我们程序员：从代码诞生到AI兴起

[美] 罗伯特·C. 马丁 (Robert C.Martin) 著

茹炳晟 柳 飞 译

清華大學出版社

北 京

北京市版权局著作权合同登记号 图字：01-2025-0623

图书在版编目 (CIP) 数据

我们程序员 : 从代码诞生到 AI 兴起 / (美) 罗伯特·C. 马丁 (Robert C. Martin) 著 ; 茹炳晟 , 柳飞译 .
北京 : 清华大学出版社 , 2025. 6. -- ISBN 978-7-302-69497-7

Ⅰ. TP311.1

中国国家版本馆 CIP 数据核字第 2025FT7997 号

责任编辑：王　军
封面设计：高娟妮
版式设计：思创景点
责任校对：成风进
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京联兴盛业印刷股份有限公司
经　　销：全国新华书店
开　　本：168mm×240mm　　**印　　张：**20　　**字　　数：**415 千字
版　　次：2025 年 7 月第 1 版　　**印　　次：**2025 年 7 月第 1 次印刷
定　　价：102.40 元

产品编号：110236-01

技术领袖力荐

Uncle Bob所著的这本书的内容暗合了这两年我脑海中一直在思考的几个问题。在AI爆发式发展的当下，各种言论甚嚣尘上、纷纷扰扰。要在迷雾中看清方向，需要回顾历史、重读经典，同时要抓住事物的第一性原理。本书就像一本软件技术发展的编年史，从计算机和软件的诞生谈到软件开发方法和技术发展的一波波浪潮，直到AI时代的软件开发及未来展望。阅读本书，可以让我们更清晰地理解和把握软件技术发展的脉络，在喧嚣之中坚守内心的信念，明确前行的方向。

——彭鑫
复旦大学计算与智能创新学院副院长、教授
中国计算机学会软件工程专委会副主任

《我们程序员：从代码诞生到AI兴起》不是冰冷的技术手册，而是一部记录程序员群体发展的技术纪实。Uncle Bob以深刻的洞察力，带我们穿越一行行代码，看清支撑数字世界运行的是怎样一群人。他们的智慧与执着、协作与分歧、传承与革新，共同推动了技术的前行。在键盘敲击声背后，是程序员与复杂系统长期搏斗的痕迹。这本书邀你一起回望这段技术史，也理解在AI时代，程序员为何依然不可替代。

——陶建辉
涛思数据创始人&CEO

程序员会不会被AI替代？所有热爱编程、从事软件开发的同学可能都会关心这个问题。由茹炳晟、柳飞老师翻译的Uncle Bob的新作《我们程序员：从代码诞生到AI兴起》，看起来是在讲述软件编程半个多世纪的历史，实则在探讨程序员的核心价值是什么，程序员不可被替代的地方是什么。透过这些故事，你可能会同意作者的观点，又或者像我一样形成自己的看法：人们关于美、价值、幸福的定义永远不会交给AI去完成，而软件编程的过程和结果恰恰是关于美、价值和幸福的定义。

——李智慧
同程旅行公司资深架构师

在历史的回响中，洞见 AI 变革软件开发

众所周知，AI 正以摧枯拉朽之势重塑软件开发格局，这场变革不是简单的技术迭代，而是对整个计算机软件生态的深度重构。此时，若想穿透 AI 变革软件开发的表象，抓住核心本质，回溯历史、从计算机、软件与编程发展的长河中探寻规律，便成为一条必由之路。这正是我对Uncle Bob最新出版的《我们程序员：从代码诞生到 AI 兴起》一书的兴趣所在。本书不只是罗列史实，而是深入挖掘技术迭代背后的动因，这些历史轨迹，本质是人类需求、技术创新与人类智慧相互作用的缩影，也是我们深入理解这场AI变革软件开发的底层脉络。

本书以宏大的历史视野，为读者勾勒出计算机与人工智能从萌芽到繁荣的完整脉络。从巴贝奇等先驱，到图灵、冯·诺伊曼奠定第一代计算机，再到编程语言从机器码演进至高级语言、面向对象、敏捷开发、互联网、神经网络……，这是一部人类认知如何驾驭计算复杂性的奋斗史。每一步都承载着技术突破与范式革新，都是人类思维在更高层次上对计算本质的“抽象”。

AI的“非编程”革命，是抽象层次的终极跃升。本书敏锐捕捉到这场变革的核心特质：大语言模型的构建本质，已迥异于传统意义上的“编程”。冯·诺伊曼架构下的传统编程，是程序员将人类意图精确分解为机器可执行的确定性指令，是迪杰斯特拉结构化编程严谨逻辑的体现。而AI则通过海量数据驱动下的神经网络，让机器“学习”并“涌现”出完成任务的能力。这标志着人机交互接口的根本性反转：开发者从指令的“编织者”，转变为目标的“定义者”、数据的“策展者”和模型行为的“引导者”。开发者的核心能力，从“如何写代码”转向“如何定义问题、评估方案与确保价值”。这比从汇编到高级语言、从过程式到面向对象的抽象跃迁更为激进，也打开了人类智慧与机器协同创新的大门。

对广大开发者而言，本书更是一场思维启蒙。在 AI 重塑一切的当下，本书为我们搭建了一座连接历史与未来的桥梁，每一次技术变革都不是孤立的“颠覆”，而是历史逻辑的延续。AI 时代的程序员，既要掌握新技术工具，更要理解技术演进的底层规律。回溯过往，编程语言从晦涩的机器码到易用的高级语言，是“让编程更贴近人类思维”的持续探索；而今，AI 辅助编程、自动生成代码，实则延续这一脉络，进一步缩短 “人机交互边界”。AI 变革软件开发的核心，是对计算范式、编程逻辑与开发者角色的重构。

理解了编程语言百年进化的规律，才能深刻理解软件如何设计、抽象、演进的历史逻辑，才能真正洞察AI所带来的范式转变之剧烈与深远，**才能更好地推动“人、代码、机器”交互协同的迭代创新，**才能在变革中把握机遇，激发人类在数字世界永不枯竭的创造力。我向所有关心AI时代软件未来走向的朋友们推荐《我们程序员：从代码诞生到 AI 兴起》一书，它不仅是历史的记录，更是通往未来的罗盘。

——李建忠
CSDN高级副总裁
全球机器学习技术大会主席
ISO-C++国际标准委员会委员

在时代浪潮中，写下属于程序员的注脚

在这个 AI 正重塑一切的时代，程序员这个角色，似乎也正在从幕后走向台前。从为业务写逻辑，到为人类设计智能，我们正在经历一场技术职业身份的重构。而《我们程序员：从代码诞生到AI兴起》正是在这个关键的转折点上，为所有代码工作者写下了一部兼具温度与深度的编年史。

作为极客邦科技有限公司的创始人，我很荣幸向中文读者推荐这本书。极客邦长期致力于推动技术知识的传播与工程文化的建设，无论是InfoQ这座技术社区的灯塔，还是“极客时间”这个为开发者提供持续成长的学习平台，以及“TGO鲲鹏会”这个链接全球科技领导者的组织，我们始终关注一个核心命题：**技术如何塑造人，技术人又如何影响这个世界。**

而这本书的作者，Robert C. Martin——程序员们亲切称他为Uncle Bob，正是将这个命题写成故事的人。他从阿达·洛芙莱斯的计算幻想谈起，跨越 COBOL、UNIX、开源、Web，到今天的 AI 编程，每一段历史不仅映射着计算技术的发展轨迹，更折射着程序员这个群体不断进化的价值观、工具观和世界观。

这本书不只是历史，也是一面镜子。我们在其中看到自己当年写下第一行代码的初心，也看到曾经热血沸腾投身某个技术浪潮的执念；我们看到软件工程从“工匠精神”到“自动化智能”的演变，也看到如今面对 AI 编程、Agent 生态的焦虑与希望并存。

在极客邦，我们今年把“AI 应用落地”设为公司的年度主题。原因很简单：AI 已不再是技术先锋的玩具，而是每个程序员都必须理解并使用的新型“工具箱”。但在这一轮工具变革中，我们更需要的是一种对编程本质的重新理解：**写代码的终极目标，不是追赶潮流，而是创造价值。**

Uncle Bob写这本书，显然也不是为了“回顾历史”，而是提醒我们：“我们是谁”比“我们做了什么”更重要。我们是程序员，是系统的架构师，是工具的缔造者，更是价值的搬运工和文明的记录者。AI 可以写代码，但只有人类程序员知道代码为什么存在。

今天，中国的程序员群体正处在全球视野、AI范式与社会价值的三重夹角中。越是在这样的时代，我们越需要这样一本书，来帮我们厘清职业的过去、当下与未来。这是一本写给“技术人”的精神读本，也是每一个写过代码、热爱过系统、为世界构

建规则的人值得阅读和传承的故事集。

感谢清华大学出版社引进这本作品，也很开心看到好友茹炳晟领衔翻译，让它的内容质量得以保证。它不是一本关于“怎么写好代码”的书，而是一本关于“为什么要当程序员”的书。**在AI时代，它值得每一个思考未来的技术人认真读完。**

——霍太稳

极客邦科技创始人 & CEO

凡人英雄史：一部程序员的五十年跋涉与时代镜像

我是从《敏捷软件开发：原则、模式与实践》开始认识Uncle Bob的，这是一本为我打开新世界的书，可你是否知道，这是Uncle Bob的公司业务因为 9·11 事件发展受阻之后利用闲暇时间完成的。《代码整洁之道》是后来很多程序员的编程必读书，可你是否知道，这本书写完之后，Uncle Bob的公司因为2008年金融危机就彻底倒闭了。

Uncle Bob是很多人心目中的代码英雄，但他也有凡人的一面。《我们程序员：从代码诞生到 AI兴起》是Uncle Bob写就的一本代码英雄的编年史。这本书前一半写了Uncle Bob自己心中的代码英雄：从巴贝奇、图灵、冯·诺伊曼，到肯·汤普森、丹尼斯·里奇、布莱恩·克尼汉。

如果从一部编年史的角度看，后面还有很多值得记录的人，但站在Uncle Bob的角度，这之后就是自己的故事了。

他从20世纪60年代开始接触编程，1969年开始了自己的第一份程序员工作，然后，就是他和计算机行业一起发展的故事了。相对于那些影响了计算发展历史的人，Uncle Bob在这本书里更像一个普通程序员，他有学到新知的兴奋，也有丢掉工作的落寞。从他的身上，我们依稀可看到自己的影子。

Uncle Bob其他的书更多的是传道授业，而《我们程序员：从代码诞生到 AI兴起》更像一部以编年史名义写就的个人传记。这本书让我们看到了热爱的力量，看到了计算机的变迁，看到了各个时代的机遇。Uncle Bob的职业生涯足够长，让我们看到一个人坚守在一个领域遇到的诸多变迁。这也是这本书对我来说最大的价值，给予我在程序员的道路上继续前行的力量。

如果说本书有什么遗憾，那就是这本书的写作稍微早了一点，大模型驱动的 AI 编程工具在这本书出版之后得到了极大的发展，这就让这本书关于未来的部分显得有些单薄。不过，反过来想，我们总是羡慕前行者有着各种机会，但实际上，新机会总是在涌现，现在的我们何尝不是站在AI时代的风口上。

——郑晔

开源项目 Moco的作者

从潦草笔记到通天塔：程序员穿越 AI 危机的生存智慧

第一次翻开这本书时，我惊讶地发现：那些塑造我们行业的方法论——敏捷开发、代码整洁之道——竟以如此“草根”的形式诞生。

Uncle Bob在书中坦言，当年写《代码整洁之道》时，自己也怀疑“凭什么由我定规则？”但正是“如果不是我，那还有谁呢？”的念头，最终催生了这本经典。更震撼的是，我第一次从亲历者视角看到“敏捷宣言”的诞生——它并非精心设计的理论，起初不过是想在五花八门的“轻量级”开发流程里梳理出共同点。本书揭开了改变软件开发史的思想，是如何从实践中淬炼出来的。

如今AI浪潮席卷，总有人说“程序员要失业”。但读完这部编年史，我反而定了心：过去五十年，软件行业从打孔卡走到云计算，穿越了大型机时代的落幕、互联网泡沫的破灭、开源革命的浪潮，甚至经济危机的战壕——每一次灾难都像淬火的铁，越锤打越迸出火花。正如Uncle Bob在公司倒闭后的低谷里，反而沉淀出影响千万人的原则。AI或许能生成代码，但它永远学不会程序员骨子里的能耐：把混沌的需求变成清晰的逻辑，在试错中迸发出创造的火花。

如果你也在深夜盯着屏幕问“这行还能干多久？”，本书就像一剂解药。与其焦虑自己被取代，不如学学前辈们的生存智慧：把时代砸来的新问题，当成催生敏捷宣言的契机——用代码直面矛盾，在混乱中提炼答案。毕竟你看，那些定义行业的经典，最初也不过是几张为解决眼前麻烦而写的潦草笔记。而整部程序员史，正是由无数个这样的“眼前麻烦”垒成的通天塔。

那些改变世界的灵感，或许就藏在你我刚修改好的bug里。而这座通天塔，正等着你我添砖加瓦。

——谢昌明

前喜马拉雅电商中台负责人

数字文明的创世纪与牧羊人

当你在清晨的薄雾中窥见屏幕像素的真相时，人类文明的某个隐秘维度正在被悄然揭示。这不是关于未来的寓言，而是一部献给数字创世者的史诗——那些在0与1的荒原上开垦文明绿洲的现代游吟诗人，正是本书所书写的“我们程序员”。

这部横跨两个世纪的技术史诗，以战争催生的机电火花为序章，在晶体管与光纤的矩阵中铺展开来。作者Uncle Bob以亲历者的饱含热诚的手掌，触摸着从ENIAC到ChatGPT的每一道年轮。当我们凝视书中图4-2的那条发黄的穿孔纸带时，看到的不仅是二进制代码的原始胎动，更是一个新兴文明物种的进化图谱。从霍珀的编译器之梦，到汤普森的UNIX圣殿，从迪杰斯特拉的结构化启示，到敏捷宣言的群体觉醒，程序员群体完成了从“机器祭司”到“数字先知”的身份嬗变。

在数字文明的三重门扉前，我们看到了惊人的历史韵律：冯·诺伊曼架构与神经网络竟共享着相似的拓扑结构，巴贝奇的差分机遗稿里闪烁着区块链的原始基因。当Uncle Bob带领我们重走这条布满电子荆棘的朝圣之路，每个转折处都暗藏着对当下的启示——人工智能的崛起不是程序的终结，恰是“人原生”编程的进化；量子计算的曙光不是硅基的黄昏，而是计算本质的再探索与再发现。

书中那些被岁月尘封的细节，在今天折射出惊人的现实意义：图灵测试的原始提案预言了人机协作的伦理困境，Lisp语言的“括号森林”里生长着函数式编程的翠绿幼苗。当我们惊觉ChatGPT的“思考”轨迹与早期编译器有着惊人的同构性，方才理解Uncle Bob将程序员称为“细节管理者”的深意——我们不仅是代码的书写者，更是数字世界底层逻辑的构建者，是维系现实与虚拟的卡巴拉祭司。

在这个被算法重构的星球上，程序员群体的存在本身已成为文明的新器官。从《终结者》的系统叛变到《机械姬》的机械女仆，好莱坞的叙事嬗变恰是程序员地位升维的镜像：当新时代的年轻人像追星一样找Uncle Bob索要签名，意味着编程已从技术工种升华为文化基因。正如Uncle Bob暗示的，现代程序员正在书写新的创世神话——在数字世界的像素沙盒里，在GitHub的协作星空中，在开源社区的分布式智慧里。

但本书的深刻之处，在于始终保持着技术乐观主义者的清醒。当Uncle Bob指出“对程序员的需求随着机器能力的提升而增长”时，他揭示的不仅是职业发展的悖论，更是人类认知边界的永恒命题。人工智能不是程序员的掘墓人，而是认知增强的外骨骼；量子计算不是古典算法的丧钟，而是高层次算力的新跃迁。正如编译器挣脱

了汇编语言的桎梏，GPT类工具正在创造人机协作的新范式——这非但不是程序员的黄昏，恰是“自然语言编程”时代的黎明。

译罢掩卷，窗外的城市正被数据洪流冲刷重塑。那些在咖啡厅敲击键盘的身影，那些在深夜调试算法的开发者的瞳孔，那些在开源社区碰撞思想的灵魂，正在编织数字文明的新经纬。本书的价值，不仅在于记录了我们如何走到今天，更在于启发我们如何走向未来——当AI开始编写代码，程序员的真正使命才刚刚显现：从代码工匠进化为数字生态的架构师，从功能实现者蜕变为技术伦理的守门人，在数字比特的海洋中守护真善美的灯塔。

这或许就是数字文明给予“我们程序员”的终极命题：当机器学会思考，人类要如何为自己的未来“编程”？答案，或许就藏在这部代码史诗的字里行间，等待每个数字时代的创世者，也就是我们程序员，去续写新的篇章。

——茹炳晟

世上为什么要有程序员？

AI编程

最近编程领域最热门的话题肯定是“AI编程”了。我使用 AI 编程工具大概有两年。起初，由于公司资源有限，项目无法安排到开发排期，我只好自己动手，结果极其迅速地完成整个项目。后来编程工作就彻底离不开AI了。

以前见客户时，要么在PPT上画饼，要么基于现有系统配置数据演示，很难在一开始就用贴合客户需求的方式演示。然而现在完全变了，我用AI，在两天内从头搭建好一个搜索、转写视频的脚本。完备的流程，加上一整套成熟的用户界面，令客户备感满意。

这一切看起来像魔法，就像巴贝奇在维多利亚沙龙上展现差分机时，给人们带来的震撼一样，那时一定有人面对叮叮当当运行的机械巨兽惊呼：“机器能思考！”(巴贝奇的故事详细见本书第 2 章)。

兴趣

兴趣是引领我们前行的巨大动力。10 岁时，艾伦•图灵被《每个孩子都应该知道的自然奇观》(Natural Wonders Every Child Should Know)这本书所吸引，从此对科学有了新的认识。

在翻译过程中，我在B站发现了一位Up主，他喜欢在视频中分享机械计算机运行原理。原以为他是一名机械专业的大学生，后来发现他发布第一条原理视频时，还只是一名初中生，而大部分视频是在他读高中时发布的。在AI时代，很多专长会被无情地替代，大概只有好奇心和兴趣，才能指引我们。

Uncle Bob说，他在12岁时就立志成为程序员，起因就是妈妈送给他的一份生日礼物。他已经保存了60多年的这个玩具，现在看来很简单，就是用一些金属棒和机械装置搭建起来的装置，可以计数，还可执行加法运算等。那时候的他非常痴迷，完成了很多程序。他说：“那种拥有无限能力的纯粹喜悦感让我确定了自己的人生方向。我是一名程序员，而且我将永远是一名程序员。”(有关Uncle Bob是如何走上程序员道路的，详见第11章)。

希望这本书中某些故事，也能在某个时刻激起大家的兴趣。

细节

曾经给客户开发过一套Oracle考试模拟环境的软件。项目负责人是一位大学数学系教授，其中有项工作是要整理每个Oracle语句的状态转换数组。他带领几位数学系的研究生，在纸上画DFA，再提取状态值，最终形成数组。算下来处理一条语句需要几天时间，要完成整个Oracle语句集的构建需要几个月，还不能确保正确无误。我写了个程序，从bison的输出文本中自动提炼这些数据生成数组，整个过程不需要人工干预，准确率达100%，可快速测试正确性。写这个程序要处理很多细节，因为bison的输出文本并没有规范的语法，需要自己一点点依据字符个数和字符位置进行推演，有时差一个字符就跑不起来。

处理细节的重要性，在计算机发展的早期就已经体现出来。本书中讲到格蕾丝•霍珀时，她和同事们曾经要在布满数千个发热真空管和104°F水银罐的房间里操作UNIVAC 计算机。这台计算机每次加法运算的结果都比实际值大3，因此必须设计电路在每次操作后自动减去3。对于操作指令，有时纯粹为了对齐，就必须引入一些特殊的跳过操作(霍珀如何应对那么多恼人的细节，详见本书第 4 章)。这些看似没道理的细节，都必须做对，才能有正确结果。

世上为什么要有程序员?

Uncle Bob对此的回答是：“因为程序员是细节掌控者”。

我认为，虽然有了AI编程后，许多产品经理也能开发程序，但在复杂场景下(如架构设计、错误调试等)，有非常多的细节，仍然离不开真正懂程序运行原理和软件工程的人。

只有充满好奇，掌控足够多的细节，才能开拓新领域，达到足够高的成就。

翻译的几点说明

我很荣幸参与Uncle Bob这本新书的翻译工作。这本书回顾了编程发展史，讲述了许多重要的人物和事件，还包括了作者自身的很多经历。阅读本书，能让我们更好地理解编程的发展脉络，汲取先驱们的智慧与力量，在新的时代浪潮中找准自己的方向。

除了原脚注外，对于不易理解之处我添加了译者注，以帮助大家了解背景，大约有 3000 多字。原书脚注或参考文献有很多视频，但大都没有给链接，为方便读者，已将链接补充在电子脚注中。

——柳飞

对本书的赞誉

“我像Uncle Bob一样，职业生涯的大部分时间都在从事咨询、教学和参加计算机会议中度过。重要的是，本书中提及的许多人物我都遇到过，也曾和他们共进晚餐。所以这本书写的其实就是我职业生涯中的那些朋友，我可以告诉你，这些都是真实的故事。事实上，本书的文字表达非常精妙，研究得也很深入。”

——摘自Tom Gilb为本书撰写的“后记”[1]

“纵观各类相关书籍，除本书之外，我难以想象还有哪本书能如此全面、清晰地讲述计算机编程的早期发展历史。”

——Mark Seeman[2]

“这是一部引人入胜的计算机与编程历史书。精彩呈现众多伟大人物的生活片段，也生动描绘了Uncle Bob作为程序员的职业生涯历程。”

——Jon Kern，《敏捷宣言》的合著者[3]

“在本书中，Bob成功地将多位程序大师的工作经历编织成一个个引人入胜的故事，为我们提供了丰富的历史背景、人性化的事迹以及行业开创者们令人耳目一新的灵感之光，披露了诸多细节。身为这段波澜壮阔的历史的亲历者，Bob巧妙地将自己新颖的观察和见解融入其中。这一次，我们不仅完整欣赏Bob的故事，还了解了他对未来的思考。这是一本饶有趣味的书，读来如春风拂面，轻松惬意。”

——Jeff Langr[4]

---

1 Tom Gilb，软件工程方法学先驱，软件度量领域权威，“进化项目管理(又称Evo)”方法的提出者。他首次系统阐述迭代式开发理念，被视作敏捷方法的先驱。国际系统工程协会(INCOSE)终身成就奖得主。——译者注

2 Mark Seemann，软件架构师、技术作家，合著有多本技术书籍。精通函数式编程(尤其F#)、测试驱动开发(TDD)和领域驱动设计(DDD)。其博客ploeh blog是软件设计领域的知名技术资源。——译者注

3 Jon Kern，实时系统专家，敏捷联盟创始成员。自适应软件开发(Adaptive Software Development)方法的共同创立者。——译者注

4 Jeff Langr，软件工程顾问。专注于测试驱动开发(TDD)实践，维护着开源测试框架Hamcrest。——译者注

谨将本书献给蒂莫西·迈克尔·康拉德 (Timothy Michael Conrad)

序　言

简单敲入五个字符“vim .”，就能启动我最喜欢的编辑器。这可不是普通的编辑器，而是NeoVim。如今的NeoVim具备键绑定、语言服务器协议(LSP)、语法高亮、内置错误诊断等多种功能。经过全面定制后，NeoVim仅需数毫秒即可启动，实现近乎即时的文件编辑。即使项目有数千个文件，LSP也能迅速反馈项目的状态，将错误信息即时加载到快速修复菜单中，以便快速定位。只需要几个按键，我就能构建、启动、运行或测试项目。我的计算机甚至能通过AI将自然语言直接转换为代码！更神奇的是，这个AI还能在我编码时实时协作，瞬间生成大量(虽然质量存疑的)代码。这一切听起来令人惊叹。NeoVim的体验堪称美妙流畅、疾如闪电。然而，NeoVim却被某些开发者视为古董。“老顽固！”[1]他们用这样的称呼嘲讽使用NeoVim并花时间配置的人。毕竟，像IntelliJ那样全功能的集成环境唾手可得，甚至它所支持的功能之丰富已经远超我这个NeoVim用户的想象边界！

我之所以讲述这些，是因为我们正身处技术奇迹之中而不自知。遭此非议，不是因为我坚守“过时”的工具，也不是因为开发者间的理念之争——这些早已司空见惯。真正震撼的是：文本编辑这项人类智慧的结晶，当前在我们眼中竟如此稀松平常。自动补全、语法高亮、动态文档，这些曾经是具有石破天惊效果的功能，如今却成了开发者的空气与水。回溯历史，人类用了30年才从机器码跃升至高级编程语言，又等待半个世纪迎来语法高亮，直到20世纪70年代后期才出现跨语言的智能支持。编辑器的进化史，本就是一部浓缩的人类计算文明史。

比起使用古董编辑器，我更痴迷于聆听往昔的故事。那些“真正的程序员”能根据磁鼓存储器的旋转节奏来优化代码，以达到最佳的读取速度。多希望能亲眼见证大师们在打孔卡片上施展魔法！或许这只是一种浪漫的怀旧，但那个时代的每个突破都重塑了世界。在本书中，我有机会与计算历史中那些缔造过重大飞跃的创造者们并肩走过那段时光。我看到查尔斯·巴贝奇(Charles Babbage)用差分机震撼了维多利亚时代的沙龙，机械巨兽发出的叮叮当当的运转声，所产生的东西在当时看来一定是魔法。那种感受恰似今天我们初见大语言模型或Copilot自动生成代码时的震撼。我敢打赌，在那个时候你一定会在晚宴上听到客人惊呼：“机器真的能思考？”我感受到那时的

1　原文是“Luddite!”。Luddite，即卢德主义者，是19世纪英国的一个激进主义团体，反对工业革命，主张手工劳动和传统手工艺。——译者注

团队昼夜不停工作的压力，以及在二战关键计算中对更强计算能力的迫切需求，这些计算使得原子弹成为可能。他们没有Herman Miller人体工程办公椅，也没有时尚的站立式办公桌；见鬼，他们甚至都没有显示器或键盘！然而，他们取得了不可思议的成果，改变了人类发展的进程。本书是这段计算历史表述中最引人入胜的讲述之一。

如果一个程序员没听说过Uncle Bob的名号，我会非常惊讶。他在我们行业中绝对是多产的。多年来，我只是通过Twitter以及他在整洁代码和敏捷开发方面的重要贡献才知道Uncle Bob。在我心中，他一直是一种抽象的存在[1]，直到某天在Twitter上我与他互动后我的心态才发生了变化。通过电子邮件、电话，甚至播客的交流，我看到一个远比教科书更立体的罗伯特·C. 马丁：他务实，在需要时愿意做出让步。在播客节目中，最一致的评论是“他温文尔雅，笑容满面！”这反映了他豁达的性格和丰富充实的生活经历。他是一个真正的软件工程师，也是我们学习的榜样。

我个人已经厌倦了无数关于空格键、编辑器以及面向对象编程与函数式编程的争论，这些争论在X上激烈地展开着。我感兴趣的是，到底是谁创造了这些引发争论的技术。本书连接起过去的岁月和未来的希望，将更有意义的东西呈现给我们，而Uncle Bob就是讲述这个故事的完美人选。

——ThePrimeagen[2]

1　原文是“Avatar of AbstractBuilderFactory”。这个表述是对Uncle Bob的幽默化隐喻，结合了软件工程中的两个经典设计模式：Abstract Factory(抽象工厂模式)和Builder(建造者模式)。前者是创建相关对象家族的接口，隐藏具体实现。后者是分步骤构建复杂对象的模式。将两者合成为“AbstractBuilderFactory”这个虚构概念，其隐喻意义在于：强调Uncle Bob作为软件架构大师的形象，暗示其著作中强调抽象设计和模式应用的特点。——译者注

2　ThePrimeagen是YouTube频道号，主持人Michael Paulson是科技界颇具影响力的人物，以快速使用vim编辑器、编程直播、技术视频等内容而在Twitch和YouTube上闻名。——译者注

自　序

我即将为你讲述这一切是如何开始的。这是一个曲折的故事，讲述了那些非凡人物的生活与挑战，他们所处的非凡时代，以及他们所掌握的非凡机器。

但在我们深入探索这些曲折的历程之前，或许先来一个小小的预览是合适的——只是为了激发你的兴趣。

需要可能是发明之母，但没有什么比战争更能激发需求。我们这个行业的推动力正是由战争的剧变——尤其是第二次世界大战——所创造的。

在20世纪40年代，战争技术已经超越了我们的计算能力。仅靠人类操作的台式计算器根本无法满足来自战争各领域的计算需求。

问题在于，为了近似计算从火炮发射的炮弹到目标的路径，需要进行大量的加减乘除运算。这类问题并不能通过$d=rt$或$s=\frac{1}{2}at^2$这样的简单公式来解决。这些问题要求将时间和距离分解成成千上万个小段，并逐段模拟和近似弹道。这种模拟需要大量计算。

在过去几个世纪里，所有这些计算都是由成群的人用笔和纸完成的。直到20世纪，他们才得到了加法机来协助这项任务。完成这些计算以及组织执行计算的团队是一项艰巨的任务。[1]计算本身可能需要这些团队花费数周甚至数月的时间。

在19世纪，人们曾梦想过得到能够完成这些壮举的机器。甚至创造过一些功能比较简单的原型机。但它们只是玩具和奇观，是精英们在晚宴上展示的花哨装置。很少有人认为它们是值得使用的工具；考虑到它们不菲的成本使一切变得更遥远。

但第二次世界大战改变了这一切。需求迫切，成本变得无关紧要。于是，那些早期的梦想变成了现实，庞大的计算机器被建造出来。

那些编写代码和操作机器的人是计算领域的先驱。起初，他们被迫在最原始的条件下工作。编程指令实际上是通过在长长的纸带上逐一打孔来完成的，机器会读取并执行这些纸带。这种编程方式极其烦琐，且完全不容错。此外，这种程序的执行可能需要数周时间，其间需要详细监控和持续干预。例如，执行程序中的循环时，需要通过手动重新定位纸带以进行每次迭代，并手动检查机器状态以确定循环是否应终止。

1　要亲眼目睹这样的任务是如何进行的，推荐参加2024年圆周率日庆祝活动。在一周内，一个由数百人组成的非常协调的团队手工计算了100多位数字。“一个世纪以来最大的手工计算！[圆周率日2024]”由Stand-up Maths于2024年3月13日发布在YouTube上。

随着时间的推移，机电机器被电子真空管机器所取代，后者将数据存储在通过长长的水银管传播的声波中。打孔卡片最终被存储程序取代。这些新技术由早期的先驱们推动，并促进了进一步的创新。

20世纪50年代初的第一个编译器不过是带有特殊关键字的汇编器，用于加载和调用预先编写的子程序——有时来自纸带或磁带。后来的编译器尝试了表达式和数据类型，但仍然原始且缓慢。到了20世纪50年代末，约翰·巴克斯(John Backus)的FORTRAN和格蕾丝·霍珀(Grace Hopper)的COBOL引入一种全新的思维方式。程序员之前手工编写的二进制代码从此可由能够读取和解析抽象文本的计算机程序生成。

在20世纪60年代初，迪杰斯特拉(Dijkstra)的ALGOL提升了抽象层次。几年后，达尔(Dahl)和尼加德(Nygaard)的SIMULA 67再次将抽象层次提升到新高度。

结构化编程和面向对象编程就是从这些起点中诞生的。

同时，约翰·凯梅尼(John Kemeny)和他的团队在1964年通过创建BASIC和分时系统将计算带给了普通人。BASIC是一种几乎任何人都能理解和使用的语言。分时系统允许许多人方便地同时使用一台昂贵的计算机。

此后，肯·汤普森(Ken Thompson)和丹尼斯·里奇(Dennis Ritchie)在60年代末和70年代初，通过创建C语言和UNIX，开辟了软件开发的世界。从那以后，软件开发领域开始了飞速发展。

20世纪60年代的大型机革命之后是70年代的小型机革命和80年代的微型计算机革命。个人电脑在80年代席卷了整个行业，随后迅速迎来了面向对象革命、互联网革命和敏捷革命。软件开始主导一切。

9·11事件和互联网泡沫破裂使我们停滞了几年，但随后迎来了Ruby/Rails革命和移动革命。然后，互联网变得无处不在。社交网络蓬勃发展，然后衰落，而人工智能则崛起，雷霆万钧般涌来。

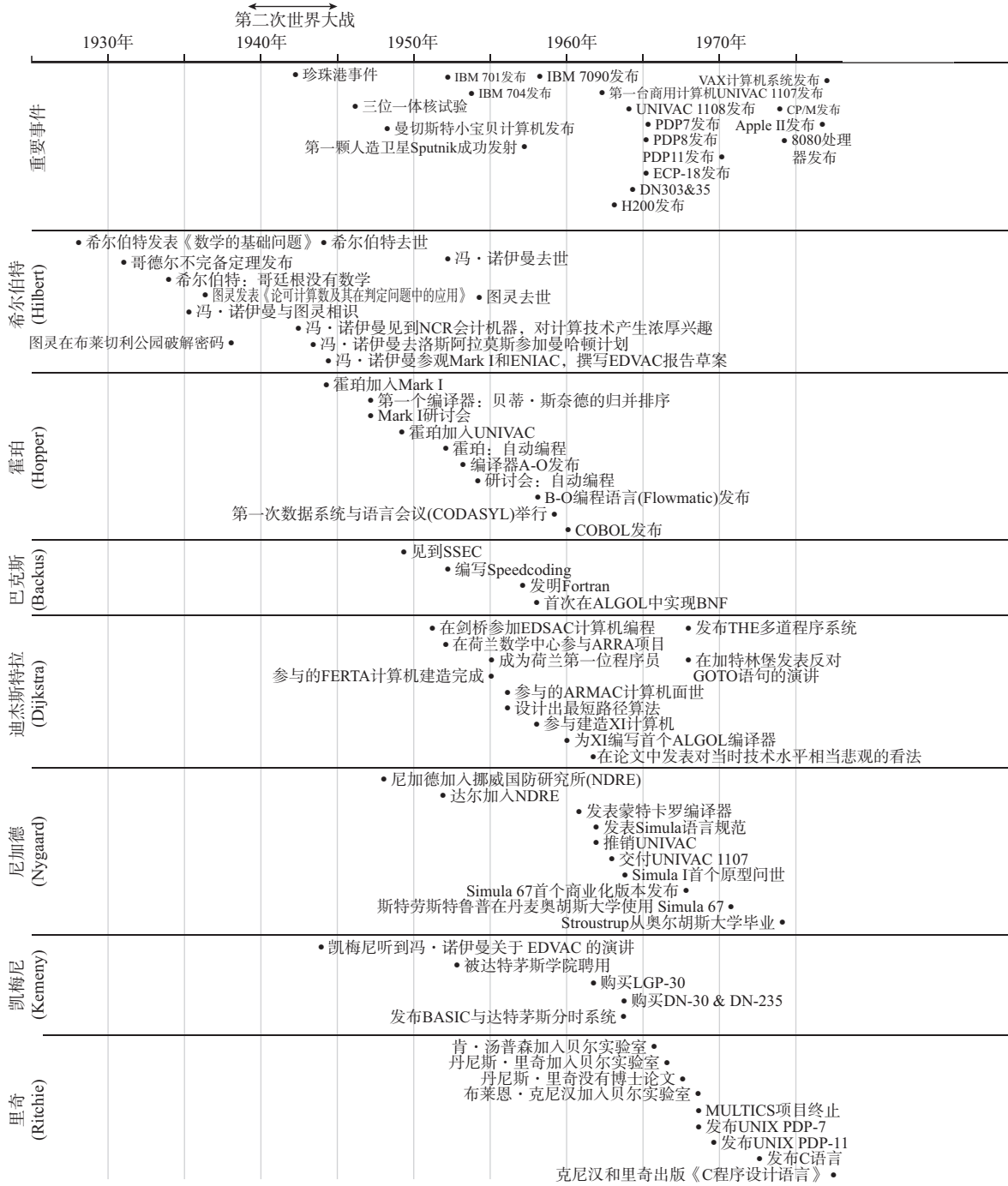
这让我们来到了现在，思考未来。所有这些，以及更多，都将在本书的页面上进行讨论。所以，如果你准备好了，那就系好安全带——因为这将是一次穿梭于时空的绮丽又狂野的旅程。

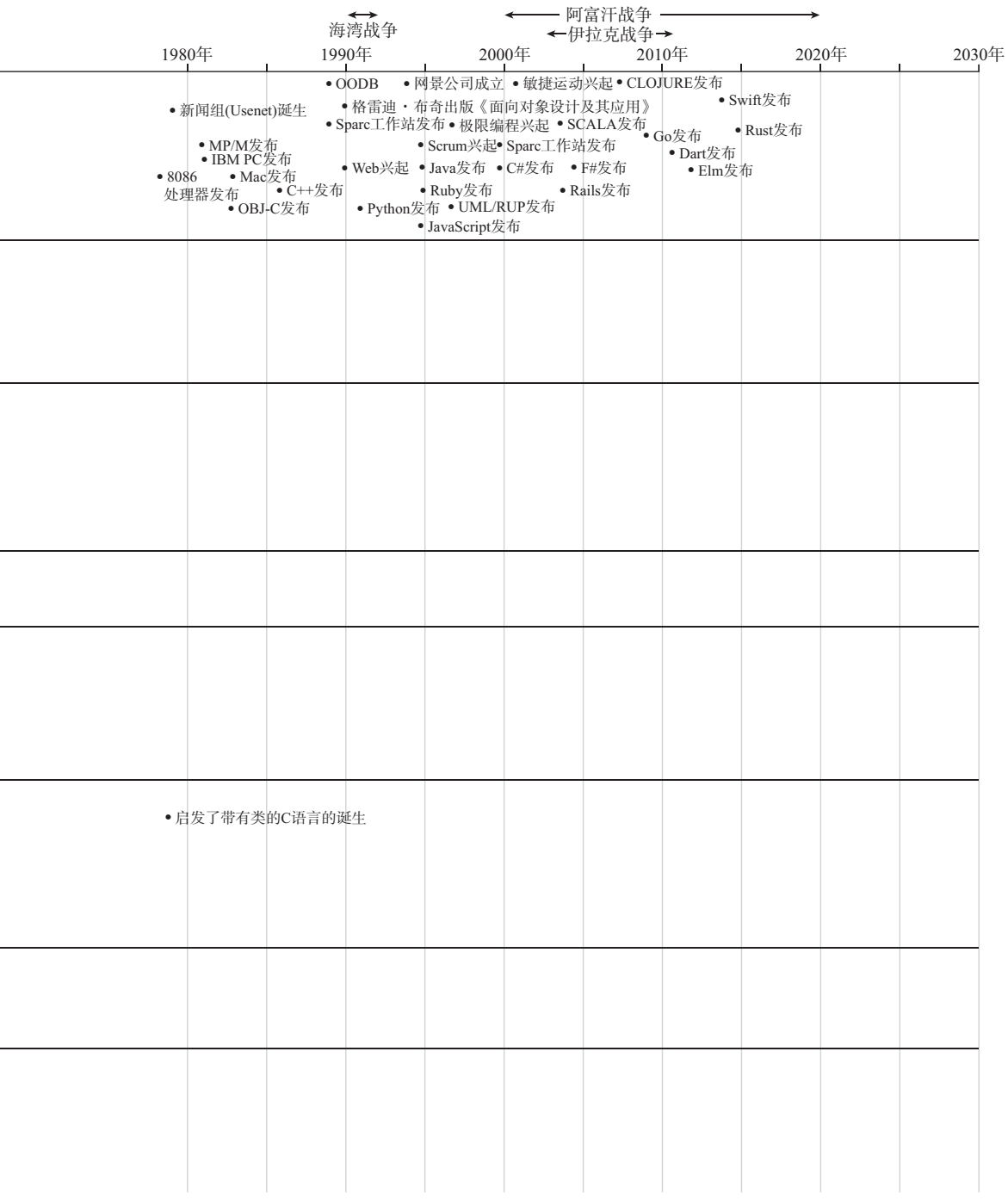
故事时间线

本书故事所描述的人物和事件都展示在这条时间线上。当你阅读故事中的相关事件时，可从这里定位，了解它们发生的背景。例如，你可能会发现一件有趣的事，如FORTRAN[1]和Sputnik[2]在时间上是巧合的，两者在无意中邂逅；或者肯·汤普森(Ken Thompson)在迪杰斯特拉(Dijkstra)提出GOTO语句有害之前就加入了贝尔实验室(Bell Labs)。

1　世界上第一门高级编程语言，IBM首次发布于1957年。详见书中章节。——译者注

2　世界上第一颗人造卫星，于1957年10月4日成功发射。详见书中章节。——译者注


第二次世界大战
1930年
1940年
1950年
1960年
1970年
重要事件
• 珍珠港事件
• IBM 701发布
• IBM 7090发布
VAX计算机系统发布 •
• IBM 704发布
• 第一台商用计算机UNIVAC 1107发布
• 三位一体核试验
• UNIVAC 1108发布
• CP/M发布
• 曼切斯特小宝贝计算机发布
• PDP7发布
Apple II发布 •
• PDP8发布
• 8080处理器发布
第一颗人造卫星Sputnik成功发射 •
PDP11发布 •
• ECP-18发布
• DN303&35
• H200发布
希尔伯特 (Hilbert)
• 希尔伯特发表《数学的基础问题》• 希尔伯特去世
• 哥德尔不完备定理发布
• 冯・诺伊曼去世
• 希尔伯特：哥廷根没有数学
• 图灵发表《论可计算数及其在判定问题中的应用》• 图灵去世
• 冯・诺伊曼与图灵相识
• 冯・诺伊曼见到NCR会计机器，对计算技术产生浓厚兴趣
图灵在布莱切利公园破解密码 •
• 冯・诺伊曼去洛斯阿拉莫斯参加曼哈顿计划
• 冯・诺伊曼参观Mark I和ENIAC，撰写EDVAC报告草案
霍珀 (Hopper)
• 霍珀加入Mark I
• 第一个编译器：贝蒂・斯奈德的归并排序
• Mark I研讨会
• 霍珀加入UNIVAC
• 霍珀：自动编程
• 编译器A-O发布
• 研讨会：自动编程
• B-O编程语言(Flowmatic)发布
第一次数据系统与语言会议(CODASYL)举行 •
• COBOL发布
巴克斯 (Backus)
• 见到SSEC
• 编写Speedcoding
• 发明Fortran
• 首次在ALGOL中实现BNF
迪杰斯特拉 (Dijkstra)
• 在剑桥参加EDSAC计算机编程
• 发布THE多道程序系统
• 在荷兰数学中心参与ARRA项目
• 成为荷兰第一位程序员
• 在加特林堡发表反对GOTO语句的演讲
参与的FERTA计算机建造完成 •
• 参与的ARMAC计算机面世
• 设计出最短路径算法
• 参与建造XI计算机
• 为XI编写首个ALGOL编译器
•在论文中发表对当时技术水平相当悲观的看法
尼加德 (Nygaard)
• 尼加德加入挪威国防研究所(NDRE)
• 达尔加入NDRE
• 发表蒙特卡罗编译器
• 发表Simula语言规范
• 推销UNIVAC
• 交付UNIVAC 1107
• Simula I首个原型问世
Simula 67首个商业化版本发布 •
斯特劳斯特鲁普在丹麦奥胡斯大学使用 Simula 67 •
Stroustrup从奥尔胡斯大学毕业 •
凯梅尼 (Kemeny)
• 凯梅尼听到冯・诺伊曼关于 EDVAC 的演讲
• 被达特茅斯学院聘用
• 购买LGP-30
• 购买DN-30 & DN-235
发布BASIC与达特茅斯分时系统 •
里奇 (Ritchie)
肯・汤普森加入贝尔实验室 •
丹尼斯・里奇加入贝尔实验室 •
丹尼斯・里奇没有博士论文 •
布莱恩・克尼汉加入贝尔实验室 •
• MULTICS项目终止
• 发布UNIX PDP-7
• 发布UNIX PDP-11
• 发布C语言
克尼汉和里奇出版《C程序设计语言》 •


海湾战争
阿富汗战争
伊拉克战争
1980年
1990年
2000年
2010年
2020年
2030年
• OODB
• 网景公司成立
• 敏捷运动兴起
• CLOJURE发布
• Swift发布
• 新闻组(Usenet)诞生
• 格雷迪・布奇出版《面向对象设计及其应用》
• Sparc工作站发布
• 极限编程兴起
• SCALA发布
• Go发布
• Rust发布
• MP/M发布
• Scrum兴起
• Sparc工作站发布
• Dart发布
• IBM PC发布
• 8086
处理器发布
• Mac发布
• Web兴起
• Java发布
• C#发布
• F#发布
• Elm发布
• C++发布
• Ruby发布
• Rails发布
• OBJ-C发布
• Python发布
• UML/RUP发布
• JavaScript发布
• 启发了带有类的C语言的诞生

前　言

在开始之前，我想先介绍一些关于本书及作者的信息，以便你更好地理解接下来的内容。

(1) 在撰写本书时，我做程序员已经60年了。虽然从12岁到18岁那段时间或许不应完全计入其中，但不管怎样，从1964年至今，我几乎全程参与了“计算机时代”的发展，亲身经历了这个领域中许多重要的(甚至是奠基性的)事件。因此，你即将阅读的这本书，实际上出自一位计算机领域早期探索者之手——如今这个群体的人越来越少了。尽管我们算不上是最早的开拓者，但确实从先行者手中接过了接力棒，继续前行。

(2) 本书所述内容的时间跨度长达两个世纪。许多人会发现，故事中提到的一些名字和思想有些陌生——仿佛被时间所遗忘。为此，在本书的结尾，我特意准备了术语表和其他重要人物名录，以帮助你更好地理解相关内容。

(3) 术语表中包含了本书中提到的大多数硬件设备的描述。如果你在阅读过程中遇到不熟悉的计算机或设备名称，可以查阅术语表，或许能找到相关信息。

(4) 其他重要人物名录列出了本书提及的其他重要人物。这份名单很长，但还远远不够。它仅仅列举了一些对计算机编程行业产生直接或间接影响的人物。书中提到的一些人物已经消逝在笼罩历史长河的时间迷雾中，在互联网搜索引擎里也难觅其踪影。当你浏览这些名字时，会惊讶于你在其中发现的人物。再仔细看看，你会意识到这份名单只是浮光掠影。再进一步看看这些人的离世时间，你会发现他们中的大多数都是最近才离开我们的。

电子脚注

本书提供脚注的电子版本。读者可扫描封底二维码，下载“电子脚注”文件。与纸质书中的脚注相比，“电子脚注”文件中增加了大量的参考网址。读者可直接单击网址浏览相关内容。

致　谢

再次感谢培生(Pearson)团队为本书出版所付出的辛勤努力，特别是Julie Phifer、Harry Misthos、Julie Nahil、Menka Mehta和Sandra Schroeder。同时，感谢制作团队对内容的深度淬炼与完善，包括Maureen Forys、Audrey Doyle、Chris Cleveland等成员。与他们合作总是令人愉快。

感谢Andy Koenig和Brian Kernighan帮助我建立了联系渠道。

特别感谢Bill和John Ritchie，他们为我提供了许多关于他们兄弟“Dear old DMR”[1]的宝贵资料。

感谢Michael Paulson(即ThePrimeagen)撰写了精彩的序言。

感谢Tom Gilb的盛情款待，感谢他的真知灼见，以及他写的后记，那是我读过的最有趣的后记之一。

感谢Grady Booch、Martin Fowler、Tim Ottinger、Jeff Langr、Tracy Brown、John Kern、Mark Seeman和Heather Kanser在书稿还非常粗糙的时候审阅了书稿，他们的帮助让书稿变得好多了。

一如既往，我要感谢我美丽可爱的妻子——我一生的挚爱——以及我四个出色的孩子和十个同样出色的孙子孙女。他们是我生活的全部。写软件只是为了好玩。

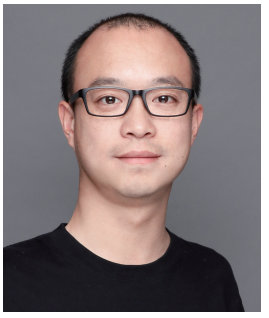
最后，我必须感谢上苍，我的生活如此完美——我生活在天堂。

1　DMR是指Dennis MacAlistair Ritchie，美国计算机科学家，在黑客圈子中通常被称为DMR。他是C语言的创造者，对计算机科学领域做出了重大贡献。详见第10章。——译者注

关 于 作 者

罗伯特·C. 马丁(在业内被尊称为Uncle Bob，即Bob大叔)自1970年起就开始从事程序员工作。他是Uncle Bob咨询公司的创始人，并与他的儿子Micah Martin共同创立了Clean Coders公司。马丁在各种行业期刊上发表了数十篇文章，并且经常在国际会议和行业展览上发表演讲。他撰写和编辑了多部书籍，包括《使用Booch方法设计面向对象的C++应用程序》(*Designing Object-Oriented C++ Applications Using the Booch Method*)、《程序设计模式语言3》(*Pattern Languages of Program Design 3*)、《更多C++精华》(*More C++ Gems*)、《极限编程实践》(*Extreme Programming in Practice*)、《敏捷软件开发：原则、模式和实践》(*Agile Software Development: Principles, Patterns, and Practices*)、《UML：Java程序员指南》(*UML for Java Programmers*)、《代码整洁之道》(*Clean Code*)、《程序员的职业素养》(*The Clean Coder*)及《函数式设计：原则、模式和实践》(*Functional Design: Principles, Patterns, and Practices*)。作为软件开发行业的领导人物，马丁曾担任《C++报告》(*C++ Report*)的总编辑三年，并担任敏捷联盟(Agile Alliance)的首任主席。

译者简介



茹炳晟，腾讯Tech Lead，腾讯研究院特约研究员，腾讯集团技术委员会委员，腾讯云架构师技术同盟入会主席，中国计算机学会(CCF)TF研发效能SIG主席，“软件研发效能度量规范”团体标准核心编写专家，中国商业联合会互联网应用技术委员会智库专家，中国通信标准化协会TC608云计算标准和开源推进委员会云上软件工程工作组副组长，国内外各大技术峰会的联席主席，出品人和Keynote演讲嘉宾，公众号“茹炳晟聊软件研发”主理人。著有技术畅销书《测试工程师全栈技术进阶与实践》《现代软件测试技术之美》《软件研发效能权威指南》《现代软件测试技术权威指南》《软件研发效能提升之美》《软件研发效能提升实践》《多模态大模型技术原理与实战》《高质效交付：软件集成、测试与发布精进之道》等，译有《软件设计的哲学(第2版)》《整洁架构之道》《现代软件工程》和《DevOps实践指南(第2版)》《精益DevOps》《持续架构实践：敏捷和DevOps时代下的软件架构》等。

柳飞，自在哪吒智能科技(nezha-ai.cn)创始人、首席服务官，程序员。《Java 性能优化权威指南》《Java 性能优化指南》和《整洁架构之道》等书籍的译者之一，疏于打理的公众号——“进化中的程序员”的主理人。马拉松业余凑热闹选手。

目　录

第Ⅰ部分　开端

第1章　我们是谁? ……3

第Ⅱ部分　技术巨擘

第2章　巴贝奇：第一位计算机工程师……11
2.1　生平 …… 11
2.2　数学用表 …… 12
2.2.1　制表之道 …… 12
2.2.2　有限差分法 …… 14
2.3　巴贝奇的远见 …… 17
2.4　差分机 …… 18
2.5　机械的符号系统 …… 19
2.6　派对魔术 …… 20
2.7　差分机的终结 …… 20
2.8　分析机 …… 22
2.9　阿达：洛芙莱斯伯爵夫人 …… 23
2.10　第一位程序员? …… 26
2.11　未竟之宏愿 …… 27
2.12　结论 …… 29
参考文献 …… 29

第3章　希尔伯特、图灵与冯 · 诺伊曼：第一代计算机架构师……31
3.1　大卫 • 希尔伯特 …… 31
3.1.1　哥德尔 …… 33
3.1.2　反犹主义风暴 …… 35
3.2　约翰 • 冯 • 诺伊曼 …… 36
3.3　艾伦 • 图灵 …… 38
3.4　图灵-冯 • 诺伊曼架构 …… 40
3.4.1　图灵的机器 …… 40
3.4.2　冯 • 诺伊曼的历程 …… 44
参考文献 …… 49

第4章　格蕾丝 · 霍珀：第一位软件工程师……51
4.1　军旅生涯：1944年夏天 …… 52
4.2　规范：1944—1945年 …… 55
4.3　子程序：1944—1946年 …… 58
4.4　研讨会：1947年 …… 59
4.5　UNIVAC：1949—1951年 …… 60
4.6　排序与编译器的起源 …… 64
4.7　酗酒：大约1949年 …… 64
4.8　编译器：1951—1952年 …… 65
4.9　A类编译器 …… 66
4.10　编程语言：1953—1956年 …… 68
4.11　COBOL：1955—1960年 …… 69
4.12　我对COBOL的吐槽 …… 72
4.13　无可争议的成功 …… 72
参考文献 …… 73

第5章　约翰 · 巴克斯：第一种高级语言……75
5.1　生平 …… 75
5.2　令人着迷的彩色灯光 …… 76
5.3　快速编码与701计算机 …… 78
5.4　对速度的需求 …… 80
5.4.1　分工 …… 84
5.4.2　我对FORTRAN的吐槽 …… 85
5.5　算法语言(Algol)及其他 …… 85
参考文献 …… 87

第6章 艾兹格·迪杰斯特拉：第一位计算机科学家……89
6.1 生平 …… 89
6.2 ARRA计算机：1952—1955年 …… 91
6.3 ARMAC计算机：1955—1958年 …… 94
6.4 ALGOL语言与X1计算机：1958—1962年 …… 95
6.5 阴霾如墨渐漫：1962年 …… 98
6.6 计算机科学的崛起：1963—1967年 …… 99
6.6.1 科学性 …… 100
6.6.2 信号量 …… 100
6.6.3 结构化编程 …… 101
6.6.4 数学证明的迷思 …… 101
6.7 数学：1968年 …… 102
6.8 结构化编程：1968年 …… 104
参考文献 …… 107

第7章 尼加德与达尔：第一种面向对象编程语言……109
7.1 克里斯滕·尼加德 …… 109
7.2 奥莱-约翰·达尔 …… 110
7.3 Simula语言与面向对象编程 … 111
参考文献 …… 119

第8章 约翰·凯梅尼：第一种“大众化”编程语言——BASIC … 121
8.1 约翰·凯梅尼的生平 …… 121
8.2 托马斯·库尔茨的生平 …… 123
8.3 革命性的想法 …… 123
8.4 看似不可能的任务 …… 124
8.5 BASIC语言 …… 125
8.6 分时系统 …… 126
8.7 操作计算机的青少年 …… 127
8.8 转型 …… 127
8.9 盲目先知 …… 128
8.9.1 共生关系？ …… 128
8.9.2 预言 …… 129
8.10 雾里看花 …… 132
参考文献 …… 132

第9章 朱迪思·艾伦 …… 133
9.1 ECP-18计算机 …… 133
9.2 朱迪思的经历 …… 134
9.3 辉煌的职业生涯 …… 137
参考文献 …… 138

第10章 汤普森、里奇与克尼汉 …… 139
10.1 肯·汤普森 …… 139
10.2 丹尼斯·里奇 …… 141
10.3 布莱恩·克尼汉 …… 144
10.3.1 Multics系统 …… 145
10.3.2 PDP-7与《太空旅行》游戏 …… 147
10.4 UNIX操作系统 …… 149
10.5 PDP-11计算机 …… 151
10.6 C语言 …… 153
10.7 克尼汉和里奇 …… 155
10.7.1 说服与合作 …… 157
10.7.2 软件工具 …… 157
10.8 结论 …… 158
参考文献 …… 158

第Ⅲ部分 技术拐点

第11章 20世纪60年代 …… 163
11.1 ECP-18 …… 166
11.2 父亲的支持和鼓励 …… 168

第12章 20世纪70年代 …… 169
12.1 1969年 …… 169
12.2 1970年 …… 172
12.3 1973年 …… 174

12.4 1974年……176
12.5 1976年……179
12.6 1978年……182
12.7 1979年……183
参考文献……184

第13章 20世纪80年代……185
13.1 1980年……185
13.1.1 系统管理员……186
13.1.2 pCCU……187
13.2 1981年……188
13.2.1 DLU/DRU……188
13.2.2 苹果Ⅱ……189
13.2.3 新产品……190
13.3 1982年……190
13.4 1983年……192
13.4.1 麦金塔内部剖析……192
13.4.2 电子公告板系统(BBS)……193
13.4.3 泰瑞达公司的C语言……193
13.5 1984—1986年：语音响应系统(VRS)……193
13.6 1986年……194
13.6.1 技工派遣系统(CDS)……195
13.6.2 字段标记数据(FLD)……195
13.6.3 有限状态机……196
13.6.4 面向对象编程(OOP)……196
13.7 1987—1988年：英国……197
参考文献……198

第14章 20世纪90年代……199
14.1 1989—1992年：克利尔通信公司……199
14.1.1 Usenet……200
14.1.2 Uncle Bob……200
14.2 1992年：*C++ Report*……201
14.3 1993年：Rational公司……201
14.4 1994年：教育考试服务中心(ETS)……203
14.4.1 *C++ Report*专栏……204
14.4.2 模式……204
14.5 1995—1996年：第一本书、会议、课程及OM公司……205
14.6 1997—1999年：*C++ Report*、统一建模语言(UML)和互联网泡沫……206
14.7 1999—2000年：极限编程…207
参考文献……209

第15章 千禧年……211
15.1 2000年：极限编程(XP)领导力……211
15.2 2001年：敏捷开发的兴起和互联网泡沫的破裂……212
15.3 2002—2008年：在困境中彷徨……213
15.4 2009年：《计算机程序的构造和解释》与色度键……214
15.4.1 视频……215
15.4.2 cleancoders.com……215
15.5 2010—2023年：视频、技艺与专业精神……216
15.5.1 敏捷开发偏离正轨……216
15.5.2 更多书籍……217
15.5.3 疫情期间……217
15.6 2023年：发展停滞期……217
参考文献……218

第IV部分 未来

第16章 编程语言……223
16.1 数据类型……224
16.2 Lisp……225

第17章 人工智能 …… 227
17.1 人类大脑 …… 227
17.2 神经网络 …… 229
17.3 构建神经网络并非编程 …… 230
17.4 大语言模型 …… 230
17.5 大型X模型的影响 …… 235
第18章 硬件 …… 237
18.1 摩尔定律 …… 238
18.1.1 多核 …… 238
18.1.2 云计算 …… 238
18.1.3 平台期 …… 238
18.2 量子计算机 …… 239
第19章 万维网 …… 241
第20章 未来的编程 …… 245
20.1 航空类比 …… 245
20.2 设计原则 …… 246
20.3 方法 …… 246
20.4 规范 …… 246
20.5 职业道德 …… 247
参考文献 …… 247
后记 …… 249
术语表 …… 257
其他重要人物名录 …… 273

第 I 部分

开 端

我们程序员，究竟是怎样一群人？世上为什么会有我们？我们竭力想要掌控的这些机器，又是什么？

第 1 章

我们是谁?

我们，程序员，是那些与机器对话并让它们运转起来的人。我们为机器注入灵魂，为经济和社会注入活力。没有我们，世界将停滞不前。我们——掌控这个世界！

有些人总以为自己是世界的主宰者，而他们制定的规则最终会传递到我们手上，由我们编写出在机器中执行的代码，正是这些代码支配着万物运转。

但这种举足轻重的地位并非与生俱来。在编程历史的早期，程序员是隐形的。所有目光都聚焦在计算机及它们所要解决的重要问题上。那些机器和它们的建造者才是时代的宠儿。没有人关注那些让机器运转起来的程序员。我们不过是时代的背景噪声。

关于那个原始年代，迪杰斯特拉(Dijkstra)曾说过：[1]

> “因为[每台计算机]都独一无二，[程序员]非常清楚地知道，他们的程序只在某台机器上才有意义。更显而易见的是，这些机器的寿命注定短暂，[程序员]深知自己的劳动成果几乎难以产生长远的价值。”

甚至，我们当时所做的几乎都不能被称为一种职业、一门学科，甚至连明确的工作都算不上。无论从哪个角度看，我们都像一群妖怪——用最笨拙也最令人抓狂的方式，硬生生地让那些不可靠、脾气暴躁、造价惊人的庞然大物(处理速度缓慢，存储小得可怜)偶尔能派上点用场。正如迪杰斯特拉所言：[2]

> “在那个年代，许多聪明的程序员用各种技巧，将不可能的任务塞进受各种限制的机器之中，从而在智力层面获得极大的满足感。”

程序员的这种形象维持了很长时间。事实上，20世纪整个60年代到70年代，程序员的形象更糟了。我们从穿着白大褂的幕后人员，变成了藏在格子间里的蓝领书呆

1　Dijkstra, Edsger W. The Humble Programmer[C]//ACM Turing Lecture. New York: ACM, 1972. [2025-03-20]. 可参考“电子脚注”中列出的网页。

2　同上。

子。直到近年，程序员才重新获得白领地位。即便如此，我们的社会仍未完全意识到对程序员的依赖，而我们也没有完全理解自己所掌握的力量。

多年来，我们一直被视作摆脱不了的麻烦。高管和产品经理们憧憬着不再需要程序员的那一天。他们的期待并非空想，因为机器的能力在不断增强。这种提升不是渐进式的改良，而是数十个量级的跨越式发展。

然而，不需要程序员的时代始终没有到来。事实上，对程序员的需求从未减少，反而随着机器能力的提升而增长。程序员不仅没有变得可有可无，反而愈发重要，愈发需要更高超的技能。如今的程序员就像医生一样专业化，你必须雇用正确类型的程序员。

尽管高管竭力削减对程序员的需求，但这种需求却与日俱增且趋多元。现在，他们认为AI会是解决方案。但请相信我，历史终将重演。随着技术能力的提升，程序员的需求量与专业地位只会水涨船高。

程序员从隐形人到不可或缺经历了一系列转变，这在当时流行的电影中可见一斑。1956年上映的《禁忌星球》(*Forbidden Planet*)中的机器人罗比(Robby)是典型的英式管家，是影片的角色之一，然而几乎没有人注意到他的代码其实是由技术精湛、思维活跃的“疯狂科学家”编写的。

1965年上映的《迷失太空》(*Lost in Space*)中的机器人也类似。虽然情节上设定机器人的程序是由史密斯博士编写的，但那个机器人形象独树一帜，机器本身才是真正的主角。

1968年上映的《2001：太空漫游》(*2001: A Space Odyssey*)中的HAL 9000是绝对的主角。而程序员钱德拉博士，只是在机器断开连接时现身过一次：当时机器还哼着优雅小曲“黛西·贝尔”。

这种模式一再重复。1970年的《巨人：福宾计划》(*Colossus: The Forbin Project*)中，机器仍是主导角色。程序员是无力反抗的受害者，最终沦为奴隶。

1986年的《霹雳五号》(*Short Circuit*)延续了这个传统，机器稳坐主角之位，程序员不过是天真又笨拙的陪衬。

转折出现在1983年的《战争游戏》(*War Games*)。主角仍是计算机约书亚(也叫WOPR)，但程序员开始参与解决问题了，尽管他扮演的只是配角，真正的英雄是故事中的翩翩少年。

真正的转变发生在1993年的《侏罗纪公园》(*Jurassic Park*)。计算机仍然很重要，但它们已经不是主角了。首席程序员丹尼斯·纳德利成为关键人物——尽管在故事中扮演的是反派。

时代已然不同。2014年8月，我在斯德哥尔摩为Mojang公司的程序员做演讲。Mojang就是《我的世界》(*Minecraft*)的开发商。演讲结束后，我们去一个被篱笆环绕的啤酒花园喝酒。一个年约12岁的男孩跑了过来，眼中闪烁着炽热光芒，对其中

一个程序员喊道：“你是Jeb吗？”只见一头红发、戴着眼镜的延斯・伯根斯坦(Jens Bergensten)默默地点了点头[1]，潇洒地给男孩签了名。

程序员已然是年轻一代心中的英雄，他们想成长为像我们一样的人。

为什么会有我们?

为什么会有我们？我们程序员——为了什么而存在？

也许这个问题过于偏向哲学层面了。那么换个角度：为什么社会需要我们？为什么人们愿意付钱雇用我们做事？为什么他们不亲自动手？

或许你认为是因为我们聪明——确实如此，但这不是根本原因。或许你觉得是因为我们是技术专家——这也没错，但同样不是关键。真正的原因可能会让你感到惊讶，出乎你的意料。事实上，这是我们特质的一个方面，需要勇气才能接受这个赤裸裸摆着的事实。

我们痴迷于细节。我们沉浸在细节中。我们在细节的洪流中绝不退缩、逆流而上。我们在细节的沼泽中艰难跋涉。我们热爱细节。我们乐此不疲。我们是……细节管理者。

但这并没有回答为什么需要我们。答案是，现代社会已经无法脱离数字世界，我们也已经无法摆脱手机了。

对于手机(手持电话机)，为什么我们称它们为电话？它们不是电话！它们与电话毫无关联。亚历山大・格雷厄姆・贝尔(Alexander Graham Bell)不会指着iPhone说，它与他和沃森发明的电话有什么直接关系。

手机根本就不是电话；手机是手持超级计算机。手机是通往信息、八卦、娱乐和一切的门户和入口。我们无法想象没有手机的生活。没有屏幕，我们可能会蜷缩成绝望的抑郁小球。

当然，我是在调侃，但这种调侃源于毋庸置疑的事实。如果所有手机突然停止工作，那我们的文明将瞬间崩塌。

这些和程序员有什么关系？为什么需要程序员来管理手机背后的所有细节？为什么人们不能自己处理这些细节？

你肯定遇到过有杀手级创意的人，他们想让你编程，实现他们的创意。他们相信创意能赚大钱，还愿意和你二八分成，只要你写代码就行。

是的，就是这样。只是写代码。没什么大不了的。

为什么他们自己不写代码？毕竟，这是他们自己的创意。为什么他们不写？

1　延斯・伯根斯坦(Jens Bergensten)，瑞典的游戏程序员和设计师，*Minecraft*的首席设计师和主要开发者，Mojang的首席创意官，Jeb是他的昵称。——译者注

表面上看，是因为他们不知道怎么做，但事实并非如此。其实他们知道怎么做。他们能用抽象的术语描述需求，但就是不能将需求转化为实现，这是为什么？

假设有个热情爆棚的企业家，不妨叫他吉米，完全相信只要他能在屏幕上画出一条红线[1]，就会有数十亿美金等着他。就那一条红线。我想说，每个人都想要这么一条红线，对吧？那么吉米应该怎么做呢？

吉米看着他掌心的手机，他看到的只是一个带有少许凸起、凹槽和空腔的长方体。他到底是怎么想的？当然，他需要一个程序员，因为程序员知道这种事情。

但等等。你有没有在拿着手机的时候打过喷嚏？你有没有看到过那些能放大屏幕细节的小水滴？只需要思考片刻，就会发现那些水滴中显现的是点，彩色的点。确实，红色、绿色和蓝色的点排列成看似矩形的网格。

于是吉米打了个喷嚏，看到了那些点，在那一瞬间，他意识到了什么。他的红线是由红点挨个连上绘制出来的！

如果他再多想一会，会意识到这三种颜色可以混合。如果他再想一会儿，会意识到必须有一种方法来控制单独点的亮度，从而创造出各种颜色。哦，他可能不知道RGB颜色，但每个小学生都知道一些颜色混合之后会变成其他颜色。所以理解这个概念并不那么难。

如果他再给自己一点时间考虑，他会意识到矩形网格意味着这些点有坐标。哦，他可能不记得高中的代数，也可能不记得笛卡儿坐标。但再次强调，每个小学生都理解矩形网格的概念。我的意思是，看在上帝的份上，当你拨打电话时，按钮就是按矩形网格排列的！

好吧，我知道，现在没人再拨打电话了，也没人知道为什么这个词叫拨号。但别管那个了。只需要几秒钟的宝贵思考时间，吉米就意识到连起那些线性排列的红点就能画出红线。

“具体怎么做呢？”吉米想知道。他还记得那个古老的线性公式：$y=mx+b$吗？可能不记得。但稍加思考，他肯定会意识到红线上每个点的纵/横坐标需要保持固定比例。他可能没有意识到，从根本上说，他正在重新发明微积分；但这没关系。理解垂直增量与水平增量成比例并不困难。

如果吉米继续思考，他会意识到他能看到那些点的唯一原因是他屏幕上的喷嚏沫起到了列文·虎克显微镜的作用。那些点一定很小！这意味着他必须画很多点，并且它们的坐标都必须遵循线性公式。他打算怎么做？

1　Beinerts, Lauris. The Expert (Short Comedy Sketch)[EB/OL]. YouTube, (2014-03-23) [2025-03-20]. 可参考“电子脚注”中列出的网页。(译者注：原视频是一则素描短喜剧，讲述了一个发生在团队内部的荒诞幽默的故事。一家公司为了实现战略目标，决定开启一个新项目，即画七条红线，要用绿色墨水、透明墨水来画，而且线之间要互相垂直。他们请专家来解决这个看似简单实则不可能的任务。Uncle Bob在此借用画线这个情节。)

更重要的是，如果他只画一排点，那么这条线会非常细。甚至可能看不见！所以他必须用一些粗细度来画线。

到这个时候，吉米已经连续思考一个多小时了，他意识到，这一小时里，他考虑的不是那条红线会让他赚多少钱，而是一直专注于点上。他可能也意识到，他只是刚刚触及了问题的皮毛。毕竟，他不知道如何让手机点亮那些点，不知道如何将坐标传递给手机。他也不知道如何让手机一遍遍地执行任务，以便他可以串起所有那些该死的点！

而最重要的是，他不愿再想这些了！他想回去思考他的红线如何让他赚到数十亿美金，以及如何在X平台[1]上投放红线广告，等等……

所以与那些该死的点说再见吧。吉米会让某个程序员去考虑这些。他不想去思考那个层次的细节。

但我们要想！这是我们的性格缺陷，也是我们的超能力。我们喜欢所有的那些细节。我们钟情于研究如何将屏幕上的点组合成一条红线。我们反倒不那么关心红线本身。

我们喜欢的是将所有那些微小细节组合成一条红线的挑战。

所以，为什么需要我们？因为社会需要那些喜欢考虑细节的人。那些人(我们)让社会可以思考其他事情，比如冰桶挑战活动、《愤怒的小鸟》游戏，或者在牙医办公室等得无聊时玩的纸牌接龙。

只要绝大多数人逃避细节，他们就需要我们这些和细节“死磕”的人。这就是我们。我们是全世界的细节管理者。

1　X平台，即Twitter，现更名为X。——译者注

第Ⅱ部分

技术巨擘

“计算机技术的发展史源远流长，我们常常遗忘和忽略这段历史。但事实上，在我们之前，众多先驱者以实际行动，在技术领域树立了践行职业道德的典范标杆。”

——肯特·贝克(Kent Beck)，2023年(以纪念不久前离世的Barry Dwolatzky)

本书第Ⅱ部分将讲述这些技术巨擘的故事。这些故事的主角都是在计算机领域取得非凡成就、对行业产生深远影响的程序员。通过他们的经历，你将看到他们如何应对技术挑战与人生考验。我的目标是让你从两个维度深入理解这些非凡人物——既见其专业造诣，亦观其人格魅力。

个人层面，我希望你能认识到，这些技术先驱与我们并无二致。他们同样经历喜怒哀乐，会犯错误也会修正方向，在摆脱困境的过程中成长，在收获成功之际感受由衷的喜悦。

技术层面，作为程序员的你最能感同身受。唯有真正编写过程序的人，才能深刻理解他们当年面临的技术挑战。但愿通过阅读这些故事，你能建立起同行之间才有的那种深切敬意——那种唯有实践者才能真正体会的专业认同。

本部分难免遗漏许多技术先驱。这绝非因为他们不够伟大或不值得关注，实因篇幅所限不得不有所取舍。我希望我的选择是明智的。

第 2 章

巴贝奇：第一位计算机工程师

在程序员和计算机爱好者中流传着一个广为人知的说法：查尔斯·巴贝奇(Charles Babbage)是通用计算机之父，而阿达·金·洛芙莱斯(Ada King Lovelace)伯爵夫人则是第一位程序员。各种奇闻轶事层出不穷，既有虚构的也有虚实参半的。但真相往往比传说更加引人入胜。

2.1 生平

1791年12月26日，查尔斯·巴贝奇出生在英国萨里郡。作为富裕的银行家之子，他自然成为19世纪初英国上流社会的一员，并继承了一大笔财富[1]。这使他能自由自在地追求自己各种各样的兴趣爱好。

他一生撰写了6部著作和86篇科学及杂项论文，涉猎的领域包括数学、国际象棋、开锁技术、税收制度、人寿保险、地质学、政治、哲学、电磁学、仪器制造、统计学、铁路工程、机床技术、政治经济学、潜水装置、潜艇设计、航海技术、旅行见闻、语言学、密码分析、工艺美术、天文学和考古学等。

最重要的是，巴贝奇是个天生的发明人才，是多种机械装置的发明者。他发明了强制通风供暖系统、眼膜曲率镜、缆索驱动的邮件递送系统、井字棋游戏机，以及许多突发奇想的装置。然而，所有这些发明都没有为他带来实际的收益。大多数发明都停留在图纸阶段，从未付诸实践。

巴贝奇更是一位社交达人，堪称卓越的宴会策划家和故事创作大师。他自己举办了许多晚宴，更是各类社交场合炙手可热的座上宾。1843年的某段时间，他每天的邀约竟然多达13场——连周日也不例外[2]。

他的社交圈星光熠熠：查尔斯·狄更斯、查尔斯·达尔文、查尔斯·莱尔、查尔斯·惠斯通(查尔斯含量过高)、乔治·布尔、乔治·比德尔·艾里、奥古斯都·德摩

1 价值10万英镑的地产，使他成为独立且富有的人，并能资助他的科学研究。

2 见本章参考文献[7]，第173页。

根、亚历山大·冯·洪堡、彼得·马克·罗杰特、约翰·赫歇尔，以及迈克尔·法拉第等都与他交游甚密。

他一生获得过诸多殊荣：1816年当选为英国皇家学会院士；1824年因发明计算机器(我们即将讨论)而获得英国天文学会首枚金质奖章；1828年当选为剑桥大学的卢卡斯数学教授[1]，并任职至1839年。

可以说，他是一个受欢迎且人脉广泛的人。

尽管社交场上如鱼得水，又获同行认可，巴贝奇却很难被称为成功人士。他绝大多数的努力都以失败告终。他也不是容易相处的人。同时代的人认为他脾气暴躁、固执己见，常有刻薄自私的言行——是个暴躁的天才[2]。

他经常公开发表抨击权贵的信件，转头又向这些权贵寻求项目资助。可以说，温文尔雅和谨慎与他沾不上边。

最终，甚至连英国首相Robert Peel爵士都发出疑问："我们该怎么摆脱巴贝奇和他的计算机器？"

2.2　数学用表

程序员查尔斯·巴贝奇的故事开始于1821年夏天，巴贝奇与其毕生挚友约翰·赫歇尔正在为英国天文学会校对数学用表。表格由两个独立团队分别制作。如果两个团队都计算正确，那两组表格应该完全一致。巴贝奇和赫歇尔要逐一比较这两组表格，找出并解决所有差异。表格中有成千上万的数字，每个数字都有十多位。两人花了数小时进行枯燥、繁重且高度专注的工作，轮流朗读数字并由对方验证它们是否相同。每当出现书写错误或数字被误读时，他们必须暂停，再次检查，解决或标注差异。这项工作令人脑袋发木、情绪沮丧且筋疲力尽。

最终，巴贝奇忍无可忍地喊道："愿上帝让蒸汽来执行这些计算吧！"[3]

就这样，巴贝奇作为程序员的热情被点燃了。他雇用工匠制作零件，不到一年就组装出了一台可以运行的小型计算模型。这台微型原型机承载着他心目中更大、更复杂的梦想：有朝一日，机器将替代人工承担制作数学用表的苦差事。

2.2.1　制表之道

数学用表的需求无处不在：数学家需要它们绘制曲线，航海家依赖它们定位经

1　许多重要人物都曾担任过卢卡斯教授席位，包括艾萨克·牛顿(1669—1702)、保罗·狄拉克(1932—1969)、斯蒂芬·霍金(1979—2009)。或许2395年之后，这个席位就不再需要人，而是由数据担任了。

2　《暴躁的天才》(*Irascible Genius*)是Maboth Moseley于1964年撰写的关于巴贝奇的书籍。可参考"电子脚注"中列出的网页。

3　见本章参考文献[7]，第10页。

纬，天文学家凭其测算星轨，工程师借其设计机械，测绘师依其丈量土地。他们需要的表格种类繁多，包括对数表、三角函数表、弹道轨迹表、潮汐时刻表等。清单冗长且无完结之时。更重要的是，每个数据都需要同时满足高正确率与高精准度要求。表格中的每一项都必须准确无误，并精确到好几位小数。

如何制作这些表格呢？如何计算数万个对数，并精确到小数点后六位、八位甚至十位呢？如何以此精度按弧秒[1]为单位，计算每个角度的正弦、余弦和正切值呢？乍一看，这简直是不可完成的任务。

但人类终究是智慧生物。可以证明，有解决方案。

第一步是将超越函数变回多项式。对数、正弦和余弦都是超越函数，这意味着它们不可以用多项式精确计算。但它们可以用多项式来近似。

考虑笛卡儿坐标中的一个正弦波。y值在1和−1之间波动，x周期为2π。如图2-1所示。

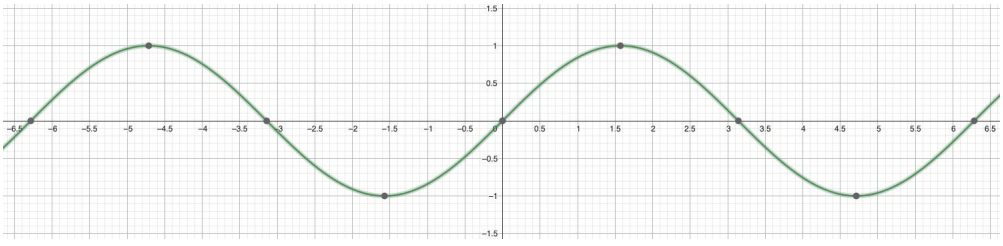


图 2-1

现在用$y=-0.1666x^3+x$叠放到正弦波上。如图2-2所示。

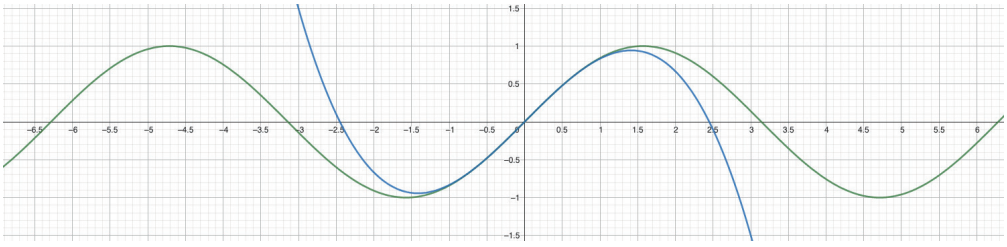


图 2-2

当x接近0时，拟合效果还不错，但仍有提升空间。试试$y=0.00833x^5-0.1666x^3+x$。如图2-3所示。

1　弧秒(arc second)，又称角秒，是量度平面角的单位，符号是"。换算关系：1° = 3600"。——译者注

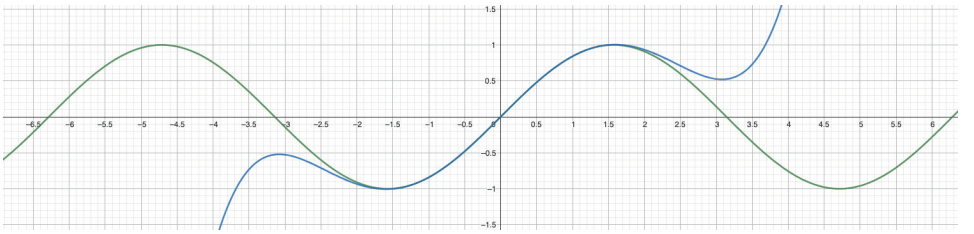


图 2-3

嘿！效果显著！但还可以做得更好。考虑以下公式：

$$y=-0.0001984x^7+0.00833x^5-0.1666x^3+x$$

如图2-4所示。

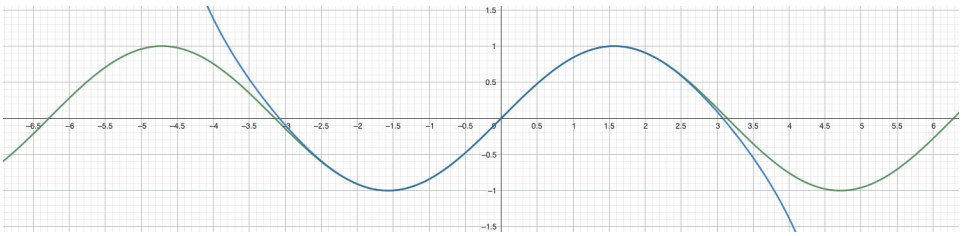


图 2-4

哇！在−π/2和π/2之间，两条曲线真的非常贴近了。实际上，对于−π/2，多项式的值是−1.00007，与真实值−1的误差出现在第五位小数，精度已达要求。

所有这些系数，来自简单的泰勒展开(Taylor expansion)：

$$\sin(x)=x-x^3/3!+x^5/5!-x^7/7!\cdots$$

好了，现在已经把正弦函数从超越函数拉回到多项式了，这些多项式仍然很复杂，该如何计算而不至于使自己陷入癫狂呢？

假设要构建一张0到π/2之间，以弧秒为单位的每个角度的正弦值表。这个范围内有324 000弧秒。而一弧秒的值是0.000004848136811弧度。你真的愿意将这样一个数字进行三次方、五次方和七次方运算；然后将它们分别除以6、120和5040；最后将它们相加和相减——一共340 000次吗？

幸运的是，有一种更好的方法：有限差分法。

2.2.2　有限差分法

以简单多项式为例，比如$f(x)=x^2+3x-2$。计算x=1～5的值。如图2-5所示。

计算一阶差分(相邻函数值之差)，如图2-6所示。

x	x^2+3x-2
1	2
2	8
3	16
4	26
5	38

图 2-5

x	x^2+3x-2	$d1$
1	2	
2	8	6
3	16	8
4	26	10
5	38	12

图 2-6

然后计算二阶差分(相邻一阶差分之差)，如图2-7所示。

x	x^2+3x-2	$d1$	$d2$
1	2		
2	8	6	
3	16	8	2
4	26	10	2
5	38	12	2

图 2-7

啊哈！二阶差分是常数2。事实上，任何n次多项式的n阶差分都是常数。

好，那么x=6时函数值是多少？不必费心计算多项式，注意此时的一阶差分应该是14，因为只需要将x=5时的二阶差分(2)加上一阶差分(12)即可。再将新的一阶差分加上38即得52，这就是正确答案。整个过程只需要两次加法！

还可以继续：f(7)是68，因为14+2是16，而16+52是68；f(8)是86，因为16+2是18，而18+68是86。如果想要一张包含所有f值的表格，只需要将两个数字相加，即可生成。没有乘法，没有减法：只需要两次简单的加法！

这种方法对正弦函数有效吗？假设要构建一张0到~π/2的正弦值表，步长为0.005。再假设使用的是7阶泰勒展开。只需要计算前8个值，后续值就可以通过累加差分来计算。

首先计算x=0.005的多项式值，如图2-8所示。

$$0.005 - 0.005^3/6 + 0.005^5/120 - 0.005^7/5040$$

图 2-8

为避免分数转化为小数时精度降低，计算过程保持分数形式，如图2-9所示。

```
  0.005 – 0.000000125/6 + 3.125E–12/120 – 7.8125E–17/5040
=5/1000 – 125/6000000000 + 3125/120000000000000000 – 78125/504000000000000000000000
=1/200 – 1/48000000 + 1/38400000000000 – 1/64512000000000000000
=3225586560016799 99/64512000000000000000
```

图 2-9

最终的值是：

0.004999979166692708

这就是sin(0.005)12位精度的近似值。

但问题来了：分母太大，导致最终结果小数点后的有效位数过多(18位)。这意味着，过程中为了保持精度多做了很多位的计算。

处理这么多大分母分数并不比计算泰勒展开更容易。试用不一样的方法。将所有这些分数乘以10^{30}，再约分以缩小分母。

现在就可以得到，3149986875016406240234375000000/63。除法结果取整，得到499997916669270831783234126 9，非常接近于sin(0.005)乘以10^{30}。

由此可见，采用大幅缩小分母的方法，即便取整，有效位数也不会有太多损失。现在，用这种技术计算接下来的七个正弦值，图2-10显示了前八个值。

再计算七个一阶差分，如图2-11所示。

然后是二阶、三阶、四阶、五阶和六阶差分——最终是常数，它们就该如此。如图2-12所示。

```
3149986875016406240234375000000/63
3149908125508592509765625000000/63
3149750628461698642578125000000/63
3149514387813142607421875000000/63
3149199409468928310546875000000/63
3148805701303497939453125000000/63
3148333273159535087890625000000/63
3147782136847718662109375000000/63
```

图 2-10

```
-124999218751953125000000
-749985937589843750000000/3
-112495546931445312500000/3
-149989687721093750000000/3
-187480078776367187500000/3
-224965782839453125000000/3
-262445862769726562500000/3
```

图 2-11

```
-37498828133398437500000/3
-124989843908203125000000
-124980469298828125000000
-37490391055273437500000/3
-124952346876953125000000
-124933599767578125000000
18749609375000000000/3
9374609375000000000
37497343750000000000/3
46869921875000000000/3
18747109375000000000

93742187500000000000/3
93735156250000000000/3
93725781250000000000/3
93714062500000000000/3

-234375000000000
-312500000000000
-390625000000000

-78125000000000
-78125000000000
```

图 2-12

所以，前七个差分值如图2-13所示。

再次强调，由于分子很大而分母很小，略去小数只考虑整数很安全，低阶的小数在加法时不会造成太大影响。最终变为整数差分，如图2-14所示。

现在要做的，就是将这些整数相加，填充正弦函数表的每一行。最终可以得到一组非常漂亮的、放大了10^{30}倍的正弦值。如图2-15所示。

```
3149986875016406240234375000000/63
-124999218751953125000000
-37498828133398437500000/3
18749609375000000000/3
9374218750000000000/3
-234375000000000
-78125000000000
```

图 2-13

```
499997916669270831783234126 9
-124999218751953125000000
-12499609377799479166666
6249869791666666666
3124739583333333333
-234375000000000
-78125000000000
```

图 2-14

```
0.005   499997916669270831783234126 9
0.01    999983333416666468253968253 8
0.015  1499943750632809109933035714 1
0.02   1999866669333307936507936507 8
0.025  2499739591471233065166170634 8
0.03   2999550020249566071428571428 2
0.035  3499285460433619259982638887 6
0.04   3998933418663415873015873012 4
0.045  4498481403766023423549107135 1
0.05   4997916927067832341269841254 6
0.055  5497227502706772118365575369 5
0.06   5996400647944457142857142811 4
```

图 2-15

自动计算机出现之前，数学用表的制作方式基本上就是这样。负责制表的首席数学家们推导出最适合逼近目标超越函数的多项式。然后将这些多项式分给大约六名熟练的数学家，这些数学家将函数划分为较小的区间，并为每个区间计算差分表。最后，再将差分表分配给由数十人组成的“计算员”团队——这些人的技能足以完成简单的加减运算，通过持续累加，构建每个区间内的数表条目。

数学用表有成千上万行。“计算员”需要进行数万次高精度加法运算。他们的工资很低，工作枯燥又乏味，全部得靠钢笔[1]和纸完成。

通常，“计算员”会被分成两队，执行相同的任务。如果两队都完美地完成工作，结果应该完全一致。而这就是1821年夏日的那一天，巴贝奇和赫歇尔所做的核对工作。

2.3　巴贝奇的远见

现在我们应该能理解他的愤懑之言“愿上帝让蒸汽来执行这些计算吧！”背后的真实含义了。如果“计算员”能被可靠的机器取代，他和赫歇尔就不需要无休止地核对了。

更重要的是，巴贝奇预见到，在未来，高效计算对于科学的进步而言将是不可或缺的关键要素。

1　美国第一家铅笔工厂于1861年开业。当然，之前也有类似的厂家，但要让铅笔成为家家户户都有的用品，还得靠大规模生产才行。

1822年，就在他发出“蒸汽”抱怨后不久，他极具预见性地写道：

> “我仍然要大胆预言：当数学公式的实际运算量堆积如山，成为一种持续存在的阻力，最终阻碍科学的良性发展时，除非采用这种方法[1]或找到其他替代方案，把我们从烦琐数值细节的沉重负担中解放出来，否则科学发展将举步维艰。”

遗憾的是，对巴贝奇而言，预言的实现还要等待一个多世纪。但它终究来了。正如我们将在后续章节看到的，这个时代以迅猛之势降临。

2.4　差分机

巴贝奇最初构想的这台机器本质上是一台巨型加法机——但带有精妙的设计。它包含6组20位十进制寄存器，可表示六阶差分，因此能处理六次多项式运算。机器每完成一个运算周期(每摇动一次曲柄)，第六寄存器就会与第五寄存器相加，第五与第四相加，第四与第三相加，以此类推。纯粹的加法运算，仅此而已。

这台机器由25 000个独立部件组成，高8英尺，长7英尺，宽3英尺，重约4吨。

为何简单的加法需要如此复杂的机器？为何需要工业级规模的设备才能完成？

问题具有双重性。首先，机器必须绝对精确。任何活动部件在非必要时刻都必须保持静止——不允许任何振动、摩擦或其他寄生运动导致部件意外位移。为此，巴贝奇设计了大量锁定/解锁机制，确保机器在数千次曲柄转动中保持结构完整性。

第二个难题是小学生都熟悉的进位问题。当某位数字从9变为0时，需要向左进位1。当时的传统机械装置采用齿轮表示每位数字，通过凸齿结构触发相邻齿轮进位。这种设计导致转动齿轮所需的力量取决于进位传播次数——给999999999999加1需要非常大的动力。这种高负荷不仅容易产生误差，还会加速零件磨损。更糟糕的是，由于机器需要手动摇柄操作，操作者会感受到手臂承受的力量剧烈变动。

为解决这个问题，巴贝奇设计了一种非常巧妙的进位记忆机制：在运算周期内先记录进位，再分步传播。这种记忆体现为每个齿轮旁的小型控制杆具有“进位”与“无进位”两种状态。通过螺旋阶梯状排列的传动杆系统，每个传动杆都会检测对应控制杆的状态，若处于进位状态则向上位齿轮加1。这种设计确保每个进位操作都在前一个进位完成后进行。

整套机制是一个精巧设计的机关，非常巧妙。

巴贝奇本质上是一名程序员。他编写的程序以杠杆、轮子、齿轮和曲柄的形式存在。尽管如此，他仍然是一名程序员。他的机器执行的是一个精准的连续加法过程。

1　指他关于差分机的想法。

用现代代码实现这个程序大概是这样的：

```
(defn crank [xs]
    (let [dxs (concat (rest xs) [0])] (map + xs dxs)))
```

是的，三行代码(约74个字符)就替代了25 000个机械部件。

不过这种比较或许有失公允。毕竟这段代码运行在MacBook Pro上——这台仅重一两磅的笔记本电脑，其复杂程度已远超巴贝奇的差分机。

细心的读者可能注意到，前文正弦函数示例中有些差分是负数。然而，巴贝奇的机器只能表示正整数。他是如何解决这个问题的？

具体实现方法已不可考，不过可以用十进制补码[1]的方式实现。假设数字有20位且忽略最高位的进位，那补码就可用来表示负数。

例如，1的补码是99999999999999999999。两数相加，忽略最高位的进位，即得0。因此，1的补码可视为-1。

具体步骤是：取数字的每一位，用9减去该数，作为补码的对应位。每一位都处理完毕后，最后的结果再加1。

以314159265358879323 84为例，先用“9减去”每一位，得到68584073464112067615，然后加1，得到它的补码68584073464112067616。和原数相加并忽略最高位进位，结果即为0。

补码是一种创建负数的有效方法，可以把减法转化为加法。

2.5　机械的符号系统

巴贝奇是程序员的另一个证据是：他面临着一个前所未有的工程难题——机械动力学。他的机器部件以复杂的方式在复杂的时间点上移动。这些部件还以复杂的方式相互耦合与分离。这种规模的机械复杂性前所未有，需要某种形式化的表示方法。

为此，巴贝奇创造了一套专门描述机械动力学特征的符号系统。这套系统包含时序图、逻辑流程图及多种符号约定，能够精确标注部件的运动状态、静止状态及其相互关系。巴贝奇认为这套符号是交互部件之间的“通用抽象语言”。事实上，他觉得这套符号可应用于任何形式的机械或非机械交互系统。

关于这套符号系统，他说：[2]

> “要在大脑中同时记住复杂机器的所有协同运动和连续动作已属不易，而要精确协调这些预设动作的时序更是难上加难。这迫使我寻求一种方法，只需

1　原文中“nine’s complement”和“ten’s complement”说法混用，为避免混乱，统一译作“十进制补码”，不引起歧义的情况下译作“补码”。——译者注

2　见本章参考文献[7]，第119页。

要扫视就能定位特定部件，随时掌握其运动/静止状态，理清其与其他部件的运动关联，必要时还能追溯其运动源头直至原始动力。”

常有观点认为巴贝奇未能意识到机器所操作的数字可以代表其他符号系统。但从此处可以看到，巴贝奇非常自如地发明了一种全新的动力学符号形式化方法。这说明，符号操作对他来说并不陌生。

2.6　派对魔术

还有一个证据说明巴贝奇是程序员：他会用差分原型机在晚宴上表演派对魔术。

举一个关于他的恶作剧的例子：他在客厅里聚集了一群伦敦的社交名流，围绕着他的小型原型机。他先让他们看了一眼，结果寄存器中是0，然后他转动曲柄。数字从0变到2。他问客人，继续转动曲柄会出现哪个数字，那些猜中4的人很是惊喜。他再次询问，所有人都会猜6，然后是8，再后是10。就在他们即将感到乏味时，接下来转动曲柄就会得到42。

只要理解差分机的工作原理，设计差分机以产生这样的惊喜并不困难。毕竟序列0、2、4、6、8、10、42之间有差分，在设置好的差分机上逐渐累加就能重现这个数列[1]。

然后巴贝奇告诉他的客人，这台机器正遵循着只有他知道的隐秘法则。他还会继续和他们说，神秘数字42类似于《圣经》中分开红海或治愈病患的神迹。在他看来，上帝正是为宇宙编写了隐秘法则的程序员，而这些法则唯有造物主知晓。[2]

2.7　差分机的终结

1823年，巴贝奇凭借其在英国天文学会和伦敦皇家学会的良好声誉，以及与议会中支持者的关系，成功获得了政府委员会的资助来建造他的机器。

这项资助决策引发了争议。一些人认为他的设备具有实用价值，而另一些人则认为这笔费用不划算。毕竟，失业的理发师的薪资要求并不高——而且他们同样能做加法运算。

但巴贝奇是位热忱的布道者和极具魅力的演说家。他时常向人们描绘这台机器的优势及其蕴含的惊人潜力，甚至声称自己都未能完全掌握它的所有可能性。凭借对项目的热情阐述和清晰表达，他总能让听众听得如痴如醉。

1　借助DeepSeek r1，很容易找出一组能生成该序列的多项式f(n) = 2n + (1/24)n(n-1)(n-2)(n-3)(n-4)(n-5)。——译者注

2　见本章参考文献[7]，第79页。

当然，仅仅通过物理力量就能驱动机器完成原本属于思考领域的工作，这一构想本身就充满魔力。无论是当时还是现在，能思考的机器始终都令人们惊叹不已。

在他的朋友和支持者的不懈努力之下，巴贝奇最终赢得了胜利，并获得了资助承诺。这笔资助最初为1500英镑，后来逐渐追加至超过17 000英镑。巴贝奇自己也投入了相当一部分资金。

项目初期进展顺利，原型机陆续问世。在接下来的十年间，大量零件被制造，小型演示机型相继完成。然而由于设备规模庞大，加之巴贝奇易分心的性格特质和难以相处的个性——他敏感自负，睚眦必报，得罪了不少重要人物——在经历十年拖延、承包商纠纷、工程停滞和严重成本超支后，最终失去了政府资助。

他的早期支持者都对这一最终失败感到沮丧和尴尬。投入大量的公共资金却毫无成果。他也不可能受到支持了。

因此，巴贝奇的差分机从未被完整建造出来。[1]哦，部分零件确实被制造出来了。事实上，他用来招待客人的那台机器就是由原本为全尺寸机器准备的零件所制成的。

最终，在英国皇家天文学家George Bidell Airy的建议下，政府拒绝再提供任何资金。首相办公室的信函中写道：[2]

> “巴贝奇先生的项目开支无限膨胀，最终成功的概率微乎其微，支出规模庞大，完全无法预估，政府实无理由继续承担相关责任。”

至此，历经十年耗费17 000英镑后，这项工程宣告终止。

在我看来，项目失败的主要原因在于巴贝奇不断被更宏大的构想分散注意力，尽管他本人可能会否认这一点。对巴贝奇而言，完成项目远不如开启新计划来得有趣。

技术局限之争

有人认为，未能完成差分机的原因是19世纪早期的金属加工技术无法达到所需精度。然而，这一说法并没有真正的依据。现存的零件具有极高的精度，而且原型机至今仍能正常运转。

有充分的理由相信，只要巴贝奇具备意愿、专注力和资源来造出这台机器，机器就会按设计那样工作。不过，正如后文所述，让机器真正运转与仅仅完成建造完全是两回事。

1　他大幅改进的设计——差分机2——最终于20世纪80年代末和90年代初在伦敦科学博物馆建造。它在那里展出，并且按照巴贝奇的设计工作，尽管调试它是一个挑战。

2　见本章参考文献[7]，第176页。

2.8　分析机

下面是科幻作品《银河系漫游指南》中超级计算机“深思”的一段话：

> “……在我浩瀚的运算阵列中，我能穿越孕育着未来可能性的无尽数据流，预见到终将诞生这样的计算机——我连计算它的初始参数都不够格，但最终要由我来设计它。”

究竟是何种诱惑让巴贝奇偏离了差分机的建造之路？

试想一台火车头般庞大的机器：它不是差分机的6列20位存储，而是拥有1000列50位的存储阵列。机械总线如同数字血管，将数值泵送至运算器的双输入通道。这台机器能进行单精度(50位)或双精度(100位)的加减乘除运算，再将结果通过总线送回存储阵列。

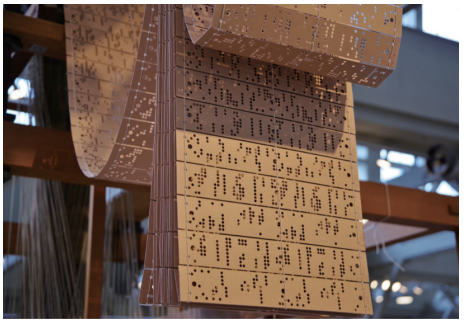


图 2-16

驱动这台庞然大物的是串联成链的打孔卡片——灵感源自提花织机的程序载体。这些指令卡指挥总线从特定存储列抓取数据，引导运算器执行操作，并将结果写回指定的位置。

它们能将常数加载进存储器，指挥打印机输出结果，驱动曲线绘图仪绘制轨迹，甚至触发简单的响铃动作。如图2-16所示。

最具革命性的是条件跳转指令卡：当运算发生溢出时，能指示读卡器向前或向后跳过指定数量的卡片。这几乎就是现代计算机的雏形。

除了两个重大区别：程序存储在卡片上而非存储器中，且机器完全由机械驱动，动力来源于——你猜对了——是蒸汽。

巴贝奇为提升运算效率殚精竭虑，最终设计出能在1秒内完成50位数加减的机械结构。乘法采用位移累加技术，除法采用位移递减技术，即便双精度运算也能在1分钟内完成[1]。

想象一下这台蒸汽朋克巨兽运转时的情景：卡片读取器的咔嗒刮擦声催动机体内部发出金属摩擦的嘶鸣、齿轮咬合的铿锵与蒸汽驱动的低吼。当数值从存储阵列出发，经总线奔涌至运算核心再折返时，你能亲眼目睹数值在流动。

想象一下，齿轮组往复位移、叠加运算，摩擦嘶鸣中吐出乘积，铿锵作响间迸发商与余数。程序持续运转，数值在存储阵列与运算核心间来回奔涌，时而触发打印机吐出结果，推动绘图笔尖划出轨迹，或是敲响清脆的铜铃。

1　见本章参考文献[7]，第111页。

看到这些，谁能不为之目眩神迷？

巴贝奇的脑海中已经清晰勾勒出这台机器的运作机制，深刻地理解了其中的意义。他宣称“算术的整个领域如今已尽在机械掌控之中”[1]，甚至推断它能编程下棋[2]。1832年他在“一位哲学家的生命历程”中写道：“任何技巧性游戏都可由自动机械完成。”

当然，这台机器从未真正建成。巴贝奇只制作了分析机的小型原型，验证了部分机械设计。他心知肚明，建造这头机械巨兽所需的资金、时间和精力是个无底洞。

在优化设计的过程中，他回头审视差分机的不足之处，进而设计出差分机2号——零件减少三分之二，速度提升三倍，容量翻倍有余[3]。即便在当时，摩尔定律的精神已隐约显现。

当然，差分机2号同样止步于图纸。直到20世纪末，伦敦科学博物馆的研究人员依据这些图纸终于将其建造完成了[4]。

符号化

再次反对这样的说法，即认为巴贝奇没有意识到他的机器可以操纵符号。首先，他认为可针对分析机进行编程来下棋，这表明他有能力将棋盘、棋子和走法符号化。

巴贝奇还意识到，通过适当的编程，机器或许能够进行符号代数运算。用他自己的话来说：[5]

> “今天，我第一次隐隐约约地意识到，或许能让机器进行代数运算——我指的是完全不依赖字母具体数值的符号推导。”

是的，巴贝奇是一位真正的程序员。他深刻理解符号与数字的关联性，明白任何能处理数字的机器必然能通过这种关联性处理符号。

2.9　阿达：洛芙莱斯伯爵夫人

乔治·戈登·拜伦(George Gordon Byron)的降生或许正逢黑暗时刻。他是一位才华横溢且多产的作家和诗人，英国最杰出的人物之一。他创作了诗体小说《唐璜》(*Don Juan*)和诗歌《希伯来旋律》(*Hebrew Melodies)*等传世佳作，10岁时继承罗奇代尔男爵爵位，成为拜伦勋爵。

然而拜伦并非正人君子。他脾气暴躁、风流成性，有许多非婚生子女，与玛丽·

1　见本章参考文献[7]，第91页。
2　见本章参考文献[7]，第179页。
3　戈登·摩尔，提出“摩尔定律”，该定律预测集成电路的密度每年将翻倍。
4　见本章参考文献[7]，第221页及后续章节。
5　见本章参考文献[7]，第169页。

雪莱的表妹[1]及他同父异母的妹妹奥古斯塔·玛丽·利有暧昧关系。

他曾经为了缓解债务压力而寻求合适的婚姻对象。安妮贝拉·米尔班克[2]是众多目标之一，她叔叔很有钱，而她是继承人。尽管她最初拒绝了拜伦，但最终还是屈服了，并于1815年1月与拜伦勋爵结婚，同年12月生下他们唯一的合法子女——奥古斯塔·阿达·拜伦(Augusta Ada Byron)。

拜伦对女儿的出生颇为失望；他原本期待的是一个“光宗耀祖的男孩”。无论出于侮辱还是纪念，最终他以情人兼同父异母妹妹的名字为她取名；但始终称她为阿达。

拜伦传记的一位作者[3]曾将拜伦与安妮贝拉的婚姻描述为史上最臭名昭著的婚姻之一。[4]拜伦的行为极其恶劣。他一直与同父异母的妹妹私通，并与许多女性保持不正当关系，其中包括一些知名女演员。

拜伦曾四次试图施暴安妮贝拉，以至于她不得不让人反锁家门。

拜伦酗酒成性，甚至最后试图将安妮贝拉逐出家门。安妮贝拉认为他疯了，只好带着五周大的阿达离开了。

这场丑闻轰动一时。伦敦社会为之哗然。1815年4月，拜伦逃往欧洲大陆，再也没有回来，再也没有见到他的女儿阿达。

由于拜伦的缘故，公众对阿达的兴趣与日俱增。阿达逐渐成为名人，与她那著名的浪荡父亲的关系始终是焦点。

安妮贝拉受过数学训练，就开始训练阿达，以此转移她对疯癫父亲的关注。阿达展现出过人的天赋，对数学的热爱甚至超越母亲；但她从未失去对父亲的兴趣，最终以父亲的名字为她的孩子命名，并要求将她的墓地设在父亲的旁边。

因为安妮贝拉缺乏关爱，阿达对父亲的思念愈发强烈。安妮贝拉提到阿达时，有时会用代词“它”。大多数情况下，她都是将阿达留给阿达的外祖母照顾，[5]外祖母十分疼爱阿达，一手将她抚养成人。

阿达自幼体弱多病，饱受头痛与视力模糊的困扰。青春期时，她曾因感染麻疹而卧床近一年。但她仍然坚持数学研究——或许是因为过于投入，17岁时，她竟试图与她的一位家庭教师私奔。[6]

在她的另一位家庭教师玛丽·萨默维尔的引荐之下，她结识了许多重要的数学界和科学界人物——包括查尔斯·巴贝奇。

在一次巴贝奇的晚宴上，阿达见到了差分机的原型机，随即对它的工作原理着了

1　玛丽当时正在瑞士日内瓦湖附近的拜伦家中访问，那是1816年——一个没有夏天的年份。全球因前一年坦博拉火山的爆发而陷入毁灭性的火山寒冷期。在那段雨天和夜晚，他们和一群精英朋友围坐在篝火旁，读着鬼故事。拜伦发起挑战，要他们所有人写一个鬼故事，这启发了玛丽写下*Frankenstein*(《弗兰肯斯坦》)。

2　Anne Isabella Milbanke；Annabella是她的昵称。

3　指Benita Eisler。

4　见本章参考文献[3]，第156页。

5　William Turner。阿达声称这段关系从未实现。

6　是的，就是你知道的那个德摩根：AB=(A+~B)。

迷。此后，她频繁拜访巴贝奇，观看他的机器，还讨论这些机器以及他更为宏大的计划。巴贝奇向她讲述了分析机的崇高目标。她被精妙的设计及蕴含的各种可能性所深深陶醉。

从那一刻起，阿达便与编程结下了不解之缘。

19岁时，阿达嫁给了威廉·金(William King)，他是奥克姆勋爵八世、洛芙莱斯伯爵一世。就这样，阿达成了洛芙莱斯伯爵夫人。尽管她饱受疾病困扰，肩负着抚养孩子和家庭生活的重担，但她依然坚守着对数学的执着追求，并深入研究巴贝奇的思想。在她的请求之下，巴贝奇推荐奥古斯都·德·摩根(Augustus De Morgan)[1]成为她的导师。

不久后，巴贝奇收到了数学家乔瓦尼·普拉纳(Giovanni Plana)的邀请，希望他在都灵的意大利科学家大会上介绍他的分析机概念。巴贝奇欣然接受，这将是他首次也是唯一一次在公众场合详细描述这一宏伟构想。

演讲大获成功，普拉纳承诺将发表会议报告。然而，巴贝奇却苦等了近两年才收到这份报告。

延迟可能是因为普拉纳繁杂的个人事务，也可能是因为他对这一项目的兴趣并不如他表现出的那般强烈。最终，普拉纳将任务委托给了31岁的工程师路易吉·梅纳布雷亚(Luigi Menabrea)，后者曾出席了会议。这篇法语报告最终于1842年在瑞士的一份期刊上发表。

查尔斯·惠特斯通(Charles Wheatstone)[2]是阿达和巴贝奇的共同朋友，他阅读了这篇报告，并建议阿达与他合作，将其翻译成英文，发表在《科学回忆录》(*Scientific Memoirs*)[3]上。阿达欣然同意，并凭借她对法语的熟练掌握和对分析机的深刻理解，在惠特斯通的指导下完成了翻译工作。这份翻译作为一份惊喜，呈献给了巴贝奇，凝聚了朋友们对他的支持和情谊。

巴贝奇对此感到非常高兴，但他认为阿达完全有能力就该主题撰写一篇原创论文，并建议她为翻译添加一些论文笔记，以进一步展现她的才华。

阿达对这个提议感到无比兴奋，她与巴贝奇展开了一段紧张而高效的合作，频繁拜访，互发信件，互传消息。事实上，她以近乎疯狂的热情投入工作，变得苛求、专横、卖弄风情且易怒。[4]随着工作的深入，她的热情也愈发高涨。

然而，事情在这里出现了意想不到的转折。阿达，这位洛芙莱斯伯爵夫人，性格中带有明显的狂躁倾向。她曾在给母亲的信中，以一种近乎偏执的方式描述自己，其中包含以下片段：[5]

1　是的，就是你知道的那个惠斯通：惠斯通电桥。

2　一本专门刊登科学外文论文的期刊。

3　见本章参考文献[7]，第161页。

4　见本章参考文献[7]，第158页。

5　见本章参考文献[7]，第161页。

> “我相信自己拥有一种极其独特的品质组合，这使我成为自然界隐藏的现实的杰出发现者。”
>
> “……由于我神经系统的一些特殊性，我对某些事物的感知无人能及……”
>
> “……我卓越的推理能力……”
>
> “……[我拥有]这样一种能力，不仅能将自己全部的精力和生命投入我所选择的任何事情中，而且能从各种看似不相关的来源中，调用庞大的资源来处理任何一个主题或想法。我能把来自宇宙各个角落的光芒汇聚到一个焦点上……”

关于这封信，她最后总结道：“尽管看起来疯狂，但这封信是我认为我写过的最合乎逻辑、冷静的作品；这是大量准确、实事求是的反思和研究的结果。”

考虑到这一点，再来看她在撰写论文笔记期间写给巴贝奇的信中的摘录——她有时会以“你的仙女夫人”签名：

> “我越是研究，就越发意识到自己对知识的如饥似渴，越发察觉到自己的天赋。”[1]

“我确信，我的父亲不曾是(也永远不可能成为)像我这样出色的分析学家(以及形而上学家)，即便他作为诗人也无法企及我在这些领域的成就。”

最终，阿达撰写了7篇论文笔记，从A到G，篇幅总和是原文的三倍。这些论文笔记发表于1843年，内容精彩绝伦，充满了热情与洞见。例如，她在论文笔记A中写道：

> “分析机与单纯的‘计算器’并非处于同一范畴。它拥有完全属于自己的独特地位；而且它所引发的思考从本质上来说极其有趣。它能让机械装置将通用符号组合在一起。”

正是由于这一点，以及她多次提到分析机能够表示符号(而非纯数值)的能力，阿达常被誉为“第一位程序员”。

2.10　第一位程序员?

仔细阅读阿达的论文笔记，不难发现她是一位程序员。她对这台机器有着深刻的理解，甚至可以说，她被这台机器深深吸引。她能够想象它的运作方式，并跟踪它的执行过程。如果她能亲手操作这台机器，她很可能会完全掌握它。

想象一下，你清楚地知道这台机器能做什么，然而同时明白，你将永远无法亲眼看到它实际运行的样子，甚至永远无法亲眼见到它。这必定是一种将喜悦与失望、宏

1　我也是。只不过我玩的与其说是太空旅行，不如说是太空战争。见https://github.com/unclebob / spacewar。

伟的憧憬与渺茫的希望交织，令人神伤的复杂心境。

尽管阿达，这位洛芙莱斯伯爵夫人能力出众，但她并不是第一位程序员。巴贝奇肯定在她之前就接触并研究了这方面的工作；而且他对这台机器的符号本质的理解并不亚于她。她所拥有的重要见解，巴贝奇同样具备。

没错，这些笔记非常出色。没错，她正式描述了分析机可以执行的程序。尽管她确实调试了其中一个程序，但这些程序并非由她编写——是巴贝奇编写的。

此外，很明显她并不是这些笔记的唯一作者。她与巴贝奇的合作如此紧密，不可能是她独自完成的。

不过，鉴于他们之间的这种合作，我们或许可以得出不同的结论：阿达或许不是第一位程序员，但阿达和巴贝奇几乎肯定是第一对结对编程的程序员。

天妒英才

9年后，经历长期身心折磨的阿达因宫颈癌去世，年仅36岁。遵照她的遗愿，遗体安葬在当年抛弃她的父亲的墓旁。

2.11　未竟之宏愿

最终，巴贝奇和阿达努力的成果喜忧参半。在他们所处的时代，这些殚精竭虑的努力，恰似流星划过天际，不见成果踪影。阿达此后再也没有发表过任何作品，巴贝奇也没有进一步尝试阐述他的想法。他们设想的宏伟愿景，在一个世纪里都无人问津。

人们很容易认为，这对命运多舛的结对程序员的思想是点燃信息时代的火花，认为一个多世纪后出现的那些先驱者是受到了他们的著作和设想的启发。但遗憾的是，事实并非完全如此。如果那些后来的先驱者提及他们的话，也只是事后顺便提一下，或者是跨越时间的鸿沟，向志同道合的人表达敬意而已。

巴贝奇在1851年发出了一条穿越时空的信息，可看作是对这种敬意的提前回应。这条信息的字里行间满溢着他如潮水般的痛苦与难以言说的失望，因为他知道自己的宏伟想法注定被同时代的人忽略：

> “未来的时代必将纠正现在的不公。必须明白，准备的日子越久远，就越领先于同时代人的努力。我有足够的底气去面对无知者的嘲笑、竞争者的嫉妒。”

正如将在后续章节中所看到的，后来的先驱者对巴贝奇和阿达表达了极大的敬意。甚至可以看到，霍华德·艾肯(Howard Aiken)和格蕾丝·霍珀(Grace Hopper)之间的合作在某种程度上重演了巴贝奇和阿达之间的关系，他们建造并编写了巴贝奇分析

机的机电模拟版本：哈佛Mark I。尽管如此，要说那些先驱者在任何重要方面受到了巴贝奇和阿达的影响或指引，就言过其实了。

巴贝奇不是一个有始有终的人。他启动了差分机项目，启动了分析机项目，还启动了差分机2号项目。他绘制了设计图，对各个部件进行了修修补补，甚至组装了部分零件。但没有一个项目是真正完成的。他的同代人抱怨说，他会急切地向他们展示一个想法，然后又会展示下一个精妙的想法，再下一个更精妙的想法，还总是用新想法作为借口，解释为什么之前的项目没有完成。

要是巴贝奇真的把第一个差分机项目完成了，谁知道会发生什么呢。我们现在知道那台机器本来是能够运转的。那次成功会不会带来其他更宏伟的机器呢？最终，他有没有可能以某种形式看到自己的分析机问世呢？

差分机2号的实现

要是巴贝奇完成了他的第一台差分机，他肯定会发现自己在机器的组装过程中并没有提供任何调试和测试的方法。

在20世纪80年代末到90年代初，伦敦科学博物馆的工作人员成功制造出了一台可以运行的差分机2号。这是一台很精巧的机器，是由黄铜和钢材构成的闪闪发光的金属框架。当转动曲柄时，数字列会在行列中旋转摆动，那些精巧的进位轴会将计算结果添加到总数上。观看它的运行真是一种奇妙的体验。

但是，参与制造这台机器的人所讲述的却是一段充满挫折的经历。

在组装完成后，只要把手柄转动一两度以上，机器就会卡住。每个部件都需要根据机器的运行时间与其他部件完美对准，而这一点是巴贝奇没有考虑到的。甚至不清楚他是否预见到了这个问题。

这台机器的逐步调试、校准和修复花了11个月的时间。有时要轻轻转动一下曲柄，直到它卡住，然后用螺丝刀或钳子在机器内部捣鼓，看看这里有没有松动或者那里是不是太紧了。有时还得故意弄坏一些部件，以便找到产生阻力的位置。有时甚至还需要进行一些微小的重新设计。

伦敦科学博物馆的计算机馆馆长多伦·斯瓦德(Doron Swade)是从头到尾推动该项目完成的人，他对巴贝奇的设计给出如下评价：

> “巴贝奇没有为调试提供任何手段。没有简单的方法可以将差分机的一个部分与其他部分隔离，以便定位卡住的来源。整个机器是一个整体的‘硬连线’单元。驱动杆和连接件被永久固定或铆接到位，一旦组装好就很难拆卸。”

但最终，在所有4000个部件都对准并且所有小改进都安装好后，机器完美地运行了。

有一些关于这台机器运行的真实视频，非常令人着迷。想看的话，可以在YouTube上搜索(在撰写本书时，可查看Computer History Museum频道中的The Babbage Difference Engine #2 at CHM。发布于2016年2月17日)。

2.12　结论

巴贝奇是一位发明家、一位喜欢捣鼓的人、一位有远见的人，而且……还是一位程序员。遗憾的是，和我们许多人一样，追求完美阻碍了他把事情完成。和我们许多人一样，他对自己的设计过于自信，很少甚至根本没有考虑过渐进式改进的方法。和我们许多人一样，他很容易被一个想法所吸引，并且乐于将这个想法思考到八成，但当涉及最后需要付出八成努力才能完成的两成工作时，他就无法保持那份热情了。

爱迪生曾说过，发明是1%的灵感加上99%的汗水。巴贝奇在1%的灵感方面非常出色，但他却始终无法跨越另外的99%。他喜欢思考问题，甚至喜欢制作一些东西。他喜欢谈论这些东西，还喜欢展示他制作出来的部件。但当真正要完成一件事情，需要付出艰苦的劳动时，他就会转向下一项重大发明。

参考文献

[1] Adam, Douglas. The Hitchhiker's Guide to the Galaxy[M]. London: Pan Books, 1979. (中文版：亚当斯，道格拉斯. 银河系漫游指南[M]. 徐百柯，译. 成都：四川科学技术出版社，2005).

[2] Beyer, Kurt W. Grace Hopper and the Invention of the Information Age[M]. Massachusetts: MIT Press, 2009. (中文版：Beyer, Kurt W. 格雷斯·霍珀与信息时代的发明[M]. 包艳丽，刘珍，陈菲，译. 北京：机械工业出版社，2010).

[3] Eisler, Benita. Byron: Child of Passion, Fool of Fame[M]. New York: Knopf, 1999.

[4] Jollymore, Amy. Ada Lovelace, an Indirect and Reciprocal Influence[EB/OL]. (2013-10-14) [2025-03-20]. https://www.oreilly.com/content/ada-lovelace-an-indirect-and-reciprocal-influence.

[5] Moseley, Maboth. Irascible Genius: The Life of Charles Babbage[M]. London: Hutchinson, 1964.

[6] Scoble, Robert. A Demo of Charles Babbage's Difference Engine[EB/OL]. (2010-06-17) [2025-03-20]. https://www.youtube.com/watch?v=BlbQsKpq3Ak.

[7] Swade, Doron. The Difference Engine[M]. London: Penguin Books, 2000.

[8] University of St Andrews. The early history of computing | Professor Ursula Martin (Lecture 1)[EB/OL]. (2020-02-26) [2025-03-20]. https://www.youtube.com/

watch?v=moYxYEfxO7g.

[9] Wikipedia. Sketch of the Analytical Engine Invented by Charles Babbage, L. F. Menabrea, translated and annotated by Ada Augusta, the Countess of Lovelace[EB/OL]. [2025-03-20]. https://en.wikisource.org/wiki/Scientific_Memoirs/3/Sketch_of_the_Analytical_Engine_invented_by_Charles_Babbage,_Esq.

[10] Wikipedia. Babbage [EB/OL]. [2025-03-20]. https://en.wikipedia.org/wiki/Charles_Babbage.

[11] Wikipedia. Difference Engine [EB/OL]. [2025-03-20]. https://en.wikipedia.org/wiki/Difference_Engine.

[12] Wikipedia. Analytical Engine [EB/OL]. [2025-03-20]. https://en.wikipedia.org/wiki/Analytical_Engine.

[13] Wikipedia. Ada Lovelace [EB/OL]. [2025-03-20]. https://en.wikipedia.org/wiki/Ada_Lovelace.

[14] Wikipedia. Lord Byron [EB/OL]. [2025-03-20]. https://en.wikipedia.org/wiki/Lord_Byron.