

PEARSON

编程珠玑

续
修订版

[美] Jon Bentley◎著 钱丽艳 刘田 等◎译



More
Programming Pearls



中国工信出版集团



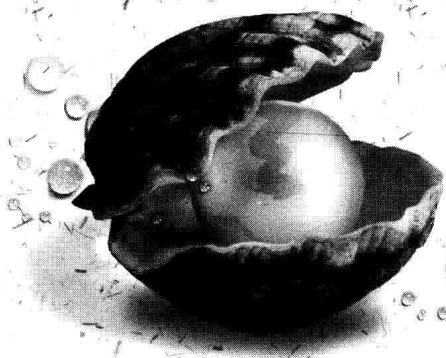
人民邮电出版社
POSTS & TELECOM PRESS

PEARSON

编程珠玑

续
修订版

[美] Jon Bentley◎著 钱丽艳 刘田 等◎译



More
Programming Pearls

人民邮电出版社
北京

图书在版编目 (C I P) 数据

编程珠玑 : 续 / (美) 本特利 (Bentley, J.) 著 ;
钱丽艳等译. — 3版 (修订本). — 北京 : 人民邮电出
版社, 2015. 9

书名原文: More Programming Pearls

ISBN 978-7-115-37372-4

I. ①编… II. ①本… ②钱… III. ①程序设计
IV. ①TP311.1

中国版本图书馆CIP数据核字 (2015) 第184534号

内 容 提 要

本书是计算机科学方面的经典名著《编程珠玑》的姊妹篇, 讲述了对于程序员有共性的知识。本书延续了《编程珠玑》的特色, 通过一些精心设计的有趣而又颇具指导意义的程序, 对实用程序设计技巧及基本设计原则进行透彻而睿智的描述, 为复杂的编程问题提供清晰而完备的解决思路。书中涵盖了程序员操纵程序的技术、程序员取舍的技巧、输入和输出设计以及算法示例, 这些内容结合成一个有机的整体, 如一串串珠玑展示给程序员。本书对各个层次的程序员都具有很高的阅读价值。

-
- ◆ 著 [美] Jon Bentley
 - 译 钱丽艳 刘 田 等
 - 责任编辑 杨海玲
 - 责任印制 张佳莹 焦志炜
 - ◆ 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>
 - 固安县铭成印刷有限公司印刷
 - ◆ 开本: 720×960 1/16
 - 印张: 14
 - 字数: 253 千字 2015 年 9 月第 3 版
 - 印数: 1—3 000 册 2015 年 9 月河北第 1 次印刷

著作权合同登记号 图字: 01-2007-0864 号

定价: 35.00 元

读者服务热线: (010)81055410 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

版 权 声 明

Authorized translation from the English language edition, entitled *More Programming Pearls: Confessions of a Coder*, 0201118890 by Jon Bentley, published by Pearson Education, Inc., publishing as Addison Wesley, Copyright © 1988 by AT&T Bell Laboratories.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD. and POSTS & TELECOM PRESS Copyright © 2015.

本书中文简体字版由 Pearson Education Asia Ltd. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

本书封面贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

译者序

本书作者 Jon Bentley 是美国著名的程序员和计算机科学家,他于 20 世纪 70 年代前后在很有影响力的《ACM 通讯》(*Communications of the ACM*) 上以专栏的形式连续发表了一系列短文,成功地总结和提炼了自己在长期的计算机程序设计实践中积累下来的宝贵经验。这些短文充满了真知灼见,而且文笔生动、可读性强,对于提高职业程序员的专业技能很有帮助,因此该专栏大受读者欢迎,成为当时该学术期刊的王牌栏目之一。可以想象当时的情形,颇似早年金庸先生在《明报》上连载其武侠小说的盛况。后来在 ACM 的鼓励下,作者经过仔细修订和补充整理,对各篇文章做了精心编排,分别在 1986 年和 1988 年结集出版了 *Programming Pearls* (《编程珠玑》) 和 *More Programming Pearls* (《编程珠玑(续)》) 这两本书,二者均成为该领域的名著。《编程珠玑(第 2 版)》在 2000 年问世,书中的例子都改用 C 语言书写,并多处提到如何用 C++ 和 Java 中的类来实现。《编程珠玑(续)》虽未再版,例子多以 Awk 语言写成,但其语法与 C 相近,容易看懂。

作者博览群书,旁征博引,无论是计算机科学的专业名著,如《计算机程序设计艺术》,还是普通的科普名著,如《啊哈!灵机一动》,都在作者笔下信手拈来、娓娓道出,更不用说随处可见的作者自己的真知灼见了。如果说《计算机程序设计艺术》这样的巨著代表了程序员们使用的“坦克和大炮”一类的重型武器,这两本书则在某种程度上类似于鲁迅先生所说的“匕首与投枪”一类的轻型武器,更能满足职业程序员的日常需要。或者说前者是武侠小说中提高内力修为的根本秘籍,后者是点拨临阵招数的速成宝典,二者同样都是克敌制胜的法宝,缺一不可。在无止境地追求精湛技艺这一点上,程序员、数学家和武侠们其实是相通的。

在美国,这两本书不仅被用作大学低年级数据结构与算法课程的教材,还用作高年级算法课程的辅助教材。例如,美国著名大学麻省理工学院的电气工程与计算机科学开放式核心课程算法导论就将这两本书列为推荐读物。这两本书覆盖了大学算法课程和数据结构课程的大部分内容,但是与普通教材的侧重点又不一样,不强调单纯从

数学上进行分析的技巧，而是强调结合实际问题来进行分析、应用和实现的技巧，因此可作为大学计算机专业的算法、数据结构、软件工程等课程的教师参考用书和优秀课外读物。书中有许多真实的历史案例和许多极好的练习题以及部分练习题的提示与解答，非常适合自学。正如作者所建议的那样，阅读这两本书时，读者需要备有纸和笔，最好还有一台计算机在手边，边读边想、边想边做，这样才能将阅读这两本书的收益最大化。

人民邮电出版社引进版权，同时翻译出版了《编程珠玑（第2版）》和《编程珠玑（续）》，使这两个中译本珠联璧合，相信这不仅能极大地满足广大程序员读者的需求，还有助于提高国内相关课程的授课质量和学生的学习兴趣。

本书主要由钱丽艳和刘田翻译，翻译过程中得到了严浩、李梁、任铁男三位研究生的帮助，在此一并表示感谢。由于本书内容深刻，语言精妙，而译者的水平和时间都比较有限，错误和不当之处在所难免，敬请广大读者批评指正。

前言

计算机编程充满乐趣，有时候，它又是一门优雅的科学，还要靠它去开发和使用新的软件工具。编程与人息息相关：客户实际想解决什么问题？如何让用户容易与程序沟通？是编程让我接触到相当广泛的话题，从有机化学到拿破仑战争。本书描述了编程的所有这些方面的知识，而且远不止这些。

这是一部短文集，每篇短文独立成章，但所有短文又依据逻辑分成了几组。第 1 章至第 4 章描述操纵程序的技术；第 5 章至第 8 章给出了一些程序员的实用技巧，这是本书技术性最低的部分；第 9 章至第 12 章讲解输入和输出设计；第 13 章至第 15 章介绍了 3 个有用的子程序。这些分类主题在每个部分的引言中进行了详细说明。

本书大多数章都是以我在《ACM 通讯》杂志中的“编程珠玑”(Programming Pearls) 专栏文章为基础的。各部分的引言中描述了这些文章的发表历史。既然已经发表过，为什么我还要费劲写这本书呢？自首次发表以来，这些专栏文章发生了很大变化，有了数千处小改进：有了新的问题和解决方案，纠正了小错误，并采纳了很多读者的意见。与此同时，我删除了一些旧的内容以免重复，并加入了很多新的内容，其中有一章是全新的。

然而，写本书的最大理由是，我想把各章组成一个有机的整体，我想展示一整串珠玑。我 1986 年出版的《编程珠玑》是类似的 13 篇短文的结集，围绕性能这个中心主题来组织，该主题在最早两年的《ACM 通讯》专栏中占据了突出位置。本书中也有几章再次谈及效率的话题，但全书考察的编程领域范围要大得多。

读者阅读本书时不要太快，一次一章，仔细地读。试解一下书中提出的问题——有些问题并不像看起来那样容易。有些章末尾的“深入阅读”并不是学术意义上的参考文献列表，而是我推荐的一些好书，这些书是我个人藏书的重要部分。

我很高兴能借此机会感谢许多人所作的重要贡献。感谢 Al Aho、Peter Denning、Brian Kernighan 和 Doug McIlroy 对各章提出了详细的意见。我还要感谢以下诸位提出

有益的见解: Bill Cleveland、Mike Garey、Eric Grosse、Gerard Holzmann、Lynn Jelinski、David Johnson、Arno Penzias、Ravi Sethi、Bjarne Stroustrup、Howard Trickey 和 Vic Vyssotsky。感谢允许我引用他们信件的几个人, 特别是 Peter Denning、Bob Floyd、Frank Starmer、Vic Vyssotsky 和 Bruce Weide。我特别要感谢 ACM 鼓励我把专栏文章出版成书, 还要感谢《ACM 通讯》的许多读者, 他们对原始专栏文章提出了不少意见, 使得本书这个扩充版本的出版十分必要。贝尔实验室(特别是其计算科学研究中心)在我写这些专栏文章时, 提供了极佳的支持环境。感谢所有的人。

Jon Bentley
于新泽西州 Murray Hill

目 录

第一部分 编程技术

第 1 章 性能监视工具	3
1.1 计算素数	3
1.2 使用性能监视工具	7
1.3 专用的性能监视工具	9
1.4 开发性能监视工具	11
1.5 原理	12
1.6 习题	13
1.7 深入阅读	14
第 2 章 关联数组	15
2.1 Awk 中的关联数组	16
2.2 有穷状态机模拟器	18
2.3 拓扑排序	20
2.4 原理	23
2.5 习题	24
2.6 深入阅读	25
第 3 章 程序员的忏悔	27
3.1 二分搜索	28
3.2 选择算法	30
3.3 子程序库	33
3.4 原理	35
3.5 习题	36
第 4 章 自描述数据	39
4.1 名字-值对	39
4.2 记录来历	42
4.3 排序实验	44

4.4 原理	45
4.5 习题	46

第二部分 实用技巧

第 5 章 劈开戈尔迪之结	49
5.1 小测验	49
5.2 解答	50
5.3 提示	51
5.4 原理	54
5.5 习题	55
5.6 深入阅读	56
5.7 调试 (边栏)	56
第 6 章 计算机科学箴言集	59
6.1 编码	60
6.2 用户界面	61
6.3 调试	62
6.4 性能	63
6.5 文档	64
6.6 软件管理	65
6.7 其他	67
6.8 原理	67
6.9 习题	68
6.10 深入阅读	69
第 7 章 粗略估算	71
7.1 头脑热身	71
7.2 性能的经验法则	73

7.3	Little 定律.....	75
7.4	原理.....	76
7.5	习题.....	76
7.6	深入阅读.....	78
7.7	日常速算 (边栏).....	78
第 8 章	人员备忘录.....	81
8.1	备忘录.....	82
8.2	原理.....	83
8.3	深入阅读.....	83

第三部分 人性化 I/O

第 9 章	小语言.....	87
9.1	Pic 语言.....	88
9.2	视角.....	91
9.3	Pic 预处理器.....	94
9.4	用来实现 Pic 的小语言.....	96
9.5	原理.....	100
9.6	习题.....	101
9.7	深入阅读.....	103
第 10 章	文档设计.....	105
10.1	表格.....	106
10.2	三条设计原则.....	108
10.3	插图.....	109
10.4	文本.....	111
10.5	合适的媒介.....	113
10.6	原理.....	116
10.7	习题.....	116
10.8	深入阅读.....	117
10.9	次要问题目录 (边栏).....	117
第 11 章	图形化输出.....	119
11.1	实例研究.....	119

11.2	显示结果取样.....	122
11.3	原理.....	125
11.4	习题.....	126
11.5	深入阅读.....	128
11.6	拿破仑远征莫斯科 (边栏).....	128
第 12 章	对调查的研究.....	131
12.1	有关民意调查的问题.....	131
12.2	语言.....	132
12.3	图片.....	135
12.4	原理.....	138
12.5	习题.....	139

第四部分 算法

第 13 章	绝妙的取样.....	143
13.1	取样算法一瞥.....	143
13.2	Floyd 算法.....	144
13.3	随机排列.....	146
13.4	原理.....	147
13.5	习题.....	147
13.6	深入阅读.....	148
第 14 章	编写数值计算程序.....	151
14.1	问题.....	151
14.2	牛顿迭代.....	152
14.3	良好的起点.....	154
14.4	代码.....	155
14.5	原理.....	158
14.6	习题.....	158
14.7	深入阅读.....	160
14.8	数值算法的力量 (边栏).....	161
第 15 章	选择.....	163
15.1	问题.....	163

15.2 程序.....	164	附录 A C 和 Awk 语言.....	175
15.3 运行时间分析.....	167	附录 B 子程序库.....	179
15.4 原理.....	171	部分习题答案.....	187
15.5 习题.....	172	索引.....	207
15.6 深入阅读.....	174		

第一部分 编程技术

我可没有耐心把最好的留到最后，这 4 章讨论程序员工作中最精彩的部分：你盯着计算机屏幕，敲着键盘度过的那些美好时光。

第 1 章说明如何使用性能监视工具（profiler）洞察程序的动态行为，第 2 章讨论一种强大的数据结构——关联数组（associative array），第 3 章描述用来测试和调试小的子程序的脚手架（scaffolding），第 4 章给出让数据文件自描述（self-describing）的方法。

这些技术都是用来处理真实程序的，所以要用真实系统上的真实语言来说明。第 1 章用 C 语言，第 2 章和第 3 章包含几个 Awk 程序。所有例子都使用了上述某种语言。附录 A 为不熟悉这些程序的读者简单描述了 C 和 Awk。虽然本书只使用了上述语言进行说明，但所介绍的技术可用在任何系统上。

第 1 章发表在 1987 年 7 月的《ACM 通讯》，第 2 章、第 3 章与附录 A 和附录 B 的早期版本一起发表于 1985 年 6 月和 7 月两期，第 4 章发表在 1987 年 6 月那一期。

本部分内容

- 第 1 章 性能监视工具
- 第 2 章 关联数组
- 第 3 章 程序员的忏悔
- 第 4 章 自描述数据

第1章

性能监视工具

听诊器是一种简单工具，却给医生的工作带来了革命：它让内科医生能有效地监控病人的身体。性能监视工具（**profiler**）对程序起着同样的作用。

你现在用什么工具来研究程序？复杂的分析系统很多，既有交互式调试器，又有程序动画系统。正如CT扫描仪永远代替不了听诊器一样，复杂的软件也永远代替不了程序员用来监控程序的最简单工具——性能监视工具，我们用它了解程序各部分的执行频率。

本章先用两种性能监视工具来加速一个小程序（记住真正的目的是说明性能监视工具）。后续各节简要介绍性能监视工具的各种用途、非过程语言的性能监视工具，以及开发性能监视工具的技术。

1.1 计算素数

程序P1是个ANSI标准C程序，依次打印所有小于1000的素数（如果读者不了解C，请看附录A）。

程序P1

```
int prime(int n)
{
    int i;
999    for (i = 2; i < n; i++)
78022        if (n%i == 0)
831            return 0;
168    return 1;
}
main()
{
    int i, n;
1    n = 1000;
1    for (i = 2; i <= n; i++)
999        if (prime(i))
```

```

168             printf("%d\n", i);
    }

```

如果整型参数 n 是素数，上述prime函数返回1（真），否则返回0。这个函数检验2到 $n-1$ 之间的所有整数，看其是否整除 n 。上述main过程用prime子程序来依次检查整数2~1000，发现素数就打印出来。

我像写任何一个C程序那样写好程序P1，然后在性能监视选项下进行编译。在程序运行之后，只要一个简单的命令就生成了前面所示的列表。（我稍微改变了一些输出的格式。）每行左侧的数由性能监视工具生成，用于说明相应的行执行了多少次。例如，main函数调用了1次，其中测试了999个整数，找出了168个素数。函数prime被调用了999次，其中168次返回1，另外831次返回0（快速验证：168+831=999）。prime函数共测试了78022个可能的因子，或者说为了确定素数性，对每个整数检查了大约78个因子。

程序P1是正确的，但是很慢。在VAX-11/750上，计算出小于1000的所有素数约需几秒钟，但计算出小于10 000的所有素数却需要3分钟。对这些计算的性能监视表明，大多数时间花在了测试因子上。因而下一个程序只对 n 考虑不超过 $\sqrt{n}$ 的可能的整数因子。整型函数root先把整型参数转换成浮点型，然后调用库函数sqrt，最后再把浮点型结果转换回整型。程序P2包含两个旧函数和这个新函数root。

程序P2

```

int root(int n)
5456 { return (int) sqrt((float) n); }

int prime(int n)
{   int i;
999   for (i = 2; i <= root(n); i++)
5288       if (n % i == 0)
831           return 0;
168       return 1;
}

main()
{   int i, n;
1       n = 1000;
1       for (i = 2; i <= n; i++)
999           if (prime(i))
168               printf("%d\n", i);
}

```

修改显然是有效的：程序P2的行计数显示，只测试了5288个因子（程序P1的1/14），总共调用了5456次root（测试了5288次整除性，168次由于 i 超出了 $\text{root}(n)$ 而终止循环）。不过，虽然计数大大减少了，但是程序P2运行了5.8秒，而程序P1只运行了2.4秒（本节末尾的表中含有运行时间的更多细节）。这说明什么呢？

迄今为止，我们只看到了行计数（line-count）性能监视。过程时间（procedure-time）性能监视给出了较少的控制流细节，但更多地揭示了CPU时间：

%time	cumsecs	#call	ms/call	name
82.7	4.77			_sqrt
4.5	5.03	999	0.26	_prime
4.3	5.28	5456	0.05	_root
2.6	5.43			_frexp
1.4	5.51			__doprint
1.2	5.57			_write
0.9	5.63			mcount
0.6	5.66			_creat
0.6	5.69			_printf
0.4	5.72	1	25.00	_main
0.3	5.73			_close
0.3	5.75			_exit
0.3	5.77			_isatty

过程按照运行时间递减的顺序列出。时间上既显示出总秒数，也显示出占总时间的百分比。编译后记录下源程序中main、prime和root这3个过程的调用次数。再次看到这几个计数是令人鼓舞的。其他过程没有性能监视的库函数，完成各种输入/输出和清理维护工作。第4列说明了带语句计数的所有函数每次调用的平均毫秒数。

过程时间性能监视说明，sqrt占用CPU时间的最多：该函数共被调用5456次，for循环的每次测试都要调用一次sqrt。程序P3通过把sqrt调用移到循环之外，使得在prime的每次调用中只调用一次费时的sqrt过程。

程序P3

```

int prime(int n)
{
    int i, bound;
999     bound = root(n);
999     for (i = 2; i <= bound; i++)
5288         if (n % i == 0)
831             return 0;
168         return 1;
}

```

当 $n=1000$ 时，程序P3的运行速度大约是程序P2的4倍，而当 $n=100\,000$ 时则超过10倍。以 $n=100\,000$ 的情形为例，过程时间性能监视显示，sqrt占用了程序P2的88%的运行时间，但是只占用了程序P3的48%的运行时间。这好多了，但仍然是循环的累赘。

程序P4合并了另外两个加速措施。首先，程序P4通过对被2、3和5整除的特殊检验，避免了近3/4的开方运算。语句计数表明，被2整除的性质大约把一半的输入归入合数，被3整除把剩余输入的1/3归入合数，被5整除再把剩下的这些数的1/5归入合数。其次，只考虑奇数作为可能的因子，在剩余的数中避免了大约一半的整除检验。它比程序P3大约快两倍，但是也比P3的错误更多。下面是（带错的）程序P4，你能通过检

查语句计数看出问题吗？

程序P4

```
int root(int n)
265 { return (int) sqrt((float) n); }

int prime(int n)
{ int i, bound;
999   if (n % 2 == 0)
500     return 0;
499   if (n % 3 == 0)
167     return 0;
332   if (n % 5 == 0)
67     return 0;
265   bound = root(n);
265   for (i = 7; i <= bound; i = i+2)
1530     if (n % i == 0)
100       return 0;
165   return 1;
}

main()
{ int i, n;
1    n = 1000;
1    for (i = 2; i <= n; i++)
999      if (prime(i))
165        printf("%d\n", i);
}
```

先前的程序找到168个素数，而程序P4只找到165个。丢失的3个素数在哪里？对了，我把3个数作为特殊情形，每个数都引入了一处错误：prime报告2不是素数，因为它被2整除。对于3和5，存在类似的错误。正确的检验是

```
if (n % 2 == 0)
    return (n == 2);
```

依此类推。如果n被2整除，如果n是2就返回1，否则返回0。对于n=1000、10 000和100 000，程序P4的过程时间性能监视结果总结在下表中。

n	时间百分比		
	Sqrt	prime	其他
1000	45	19	36
10 000	39	42	19
100 000	31	56	13

程序P5比程序P4快，并且有个好处：正确。它把费时的开方运算换成了乘法，如下程序片段所示。