HZ BOOKS
华章科技
Apress®

Amazon五星级畅销书，遗留代码调试和优化领域的代表性著作，资深专家10余年经验结晶

不仅从实用性角度深入、系统地讲解调试和优化遗留代码的方法、技术和最佳实践，而且从源头上阐述如何避免掉进维护遗留代码的泥潭

名家经典系列

软件驱魔

调试和优化遗留代码的艺术

（美） Bill Blunden 著
施远敏 张燎原 何军 译

Software Exorcism

A Handbook for Debugging and Optimizing Legacy Code

机械工业出版社
China Machine Press

名家经典系列

软件驱魔

调试和优化遗留代码的艺术

（美） Bill Blunden 著
施远敏 张燎原 何军 译

Software Exorcism

A Handbook for Debugging and Optimizing Legacy Code

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

软件驱魔：调试和优化遗留代码的艺术 /（美）布伦登（Blunden，B.）著；施远敏，张燎原，何军译 . —北京：机械工业出版社，2014.5
（名家经典系列）
书名原文：Software Exorcism: A Handbook for Debugging and Optimizing Legacy Code

ISBN 978-7-111-46284-2

I. 软… II. ① 布… ② 施… ③ 张… ④ 何… III. 软件开发－研究 IV. TP311.52

中国版本图书馆 CIP 数据核字（2014）第 062611 号

本书版权登记号：图字：01-2013-9163

软件驱魔：调试和优化遗留代码的艺术

[美] Bill Blunden 著

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：谢晓芳　　责任校对：殷　虹
印　　刷：三河市宏图印务有限公司　　版　　次：2014 年 5 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：16.75
书　　号：ISBN 978-7-111-46284-2　　定　　价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

译 者 序

从《画皮》里孙俪饰演的降魔人，《倩女悠魂》里古天乐饰演的猎妖师，到《西游记》里年轻的驱魔人陈玄奘（文章饰）。不知道为什么，每个故事的驱魔工作者们无一例外，都是蓬头垢面、衣衫褴褛的造型。我一直在想，是什么样的工作性质，让他们如此狼狈不堪：爬高山，钻洞穴，寂寞伴身，学会守候……

我本人所参与项目的代码库里，包含了从20世纪80年代到现在跨度将近三十年的代码，这个样本都适合做代码“考古”了，而公司目前有几百人正在从事这套软件的维护工作，维护工作的绝大多数时间就是在与遗留代码“战斗”。他们需要深入代码洞穴，攀登代码树，在错综复杂的逻辑结构中跟踪每一步可能的分支结点，除了要抓到贻害一方的缺陷，还必须在有限的时间里完成这一切工作，这就像美剧里演的一样，一个人拿着枪指着你的头，掐着秒表，逼着你在秒表归零之前完成复杂的网络攻击任务。作者把软件维护工程师比做软件驱魔人，这是多么贴切的比喻。我现在才惊醒，原来我身边的人本应如孙俪一样俊俏，像古天乐一样有型，似文章一般阳光清纯，而“伟大”的降妖除魔工作，却让他们一个个如降魔人、猎妖师和陈玄奘一般狼狈不堪。

最近听到一个真实的故事：有一位1990年出生的女同事，为了解决一个bug，不得不修改1993年编写的一段遗留代码。加班到晚上八九点之时，这位女同事大发感慨：“当我还只有三岁的时候，就有一位怪叔叔写了一段代码，等20年后我长大了再修改。”在捧腹大笑的同时，我们也在沉思，还有多少遗留代码是我们孩提时就已经存在，并且需要我们去修改的呢？或者，我们给正在成长的幼儿园宝宝们，又准备了多少的“遗产”呢？前人挖坑，后人填坑，子子孙孙无穷尽也，这会成为我们这些软件从业者的常态吗？

作者Bill Blunden试图通过本书，告诉程序员如何预防掉入软件维护的泥潭，冠冕点的说法，叫做从源头抓起，而一旦进入缺陷的修改中，程序员应该如何进行调试，针对软件性能的提升，又有什么比较好的方法。从整本书的组织角度来看，条理性比较强。作者不求面面俱到，但从实用性的角度，还是给出了比较多的方法介绍和建议，如防御性编程、单元测试、软件跟踪、调试、优化，甚至调试器的内部实现。本书非常适合初入软件维护队伍的从业者，对维护行列的老手来说，本书则会略觉浅显，但系统性的总结也是一个不错的参考。当然，对于编写软件的从业者来说，这

也是一部值得学习的图书，前面几章是关于如何预防的，防止再次编写难于维护的代码就是不错的建议，同时，关于调优的章节也一样受用。

在市面上各种编程语言介绍之类的图书大行其道的当下，这样一部专门讲授如何维护遗留代码的图书就显得异外扎眼。作者并没有提及太过高深的理论或技法，给人的感觉是更多的内容都来自作者在一线维护工作中点点滴滴的经验总结。

参加本书翻译的几位译者都是IT从业人员，有十余年的一线软件研发和维护经验，对书中的描写更能引起共鸣，这也是我们将此书介绍给国内读者的初衷。当然，译者翻译经验有限，谴词造句难免会有出入或生涩之处，请读者批评指正，多多包涵。

感谢机械工业出版社的编辑谢晓芳细致的编辑工作，她是很有耐心的一位编辑，容忍我们一次又一次的延迟交稿，如果没有她这些可贵的品质，本书不可能出版。

感谢我们团队的小伙伴们：郑建琴、王刚、余鹰伟、周峰、凌建发、宓媛珊、吴恩平（排名不分先后）。他们利用晚上的闲暇时光，帮助我们一起审读译稿；感谢公司给我们创造了一个开放宽松的学习实践环境，我们可以将书中所学传播到工作中，践行各种策略、方法和实践。

张燎原

前　言

据记载[⊖]，在典型的软件项目生命周期中，维护阶段的成本占总成本的 60% 以上。研发需要什么样的排场，什么样的环境，一般都会在设计阶段确定，一旦完成第一个版本的部署，接下来的大部分资源都将用于修复 bug，增加新的功能，并依次修复这些新的功能所产生的 bug。多数的软件应用程序在发布后的开发周期如图 1 所示。

⊖ Don Coleman et al., "Using Metrics to Evaluate Software System Maintainability," *IEEE Computer*, August 1994, pp. 44–49

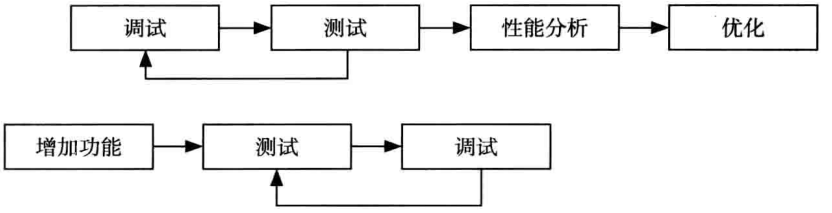


图 1　软件应用程序在发布后的开发周期

产品发布后的开发周期始于新增一个功能，或者提交一份 bug 报告。在这两种情况下，单元测试和系统测试常用来确定补丁代码是否工作正常。一旦程序的执行已经足够稳定，就可以通过剖析应用程序来定位性能瓶颈。然后实施各种各样的优化技术，以提高执行速度和减少内存占用。随着客户要求加入新特性和修复 bug，这种周期会周而复始地发生，应用程序也正是这样不停地向前演进。

关于软件工程的大多数书籍更关注一个应用程序从立项到发布的各个开发阶段（例如，需求、分析、用例、实现等）。大家都知道，从头写一个程序会是一件更容易、更开心的事。而软件维护则更像是计算机科学中后娘养的孩子。大多数教授都不愿在公众面前提及，这是因为讨论维护工作的严酷性可能会引发各种问题……一些相当危险的问题。当你最终的辛苦努力只是做一件令人沮丧的维护工作时，你会质疑当初为什么要花四年去获取一个计算机学位。如果老师告诉学生有关软件工程这一职业的真相，肯定有很多学生会逃离计算机科学专业。一个较小的院系将成为削减预算的牺牲品，并且声誉扫地，所以你最好相信，教授们更愿意避开一些不愉快的事实，以保证他们班级的规模。

这本书不同。与其避而不见，不如直视软件行业肮脏的现实，让我们重温记录，看清自己未经审查的真相。为了有备无患，本书将探讨那些方便程序员调试和优化遗留软件的工具。换句话说，这本书要讨论的是：怎样维护那些已被他人“玩死”的软件。沏杯好茶，选个合适的椅子，开始你的重温之旅，但不要抱怨我没提醒你。

一个海军陆战队侦察员曾经告诉我，武术家没有资格谈论哪些技战术具有“战斗力”，除非他们曾经在遇到生命危险的时候使用过。本着这种精神，我觉得本书不该轻率地提供关于软件维护的建议，除非作者在压力非常大的情形下使用过它们。这本书倾向于具有实战价值的战术而特意避免了象牙塔里的抽象概念。我对自己曾经所使用过的简单工具情有独钟，一般的维护工程师并没有太多的时间浪费在云图或时髦的开发方法上。维护工程师有活儿要干，并且必须立刻搞定。而本书所介绍的战术，就是在战壕里，在毫无准备的情况下就可以使用的战术。

历史动因

早在铁器时代（即 20 世纪 60 年代末和 70 年代），大多数软件工程师没有太多的计算机资产可

以使用。像房间那么大的 CDC 6600 计算机只有 65 000 个 60 位字的内存（小于一兆字节）。在这样的环境中，对于工程师而言，为了节省空间，每一个位都是非凡的长度，不容浪费。同时，处理器速度也是一个严重的问题。大多数人在前一天下午把程序留给系统操作员，等到第二天再回来取结果。因此，当时工程师们不得不认真平衡内存使用及需求，并且尽量减少消耗的 CPU 周期数。

当然，这种情况已经得到改变。1998 年，我负责的 Windows NT 工作站有两个 450MHz 奔腾处理器，2GB 内存。CDC 6600 售价为 700 万美元。而我的 NT 工作站的成本不足 5000 美元。今天的工程师不再需要面对从程序中挤压每盎司性能的压力（我几乎可以听到 CDC 工程师发牢骚，“想当年……”）。说穿了，我们可以变得懒惰，是因为硬件在帮助我们弥补空缺。如果程序运行得不够快，我们可以通过更多的硬件来解决，随着廉价配件的不断涌现，在短期内，这是一个现实的解决方案。

但总有走到尽头的一天。物理定律要求电子的电路通路大于 3 个氢原子。一旦这条通路变得比这更窄，电子就不再表现为基本粒子，而变得像逃犯一样乱窜（即出现量子隧穿）。这意味着计算机处理器的收缩尺度是有极限的。终有一天，制造商不再有免费的午餐吃。在某个时刻，为了提高计算能力，势必需要让处理器变得更大，以容纳更多的晶体管。

真到了那一天，提高软件性能的重担又将重新压到软件工程师的肩上。需要发明更好的算法及其实现方式。发生在 20 世纪 60 年代的优化工作将重新出现在下一代推动应用程序性能的急先锋身上。

注意

究竟什么时候我们会遇上这个瓶颈？我确信大多数芯片提供商并不期望那样。1989 年，英特尔发布了 80486，它的设计尺度是一微米（百万分之一米）。炭疽病菌的长度大约是 1 ～ 6 微米。人的头发的直径大约是 100 微米。据戈登·摩尔的观察，给定区域内晶体管的数量每 18 个月就会增加一倍，这就是众所周知的摩尔定律。换句话说，晶体管的设计尺度每 3 年就会减半。如果把 1989 年作为一个起点，当时晶体管的设计尺度为 1 微米，那么你应该能够通过简单计算得到，这样的演进在 2022 年将会结束。即使 CPU 厂商拿得出来一些东西，我怀疑在 2100 年后，CPU 也不可能持续地收缩变小。Zarathustra 如是说。

在铁器时代，调试工作常常伴随着读取十六进制转储日志和重建栈帧。找 bug 是痛苦费时的工作。因此，工程师们提出了精确的策略，以防止 bug 发生及在它们出现时能够快速地修复它们。今天，大多数工程师只需在代码中放置断点，通过 GUI 调试器单步调试自己的程序就可以了。GUI 调试器和霰弹式修改（shotgun surgery）已经取代了老一辈工程师们引以为豪的探测技能。

我并不是说 GUI 调试器不好，事实上，我本人也非常喜欢它。我只是说，有时仅凭 GUI 调试

器无法发现问题的根源。在这些情况下，必须通过系统的学习和实践磨炼出来的诊断技能才能解决问题。除了 GUI 调试器，可以使用这些技能来解决难题。

有效地使用工具并没有坏处，除非你完全依赖于它们。这让我想起艾萨克·阿西莫夫（Issac Asimov）写的一个叫做“电的感觉”的故事。在这个故事中，后代变得如此依赖于计算机，以至于完全忘记了如何执行基本的算术运算。在故事的结尾，主角之一在他的头脑中进行乘法运算：

9 乘以 7，舒曼深感满意，是 63，我不需要一台电脑来告诉我这个结果了。我所拥有的头脑就是一台计算器，这种感觉真是太惊人了。

随着时间的推移，软件应用程序变得越来越大。回到 1981 年，第一版的 PC DOS 大约有 4000 行汇编代码[⊖]。快进到 20 世纪 90 年代中期，Windows NT 4.0 超过了 1600 万行代码[⊖]。21 世纪，我们可能会看到达到 10 亿行代码的软件应用程序。虽然现代的调试工具令人印象深刻，但随着软件的复杂度曲线不断地攀升到最高点，当代开发人员所采用的非正规方式并不足以应对。

虽然一般软件工程师的优化和调试技巧已经退步，但随着卓越的硬件和工具不停地出现，称职的工程师依然会花时间去掌握这些被遗忘的艺术。时间和精力的投资将以技能的增长作为回报，而这些技能会使他们成为更好的工程师。没有什么比在凌晨 3:00 被愤怒的客户叫醒更惨的了。有了这本书，你可以免受打扰，安心地睡个好觉。

本书读者对象

根据美国劳工统计局数据，2001 年全美范围内有超过 100 万的软件工程师受雇用。其中大概有半数的工程师做着各种各样的维护工作。因此，这本书针对很大范围的软件从业人员。

维护编程工作没有什么吸引力。它更像是在钢厂工作：繁琐和充满危险。管理层专门高薪招聘的顾问，也只是完成早期工作而已，他会像逃避瘟疫一样地远离维护工作。为什么呢？因为维护工作够糟糕，所以咨询架构师尽力避免它。这份工作相当单调乏味，令人沮丧，平庸至极，毫无成就感可言。这就是为什么他们愿意把这份工作给你这样一个新来的家伙。

一般维护工程师通常会得到几千行代码和一些简单说明，然后被告知需要对其进行一些改进。他们很难从原始开发团队中分得一杯羹，而且往往面临的是更严格的工期。此书献给那些奋战在昏暗的隔间里，喝着隔夜咖啡，默默无闻的维护程序员。他们默默地问自己：“我是如何跑来接手这个烂摊子的？”

世界上的维护工程师们：我感受到了你们的痛苦。

⊖ Andrew Tanenbaum, *Modern Operating Systems*, Second Edition (Prentice Hall, 2001. ISBN: 0-13-031358-0)

⊖ Don Clark, “Windows NT Is Back,” *The Wall Street Journal*, July 29, 1996

本书结构

时至今日，软件开发往往会遵循一系列基本步骤，包括构建、测试、调试和调优。有很多介绍这种一般流程的模型。例如，久负盛名的瀑布模型（参见图 2）就是最早期描述这些开发步骤是如何有序开展的模型。

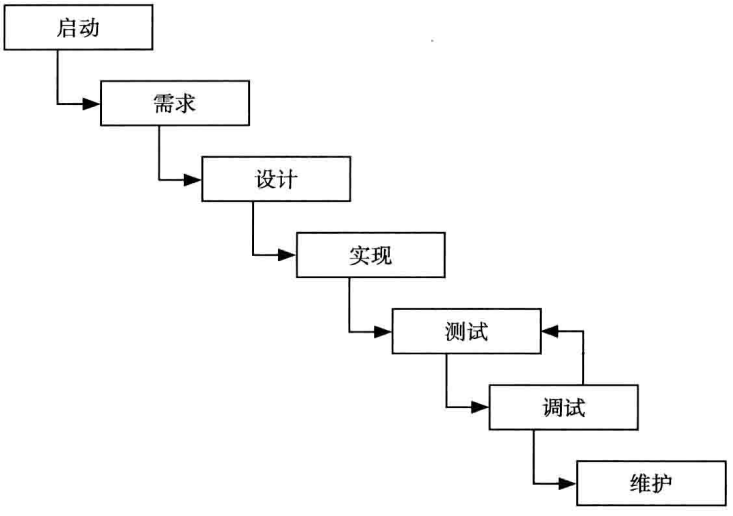


图 2　瀑布开发模型

许多工程师以各种理由嘲笑瀑布模型。在今天的硅谷时代，整个开发过程只有两个步骤，那就是实现和维护，以及由谁来决定程序执行的边界。

本书共分 7 章。我试图通过第 1 章介绍几个技巧，或许这样，你就可以跳过接下来的三章，即我提出建议，指出如何编写 bug 较少的代码。如果你没有获得一个令人羡慕的职位，从头开始编写软件，那么你可以忽略第 1 章。第 2 ～ 4 章重点讨论优雅的调试艺术。我不仅着眼于调试代码的策略，还检查调试器内部是如何运作的。

除了调试之外，大部分维护工程师将工作时间花在了提高程序性能的调优上。第 5 章和第 6 章将专门解释如何更有效地利用计算机的资源。本书最后一章会给出一些来之不易的忠告。

下面给出每个章节更详细的介绍。

第 1 章：预防药

程序的原作者几乎总是对大多数 bug 负责。那些将一个软件从无到有构建起来的工程师，具有特权地位，他们有唯一的机会制定约定，以尽量减少在他们的创作中嵌入 bug 的数量。该章重点讨论那些可以用于构建易于修改及调试的软件的相关技术。

第 2 章：调试技巧

该章介绍如何按步骤来定位和消除软件 bug。开始将讨论如何验证你所要处理的问题实际上是一个 bug。接下来，会讨论科学的方法，并解释如何将其应用到 bug 的处理中。坚持你的工作，并为后续工程师提供跟踪指南，该章着眼于可用于跟踪维护工作的方法。

第 3 章：理解问题

第 2 章提到，理解你负责处理的问题以及程序是先决条件。但当你面对 50 000 行神秘的 K&R C 代码的时候，是如何“理解问题”的呢？这可是价值 6.4 万美元的问题。该章提供了一些久经考验的方法，你可以用它来解决这个问题。努力工作没有错，但是，你可以采取更轻松的工作方法。我只能说，在我自己的防御体系里，真正做到这一点比展开这个话题要困难得多。

第 4 章：调试器内部机制

了解调试器在计算机内部的工作原理，并不是成功使用调试器的要求。但是，可能会有一些好奇的读者渴望了解调试器的操作原理。该章专门解释调试器操作的本质。该章开始介绍一些基本功能，如断点和单步执行，然后逐渐讨论更高级的功能。该章结束部分将讨论预防程序被反向工程的一些技术。

第 5 章：优化：内存占用

计算机有两个重要资源：内存和 CPU 周期。成功的优化取决于既能够减少应用程序的内存占用又能够更有效地利用处理器周期。这是一个非常微妙的平衡。该章将着眼于前者，给出减少应用程序内存使用量的相关技术，讨论程序的所有标准内存组件，包括代码段、数据段、栈和堆。

第 6 章：优化：CPU 周期

该章延续第 5 章的话题。具体而言，第 6 章介绍了许多方法，以确定哪些程序浪费了处理器周期，并对每个实例提供了解决方案。该章首先分析基本程序的控制结构，如循环和分支语句，然后再转换到更高级的主题，如异常处理和内存管理。

第 7 章：最后的赠言

有些东西在学校里教授也不会提醒你，主要是因为他们无法做到这一点：他们已经在学术上花费了太多的时间。然而，已经有一些人离开如茧般安全的大学，勇于面对软件行业中的各种元素。该章将告诉你前辈们的经验之谈。

前提条件

这本书中的例子主要是用 ANSI C、C++ 和 x86 汇编语言组合实现的。使用 C / C++ 和汇编程序不仅是要最大限度地吸引程序员，还希望能够提供足够的洞察力。要知道为什么某些策略是有效的，需要花时间查看编译器在幕后都做了什么。要做到这一点，最好的办法是查看编译器生成的汇编代码。我所使用的汇编代码相当普通，大多数工程师都能够很轻松地阅读我的代码。

起初，我想过用 Java 来实现这些例子。然而，我发现，C 与 C++ 的灵活性为滥用和误用提供了更大的机会。蜘蛛侠曾说过什么？能力越强，责任越大？ C 和 C++ 是功能强大的语言，因此，用 C/C++ 很容易制造烂摊子。内存泄漏和悬挂指针已经困扰 C 程序员几十年了，更不用说预处理器技术和任意级别的间接法。总之，C 与 C++ 给调试和优化的讨论提供了沃土。

我经常听到关于 C++ 和 Java 的相对价值的辩论。就我而言，这两种语言是用于不同工作的不同工具。这就像是在问："铅笔或钢笔，哪一个更好？"这两种语言都是面向对象的，它们之间的主要区别在于它们面向的方向。具体来说，Java 是一种应用程序语言 ，而 C++ 是一种系统语言 。

Java 程序经过编译之后在虚拟机上运行。"编写一次，到处运行" 是用 Java 实现项目的根本好处。由于这个特性，这门语言已经在各个软件公司中变得非常流行，如争取软件跨平台可移植性的 IBM。这样做的缺点是，你不能直接与原生硬件进行交互。通过争取可移植性，Java 已经与主机独立开来。

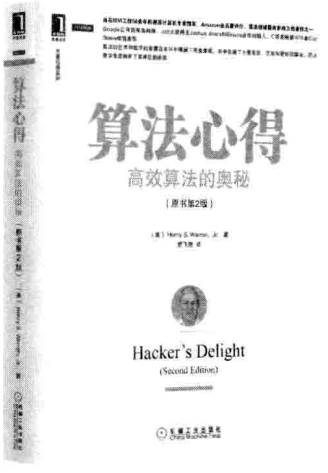
同样，构建系统软件要求你需要具备不同的能力，比如插入内联汇编代码，显式地操作内存，并生成一个原生可执行文件。只是恰巧，C++ 提供了这些功能。这毫不奇怪，然后，当前的绝大多数操作系统都是用 C 和 C++ 组合实现的。当涉及处理中断及与硬件通信时，总是不可避免地有汇编代码参与其中。C 和 C++ 中的高级结构和汇编代码交融，使汇编级例程可以隐藏在系统深处。

致谢

写一本书，几乎就如同怀孕一样。几个月下来，你会感到睡眠缺失，身心疲惫，以至于你会产生奇怪的情绪波动和对食物的渴望。当你最终完成时，你已是筋疲力尽，为苦难的结束而喜出望外。然而，即使经历过第一次磨难，人们也还是会去做这样一件事。我认为，写一本书必须要求作者有持之以恒的坚强品质。

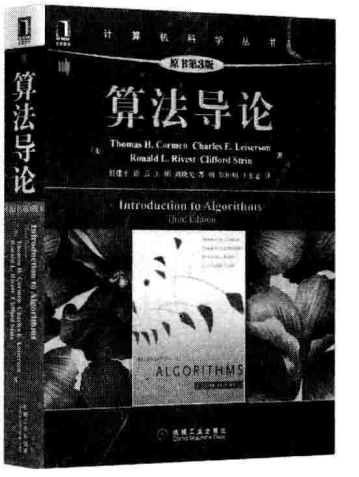
我要感谢 Apress 出版社所有鼓励我的人，他们容忍我所有的 "胡作非为"。特别地，我要感谢 Gary Cornell 让我有机会为 Apress 出版社写书。我还要感谢 Jim Sumser 和 Hollie Fischer 能够忍受我的长篇唠叨，并对可疑之处提供反馈。最后，我要感谢 Ami Knox、Kari Brooks、Beth Christmas 和 Jessica Dolcourt 在整个写作、出版过程中的帮助。

推荐阅读



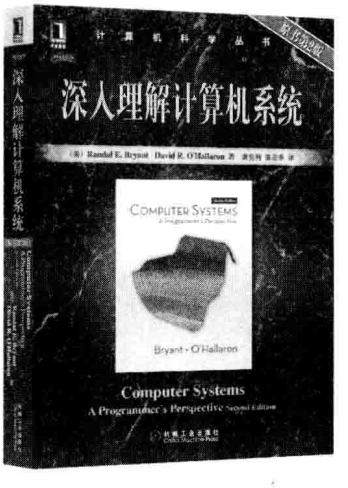
算法心得：高效算法的奥秘（原书第2版）

作者：Henry S. Warren ISBN：978-7-111-45356-7 定价：89.00元



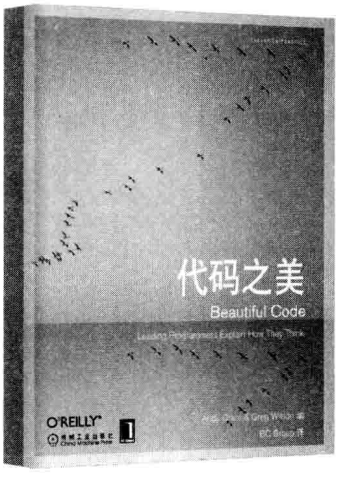
算法导论（原书第3版）

作者：Thomas H. Cormen 等 ISBN：978-7-111-40701-0 定价：128.00元



深入理解计算机系统（原书第2版）

作者：Randal E. Bryant 等 ISBN：978-7-111-32133-0 定价：99.00元



代码之美

作者：Grey Wilson ISBN：978-7-111-25133-0 定价：99.00元

目　录

第 1 章
预 防 药

概要

老实说，相比那些在事后马虎的人而言，对那些在一开始就马马虎虎的人我会毫不犹豫地将其淘汰掉。这听起来相当冷酷无情，这的确冷酷无情。但是，这并不是那种“如果你无法忍受炙热，那就远离厨房”的提醒，它更深刻。我不愿与那些马马虎虎的人共事。这是软件开发的进化论。

—— Linus Torvalds 在 Linux 内核邮件列表里关于内核调试器的讨论中如是说

维护工程师的角色是驱逐那些潜藏在遗留软件里的邪恶幽灵，他们夜以继日维护着那些软件，上下交困，怨声载道，就像在穿过公司的那条小径上孤独前行。每天，他们都面临诸如：

- 修复 bug
- 性能提升

这样的挑战。本书前面 4 章会重点关注于前者。最后两章会关注后者。