

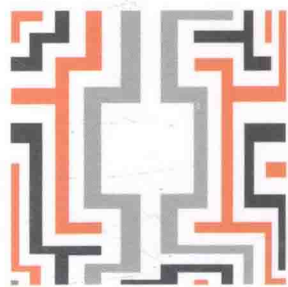
知名计算机先驱历经四十年考验的经典之作
“文津图书奖”译者高博最新译本
修炼编程功力、提高编程格调的终极宝典

the elements of
programming

编程格调

[美] Brian W. Kernighan P.J. Plauger 著
高博 徐章宁 译

STYLE



 人民邮电出版社
POSTS & TELECOM PRESS

the elements of
programming

STYLE

编程格调

[美] Brian W. Kernighan P.J. Plauger 著
高博 徐章宁 译

人民邮电出版社
北京

图书在版编目 (C I P) 数据

编程格调 / (美) 科宁汉 (Kernighan, B. W.), (美) 普劳者 (Plauger, P. J.) 著; 高博, 徐章宁译. — 北京: 人民邮电出版社, 2015. 3
ISBN 978-7-115-37952-8

I. ①编… II. ①科… ②普… ③高… ④徐… III. ①程序设计 IV. ①TP311.1

中国版本图书馆CIP数据核字(2015)第010915号

内 容 提 要

本书是编程惯用法和规则的实践指南。全书从表达、控制结构、程序结构、输入和输出、常见错误、效率和测试工具、文档等多个角度, 概括了程序设计的最佳实践或规则, 并通过代码示例加以分析和阐释。

本书两位作者都是程序设计领域的大师级人物。他们四十年前在本书中给出的 70 多条最佳实践和规划, 大多数在今天仍然适用。

本书堪称计算机专业人士和程序员必读的经典之作, 适合于不同层级的程序员和计算机相关专业的学生参考阅读。

◆ 著 [美] Brian W.Kernighan P.J.Plauger

译 高 博 徐章宁

责任编辑 陈冀康

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41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: 12.25

字数: 206 千字

2015 年 3 月第 1 版

印数: 1-3 500 册

2015 年 3 月河北第 1 次印刷

著作权合同登记号 图字: 01-2012-8414 号

定价: 45.00 元

读者服务热线: (010)81055410 印装质量热线: (010)81055316

反盗版热线: (010)81055315



版权声明

Simplified Chinese translation copyright ©2015 by Posts and Telecommunications Press

ALL RIGHTS RESERVED

The Elements of Programming Style(2nd Edition), ISBN-13: 978-0-070-34207-1

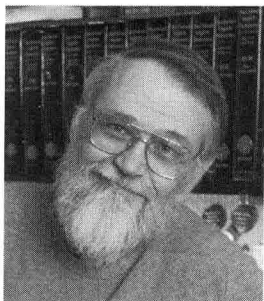
by Kernighan and Plauger

Copyright © 1978 by Kernighan and Plauger

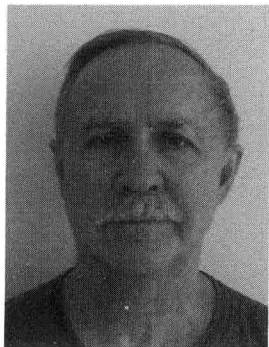
本书中文简体版由作者 Kernighan、Plauger 授权人民邮电出版社出版。未经出版者书面许可，对本书的任何部分不得以任何方式或任何手段复制和传播。

版权所有，侵权必究。

作者简介



Brian W. Kernighan, 全球知名、德高望重的计算机先驱, 在程序设计方法论和软件工程方面做了大量开创性的工作。他曾长期在贝尔实验室工作, 现在普林斯顿大学计算机科学系任教。他著有数本经典教材, 包括与 Dennis Ritchie 合著的传世之作 *The C Programming Language*、与 Rob Pike 合著的 *The Practice of Programming*, 以及最近出版的科普图书 *D is for Digital* 等。他还是 AWK 编程语言的发明者, 这种语言广泛地应用在 UNIX/Linux 应用中。“K&R C”和“AWK”中的“K”都代表 Kernighan。

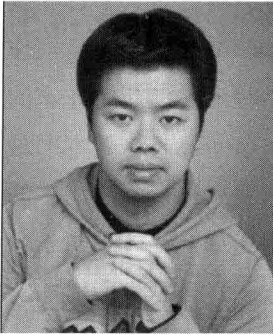


P.J. Plauger, 全球知名的计算机科学家、C/C++技术专家以及技术图书作者, 更是数个标准 C/C++程序库的作者。他曾经在贝尔实验室工作, 现在任美国 Dinkumware 公司总裁。他曾经担任 *C/C++ Users Journal* 高级编辑, 也是 *The Standard C Library*、*Standard C: A Reference* 和 *The Standard Template Library* 等图书的作者。

译者简介



高博，1983 年生，毕业于上海交通大学。目前在互联网金融创业公司任首席产品官兼首席质量官，在信息科学和工程领域有近 15 年实践和研究经验。酷爱读书和写作，业余研究兴趣涉猎广泛。译著包括图灵奖作者高德纳的《研究之美》和布鲁克斯的《设计原本》，以及 Jolt 大奖作品《元素模式》等，出版翻译作品计近百万字。



徐章宁，1984 年生，就读于上海交通大学，硕士毕业后就职于 EMC 中国卓越研发集团，现任 EMC 公司高级系统管理工程师，从事软件运维工作多年，钟爱开源软件。对各类知识有广泛兴趣，平日喜爱参与问答网站讨论，热爱读书摄影和写作。

译者序

这本尘封了近 40 年的著名教材此次重新翻译，既是我们向前辈的致敬，也是我们重读经典的努力和尝试。译者第一次看到本书，是在 1998 年参加全国信息学竞赛期间，彼时也正是计算机工业在中国进入黄金发展期的前夜，读后的心潮澎湃历历如昨。此后，在和很多业界知名的前辈如 Donald E. Knuth、Bjarne Stroustrup 和 Anders Hejlsberg 等交流的过程中，也不止一次地听到本书的名字，可见本书影响之纵深。

为什么本书能有如此的影响力？也许会有人说，是因为作者的名气，但这是事后的见解。两位作者的功底固然是力透纸背，可放在当时的历史背景下，其实主要的原因却在于，它开创了一个程序设计习惯用法（idiom）的先河。和两位作者此后的作品相比，本书中很多地方反而可以看出，他们当时还是以新锐作家的口吻，在创立自己的观点，而非早已站在制高点处向大众布道而已。这是本书最为宝贵之处，我们从中可得以窥见现在已经几乎成为常识的观点，发端于怎样的初心，并经历过怎样的磨砺，才逐渐站稳了脚跟。也正是因为如此，本书对于当时的软件业界而言，不啻是黑暗中的一道闪电，照亮了人们前行的步伐，所以才能在这么多高手的心中留下难以磨灭的印象。

而对于今天中国的软件业界，本书的重新翻译和出版，同样意义重大。求新、求异，却不重视程序设计基本功和工程素养的风气，近年来有令人不安的抬头趋势。我们思考再三，决定将本书的译名定为《编程格调》，也是为了强调书中反复申明、历久弥坚的习惯用法以及它们反映出来的宝贵思想，对于当前现实的指导意义。阅读本书，可以回答一个非常基本的问题：怎样去构建软件、构建怎样的软件，才能说成是“有格调”的，才能走得稳、走得远。

当然，对于一本近 40 年前的出版物，我们不可能要求它尽善尽美。比如书中采用的编程语言，在今天看来已经不是非常流行。但一个合格的程序员，应该可以比较迅速地掌握一门新语言的结构，至少看懂一段代码所要表达的含义。正如我国著名画家齐白石所言：“学我者生，似我者死”。我们要学会学习，吸收书中观点的精华，并和自己当下的工程实践相结合，而非邯郸学步。

感谢人民邮电出版社的陈冀康编辑为促成本书的出版而作出的大量努力，EMC 中国卓越研发集团的徐章宁高级工程师和本人一起完成了本书的重新翻译工作，这里也要向他的辛勤工作表达由衷的感谢。本书成稿的过程中，上海交通大学计算科学与技术系的张尧弼和梁阿磊教授、赛门铁克中国区技术负责人汤瑞欣总监、华为存储事业部严华兵高级经理和开源社区事业部杜玉杰工程师，以及 GitCafé CEO 姚欣宇等都参加过审阅并提出了宝贵意见。当然，由于译者能力所限，本书中仍然不可避免地存在更多的纰漏，这些理应由我们负责。我也想借此机会向在工作和生活上给了我莫大支持的父母和爱妻沈靓表达我内心最深处的敬意，希望本书的出版能给你们带来快乐。



2015 年元旦

于杭州苏堤

再版序言

自《编程格调》初版面世以来，计算机程序设计实践已经今非昔比。关于编程格调的讨论已登大雅之堂。历经“代码写过算数”的漫长岁月，学生、教师，以及计算界专业人士已经意识到程序可读性的重要。结构化程序设计已经作为一种极有价值的编程原理被普遍接受，而且人们也逐渐意识到，设计乃是一个重要的编程阶段，而这在过去经常是被忽视的。

我们对《编程格调》进行了一番彻底改写。在初版中，我们避免在任何地方直接使用“结构化程序设计”，以远离当时甚嚣尘上的宗派之争。而现在，人们的热情已经消退，我们反而感觉是时候讨论已在实践中经受住考验的结构化程序设计话题了。

本书再版时，加入了新的一整章来讨论程序结构，阐明如何运行自顶向下的设计写出结构良好的程序。全书都在讨论设计中遇到的种种问题。我们也谨慎地运用伪代码作为一种程序开发工具。

我们也重写了初版中的大量示例，以反映出（至少我们希望如此）对于如何编好程序的更深入理解。同样，书中还引入了不少新示例。初版中不少拿来作为榜样的示例，现在却成了反面教材。再版还加入了一些新的习题。最后，我们扩展、深化了作为编程格调的种种规则。

对于慷慨授权我们在书中重印他们教材中程序的作者和出版商，我们再次感觉受益匪浅。回过头来看我们自己写的程序就能明白，撰写出优秀的程序有着怎样高的要求。

我们同时要感谢阅读再版草稿的朋友，尤其要感谢 Al Aho、Jim Blue、Stu Feldman、Paul Kernighan、Doug McIlroy、Ralph Muha 和 Dick Wexelbalt，他们曾提出过非常有价值的建议。

Brian W.Kernighan

P. J. Plauger

初版序言

仅仅通过听取一般原理，就想很好地掌握程序设计，这是不可能的事。想要学好程序设计，就要一次又一次地观察体会实际的程序是如何通过运用一些久经实践考验的原理和常识，方得持续改进的。程序审阅能够教会人们如何重写程序，再进一步教会人们如何写出更好的程序。

本书就是建立在对大量“实际”程序的研究之上的成果，其中的每个程序都有着一种或数种格调方面的毛病。我们讨论了每个例子存在的缺点，并采用更好的方式将其重写，然后由这个特殊情况推出一般原理。我们的出发点是实用的、接地气的。我们关心的是要改进当前程序设计的工作，而不是想要创建一套繁杂理论来告诉人们如何进行程序设计。所以，本书可以用作任何水平的程序设计课程的补充材料，也可以让已有经验的程序员们读后获得温故而知新的效果。

书中的所有示例皆以 Fortran 和 PL/I 写就，因为这两种语言不仅被广泛采用，而且互相类似，只要会看其中一种，则另一种也不言自明（无论采用哪一种语言来写程序，我们都避免采用复杂结构，遇到不可避免的习惯用法时，也会加以解释）。而有关编程格调的原理，则适用于一切语言，包括汇编码。

我们的目标在于仅运用少量的篇幅即能讲授编程格调的妙处，所以我们有一说一。贯穿全书的是一系列规则，这样方能表明我们习得的教训。每一章都以一段总结和一组“思考要点”收尾，其中会给出一些练习，并使得读者有机会继续考查正文中没有讲全的一些议题。我们在书的末尾将规则集中列出，以飨读者。

这里特别要讲一讲本书中程序实例的来源，它们全部选自其他的程序设计教科书。换言之，我们不通过自己撰写程序来论证我们的观点，而是使用那些已经完成的产品，即那些由富有经验的程序员撰写并且发布的程序。这些程序往往就是新程序员们入行后见到的首批程序，本来我们希望这些例子能起到编程格调的示范作用，遗憾的是却常常事与愿违。这些教科书中举的例子往往本想说明自己

在某个方面做得出色，结果却适得其反，暴露了问题。（我们不会故意使坏，通过断章取义来使程序在该语境下显得写得不好。）

我们还要特意声明：我们不是要以个人或集体名义批判这些教科书的作者们。缺点的存在仅仅表明我们不过都是凡人，在编程或写书这样压力巨大的脑力劳动中，有些事情做得不完美诚属难免。毫无疑问地，本书中所提到的一些“好”程序，一定也会被未来的某位作者举作“坏”程序的反面例子。我们也诚心地希望未来的这位作者及其读者也能通过仔细研究这些程序的经验而学到东西。

一本编程格调手册的问世，离不开很多人开创性的工作，他们中有相当数量的人就写过非常出色的教科书。比如，D. D. McCracken 和 G. M. Weinberg 很久以来就在教学中呼吁简单性和明晰性的极端重要。E. W. Dijkstra 和 Harlan Mills 在结构化程序设计方面取得的成果，帮助我们明确了指定控制流的相关规则。本书的篇章安排和语言风格都深受 W. Strunk 和 E. B. White 的名著 *The Elements of Style* 一书的影响，我们通过集中精力于格调的本质和实用方面，来达到该书的简明程度。

我们受益于太多人的帮助和鼓励，特别是授权我们复用在本书正文中的程序作者和出版社，这样的合作极其难能可贵。

我们在贝尔实验室的朋友和同事们提出过大量有用的建议，正是由于这些建议，我们得以在出版前纠正了不少本来可能贻笑大方的谬误。特别要感谢 V. A. Vyssotsky 与我们一起进行了数次修订工作，对于他富有洞见的评论和对本书成形过程中每一阶段的热情支持，我们深表谢意（还有，我们在书中“无耻盗用”了他的几句格言）。我们还想感谢 A.V. Aho、M. E. Lesk、M. D. McIlroy 和 J. S. Thompson 给予本书的时间投入和莫大帮助。

我们之所以能把手稿直接录入 PDP 11/45、编辑源码、校验程序，并对最终版本排版，所有这一切都要仰仗一个独一无二的、灵活的操作系统，这就是 UNIX。K. L. Thompson 和 D. M. Ritchie 是 UNIX 的首席架构师，除了帮我们校稿以外，他们还在我们写书时帮我们最大程度地利用好 UNIX 系统。J. E. Ossanna 编写了排版程序并修改了数处，以满足我们的特殊要求。在此致谢。

Brian W. Kernighan

P. J. Plauger

目录

第 1 章 绪论	1
撰写简洁的程序——不要耍小聪明	2
第 2 章 表达	13
简单并且直接地表达你要说的意思	14
使用库函数	14
避免使用临时变量	16
代码要清晰，不要为了“效率”牺牲可读性	16
让机器干脏活	17
用函数调用替代重复的表达式	18
加括号来避免歧义	21
选择不会被混淆的变量名	21
避免使用 Fortran 的算术 IF	23
避免不必要的分支	25
使用语言好的特性，避免使用不好的特性	25
不要使用条件分支来代替一个逻辑表达式	26
用“电话测试”来检查可读性	29
第 3 章 控制结构	39
使用 DO-END 和缩进来界定语句组	40
用 IF-ELSE 强调两个操作中只有一个被执行	42
用 DO 和 DO-WHILE 来强调循环的存在	45
确保你的程序是自顶向下阅读的	46

2 编程格调

使用 IF...ELSE IF...ELSE IF...ELSE 来实现多路分支	47
使用基本的控制流结构	48
先用容易理解的伪语言编写代码，然后再翻译成你需要使用的语言	52
避免使用 THEN-IF 和空 ELSE	55
避免使用 ELSE GOTO 和 ELSE RETURN	56
判断要尽可能紧挨着与之相关的操作	58
使用数组来避免重复的控制流	61
选择可以简化程序的数据表示方法	63
不要止步于第一遍的代码草稿	66

第 4 章 程序结构 71

模块化，使用子例程	74
让模块之间的耦合变得可见	75
每一个模块都应该做好一件事	76
确保每一个模块都隐藏好一些东西	78
以数据为导向来构建程序的结构	80
不要修补烂代码——重写它	84
分块编写和测试大的程序	91
对于递归定义的数据结构使用递归过程	91

第 5 章 输入和输出 97

校验输入的合法性和合理性	100
保证输入数据不会违背程序的限制	101
利用文件结束符号或结束标志来终止输入，不要让用户去计数	102
识别出非法输入数据，如果可能则纠正之	103
使用统一的形式处理文件结束条件	105
让输入数据易于准备，并让输出数据意义不言自明	108
使用统一的输入格式	110
让输入数据易于校对	111
尽可能选择自由格式输入	112

使用含义自明的输入，指定默认值，将以上二者都输出·····	112
将输入与输出局限在子例程中·····	116
第 6 章 常见错误·····	119
确保所有的变量在使用之前都被初始化·····	120
不要停留在一个 bug 上·····	122
使用调试编译器·····	124
用 DATA 语句或 INITIAL 属性初始化常量，用可执行语句初始化变量·····	125
小心“差一”错误·····	126
要注意对不等式进行正确的分支·····	126
避免循环有多个出口·····	128
确保你的代码巧妙地“不做事情”·····	131
在边界值上测试程序·····	135
预防性编程·····	136
10.0 乘以 0.1 不等于 1.0·····	137
不要比较浮点数是否相等·····	139
第 7 章 效率和测试工具·····	145
先做对，再做快·····	147
在提高程序运行速度时，要保持其正确性·····	149
先把程序改得更简洁，再提高其运行速度·····	150
不要为了“效率”上的蝇头小利而牺牲程序的简洁性·····	151
让编译器执行平凡优化·····	151
不要勉强地复用代码，应该进行改编·····	152
保证特殊情况真的有特殊性·····	155
保持简单性，反而会更快·····	157
不要为了提高速度而画蛇添足——去寻找更好的算法·····	159
在程序中放置测试语句，“增效”之前先执行测算·····	161
第 8 章 文档·····	165

4 编程格调

确保注释和代码一致	167
不要用注释复述代码做的事情，每个注释都要有实际意义	167
不要注释糟糕的代码——重写它	169
使用含有意义的变量名	170
使用含有意义的语句标签	171
程序的格式要有助于读者的理解	171
用缩进来体现程序的逻辑结构	172
记录你的数据规划	175
不要过度注释	176
结束语	180

第 1 章 绪论

考查以下程序片断。

```
DO 14 I=1,N
DO 14 J=1,N
14 V(I,J)=(I/J)*(J/I)
```

只要对 Fortran 语言有一定的了解，我们就知道，这个双重嵌套的 DO 循环是要对具有 $N \times N$ 个元素的矩阵 V 的各个元素赋以某值。怎样的值呢？ I 和 J 是正的整型变量，在 Fortran 语言中，整型除法会向零取整。因此，当 I 小于 J 时， (I/J) 为 0。反之，当 J 小于 I 时， (J/I) 为 0。而当 I 等于 J 时，两个因数皆为 1，所以 $(I/J) * (J/I)$ 等于 1，当且仅当 I 等于 J ，否则皆为 0。这段程序意在将 V 的对角元素置 1，而其余元素置 0（从而 V 成为一单位矩阵）。多么聪明啊！

果然如此吗？

设想，你是在一个较大的程序中遭遇了这段程序。如果你对 Fortran 语言造诣颇深，那你可能会很欣赏此处对整型除法的妙用。但也有可能你会觉得吃惊不小，因为明明有更简单的做法，这里采用的做法却要做两次除法、一次乘法和一次从

整型到浮点型的强制转换。更大的可能是，你不得不重复上面的推理过程。极大的可能是，你只是有了一个囫圇的印象，知道它向数组中放置了某些有用的东西。只有当你有了强烈的动机，比如要对程序进行调试或改动时，才会回过来猜测它的确切含义。

这段程序这样写会更好。

```
C MAKE V AN IDENTITY MATRIX
  DO 14 I = 1,N
    DO 12 J = 1,N
12      V(I,J) = 0.0
14      V(I,I) = 1.0
```

先将各行置 0，然后将对角元素置 1。现在含义十分明确，代码运行得也更快了。如果我们是用 PL/I 撰写程序的，就可以表达得更明显。

```
/* MAKE V AN IDENTITY MATRIX */
  V = 0.0;
  DO I = 1 TO N;
    V(I,I) = 1.0;
  END;
```

无论哪种情况，要点在于要使代码的意图不会被误解，而非炫技。在这里，存储空间和执行时间相对而言都不重要，因为建立单位矩阵一定是整个程序中的一小部分。难懂代码的问题在于它难以调试和修改，而这些已经成为计算机程序设计中最为困难的问题。此外，过分聪明的程序会增加与原意发生偏离的危险。

撰写简洁的程序——不要耍小聪明

暂停一下，让我们反思一下到目前为止我们都做了些什么。我们研究了一段程序，该程序原样引自某本程序设计教材。我们讨论这段程序的优点和缺点，然后着手对其加以改进。（未必达到完美——仅仅是改进。）在分析和改进中，我们总结出一条规则或是一般结论，该规则听起来不免有点像是有着纯粹抽象的普适性，但它合情合理，在遇到具体情况时就能具体运用。

本书其余部分大抵如此，即从教材中选取实例，尔后是讨论和改进，再得出