quantum Computing）

领域	量子优势
密码学	Shor 算法多项式时间分解大整数，威胁 RSA
药物设计	精确模拟分子量子态，加速新药发现
材料科学	模拟电子结构，设计高温超导等新材料
优化问题	量子退火求解组合优化（路径规划、调度）

当前挑战：**退相干**（需极低温运行）、**纠错**（大量物理量子比特编码一个逻辑比特）、**规模**（目前约 1000+ 物理量子比特，实用化需数十年）。

其他前沿方向

方向	原理与前景
光计算	用光子代替电子，超低延迟、超低能耗，天然适合矩阵运算
DNA 计算	碱基配对并行计算，存储密度极高，适合特定组合问题
边缘计算	计算推向数据源端，减少延迟，支撑 IoT 和实时应用
可持续计算	关注碳足迹，绿色数据中心、低功耗芯片、算法效率优化

思考与展望

计算技术的核心趋势

回顾历史，计算技术的演进遵循几条主线：

1. **更快**：从秒级到纳秒级，从单机到并行
2. **更小**：从房间大小到指甲大小，从桌面到口袋到嵌入万物
3. **更便宜**：从百万美元到几美元，计算成为基础设施
4. **更易用**：从打孔卡片到自然语言，编程门槛持续降低
5. **更智能**：从执行指令到理解意图，从工具到助手到伙伴

AI 时代的新范式

大语言模型正在改变编程本身：

变化	影响
自然语言编程	用描述代替代码；但“说清需求”比“写对代码”更难
代码生成	AI 辅助编写 60–80% 代码，人类聚焦架构和问题拆解
调试进化	从“找 bug”到“审阅 AI 输出”，基础功底考验不减反增
学习路径	从“背语法”到“理解原理”，抽象思维和系统设计更重要

AI 时代的新范式

AI 不会取代程序员，但**会用 AI 的程序员**会取代不会用的。计算思维——抽象、分解、建模、实验——在 AI 时代更加重要。

给同学们的建议

方面	建议
基础能力	打好数学、物理、计算机基础；基础越牢，适应新技术越快
工具思维	编程是解决问题的工具，而非目的本身
持续学习	保持好奇心和学习能力比掌握特定技术更重要
交叉视野	计算技术渗透所有学科，跨学科能力是关键竞争力
实践导向	用项目驱动学习，从“会用”到“用好”需要大量实践

本讲小结

知识要点

1. **计算历史**：机械计算 → 电子计算机 → 个人电脑 → 移动云
2. **冯·诺依曼架构**：存储程序、二进制、五大部件
3. **编程语言演进**：机器码 → 汇编 → 高级语言 → OOP → 多范式
4. **Python 成功之道**：可读性、生态、胶水能力，而非纯性能
5. **超级计算**：异构并行、Exascale，支撑科学前沿
6. **类脑计算**：模仿大脑的高效、自适应、低功耗模式
7. **量子计算**：叠加和纠缠，特定问题可能指数级加速
8. **AI 时代**：编程方式在变革，计算思维永恒重要

课程结语

本课程从图像处理到科学计算，从 Python 基础到 C++ 工程实践，覆盖了程序设计与计算思维的核心内容。

技术会过时，但思维不会。希望同学们将课程中学到的**抽象、分解、建模、实验**能力带到未来的学习和工作中，成为**用计算思维解决问题**的人。

祝同学们期末顺利！