TURING

Apress®

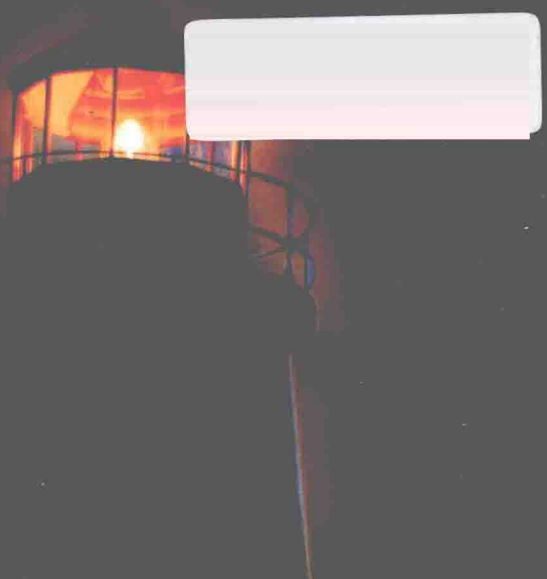
CODERS AT WORK

编程人生

15位软件先驱访谈录

【美】Peter Seibel○著　图灵社区○译

（下卷）



人民邮电出版社
POSTS & TELECOM PRESS

TURING

CODERS AT WORK

编程人生

15位软件先驱访谈录

【美】Peter Seibel○著　图灵社区○译

（下卷）

人民邮电出版社
北　京

图书在版编目（CIP）数据

编程人生 : 15位软件先驱访谈录. 下卷 / (美) 塞贝尔 (Seibel,P.) 著 ; 图灵社区译. -- 北京 : 人民邮电出版社, 2015.1
书名原文: Coders at work
ISBN 978-7-115-35608-6

Ⅰ. ①编… Ⅱ. ①塞… ②图… Ⅲ. ①计算机科学－科学家－访问记－世界 Ⅳ. ①K816.16

中国版本图书馆CIP数据核字(2014)第100286号

内 容 提 要

这是一本访谈笔录，记录了当今最具个人魅力的15位软件先驱的编程生涯。包括Donald Knuth、Jamie Zawinski、Joshua Bloch、Ken Thompson等在内的业界传奇人物，为我们讲述了他们是怎么学习编程的，在编程过程中发现了什么以及他们对未来的看法，并对诸如应该如何设计软件等长久以来一直困扰很多程序员的问题谈了自己的观点。本书是下卷，包含7篇访谈。

本书适合所有程序员，也适合所有对计算机行业、对软件开发感兴趣的人。

◆ 著　[美] Peter Seibel
译　图灵社区
责任编辑　朱　巍

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京铭成印刷有限公司印刷

◆ 开本：720×960　1/16
印张：15.25
字数：318千字　2015年1月第1版
印数：1－3 000册　2015年1月北京第1次印刷
著作权合同登记号　图字：01-2013-4950号

定价：39.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第0021号

前　言

李清　译

抛开 Ada Lovelace（被誉为世界上第一位程序员，著名诗人拜伦之女，她为 Charles Babbage 未完成的分析机设计了算法）的工作不说，人类在计算机编程领域奋斗的时间还不及一个人的寿命长，从 1941 年 Konrad Zuse 完成 Z3 电子机械计算机（首个可运行的通用计算机）算起，只有短短 68 年。曾有 6 位女性（Kay Antonelli、Jean Bartik、Betty Holberton、Marlyn Meltzer、Frances Spence 和 Ruth Teitelbaum）在美军的计算机部队工作，她们手工计算弹道数据表，后来被调去做 ENIAC（首台通用电子计算机）最早的程序员。若从这时候算起，人类的编程历史则只有 64 年。在婴儿潮早期出生的人们以及他们的父母，很多至今仍然健在，他们出生时世界上还不存在计算机程序员。

当然，这些已经是历史了。现在世界上有很多程序员。劳动统计局在 2008 年对美国 125 万人进行了统计，大约每 106 个工作者当中就有 1 个是计算机程序员或软件工程师。这还没算美国之外的职业程序员、数不清的学生和业余编程爱好者，还有很多人从事其他正式工作，但却花费了一部分或很多时间来试图驯服计算机。虽然有数以百万计的人写过代码，虽然在编程出现后人们写过的代码没有数万亿行也有数十亿行，我们仍然不断地在这一领域进行创造。人们仍然在争论编程到底是数学还是工程，是工艺、艺术还是科学。我们仍然在（经常是带有强烈情绪地）争论编程的最佳方式，因特网上有无数的博客文章和论坛帖子来讨论这些问题。书店也摆满了各种论述新编程语言、新编程方法、新编程思想的书。

本书按照文学期刊《巴黎评论》（*The Paris Review*）的传统，采取了一种不同的方法来讲述什么是编程。这家期刊曾派了两位教授去采访小说家 E. M. Forster，这次采访和随后的一系列问答式的采访后来辑录为 *Writers at Work* 一书。

我采访了 15 位成就斐然、经验丰富的程序员，其中有些人是系统黑客，如 Ken Thompson（Unix 的发明者）和 Bernie Cosell（ARPANET 早期实现者之一）；有些人既有强大学术实力本身，又是著名黑客，如 Donald Knuth、Guy Steele 和 Simon Peyton Jones；有些人是业界的研究员，如 IBM 的 Fran Allen，爱立信的 Joe Armstrong，Google 的 Peter Norvig，以及曾在施乐帕克研究中心工作过的 Dan Ingalls 和 L Peter Deutsch；有些人是 Netscape 的早期实现者，如 Jamie Zawinski 和 Brendan Eich；有些人参与设计和实现了现在的万维网，如刚才提到的 Eich，以及 Douglas Crockford 和 Joshua Bloch；还有 Live Journal 的发明人 Brad Fitzpatrick（在伴随 Web 成长起来的一代程序员当中，他是一个当之无愧的典型）。

在采访中，我问他们有关编程的问题，问他们是怎么学习编程的，在编程过程中发现了什么，以及他们对未来的看法。而且我很用心地请他们多谈谈长久以来程序员一直在苦苦思索的那些问题：我们应该如何设计软件？编程语言在帮助我们提高生产力和避免错误方面扮演了什么角色？有什么办法可以更容易地查出难以发现的 bug？

这些问题远远还没有解决，所以我的采访对象持有非常不同的观点也就不那么奇怪了。Jamie Zawinski 和 Dan Ingalls 强调尽早让代码跑起来的重要性，而 Joshua Bloch 则描述了在实现之前，他如何设计 API 并测试它们能否支持要写的代码。Donald Knuth 讲述了他在编写排版软件 TeX 的时候，怎样在敲代码之前先用铅笔在纸上完整地实现整个系统。Fran Allen 大力批判近几十年来人们躺在 C 语言的脚下对计算机科学的兴趣越来越低，Bernie Cosell 称之为“现代计算机最严重的安全问题”，Ken Thompson 却认为安全问题是程序员而不是编程语言造成的，Donald Knuth 也说 C 的指针是他所看到过的“最令人赞叹的记法改进之一”。一些受访者对“形式化证明可能有助于改善软件质量”这一观点嗤之以鼻，而 Guy Steele 则漂亮地展示了这种做法的优点和限制。

然而，仍然有一些主题是大家都认同的。几乎所有人都强调保持代码可读性是很重要的。大部分受访者都认为最难查找的 bug 出现在并发代码中。没有人认为编程问题已经完全解决了，他们大多数人仍然在寻找编写软件的更好办法，比如怎样自动分析代码，如何让程序员更好地协作，或者寻找（或设计）更好的编程语言。同时几乎所有人都认为多核 CPU 的大量采用将会给软件开发带来重大改变。

这些谈话发生在计算机发展史的一个特定时刻。因此，本书中讨论的一些话题在当前是紧迫问题，今后将不再是问题而变成了历史。但即使是像编程这种新兴领域，历史也能为我们提供很多教训。除此之外，我觉得我的受访者们可能有一些共识，包括什么是编程，如何更好地编程，等等，不仅现在的程序员会从中受益，未来几代程序员也将从中受益。

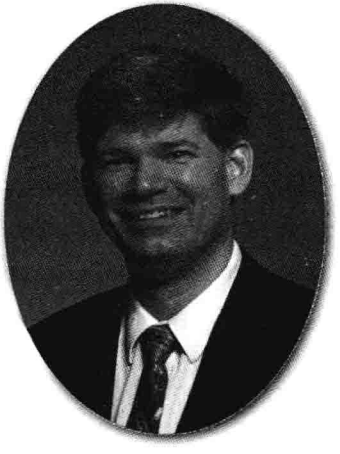
最后提一下本书的书名：*Coders at Work*。这个书名与前面提到的《巴黎评论》出的 *Writers at Work* 系列以及 Apress 的 *Founders at Work*[①]（该书讲述如何创办技术公司，而本书讨论计算机编程）相呼应。我意识到编程涵盖的范围太广了，而“编码”（coding）则可以特指其中一个很窄的部分。我个人从不认为一个糟糕的程序员会是一个优秀的编码者，也不相信好的程序员会不是出色的设计者、沟通者和思考者。毋庸置疑，这些受访者都是优秀的编码者、程序员、设计者和思考者，而且还不仅仅如此。我相信接下来你在阅读他们的谈话内容时一定能够体会到这一点。希望你能喜欢这本书！

① 中文版《创业者》已由机械工业出版社出版。——编者注

目　录

第 1 篇

Guy Steele



叶淮光　译

Guy Steele是个真正的程序语言多面手。当我问他曾经认真地使用过哪些语言的时候，他列出了下面这一长串：COBOL、Fortran、IBM 1130汇编、PDP-10机器语言、APL、C、C++、Bliss、GNAL、Common Lisp、Scheme、Maclisp、S-1 Lisp、*Lisp、C*、Java、JavaScript、Tcl、Haskell 、FOCAL、BASIC、TECO以及TeX。他还说："这些是其中主要的语言。"

他参与了现存两种主要的通用Lisp方言——Common Lisp和Scheme的创建。他也在Common Lisp、Fortran、C、ECMAScript和Scheme的标准化组织中工作，Bill Joy还邀请他帮助制定Java的官方语言规范。现在他正致力于Fortress的设计，这是一种用于高性能科学计算的新语言。

Steele在哈佛大学获得文学学士学位，在MIT获得科学硕士学位和博士学位。在

MIT期间，他和Gerald Sussman合著了一系列著名的论文，现在被称作“The Lambda Papers”，其中包括了Scheme程序语言的初始定义。他还曾经是一名黑客文化编年史的作者，JargonFile的最初编撰者之一，以及《黑客词典》一书的编者（该书后来由Eric S. Raymond更新修订为《新黑客词典》）。他还在Emacs的诞生中扮演了重要角色，同时也是最早移植Donald Knuth的程序TeX的程序员之一。

Steele是ACM会员和美国艺术与科学院院士，也是美国国家工程院院士。1988年他获得ACM Grace Murray Hopper奖，2005年获得Dr. Dobb's程序设计杰出奖。

在这篇访谈中，他讨论了软件设计，以及写作和编程的关系，他还给出了我所听过的关于正确性的形式证明的价值及其局限的最佳解释。

Seibel：你是怎样接触编程的？

Steele：嗯，当我还是个小学生时，我就已经深深迷恋科学和数学了，我读了很多这方面的书，比如Irving Adler的 Magic House of Numbers，它是我的最爱。我也喜欢儿童科幻小说，比如Danng Dunn 系列，等等。总的来说，我对科学和数学有着广泛的兴趣。所有我能找到的关于科学和数学的东西，我都读了，同时我也读到了一点关于即将到来的新奇的计算机的介绍。

Seibel：这些都是什么时候的事情？

Steele：1960 年 ~ 1966 年，在我读小学期间。不过我想转折点发生在我开始就读于波士顿拉丁学校[①]——大约相当于 9 年级。一个朋友问我：“你听说了那些放在地下室的新计算机吗？”我以为那不过是个恶搞[②]，之前有人说第四层有个游泳池，而学校教学楼一共只有三层。不过他说：“不，这是真的，它就摆在那里呢。”

---

① 拉丁学校，以拉丁语和希腊语为主科的文科中学，作为大学的预料。13 世纪始于欧洲。——编者注

② 原文为 story，双关语，既可以解释为“恶搞”、“故事”，也可解释为“楼层”。这一句颇有递归的意思，因为 Gug Steele 的背景是 Lisp。

后来证实是文森特·利尔森先生[①]赠送给波士顿拉丁学校的IBM 1130小型计算机。他是校友，并且显然是一个慷慨的人。我朋友向我展示了一段5行的Fortran程序，我马上就着迷了。

我找到我们的数学老师，问他是不是有些什么书可以让我来学习。他给了我几本，并且以为至少可以让我忙活一个月了，不过实际上，仅仅一个周末我就搞定了它们。在1968年感恩节的那个周末——是的，那是个长周末[②]——我自学了Fortran。从此我就被编程套牢了。

拉丁学校的朋友和我因为IBM 1130而迷上了IBM，我们每隔个月就去一趟市中心的IBM办公室，和那里的人们交谈，偶尔用我们仅有的一点钱买一些刊物。

市中心也有一个书店，那里卖一些新奇的语言（比如PL/I）的书，我们偶尔也在那里买几本。就这样，在拉丁学校我们熟悉了IBM的设备。我们只有1130，很眼馋360系统。我们在书本上见过它，不过没有机会接触真正的。

1969年的春天，在高中学习项目上，我有机会进入到了MIT。这非常棒——星期六早上去，那些大学生会教你一些很酷的玩意。我学习了群论和计算机程序设计，其他的我已经忘记了。我就深陷其中，并且对MIT的氛围熟悉了起来。通过高中学习项目，我们有机会接触到IBM 1130和DEC PDP-10[③]。这样我们也开始了解DEC公司产品线了。

我们这些高中生很快熟悉了中央广场的DEC公司办公室，那里本来是接待MIT的学生的。他们不介意高中生进办公室来要一些参考手册，这可真是太好了。我还是拉丁学校的三年级还是四年级的学生的时候，就和一个朋友一起向DEC公司提交了一份在PDP-8上实现APL语言的计划书。他们认真评估了这份计划书。大约一个星期后，他们告诉我说："这样的，我们不赞成这个点子，但是很感谢你做的计划书。"

Seibel：*你编写的第一个有趣的程序是什么？*

Steele：嗯，我首先学习了 Fortran 语言，不过在我开始学习 IBM 1130 汇编语言之后，事情才变得真正有趣。我能想起来的最早的有趣的程序是一段能产生上下文关键字索引的东西。IBM 为他们的用户手册提供一个被称作是快速索引的东西：给定一个关键

---

① IBM 前 CEO。

② 长周末指赶上节日的周末，可以多休一两天。——编者注

③ DEC 公司生产的小型机系列的代号。PDP 是 Programmed Data Processor（程序数据处理器）的首字母缩写。——编者注

字，你可以从一个按字母排序的索引中查找，关键字的前后是这个关键字的上下文的一些单词。

Seibel：关键字出现处的上下文？

Steele：在原始文档中关键字出现处的上下文。每一栏中间是按照字母排序的关键字，左右两栏是大量的上下文。我想我在 1130 上面解决了这个问题。考虑到 130 只有 4k 字的内存，很明显我得把记录保存在磁盘上。因此我学到了很有效率的out-of-core排序算法[①]。这段程序有趣的地方不仅在于产生了上下文关键字的索引，还实现了多路 out-of-core 合并排序。它相当有效。不过内存中的排序用的是冒泡算法，因为我还没有那么老到。在内存中我本应该也使用合并排序，不过那时我并没有意识到。

Seibel：从你意识到地下室真的有一台计算机到你编写这段程序之间，过了多久呢？一个月？还是一个星期？

Steele：应该是在最初的两年当中，我不记得是不是在第一年。1968 年秋天我学习了 Fortran 语言。APL 语言是我学的第三种语言，所以我应该是在圣诞节期间或者刚过完圣诞节时学习了汇编语言。我清楚记得 1969 年的春天我学习了 APL，因为那时春季计算机联交会在波士顿举行。

IBM的展位上展示了所有的IBM产品，特别是APL 360，而我就在他们展位附近处闲逛。交易会结束后他们准备扔掉一大堆电动打字机终端的演示纸。就在那时我走上前去问："您准备扔掉这些纸吗？"那位女士困惑地看了看我，说："那，给你吧。"好像她给我的是一个圣诞节大礼包。不过对我来说这的确算是圣诞大礼包。

Seibel：那些纸上是什么呢？

Steele：这些从电动打字机终端上出来的、有着扇形褶皱的纸，是他们过去两天中展示 APL 用的。那上面刚好有他们随意键入的一些程序小例子。从这些程序示例还有交易会上那些小册子中，我自学了 APL。

Seibel：你在MIT很自在，但最终还是去了哈佛读书，而同时又在MIT打工，这是怎么回事呢？

---

① 也就是基于外存的排序算法。

Steele：我申请大学的时候，申请了三所学校，MIT、哈佛还有普林斯顿。我最想去的是 MIT。三所学校同时都录取了我。波士顿拉丁学校的校长 Wilfred L. O'Leary 是个老派的学者。老先生人非常好，打电话给我父母说："你们知道令郎拿着哈佛的通知书实际上却考虑去 MIT 吗？"他就这样向我父母施压，我父母转而对我施压，最终我决定去哈佛了。

我父母继续找我的麻烦，让我去打一份夏季工，而不是在家待着——你知道，做父母的都会这样。我很清楚自己的兴趣是计算机，我可不想去快餐店摆弄汉堡包。我面试了打孔工的工作，并且自以为是完全能够胜任的。但是没有人愿意雇用我，部分原因是我还不满18岁，可找到后才明白。他们听了我的叙述后说："不要打电话给我们，我们会打给你的。"然后就杳无音讯了。

大约7月初我听说MIT的Bill Martin正在寻找Lisp程序员。我想："啊哈，机会来了，我了解Lisp啊。"我过去经常出没于MIT的时候，从AI实验室搞到了一些Lisp文档的副本，我也曾偷偷溜进实验室摆弄过计算机。那些日子里大门是敞开的，反越战抗议发生后门才被锁上。我在高中四年级时在IBM 1130计算机上实现过我自己的Lisp程序。

于是，我这个不知道哪里冒出来的小瘦猴儿，跑到Bill Martin的办公室，从门口探进头说："我听说你在招Lisp程序员。"他并没有嘲笑我，只是打量了我一下，然后说："你得先做做我出的Lisp考题。""没问题，现在考怎么样？"我就坐了下来，花了两个小时来答题。完成后我把试卷递给他，他用了十分钟浏览了一遍，然后对我说："你被录取了。"

Seibel：*是不是你在高中学习项目当中学习过Lisp呢?*

Steele：一点点，更多的是 Fortran 和其他的语言。

Seibel：*在你起步时有没有遇到对你很重要的导师呢?*

Steele：在拉丁学校期间我的数学老师对我的适当鼓励刺激很重要。9 年级的 Ralph Wellings，就是在那个感恩节周末借我书的那位老师，和我做了一个交易。他说："我注意到你在所有数学测验中都得到了 100 分。我可以让你在每周的前 4 节数学课都呆在计算机室，不过在周五数学课的测试上你必须得到 100 分，否则，交易就自动终止。"看，这就是激励。在那年余下的时间中我变成了测试高手——我特别刻苦地学习数学，因为这能让我接触到计算机。更好的是，第二年我的数学老师没有与我做同样的交易，

这正好，因为我对那一年的数学了解不多。他们做出了恰当的评估。我的老师都是非常好的老师，我要学什么他们总是为我大行方便。

Seibel：*在那之后，随着你更深入地学习计算机，有没有特别的人在这领域帮助你呢？*

Steele：有，当然就是雇用我的 Bill Martin。还有Joel Moses，他领导着Macsyma项目，我受雇于 MIT 期间就在这个项目组里。

Seibel：*在整个大学期间，你一直在做这个项目吗？*

Steele：是的，我在哈佛读书的时候就一直是 MIT 的一名雇员。在暑假时是一份全职工作，开学后它就变成了一份下午的兼职工作。我尽可能地把哈佛的课程安排到早上，这样我就可以搭乘地铁去 MIT，用两三个小时来编程，然后再回家。

Seibel：*一直在用Lisp做Macsyma项目吗？*

Steele：是的。具体说就是当 Maclisp 解释器的维护人员。JonL White原本同时负责解释器和编译器的工作。他后来成为了一位相当厉害的编译器大师，而我则负责解释器，这个分工不错。就这样，JonL White 成了我的导师。Macsyma 项目组里所有的人都很关照我。我也得以结识一些 AI 实验室的人，所以当我申请读 MIT 的研究生的时候，很容易就被录取了，因为他们已经了解我，并且知道我在干什么。

Seibel：*你得到了计算机科学的学士学位？*

Steele：是的，我本来打算主修纯数学，并且都安排好了我的课程。后来发现我对什么无穷维巴拿赫空间完全没有感觉，简直要害死我了。幸好，我已经在业余时间学习了足够多的计算机课程，这让我可以在转换主修专业的时候很主动。确切地说，我转去修应用数学专业，而计算机科学是应用数学的一个分支，在哈佛应用数学又属于工程学的一部分。

Seibel：*在哈佛你使用哪种机器？*

Steele：DEC PDP-10。在学校里有一台 PDP-10，不过我想它主要是给研究生用的。本科生只能通过电传打字机终端接入商业系统，那是哈佛租来的。

Seibel：如果有可能让你重新学习编程，你会怎样重新开始？有什么事情你希望更早一点完成的吗？

Steele：并不是一开始在我的脑海里就有特定的目标。我对我选择的这条路也不后悔。回首往事，我想我是一个幸运儿，受惠于一系列有趣的巧合，或者说，恩赐。

现在我意识到，实际上同时在MIT和哈佛的经历是很不寻常的。我可以跑来跑去，然后说："这条河[①]那一边的教授是这样说的。"而这一边的教授就会说："哦，别信他，你应该这么想。"这很快让我的视野更开阔。

作为高中生就能进入MIT是另一个相当不寻常的经历。我15岁时就可以摆弄那些价值数百万美元的机器，在那时一百万可真是一笔相当大的钱。所以，我当然没有抱怨，没有后悔，也不会有任何得陇望蜀的想法。我本性也是个随遇而安的，既来之，则安之。

Seibel：与那时相比，对于编程的思维方式，有哪些大的改变？除了认识到冒泡排序不是最好的排序算法之外。

Steele：我想对我来说，如今最大的变化是认识到你不大可能明白运行在你计算机上的所有东西。有些事情绝对超出了你的控制，因为不可能了解所有软件的一切细节。而在上世纪 70 年代计算机仅仅有 4k 字的内存，你完全可以做一个内存转储，然后一个字一个字地去检查是不是你期望的。阅读操作系统的源码，了解它是如何运行的自然也正常。我也确实这样干过——我研究过磁盘管理程序和卡片阅读机程序，然后实现了我自己的版本。我觉得我自己了解整个 IBM 1130 是如何运作的，或者至少可以说我了解我自己想了解的所有事情。不过现在你再也不能这样干了。

Seibel：在你学习编程的时候，有没有什么书对你特别重要？

Steele：在 70 年代的是当然是 Knuth 的 TAOCP《计算机程序设计艺术》。

Seibel：你从头到尾地读过吗？

Steele：差不多每页都读过，差不多。我做了几乎所有我能做的习题。一些被称作是高等数学之类的东西我不太明白，我做了些注释或跳过了那些我不懂的。不过头两卷

① 哈佛大学与 MIT 只有一河之隔，即查尔斯河。——编者注

和第三卷的大部分我都很认真地读过。还有 Aho、Hopcroft 和 Ullman 编著的那本算法书[①]——我想我是从这里面真正学到如何排序的。我得从我的书库里查找一下，看看能不能还记得有什么别的书。我是个收集狂人——这类书我都有。不过这两本是我首先想到的。还有 Lisp 的书。Berkeley 和 Bobrow 编辑的III Lisp[②]：是主题各异的论文集，不过我从中学到了很多有意思的东西。然后我开始读《SIGPLAN 公报》和《ACM 通讯》。那些日子里的 ACM 通讯可是有很多真正的技术内容，非常值得一读。

我要提及两件事。第一件，当我在拉丁学校开始对科学感兴趣的时候，我决定从事计算机科学相关的事情。有一天有个学督问我："你考虑过成为ACM的学生会员吗？"我不知道他的名字。不过我从那时起就非常感谢他，这给了我很大的鼓励。

而我上哈佛以后，每当早晨有点空闲时间，我就会去Lamont图书馆做两件事：按照我的方式，从后往前阅读《科学美国人》，从前往后阅读《ACM通讯》。对《科学美国人》，我特别注意Martin Gardner[③]的所有数学游戏专栏。对《ACM通讯》则阅读所有我感兴趣的文章。在1972年这本期刊还只有15年的历史，所以不难把它们全部都过一遍。

Seibel：阅读所有文章在那时比在今天要容易得多，一个人还是希望了解这整个领域的。

Steele：是的，你可以期望了解整个领域。有很多只有一页长短的文章，让你知道："这儿有一项新的散列技术。"我读了很多这类的文章。

Seibel：旧的文章我个人常常觉得不太容易理解，因为它们和一些旧硬件或者旧语言联系得比较紧密。

Steele：是这样的，需要是创新之母——一个想法的出现是因为在一个特定的环境下需要它。过了一段时间，大众认识到这个想法很重要。然后你需要摆脱环境的局限，核心思想本身就凸现出来，这样就可以流行好几年了。"这个精妙的技巧可以按位逆转字。"他们给出了 7090 汇编语言的一些东西。有些很有趣的数学思想在里头，不过他们还不能从中抽象出来。

---

① 是指 *The Desing and Analysis of Computer Alogorithms*（《算法分析与设计》）一书。——编者注

② III 指 Intformation International，Inc.。——编者注

③ 马丁·加德纳，《啊哈，灵机一动》的作者。

Seibel：我猜那是Knuth的工作，对不？

Steele：Knuth 和像他一样的人，当然了。

Seibel：的确很多人从学校里开始，在指导下学习计算机科学知识。但是还有很多程序员是没有正规学历背景的，只是边干边学。对这个你有什么建议吗？你怎么开始阅读那些技术论文，如何抓住要点并理解它呢？应该从ACM最初读起，一直读到现在吗？

Steele：嗯，首先，我得说通读《ACM 通讯》并不是我刻意博览群书成为一名伟大的计算机科学家的计划。我阅读是因为我有兴趣，有内在的动力去学习那些资料。所以我觉得这里有两个因素：第一是有内在的动力，想要阅读它们，因为你有兴趣或者说你觉得能提高你的技能。

另一个问题是你怎样才能发现好的东西？当然，对“好”的认识也是三十年河东，三十年河西。今年你觉得是真正好的十年后说不定过时了。我觉得你可以请教一个曾经仔细考虑过这个问题的导师，问他觉得什么才是好的东西。对我而言就是Knuth，就是Aho，Hopcroft和Ullman。还有Gerald Weinberg的《程序设计心理学》，那本书今天还是非常值得读一读的。Fred Brook的《人月神话》也给我一些启示。

那时候我流连于MIT书店的计算机科学书架，下决心每个月到那里去一次，去翻翻那些书架。今天你再去一个书店，它的计算机书架规模可能是那时的10倍了，不过其中大部分图书都是如何使用C或者Java的。但还是会有一部分理论背景、算法这类书。

Seibel：另一种阅读——我知道你认为很重要的——是代码阅读。你是怎么样以你的方式切入不是你编写的一大段代码的呢？

Steele：如果那个软件我知道如何使用，但不了解内部的工作机制，我通常会选择一个特定的命令或者交互行为然后追踪下去。

Seibel：执行路径吗？

Steele：是的。如果我要开始阅读 Emacs 源代码的话，我会说：“让我们看看‘向前移动一个字符’的那部分代码吧。”即使我不能完全理解，我至少会知道它使用的一些数据结构以及缓冲区是怎么表示的。如果我足够幸运，我能找到缓冲区增加一个的地方。一旦我理解了之后，我接下来会尝试“后退一个字符”、“删除一行”。通过我的

方式就了解越来越多的使用方法或者交互，直到我觉得我能够按照这种方式追踪代码的其他更重要的部分。

Seibel：“追踪”是指查看源代码在心里执行它呢，还是要在调试器中启动它，然后单步执行进去呢？

Steele：两种方式我都会做，我会用单步调试器对付那些 70 年代或者 80 年代的那种小一点的程序。今天的问题是从启动程序到它真正可以做点什么，这中间可以说是一段很长的初始化过程。所以更好的办法是找到主命令循环或者中央控制子程序，从那里开始追踪。

Seibel：当你找到了那些以后，你是设置一个断点然后单步跟进去呢，还是仅仅在脑海中想像它们的执行过程？

Steele：我更愿意做桌面检查——就是阅读代码想象它会做什么。如果我确实需要理解整段代码，我会坐下来试图按照我的方式来通读代码。不过你不能一上来就这样做，应该先在脑子中有了事情的组织框架。现在，如果你足够幸运，程序员会留下一些文档或者规则的命名，或者合理组织文件中的顺序，方便你快速阅读他们的代码。

Seibel：什么是合理组织文件中的顺序？

Steele：很好的问题。使我想起了诸如 Pascal 这样的程序语言的一个问题，Pascal 是为只过一遍的编译器设计的，源文件中的过程倾向按照自底向上的方式组织，因为在使用过程之前你必须定义过它们。也就是说，阅读 Pascal 程序最好的方法实际上是从后面读起，因为这样你就会看到程序自顶向下的结构。现在（编译器）形式如此多样，你也就不能指望什么了，除非程序员有一颗很强的责任心，将一切事情按照帮助你理解的目的安排得井井有条。不过，第三点，我们现在也有很棒的 IDE 来帮助你查看交叉引用，也许程序的线性组织也不再是那么重要了。

第四点，我个人非常不喜欢IDE的一个原因是，你看完了所有内容后，还是很难理解。在网状结构迷宫里乱蹿，很难知道所有地方都走到了。但如果你得到的是线性顺序，就会确保所有事情都会梳理到。

Seibel：那么在你写代码的这些日子里，你是不是尽量按照自顶向下来组织代码呢？高层函数出现在它们依赖的低层函数之前？