程序员必读书单 1.0

2015年 2月25日 | 评论

作者：Lucida

- 微博：@peng_gong
- 豆瓣：@figure9

原文链接：http://lucida.me/blog/developer-reading-list/

本文把程序员所需掌握的关键知识总结为三大类19个关键概念，然后给出了掌握每个关键概念所需的入门书籍，必读书籍，以及延伸阅读。旨在成为最好最全面的程序员必读书单。

前言

> *Reading makes a full man; conference a ready man; and writing an exact man.*
>
> *Francis Bacon*

优秀的程序员应该具备两方面能力：

- 良好的程序设计能力：
  - 掌握常用的数据结构和算法（例如链表，栈，堆，队列，排序和散列）；
  - 理解计算机科学的核心概念（例如计算机系统结构、操作系统、编译原理和计算机网络）；
  - 熟悉至少两门以上编程语言（例如C++，Java，C#，和Python）；
- 专业的软件开发素养：
  - 具备良好的编程实践，能够编写可测试（Testable），可扩展（Extensible），可维护（Maintainable）的代码；
  - 把握客户需求，按时交付客户所需要的软件产品；
  - 理解现代软件开发过程中的核心概念（例如面向对象程序设计，测试驱动开发，持续集成，和持续交付等等）。

和其它能力一样，程序设计能力和软件开发素养源自项目经验和书本知识。项目经验因人而异（来自不同领域的程序员，项目差异会很大）；但书本知识是相通的——尤其是经典图书，它们都能够拓宽程序员的视野，提高程序员的成长速度。

在过去几年的学习和工作中，我阅读了大量的程序设计/软件开发书籍。随着阅读量的增长，我意识到：

- 经典书籍需要不断被重读——每一次重读都会有新的体会；
- 书籍并非读的越多越好——大多数书籍只是经典书籍中的概念延伸（有时甚至是照搬）；

意识到这两点之后，我开始思考一个很功利的问题：**如何从尽可能少的书中，获取尽可能多的关键知识？**换句话说：

- 优秀的程序员应该掌握哪些关键概念？
- 哪些书籍来可以帮助程序员掌握这些关键概念？

这即是这篇文章的出发点——我试图通过程序员必读书单这篇文章来回答上面两个问题。

标准

进入必读书单之前，我先介绍下书单里的书籍选择标准和领域选择标准。当然你也点击这里直接跳转到书单开始阅读。

书籍选择标准

1. **必读**：什么是必读书籍呢？如果学习某项技术有一本书无论如何都不能错过，那么这本书就是必读书籍——例如Effective Java于Java，CLR via C#于C#；
    - 注意我没有使用“经典”这个词，因为**经典**计算机书籍往往和**计算机科学**联系在一起，而且经典往往需要10年甚至更长的时间进行考验；
2. **注重实践，而非理论**：所以这个书单不会包含过于原理性的书籍；
3. **入门—必读—延伸**：必读书籍的问题在于：1. 大多不适合入门；2. 不够全面。考虑到没有入门阅读和延伸阅读的阅读列表是不完整的——所以书单中每个关键概念都会由一本入门书籍，一本必读书籍（有时入门书籍和必读书籍是同一本），和若干延伸阅读书籍所构成。

概念选择标准

1. **全面**：全面覆盖软件开发中重要的概念；
2. **通用**：适用于每一个程序员，和领域特定方向无关；
3. **注重基础，但不过于深入**：优秀的程序员需要**良好**的计算机科学基础，但程序员并没必要掌握**过于深入**的计算机科学知识。以算法为例，每个程序员都应该掌握排序、链表、栈以及队列这些基本数据结构和算法，但计算几何、线性规划和网络流这些算法可能就不是每个程序员都需要掌握的了；

通过这几个标准，我把程序员应掌握的关键概念分为程序设计，软件开发，以及个人成长三大类，每一大类均由若干关键概念组成。

快速通道

自从开博以来，经常会有朋友在论坛，微博，和QQ上提问学习X技术读什么书合适（例如：学习Java读什么书合适？如何学习程序设计？）所以我在这里列出了一个“快速通道”——把常见的问题集中在一起，点击问题，即可直接进入答案。（当然，如果你把本文从头读到尾帮助会更大 :–)）

- 如何学习计算机基础知识？
- 如何学习C语言？
- 如何学习C++？
- 如何学习Java？
- 如何学习C#？
- 如何学习JavaScript？
- 如何学习Python？
- 如何加深对编程语言的理解？
- 如何学习程序设计技巧？
- 如何学习算法？
- 如何高效的调试程序？
- 如何掌握良好的编程实践？
- 如何学习面向对象程序设计？
- 如何对代码进行重构？
- 如何更好的进行软件测试？
- 如何管理软件团队/软件项目？
- 如何成为一名更专业的程序员？
- 程序员如何学习设计？
- 程序员如何进行职业规划？
- 如何提高自己的思维能力？
- 如何进行高效求职面试？
- 如何提高自己的英语写作能力？

程序员必读书单

入门书籍

程序设计：

1. 基础理论：编码：隐匿在计算机软硬件背后的语言
2. 编程语言：
    - C：C和指针
    - C++：C++程序设计原理与实践
    - Java：Java核心技术（第9版）
    - C#：精通C#（第6版）
    - JavaScript：JavaScript DOM编程艺术（第2版）

    - Python：Python基础教程（第二版）
3. 编程语言理论：编程语言实现模式
4. 程序设计：程序设计方法
5. 算法与数据结构：算法（第4版）
6. 程序调试：调试九法——软硬件错误的排查之道

软件开发：

1. 编程实践：程序设计实践
2. 面向对象程序设计：Head First设计模式
3. 重构：重构
4. 软件测试：How to Break Software
5. 项目管理：极客与团队
6. 专业开发：程序员修炼之道：从小工到专家
7. 大师之言：奇思妙想：15位计算机天才及其重大发现
8. 界面设计：写给大家看的设计书
9. 交互设计：通用设计法则

个人成长：

1. 职业规划：软件开发者路线图
2. 思维方式：程序员的思维修炼：开发认知潜能的九堂课
3. 求职面试：金领简历：敲开苹果微软谷歌的大门
4. 英语写作：The Only Grammar Book You'll Ever Need

必读书籍

程序设计：

1. 基础理论：深入理解计算机系统（第2版）
2. 编程语言：
    - C：C程序设计语言（第2版）
    - C++：C++程序设计语言（第4版）
    - Java：Effective Java（第2版）
    - C#：CLR via C#（第4版）
    - JavaScript：JavaScript语言精粹
    - Python：Python参考手册（第4版）
3. 编程语言理论：程序设计语言——实践之路（第3版）
4. 程序设计：计算机程序的构造与解释（第2版）
5. 算法与数据结构：编程珠玑（第2版）
6. 程序调试：调试九法——软硬件错误的排查之道

软件开发：

1. 编程实践：代码大全（第2版）
2. 面向对象程序设计：设计模式
3. 重构：修改代码的艺术
4. 软件测试：xUnit Test Patterns
5. 项目管理：人月神话
6. 专业开发：程序员职业素养
7. 大师之言：编程人生：15位软件先驱访谈录
8. 界面设计：认知与设计：理解UI设计准则（第2版）
9. 交互设计：交互设计精髓（第3版）

个人成长：

1. 职业规划：软件开发者路线图
2. 思维方式：如何把事情做到最好
3. 求职面试：程序员面试金典（第5版）
4. 英语写作：风格的要素

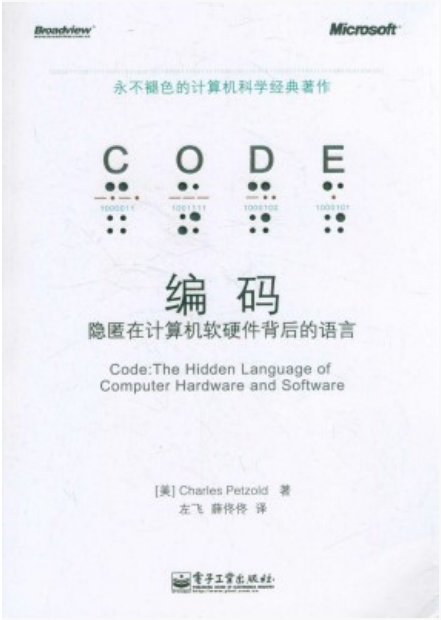
这个阅读列表覆盖了软件开发各个关键领域的入门书籍和必读书籍，我相信它可以满足绝大多数程序员的需求，无论你是初学者，还是进阶者，都可以从中获益：

- 基础理论包括了程序员应该掌握的计算机基础知识；
- 编程语言对软件开发至关重要，我选择了C，C++，Java，C#，Python，和JavaScript这六门主流编程语言进行介绍，如果想进一步理解编程语言，可以阅读编程语言理论里的书目；
- 在理解编程语言的基础上，优秀的程序员还应该了解各种程序设计技巧，熟悉基本的算法数据结构，并且能够高效的进行程序调试。
- 良好的程序设计能力是成为优秀程序员的前提，但软件开发知识也是必不可少的：优秀的程序员应具备良好的编程实践，知道如何利用面向对象，重构，和软件测试编写可复用，可扩展，可维护的代码，并具备软件项目管理知识和专业开发素养；
- 就像我们可以从名人传记里学习名人的成功经验，程序员也可以通过追随优秀程序员的足迹使自己少走弯路。大师之言包含一系列对大师程序员/计算机科学家的访谈，任何程序员都可以从中获益良多；
- 为了打造用户满意的软件产品，程序员应当掌握一定的界面设计知识和交互设计知识（是的，这些工作应该交给UI和UX，但如果你想独自打造一个产品呢？）；
- 专业程序员应当对自己进行职业规划，并熟悉程序员求职面试的流程，以便在职业道路上越走越远；
- 软件开发是一项需要不断学习的技能，学习思维方式可以有效的提升学习能力和学习效率；
- 软件开发是一项国际化的工作，为了让更多的人了解你的代码（工作），良好的英语写作能力必不可少。

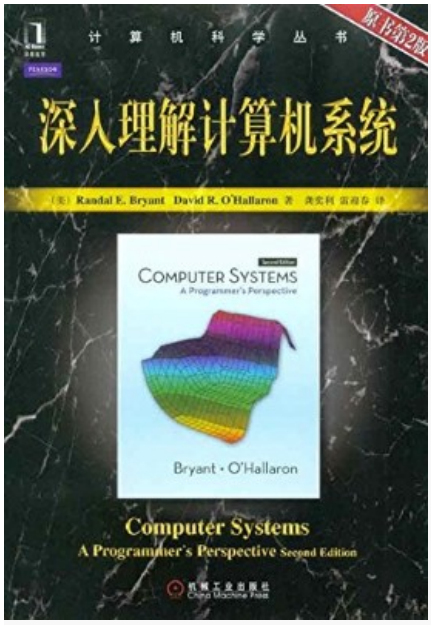
尽管我尽可能的去完善这个书单，但受限于我的个人经历，这个书单难免会有所偏颇。所以如果你有不同的意见，或者认为这个书单漏掉了某些重要书籍，请在评论中指出，我会及时更新。:–)

程序设计

1. 基础理论



编码：隐匿在计算机软硬件背后的语言这本书其实不应该叫编码——它更应该叫“Petzold教你造计算机”——作者Charles Petzold创造性的以编码为主题，从电报机和手电筒讲到数字电路，然后利用数字电路中的逻辑门构造出加法器和触发器，最后构造出一个完整的存储程序计算机。不要被这些电路概念吓到——编码使用大量形象贴切的类比简化了这些概念，使其成为最精彩最通俗易懂的计算机入门读物。



深入理解计算机系统（第2版）这本书的全名是：Computer Systems：A Programmer's Perspective（所以它又被称为CSAPP），我个人习惯把它翻译为程序员所需了解的计算机系统知识，尽管土了些，但更名副其实。

深入理解计算机系统是我读过的最优秀的计算机系统导论型作品，它创造性的把操作系统，计算机组成结构，数字电路，以及编译原理这些计算机基础学科中的核心概念汇集在一起，从而覆盖了指令集体系架构，汇编语言，代码优化，计算机存储体系架构，链接，装载，进程，以及虚拟内存这些程序员所需了解的关键计算机系统知识。如果想打下扎实的计算机基础又不想把操作系统计算机结构编译原理这些书统统读一遍，阅读深入理解计算机系统是最有效率的方式。

延伸阅读：

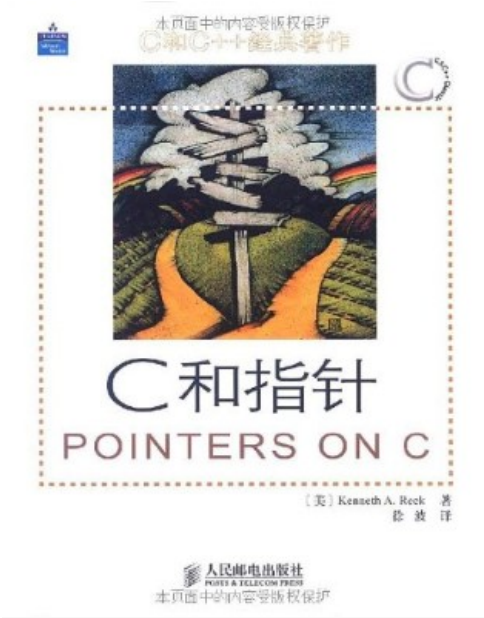
- 世界是数字的：K&R中的K（Brian Kernighan）的近作，这本书源自Brian在普林斯顿大学所教授的计算机基础课程，以通俗易懂的方式讲述了现代人所应了解的计算机知识和网络知识；
- 图灵的秘密：他的生平、思想及论文解读：Charles Petzold的另一部作品，这本书以图灵的论文**论可计算数及其在判定问题上的应用**（On Computable Numbers, with an Application to the Entscheidungsproblem）为主题，阐述了图灵机（现代计算机的始祖）的构造，原理，以及应用。
- 计算机系统概论（第2版）：另一部优秀的计算机系统导论型作品，和深入理解计算机系统不同，这本书采用自下而上的方式，从二进制，和数字逻辑这些底层知识一步步过渡到高级编程语言（C），从而以另一种方式理解计算机系统。

2. 编程语言

编程语言是程序员必不可少的日常工具。工欲善其事，必先利其器。我在这里给出了C，C++，Java，C#，JavaScript，和Python这六种常用编程语言的书单（我个人不熟悉Objective-C和PHP，因此它们不在其中）。

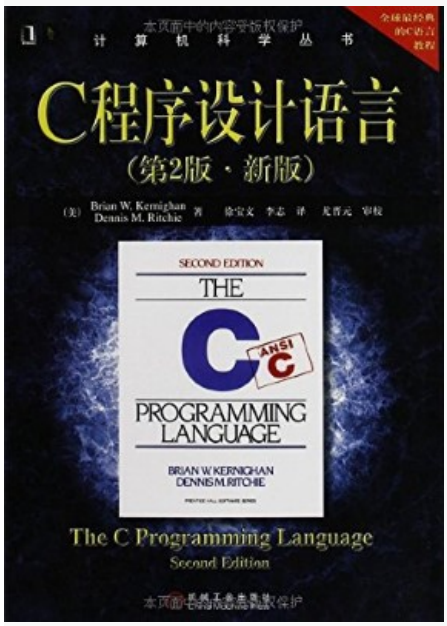
需要注意的是：我在这里给出的是编程语言（Programming Language）书籍，而非编程平台（Programming Platform）书籍。以Java为例，Effective Java属于编程语言书籍，而Android编程权威指南就属于编程平台书籍。

C



忘记谭浩强那本糟糕不堪的C程序设计，C和指针才是C语言的最佳入门书籍。它详细但又不失简练的介绍了C语言以及C标准库的方方面面。

对于C语言初学者，最难的概念不仅仅是指针和数组，还有指向数组的指针和指向指针的指针。C和指针花了大量的篇幅和图示来把这些难懂但重要的概念讲的清清楚楚，这也是我推荐它作为C语言入门读物的原因。

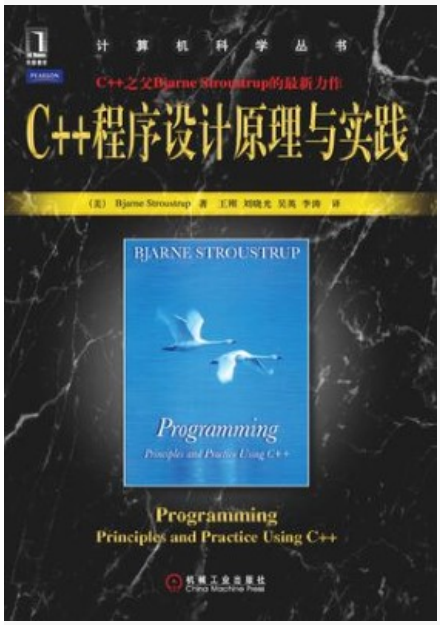


尽管C程序设计语言是二十多年前的书籍，但它仍然是C语言——以及计算机科学中最重要的书籍之一，它的重要性不仅仅在于它用清晰的语言和简练的代码描述了C语言全貌，而且在于它为之后的计算机书籍——尤其是编程语言书籍树立了新的标杆。以至于在很多计算机书籍的扉页，都会有“感谢Kernighan教会我写作”这样的字样。

延伸阅读：

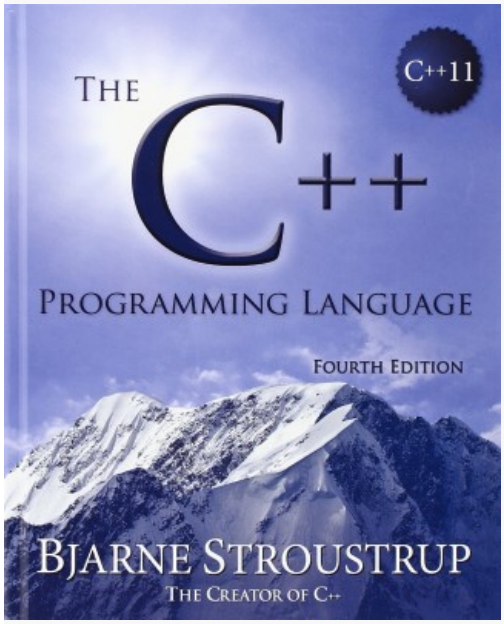
- C专家编程：不要被标题中的“专家”吓到，这实际是一本很轻松的书籍，它既包含了大量C语言技术细节和编程技巧，也包含了很多有趣的编程轶事；
- C陷阱与缺陷：书如其名，这本书介绍了C语言中常见的坑和一些稀奇古怪的编程“技巧”，不少刁钻的C语言面试题都源自这本小册子；
- C语言参考手册：全面且权威的C语言参考手册，而且覆盖C99，如果你打算成为C语言专家，那么这本书不可错过；
- C标准库：给出了15个C标准库的设计思路，实现代码，以及测试代码，配合C程序设计语言阅读效果更佳；
- C语言接口与实现：这本书展示了如何使用C语言实现可复用的数据结构，其中包含大量C语言高级技巧，以至于Amazon上排行第一的评论是“Probably the best advanced C book in existance”，而排行第二的评论则是“By far the most advanced C book I read”。

C++



作为C++的发明者，没有人能比Bjarne Stroustrup更理解C++。Bjarne在Texas A&M大学任教时使用C++为大学新生讲授编程，从而就有了C++程序设计原理与实践这本书——它面向编程初学者，既包含C++教程，也包含大量程序设计原则。它不但是我读过最好的C++入门书，也是我读过最好的编程入门书。

比较有趣的是，C++程序设计原理与实践直到全书过半都没有出现指针，我想这可能是Bjarne为了证明不学C也可以学好C++吧。

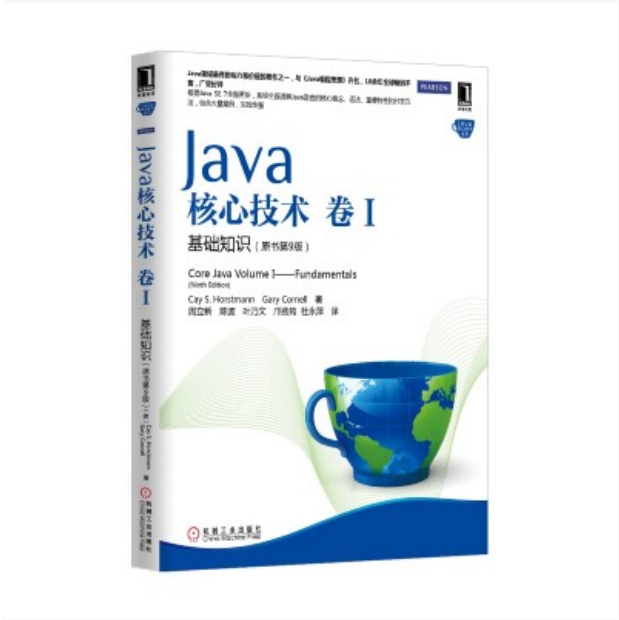


同样是Bjarne Stroustrup的作品，C++程序设计语言是C++**最权威且最全面**的书籍。第4版相对于之前的版本进行了全面的更新，覆盖了第二新的C++ 11标准，并砍掉了部分过时的内容。

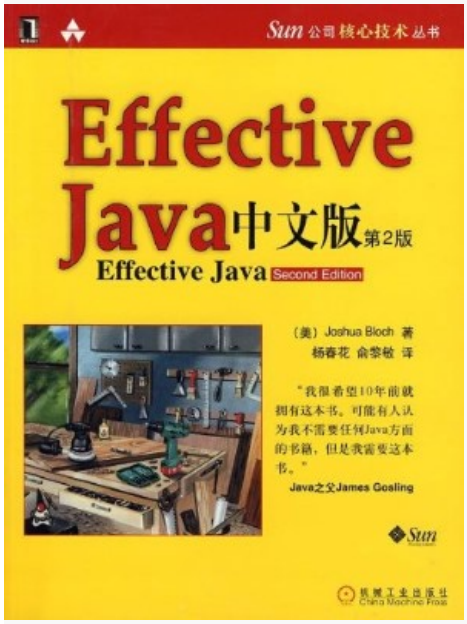
延伸阅读：

- A Tour of C++：如果你觉得C++程序设计语言过于庞大，但你又想快速的浏览一遍新版C++的语言特色，那么可以试试这本小红书；
- C++语言的设计与演化：C++的“历史书”，讲述了C++是如何一步一步从C with Classes走到如今这一步，以及C++语言特性背后的故事；
- C++标准库（第2版）：相对于其它语言的标准库，C++标准库虽然强大，但学习曲线十分陡峭，这本书是学习C++标准库有力的补充；
- 深度探索C++对象模型：这本书系统的讲解了C++是如何以最小的性能代价实现对象模型，很多C++面试题（包括被问烂的虚函数指针）都可以在这本书里找到答案；
- Effective C++和More Effective C++：由于C++的特性实在繁杂，因此很容易就掉到坑里。Effective系列既讲述了C++的良好编程实践，也包含C++的使用误区，从而帮你绕过这些坑。

Java



平心而论Java核心技术（即Core Java）并不算是一本特别出色的书籍：示例代码不够严谨，充斥着很多与C/C++的比较，语言也不够简洁——问题在于Java并没有一本很出色的入门书籍，与同类型的Java编程思想相比，Java核心技术至少做到了废话不多，与时俱进（Java编程思想还停留在Java 6之前），矮子里面选将军，Java核心技术算不错了。



尽管Java没有什么出色的入门书籍，但这不代表Java没有出色的必读书籍。Effective Java是我读过的最好的编程书籍之一，它包含大量的优秀Java编程实践，并对泛型和并发这两个充满陷阱的Java特性给出了充满洞察力的建议，以至于Java之父James Gosling为这本书作序：*“我很希望10年前就拥有这本书。可能有人认为我不需要任何Java方面的书籍，但是我需要这本书。”*

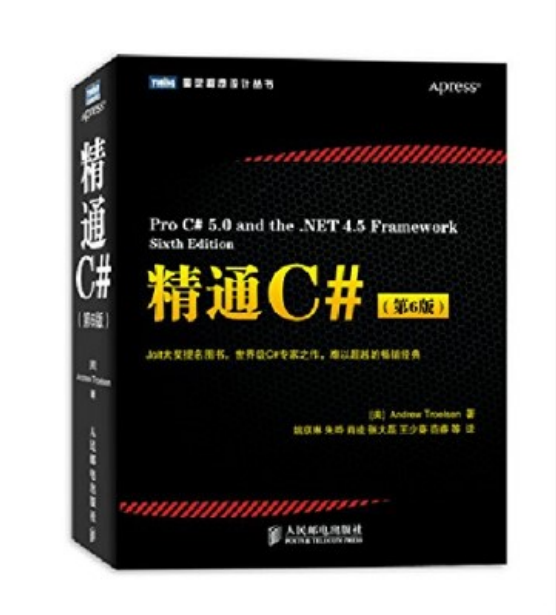
延伸阅读：

- 深入理解Java虚拟机（第2版）：非常优秀且难得的国产佳作，系统的介绍了Java虚拟机和相关工具，并给出了一些调优建议；
- Java程序员修炼之道：在这本书之前，并没有一本Java书籍系统详细的介绍Java 7的新特性（例如新的垃圾收集器，`try using`结构和`invokedynamic`指令），这本书填补了这个空白；
- Java并发编程实践：系统全面的介绍了Java的并发，如何设计支持并发的数据结构，以及如何编写正确的并发程序；
- Java Puzzlers：包含了大量的Java陷阱——以至于读这本书时我说的最多的一个词就是WTF，这本书的意义在于它是一个反模式大全，Effective Java告诉你如何写好的Java程序，而Java Puzzlers则告诉你糟糕的Java程序是什么样子。更有意思的是，这两本书的作者都是Joshua Bloch。

视频教程：

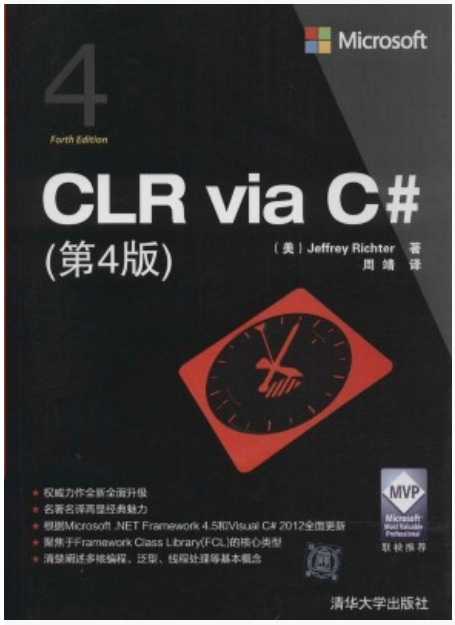
- Java语言学习极速之旅：系统全面的Java语言教程，6个阶段Java基础入门，循序渐进掌握Java面向对象精髓。3个Java进阶方向，Java SE、Java EE、Android开发，每一个都包含相应的知识点精讲和项目开发实例，快速上手。
- Android 0基础极速养成计划：Android开发快速入门视频教程，通过对Android基础知识讲解，详细介绍Android开发环境搭建，同时包含Android实战案例解析，通过全新实战的Android课程，系统学习Android开发。

C#



可能你会疑问我为什么会推荐这本接近1200页的“巨著”用作C#入门，这是我的答案：

1. C#的语言特性非常丰富，很难用简短的篇幅概括这些特性；
2. 精通C#之所以有近1200页的篇幅，是因为它不但全面介绍了C#语言，而且还覆盖了ADO.NET，WCF，WF，WPF，以及ASP.NET这些.Net框架。你可以把这本书视为两本书——一本500多页的C#语言教程和一本600多页的.Net平台框架快速上手手册。
3. 尽管标题带有“精通”两字，精通C#实际上是一本面向初学者的C#书籍，你甚至不需要太多编程知识，就可以读懂它。



CLR via C#是C#/.Net最重要的书籍，没有之一。它全面介绍了.Net的基石——CLR的运行原理，以及构建于CLR之上的C#类型系统，运行时关系，泛型，以及线程/并行等高级内容。任何一个以C#为工作内容的程序员都应该阅读此书。

延伸阅读：

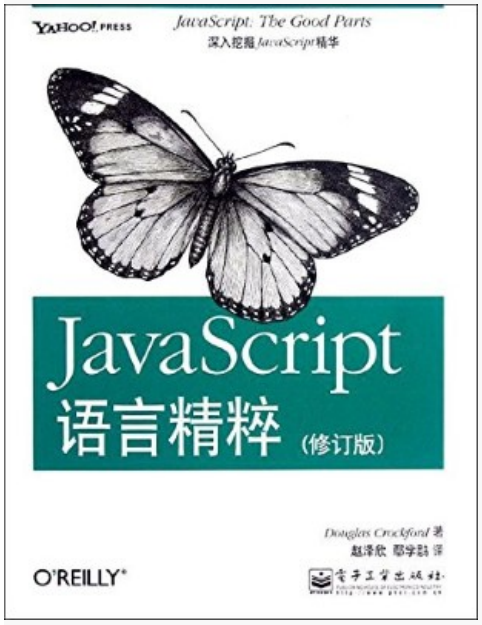
- 深入理解C#（第3版）：C#进阶必读，这本书偏重于C#的语言特性，它系统的介绍了C#从1.0到C#

4.0的语言特性演化，并展示了如何利用C#的语言特性编写优雅的程序；

- .NET设计规范（第2版）：C#专业程序员必读，从变量命名规范讲到类型系统设计原则，这本书提供了一套完整的.Net编程规范，使得程序员可以编写出一致，严谨的代码，
- C# 5.0权威指南：来自O'Reilly的C#参考手册，严谨的介绍了C#语法，使用，以及核心类库，C#程序员案头必备；
- LINQ to Objects Using C# 4.0和Async in C# 5.0：LINQ和`async`分别是.Net 3.5和.Net 4.5中所引入的最重要的语言特性，所以我认为有必要在它们上面花点功夫——这两本书是介绍LINQ和`async`编程的最佳读物。

JavaScript

尽管JavaScript现在可以做到客户端服务器端通吃，尽管JQuery之类的前端框架使得一些人可以不懂JavaScript也可以编程，但我还是认为学习JavaScript从HTML DOM开始最为适合，因为这是JavaScript设计的初衷。JavaScript DOM编程艺术系统的介绍了如何使用JavaScript，HTML，以及CSS创建可用的Web页面，是一本前端入门佳作。



JavaScript语言包含大量的陷阱和误区，但它却又有一些相当不错的特性，这也是为什么Douglas Crockford称JavaScript为世界上最被误解的语言，并编写了JavaScript语言精粹一书来帮助前端开发者绕开JavaScript中的陷阱。和同类书籍不同，JavaScript语言精粹用精炼的语言讲解了JavaScript语言中好的那部分（例如闭包，函数是头等对象，以及对象字面量），并建议读者**不要**使用其它不好的部分（例如混乱的类型转换，默认全局命名空间，以及奇葩的相等判断符），毕竟，用糟糕的特性编写出来的程序往往也是糟糕的。

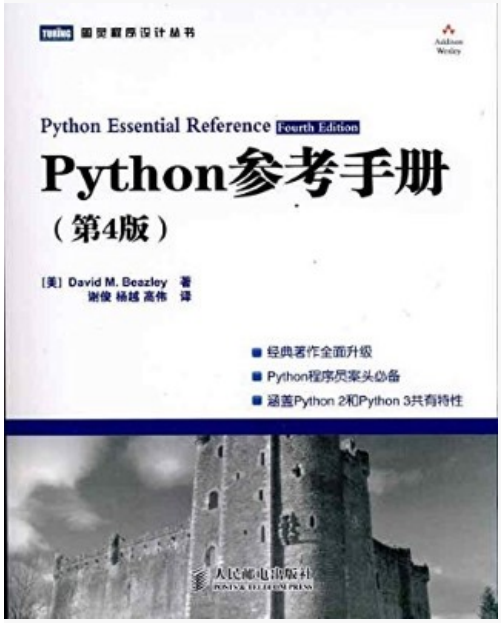
延伸阅读：

- JavaScript高级程序设计（第3版）：详尽且深入的介绍了Javascript语言，DOM，以及Ajax，并针对HTML5做了对应更新；
- JavaScript权威指南（第6版）：这本书的第5版曾被前端专家Douglas Crockford称之为*“唯一靠谱的JavaScript书”*。相对于JavaScript高级程序设计，JavaScript权威指南更像是一本案头参考书，当然如果你感兴趣也可以从头读到尾；
- 编写可维护的JavaScript：书如其名，这本书给出了大量的优秀JavaScript编程实践，使得程序员编写出健壮且易于维护的JavaScript代码；
- JavaScript异步编程：和常见的支持并发的编程语言（例如Java和C#）不同，JavaScript本身是**单线程**的，因此不能把其它语言处理并发的方式照搬到JavaScript。JavaScript异步编程系统的介绍了JavaScript中的并发原理，并阐述了如何使用`Promise`、`Deferred`以及`Async.js`编写出简洁高效的异步程序。

Python

Python的入门书籍很多，而且据说质量大多不错，我推荐Python基础教程的原因是因为它是我的Python入门读物——简洁，全面，代码质量很不错，而且有几个很有趣的课后作业，使得我可以快速上手。

这里顺便多说一句，不要用Python学习手册作为Python入门——它的废话实在太多，你能想象它用了15页的篇幅去讲解`if`语句吗？尽管O'Reilly出了很多经典编程书，但这本Python学习手册绝对不在其中。

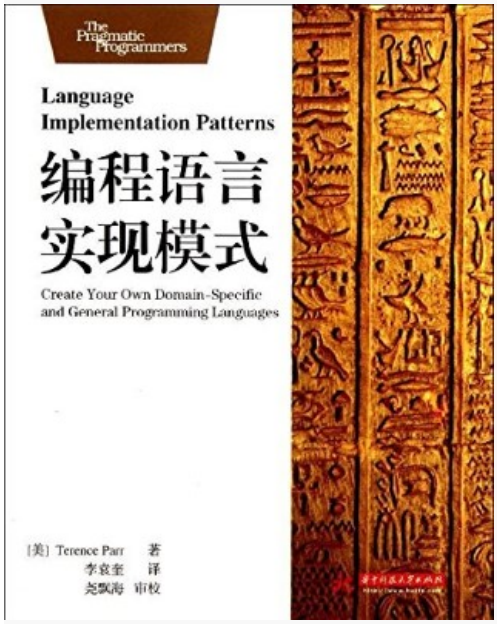


权威且实用的Python书籍，覆盖Python 2和Python 3。尽管它名为参考手册，但Python参考手册在Python语法和标准库基础之上对其实现机制也给出了深入的讲解，不容错过。

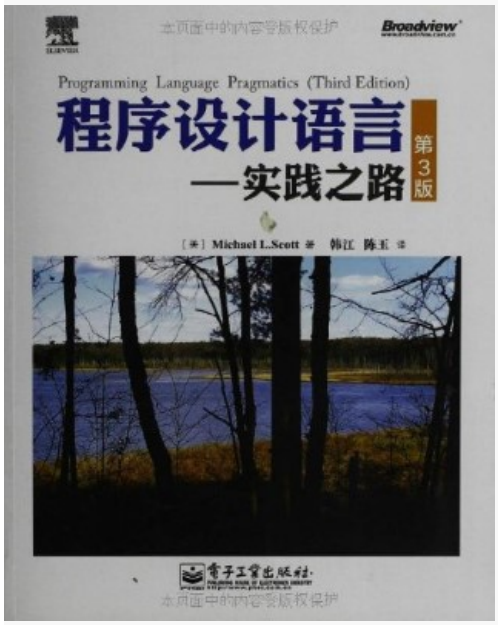
延伸阅读：

- Python袖珍指南（第5版）：实用且便携的Python参考手册，我会说我在飞机上写程序时用的就是它么 -_-#；
- Python Cookbook（第3版）：非常好的Python进阶读物，包含各种常用场景下的Python代码，使得读者可以写出更加Pythonic的代码；
- Python编程实战：运用设计模式、并发和程序库创建高质量程序：Python高级读物，针对Python 3，2014年的Jolt大奖图书，不可错过；
- Python源码剖析：少见的国产精品，这本书以Python 2.5为例，从源代码出发，一步步分析了CPython是如何实现类型，控制流，函数/方法的声明与调用，类型以及装饰器等Python核心概念，读过之后会大大加深对Python的理解。尽管这本书有些过时，但我们仍然可以按照它分析源代码的方式来分析新版Python。

3. 编程语言理论



大多数程序员并不需要从头编写一个编译器或解释器，因此龙书（编译原理）就显得过于重量级；然而多数程序员还是需要解析文本，处理配置文件，或者写一个小语言，编程语言实现模式很好的满足了这个需求。它把常用的文本解析/代码生成方法组织成一个个模式，并为每个模式给出了实例和应用场景。这本书既会提高你的动手能力，也会加深你对编程语言的理解。Python发明者Guido van Rossum甚至为这本书给出了*“Throw away your compiler theory book!”*这样的超高评价。



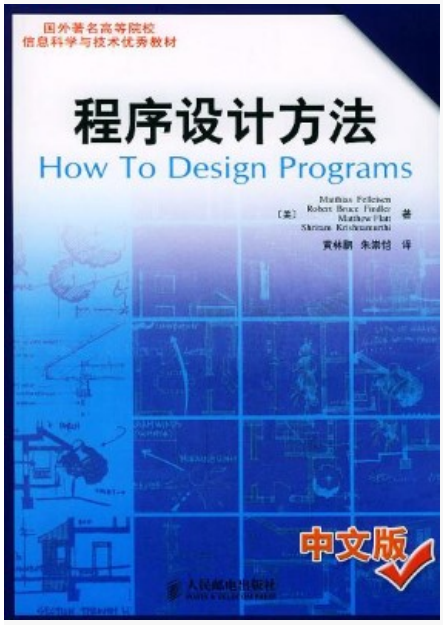
程序员每天都要和编程语言打交道，但是思考编程语言为什么会被设计成这个样子的程序员并不多，程序设计语言——实践之路完美的回答了这个问题。这本书从编程语言的解析和运行开始讲起，系统了介绍了命名空间，作用域，控制流，数据类型以及方法（控制抽象）这些程序设计语言的核心概念，然后展示了这些概念是如何被应用到过程式语言，面向对象语言，函数式语言，脚本式，逻辑编程语言以及并发编程语言这些具有不同编程范式的编程语言之上。这本书或极大的拓宽你的视野——无论你使用什么编程语言，都

会从这本书中获益良多。理解这一本书，胜过学习十门新的编程语言。

延伸阅读：

- 七周七语言：理解多种编程范型：尽管我们在日常工作中可能只使用两三门编程语言，但是了解其它编程语言范式是很重要的。七周七语言一书用精简的篇幅介绍了Ruby，Io，Prolog，Scala，Erlang，Clojure，和Haskell这七种具有不同编程范式的语言——是的，你没法通过这本书变成这七种语言的专家，但你的视野会得到极大的拓宽；
- 自制编程语言：另一本优秀的编译原理作品，自制编程语言通过从零开始制作一门无类型语言`Crowbar`和一门静态类型语言`Diksam`，把类型系统，垃圾回收，和代码生成等编程语言的关键概念讲的清清楚楚；
- 计算的本质：深入剖析程序和计算机：披着Ruby外衣的计算理论入门书籍，使你对编程语言的理解更上一层楼。

4. 程序设计



现代编程语言的语法大多很繁杂，初学者使用这些语言学习编程会导致花大量的时间在编程语言语法（诸如指针，引用和类型定义）而不是程序设计方法（诸如数据抽象和过程抽象）之上。程序设计方法解决了这个问题——它专注于程序设计方法，使得读者无需把大量时间花在编程语言上。这本书还有一个与之配套的教学开发环境DrScheme，这个环境会根据读者的程度变换编程语言的深度，使得读者可以始终把注意力集中在程序设计方法上。

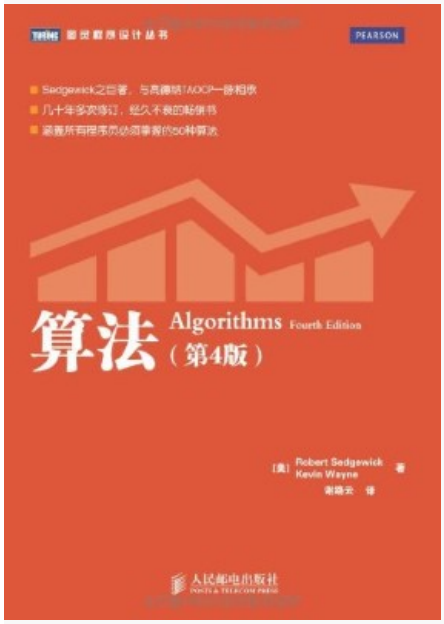
我个人很奇怪程序设计方法这样的佳作为什么会绝版，而谭浩强C语言这样的垃圾却大行其道——好在是程序设计方法第二版已经被免费发布在网上。

计算机程序的构造与解释是另一本被国内大学忽视（至少在我本科时很少有人知道这本书）的教材，这本书和程序设计方法有很多共同点——都使用Scheme)作为教学语言；都专注于程序设计方法而非编程语言本身；都拥有相当出色的课后题。相对于程序设计方法，计算机程序的构造与解释要更加深入程序设计的本质（过程抽象，数据抽象，以及元语言抽象），以至于Google技术总监Peter Norvig给了这本书超高的评价。

延伸阅读：

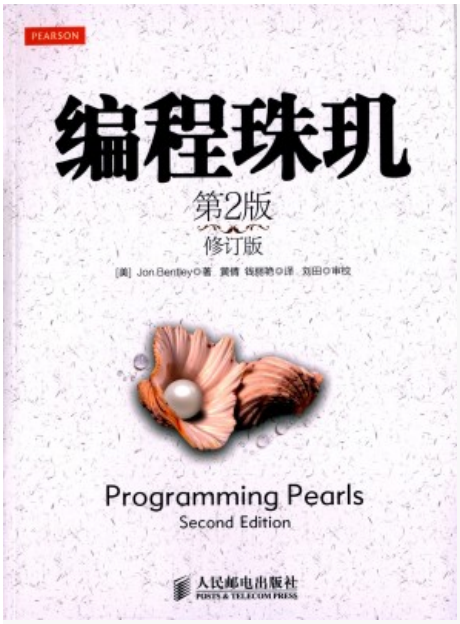
- 编程原本：STL作者的关于程序设计方法佳作——他把关系代数和群论引入编程之中，试图为程序设计提供一个坚实的理论基础，从而构建出更加稳固的软件。这本书是程序设计方法和计算机程序的构造与解释的绝好补充——前者使用函数式语言（Scheme）讲授程序设计，而编程原本则使用命令式语言（C++）；
- 元素模式：设计模式总结了**面向对象程序设计**中的模式，而元素模式这本书分析了**程序设计**中的常见模式的本质，阅读这本书会让你对程序设计有更深的理解；
- The Science of Programming：会编程的人很多，但能够编写正确程序的人就少多了。The Science of Programming通过**前条件——不变式——后条件**以及逻辑谓词演算，为编写正确程序提供了强有力的理论基础，然后这本书通过实例阐述了如何应用这些理论到具体程序上。任何一个想大幅提高开发效率的程序员都应阅读此书。

5. 算法与数据结构



我在算法学习之路一文中提到我的算法入门教材是数据结构与算法分析：C语言描述，我曾经认为它是最好的算法入门教材，但自从我读到Sedgewick的算法之后我就改变了观点——这本算法才是最好的算法入门教材：

- 使用更为容易的Java语言作为教学语言；
- 覆盖所有常用的数据结构和算法，并均给出其完整实现；
- 包含大量的图示用于可视化算法——事实上这是我读过的图示最为丰富形象的书籍，这也是我称其为最好的算法入门书籍的原因。

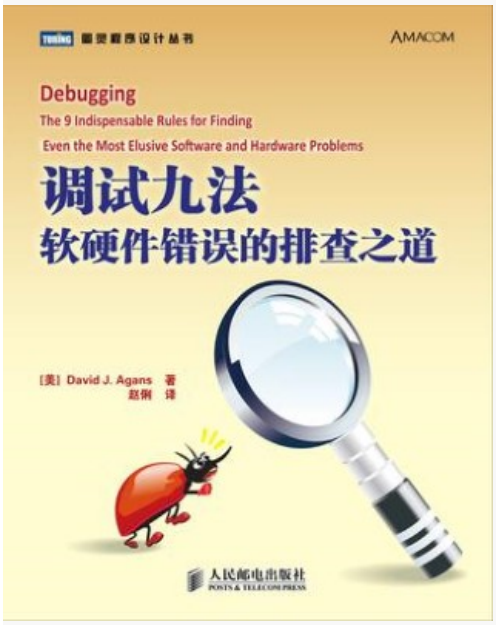


编程珠玑（第2版）是一本少见的实践型算法书籍——它并非一一介绍数据结构/算法的教材，而是实践性极强的算法应用手册。作者（Jon Bentley）从他多年的实际经验精选出一些有趣而又实用的问题，然后展示了他解决这些问题的过程（分析问题，选择合适的算法，解决问题，以及验证答案）。任何程序员都可以从中获益。

延伸阅读：

- 编程珠玑（续）：严格来说这本书并非编程珠玑的续作，而是一本类似于番外篇的编程技巧/实践手册；它不像编程珠玑那般重视算法的应用，而是全面覆盖了程序员所需的能力；
- 算法导论（第3版）：尽管我在这边文章开头提到会尽量避免理论性的书籍，但没有算法导论的算法阅读列表是不完整的，我想这本书就不需要我多介绍了；:–)
- 算法设计与分析基础（第3版）：侧重于算法设计，这本书创新的把常见算法分为分治，减治，变治三大类，并覆盖了动态规划，回溯，以及分支定界等高级算法设计方法，属于算法设计的入门佳作；

6. 程序调试



一个让非编程从业人员惊讶的事实是程序员的绝大多时间都花在调试上，而不是写程序上，以至于Bob大叔把**调试时间占工作时间的比例**作为衡量程序员开发能力的标准。调试九法——软硬件错误的排查之道既是调试领域的入门作品，也是必读经典之作。调试九法的作者是一个具有丰富实战经验的硬件工程师，他把他多年的调试经验总结成九条调试法则，并对每一条法则都给对应的实际案例。任何程序员都应通过阅读这本书改善调试效率，即便是非程序员，也可以从这本书中学到系统解决问题的方法。

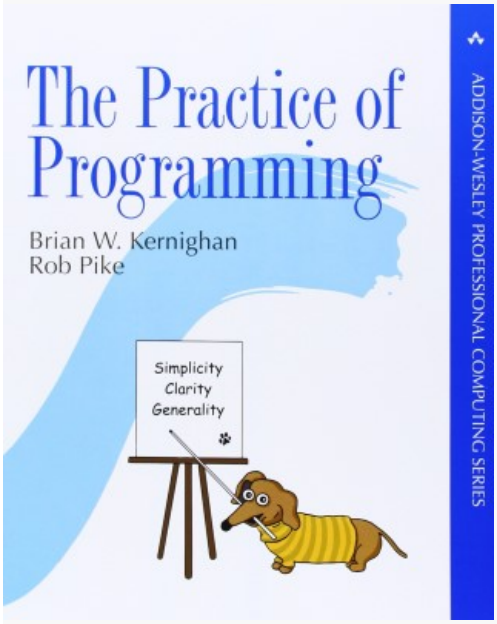
延伸阅读：

- Writing Solid Code：**最好的调试是不调试**——Writing Solid Code介绍了断言，设计清晰的API，以及

单步代码等技巧，用于编写健壮的代码，减少调试的时间；
- 软件调试的艺术：调试工具书——这本书详细的介绍了常见的调试器工具，并通过具体案例展示了它们的使用技巧；

软件开发

1. 编程实践



Brian Kernighan是这个星球上最好的计算机书籍作者：从上古时期的Software Tools，到早期的Unix编程环境和C程序设计语言，再到这本程序设计实践，每本书都是**必读**之作。

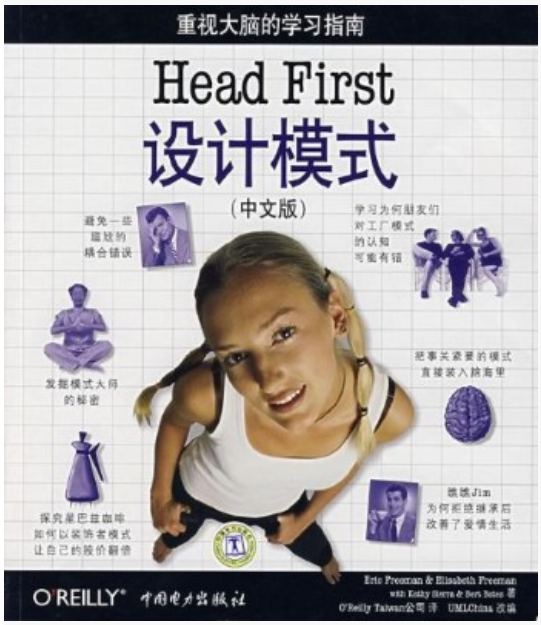
尽管程序设计实践只有短短200余页，但它使用精炼的代码和简要的原则覆盖了程序设计的所有关键概念（包括编程风格，算法与数据结构，API设计，调试，测试，优化，移植，以及领域特定语言等概念）。如果你想快速掌握良好的编程实践，或者你觉着900多页的代码大全过于沉重，那么程序设计实践是你的不二之选。我第一次读这本书就被它简洁的语言和优雅的代码所吸引，以至于读研时我买了三本程序设计实践——一本放在学校实验室，一本放在宿舍，一本随身携带阅读。我想我至少把它读了十遍以上——每一次都有新的收获。

无论在哪个版本的程序员必读书单，代码大全都会高居首位。和其它程序设计书籍不同，代码大全用通俗清晰的语言覆盖了软件构建（Software Construction）中各个层次上**所有**的重要概念——从变量命名到类型设计，从控制循环到代码结构，从测试和调试到构建和集成，代码大全可谓无所不包，你可以把这本书看作为程序员的一站式（Once and for all）阅读手册。更珍贵的是，代码大全在每一章末尾都给出了价值很高的参考书目（参考我之前的如何阅读书籍一文），如果你是一个初出茅庐的程序员，代码大全是绝好的阅读起点。

延伸阅读：

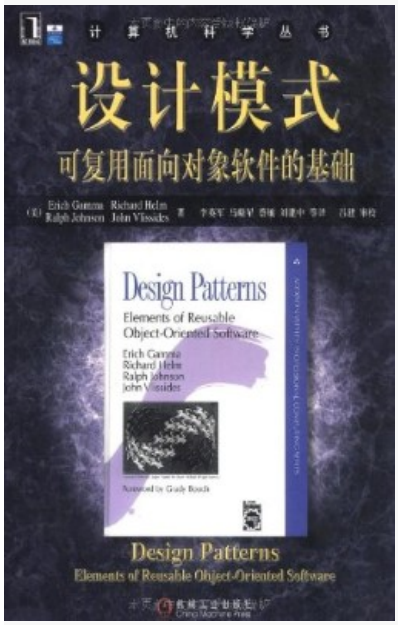
- 编写可读代码的艺术：专注于代码可读性（Code Readability），这本书来自Google的两位工程师对Google Code Readability的总结。它给出了大量命名，注释，代码结构，以及API设计等日常编码的最佳实践，并包含了很多看似细微但却可以显著提升代码可读性的编程技巧。这本书的翻译还不错，但如果你想体会书中的英语幽默（例如Tyrannosaurus——Stegosaurus——Thesaurus），建议阅读它的英文影印版；
- 卓有成效的程序员：专注于生产效率（Productivity），它既包含源自作者多年经验的高生产率原则，也包含大量的提高生产率的小工具，每个追求高生产率的程序员都应该阅读这本书；
- UNIX编程艺术：专注于程序设计哲学，这本书首先总结出包括模块化，清晰化，可组合，可分离等17个Unix程序设计哲学，接下来通过Unix历史以及各种Unix编程工具展示了这些原则的应用。尽管个人觉的这本书有些过度拔高Unix且过度贬低Windows和M$，但书中的Unix设计哲学非常值得借鉴。

2. 面向对象程序设计



无论是在Amazon还是在Google上搜索设计模式相关书籍，Head First设计模式都会排在首位——它使用风趣的语言和诙谐的图示讲述了观察者，装饰者，抽象工厂，和单例等关键设计模式，使得初学者可以迅速的理解并掌握设计模式。Head First设计模式在Amazon上好评如潮，就连设计模式原书作者Erich Gamma都对它给出了很高的评价。

需要注意，Head First设计模式是非常好的设计模式入门书，但**千万不要**把这本书作为学习设计模式的唯一的书——是的，Head First设计模式拥有风趣的语言和诙谐的例子，但它既缺乏**实际**的工程范例，也没有给出设计模式的应用/适用场景。我个人建议是在读过这本书之后立即阅读"四人帮")的设计模式或Bob大叔的敏捷软件开发，以便理解设计模式在实际中的应用。



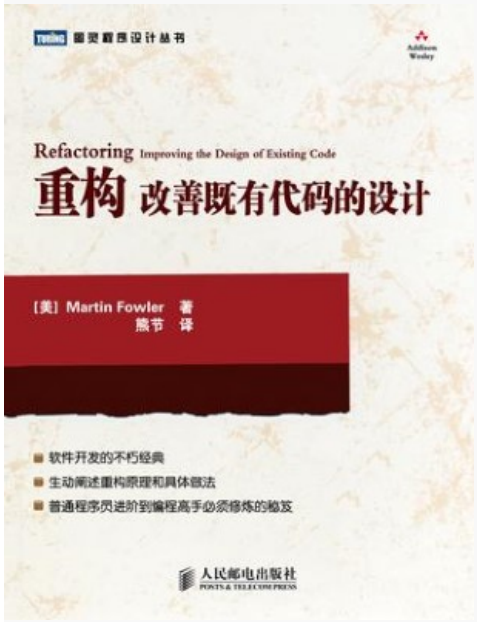
设计模式作为设计模式领域的开山之作，Erich Gamma，Richard Helm，Ralph Johnson等四位作者将各个领域面向对象程序开发的经验总结成三大类23种模式，并给出了每个模式的使用场景，变体，不足，以及如何克服这些不足。这本书行文严谨紧凑（四位作者都是PhD），并且代码源自实际项目，属于设计模式领域的必读之作。

需要注意：设计模式**不适合**初学者阅读——它更像是一篇博士论文而非技术书籍，加上它的范例都具有很强的领域背景（诸如GUI窗口系统和富文本编辑器），缺乏实际经验的程序员很难理解这本书。

延伸阅读：

- 敏捷软件开发（原则模式与实践）：尽管标题带有“敏捷”，但这本书实际是一本面向对象程序设计读物——Bob大叔通过丰富的例子讲解设计模式的应用和SOLID面向对象设计原则)，如果你觉着设计模式过于晦涩，那么你完全可以从这本书开始学习。这本书使用Java作为讲解语言，它也有对应的C#版本；
- 代码整洁之道：同样是Bob大叔的作品，这本书教导读者使用面向对象+敏捷开发原则编写清晰可维护的代码；
- 企业应用架构模式：这本书专注于架构，作者Martin Fowler针对企业应用的特点（诸如持久化数据，多人访问，操作数据的界面以及复杂的业务逻辑），总结出若干企业架构模式，以便程序员构建强大且可扩展的企业应用。

3. 重构



任何产品代码都不是一蹴而就，而是在反复不断的修改中进化而来。重构正是这样一本介绍如何改进代码的书籍——如何在保持代码行为的基础上，提升代码的质量（这也是重构的定义）。

我见过很多程序员，他们经常声称自己在重构代码，但他们实际只做了第二步（提升代码的质量），却没有保证第一步（保持代码行为），因此他们所谓的重构往往会适得其反——破坏现有代码或是引入新bug。这也是我推荐重构这本书的原因——它既介绍糟糕代码的特征（Bad smell）和改进代码的方法，也给出了重构的完整流程——1. 编写单元测试保持（Preserve）程序行为；2. 重构代码；3. 保证单元测试通过。重构还引入了一套重构术语（诸如封装字段，内联方法，和字段上移），以便程序员之间交流。只有理解了这三个方面，才能算是理解重构。

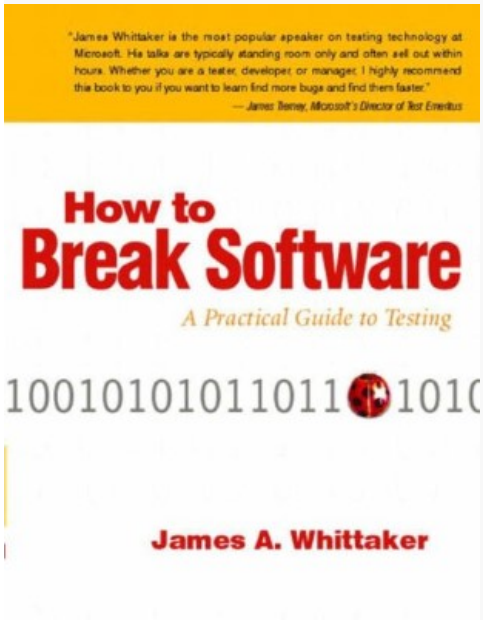
这里再重复一遍重构的定义——**在保持代码行为的基础上，提升代码的质量。**重构专注于第二步，即如何提升代码的质量，而修改代码的艺术专注于第一步，即如何保持代码的行为。

提升代码质量并不困难，但保持代码行为就难多了，尤其是对没有测试的遗留代码（Legacy Code）而言——你需要首先引入测试，但遗留代码往往可测试性（Testability）很差，这时你就需要把代码变的可测试。修改代码的艺术包含大量的实用建议，用来把代码变的可测试（Testable），从而使重构变为可能，使提高代码质量变为可能。

延伸阅读：

- 重构与模式：这本书的中文书名存在误导，它的原书书名是Refactoring to Patterns——通过重构，把模式引入代码。这本书阐述了重构和设计模式之间的关系，使得程序员可以在更高的层次上思考重构，进行重构。

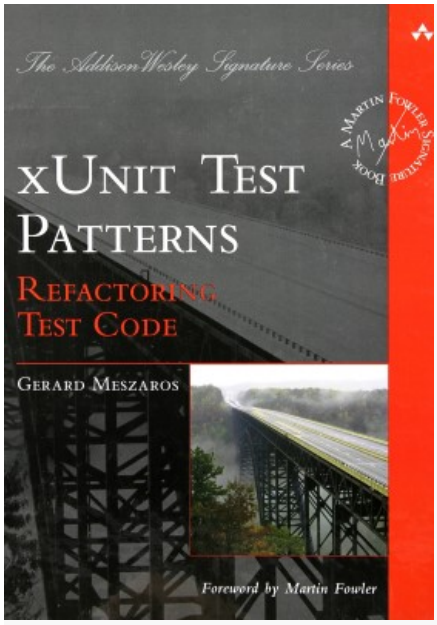
4. 软件测试



关于软件测试的书籍很多，但很少有一本测试书籍能像How to Break Software这般既有趣又实用。不同于传统的软件测试书籍（往往空话连篇，无法直接应用），How to Break Software非常实际——它从程序员的心理出发，分析软件错误/Bug最可能产生的路径，然后针对这些路径进行**残酷**的测试，以保证软件质量。

我在第一次阅读这本书时大呼作者太过“残忍”——连这些刁钻诡异的测试招数都能想出来。但这种毫不留情（Relentless）的测试风格正是每个专业程序员所应具备的心态。

注意：如果你是一个测试工程师，那么在阅读这本书前请三思——因为阅读它之后你会让你身边的程序员苦不堪言，甚至连掐死你的心都有 :-D。

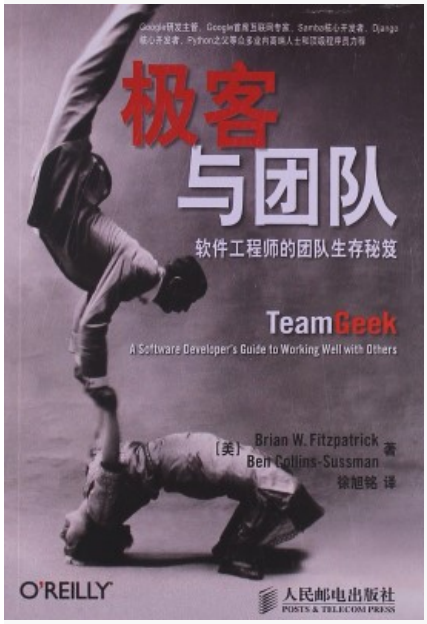


How to Break Software注重黑盒测试，而这本xUnit Test Patterns则注重白盒测试。正如书名所示，xUnit Test Patterns覆盖了单元测试的每个方面：从如何编写良好的单元测试，到如何设计可测试（Testable）的软件，再到如何重构测试——可以把它看作为单元测试的百科全书。

延伸阅读：

- Practical Unit Testing with JUnit and Mockito：尽管xUnit Test Patterns覆盖了单元测试的方方面面，但它的问题在于不够与时俱进（07年出版）。Practical Unit Testing弥补了这个缺陷——它详细介绍了如何通过测试框架JUnit和Mock框架Mockito编写良好的单元测试，并给出了大量优秀单元测试的原则；
- 单元测试的艺术（第2版）：可以把这本书看作为前一本书的.Net版，适合.Net程序员；
- Google软件测试之道：这本书详细介绍了Google如何测试软件——包括Google的软件测试流程以及Google软件测试工程师的日常工作/职业发展。需要注意的是：这本书中的测试流程在国内很可能行不通（国内企业缺乏像Google那般强大的基础设施（Infrastructure）），但它至少可以让国内企业有一个可以效仿的目标；
- 探索式软件测试：James Whittaker的另一本测试著作，不同于传统的黑盒/白盒测试，这本书创造性的把测试比喻为“探索”（Exploration），然后把不同的探索方式对应到不同的测试方式上，以便尽早发现更多的软件错误/Bug。

5. 项目管理



很多程序员都向往成为横扫千军（One-man Army）式的“编程英雄”，但卓越的软件并非一人之力，而是由团队合力而成。极客与团队就是这样一本写给程序员的如何在团队中工作的绝好书籍，它围绕着HRT三大原则（Humility谦逊，Respect尊重，和Trust信任），系统的介绍了如何融入团队，如何打造优秀的团队，

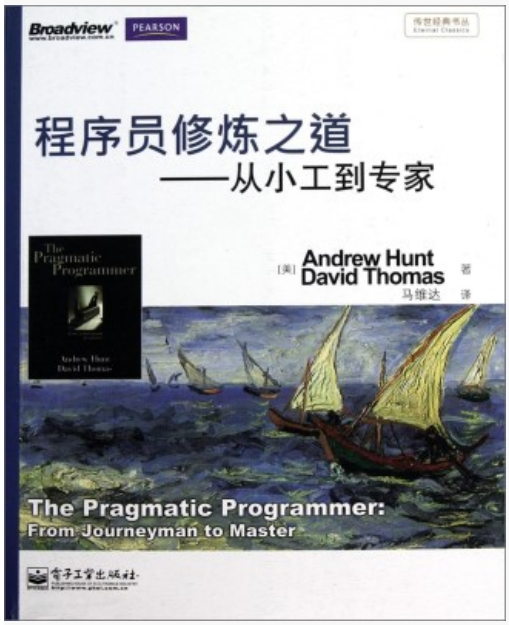
如何领导团队，以及如何应对团队中的害群之马（Poisonous People）。这本书实用性极强，以至于Python之父Guido van Rossum都盛赞这本书*“说出了我一直在做但总结不出来的东西”*。

尽管人月神话成书于40年前，但它仍是软件项目管理最重要的书籍。人月神话源自作者Fred Brooks领导并完成System/360和OS/360这两个即是放到现在也是巨型软件项目的里程碑项目的经验总结。它覆盖了软件项目各个方面的关键概念：从工期管理（Brooks定律）到团队建设（外科团队），从程序设计（编程的本质是使用正确的数据结构）到架构设计（概念完整性），从原型设计（Plan to Throw one away）到团队交流（形式化文档+会议）。令人惊讶的是，即便40年之后，人月神话中的关键概念（包括焦油坑，Brooks定律，概念完整性，外科团队，第二版效应等等）依然适用，而软件开发的核心复杂度仍然没有得到解决（没有银弹）。

延伸阅读：

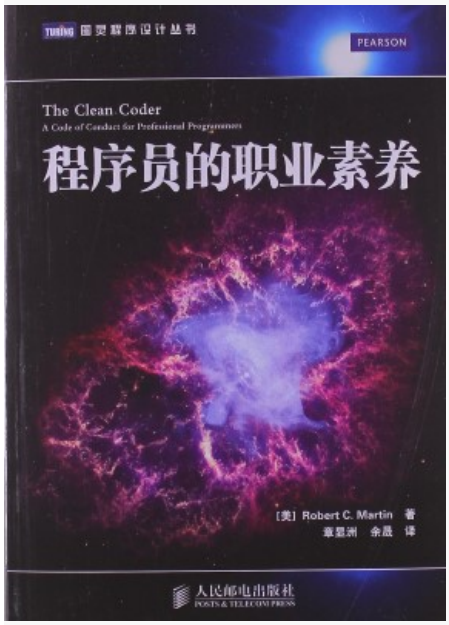
- 人件（原书第3版）：从人的角度分析软件项目。人件从雇佣正确的人，创建健康的工作环境，以及打造高效的开发团队等角度阐述了如何改善人，从而改善软件项目；
- 门后的秘密：卓越管理的故事：这本书生动的再现了软件项目管理工作的场景，并给出了各种实用管理技巧，如果你有意转向管理岗位，这本书不容错过；
- 大教堂与集市：这本书从黑客的历史说起，系统而又风趣的讲述了开源运动的理论和实践，以及开源软件项目是如何运作并发展的。了解开源，从这本书开始。

6. 专业开发



不要被庸俗的译名迷惑，程序员修炼之道是一本价值极高的程序员成长手册。这本书并不局限于特定的编程语言或框架，而是提出了一套切实可行的实效（Pragmatic）开发哲学，并通过程序设计，测试，编程工

具，以及项目管理等方面的实例展示了如何应用这套开发哲学，从而使得程序员更加高效专业。有人把这本书称之为迷你版代码大全——代码大全给出了大量的优秀程序设计实践，偏向术；而程序员修炼之道给出了程序设计实践背后的思想，注重道。

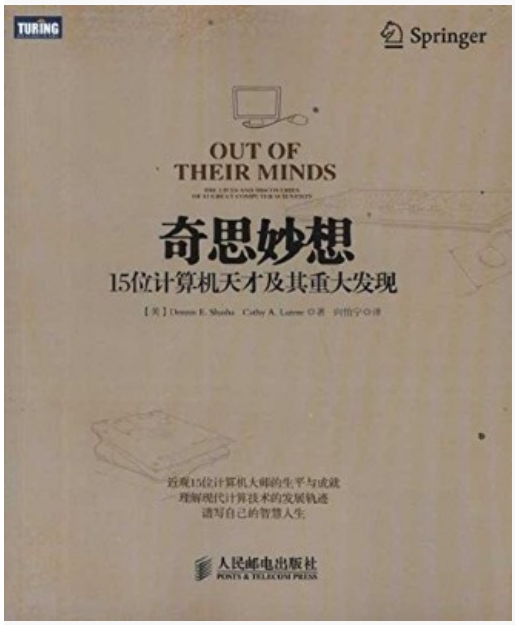


程序员修炼之道指出了**如何**成为专业程序员，这本程序员职业素养则指出了专业程序员应该是**什么**样子——承担责任；知道自己在做什么；知道何时说不/何时说是；在正确的时间编写正确的代码；懂得自我时间管理和工期预估；知道如何应对压力。如果你想成为专业程序员（Professional Developer）（而不是码农（Code Monkey）），这本书会为你指明前进的方向。

延伸阅读：

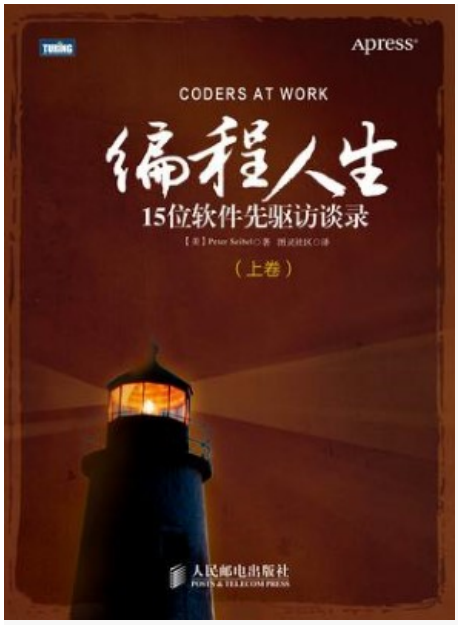
- 高效程序员的45个习惯：“敏捷”版的程序员修炼之道，可以把这本书作为程序员修炼之道的补充；
- 精益创业：尽管这是一本讲如何创业的书，但其中的精益生产，最小价值产品，以及构建-度量-学习循环都值得程序员借鉴。

7. 大师之言



奇思妙想：15位计算机天才及其重大发现是一本极具眼光的技术访谈书籍——在这本书访谈的15位计算机科学家中，竟出现了12位图灵奖获得者——要知道图灵奖从1966年设奖到现在也只有六十几位获奖者而已。

奇思妙想把计算机科学分为四大领域：编程语言；算法；架构；人工智能。并选取了每个领域下最具代表性的计算机科学家进行访谈。因为这些计算机科学家都是其所在领域的开拓者，因此他们能给出常人无法给出的深刻见解。通过这本书，你可以了解前三十年的计算机科学的发展历程——计算机科学家做了什么，而计算机又能做到/做不到什么。从而避免把时间浪费在前人已经解决的问题（或者根本无法解决的问题）上面。

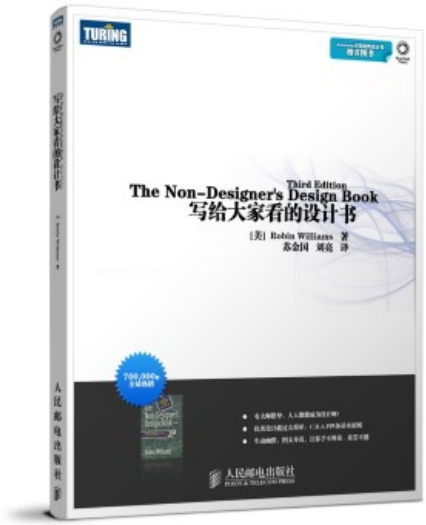


同样是访谈录，同样访谈15个人，编程人生把重点放在程序员（Coders at work）上。它从各个领域选取了15位顶尖的程序员，这些程序员既包括Ken Thompson和Jamie Zawinski这些老牌Unix黑客，也包括Brad Fitzpatrick这样的80后新生代，还包括Frances Allen和Donald Knuth这样的计算机科学家。这种多样性（Diversity）使得编程人生兼具严谨性和趣味性，无论你是什么类型的程序员，都能从中受益良多。

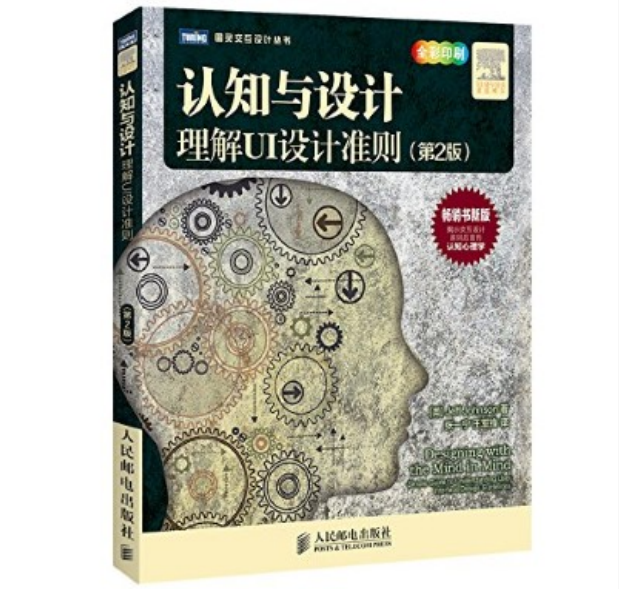
延伸阅读：

- 图灵和ACM图灵奖（1966-2011）：通过图灵奖介绍整个计算机科学发展史，非常难得的国产精品图书；
- 编程大师访谈录：可以把这本书看作为二十年前的编程人生，被访谈者都是当时叱咤风云的人物（例如微软的创造者Bill Gates，Macintosh的发明者Jeff Raskin，以及Adobe的创始人John Warnock等等）。有趣的是这本书中大量的经验和建议到如今依然适用；
- 编程大师智慧：类似于编程人生，不同的是被访谈者都是编程语言的设计者——这本书覆盖了除C语言以外的几乎所有主流编程语言。通过这本书，你可以从中学到编程语言背后的设计思想——编程语言为什么要被设计成这样，是什么促使设计者要在语言中加入这个特性（或拒绝那个特性）。从而提升对编程语言的理解。

8. 界面设计



书如其名，写给大家看的设计书是一本面向初学者的快速设计入门。它覆盖了版式，色彩，和字体这三个设计中的关键元素，并创造性的为版式设计总结出CRAP四大原则（Contrast对比，Repetition重复，Alignment对齐，Proximity亲密）。全书使用丰富生动的范例告诉读者什么是好的设计，什么是不好的设计，使得即便是对设计一无所知的人，也可以从这本书快速入门。

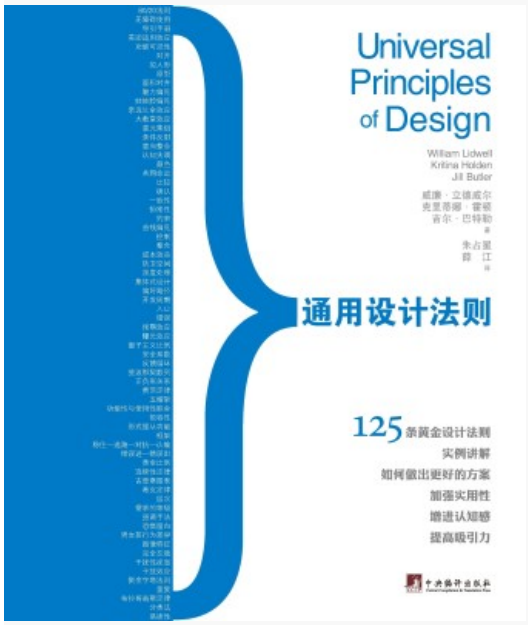


写给大家看的设计书强调实践，即**如何**做出好的设计；认知与设计：理解UI设计准则强调理论，即**为什么**我们会接受这样的设计而反感那样的设计。如果你想要搞清楚设计背后的心理学知识，但又不想阅读大部头的心理学著作，那么认知与设计是你的首选。

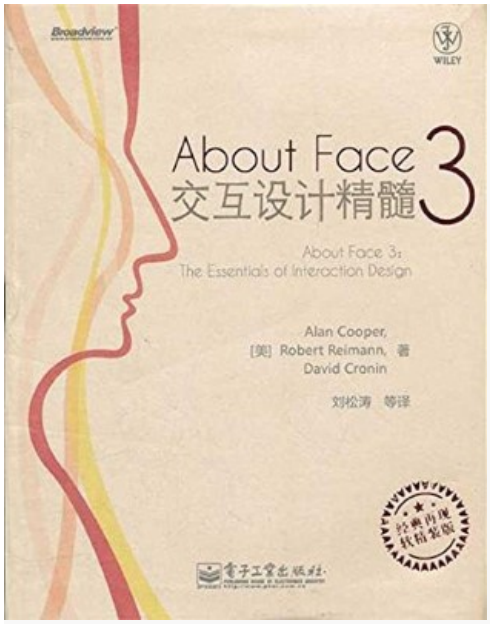
延伸阅读：

- GUI设计禁忌 2.0：这本书指出了GUI设计的原则和常见误区，然后通过具体范例指出了如何避免这些误区。如果你的工作涉及到用户界面，那么这本书会为你减少很多麻烦；
- 界面设计模式（第2版）：这本书将用户界面中的常见元素/行为组织成彼此关联的模式，以便读者理解并举一反三，从而将其运用到自己的应用中；
- 移动应用UI设计模式：类似于界面设计模式，但面向移动平台。它给出了iOS，Android，以及Windows Phones上常用的90余种界面设计模式，从而使得你不必把这些平台的应用挨个玩一遍也可以掌握各个平台的设计精髓。如果你主攻Android平台，那么Android应用UI设计模式会是更好的选择；
- 配色设计原理和版式设计原理：如果你读过写给大家看的设计书之后想继续深入学习设计，这两本书是不错的起点。

9. 交互设计



书如其名，通用设计法则给出了重要的125个设计原则，并用简练的语言和范例展示了这些原则的实际应用。每个原则都有对应的参考文献，以便读者进一步学习。我之所以推荐这本书，是因为：1. 程序员需要对设计有全面的认识；2. 程序员并不需要知道这些设计原则是怎么来的，知道怎么用即可。这本书很好的满足了这两个要求。



交互设计精髓是交互设计领域的圣经级著作。交互设计专家（以及VB之父）Alan Cooper在这本书中详细介绍了交互设计的原则，流程，以及方法，然后通过各种范例（主要来自桌面系统）展示了如何应用这些原则。

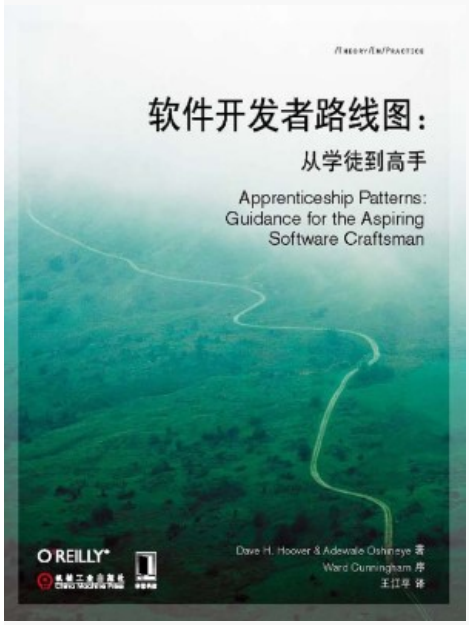
需要注意的是这本书的第4版已经出版，它在第三版的基础上增加了移动设计以及Web设计等内容。

延伸阅读：

- The Design of Everyday Things：交互设计领域的另一本经典之作，它通过解读人类行动背后的心理活动，展示了设计问题的根源，并给出了一系列方法用以解决设计问题（需要注意，尽管这本书有中译版，但中译版对应的是02年的旧版，而非13年的新版）；
- The Inmates Are Running the Asylum：Alan Cooper的另一本经典，这本书非常辛辣的指出让不具备人机交互知识的程序员直接编写面向用户的软件就像让精神病人管理疯人院（The Inmates Are Running the Asylum），然后给出了一套交互设计流程以挽救这个局面；
- 简约至上：交互式设计四策略：专注于把产品变的更加简单易用。作者通过删除，组织，隐藏，和转移这四个策略，展示了如何创造出简约优质的用户体验。

个人成长

1. 职业规划



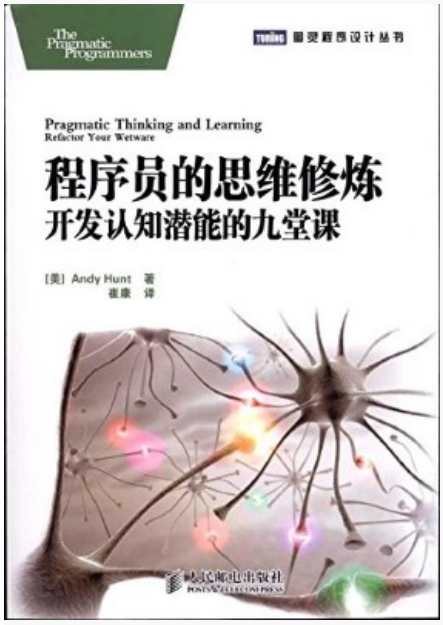
软件开发者路线图是一本优秀且实用的程序员职业规划手册。这本书由若干个模式组成，每个模式都对应于程序员职业生涯中的特定阶段。通过这本书，读者可以很方便的找到自己目前所处的模式（阶段），应该做什么，目标是什么，以及下一个模式（阶段）会是什么。如果你时常感到迷茫，那么请阅读这本路线

图，找到自己的位置，确定接下来的方向。

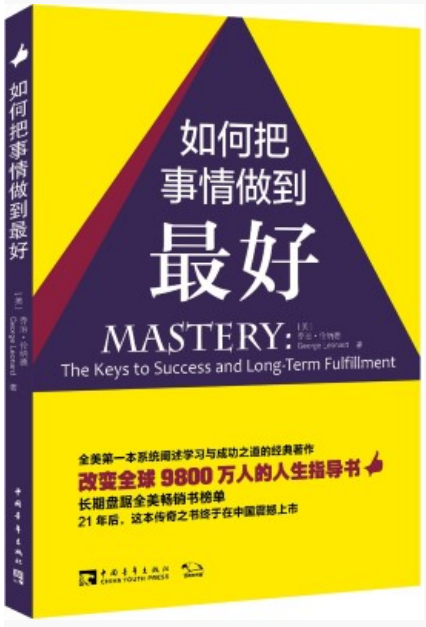
延伸阅读：

- 卡耐基全集：非常著名的为人处世书籍。很多人把这本书归类到成功学，但我并不这么认为——在我看来，这本书教的更多的是如何成为一个让大家喜欢的人。作为天天和机器打交道的程序员，这套书会帮助我们与人打交道；
- 沃顿商学院最受欢迎的谈判课：这本书不是教你去谈判，而是教你通过谈判（Negotiation）去得到更多（Getting more，这也是这本书的原书书名）。小到买菜砍价，大到争取项目，这本书中的谈判原则会让你收益良多；
- 程序员健康指南：作为长期与计算机打交道的职业，程序员往往会受到各式各样疾病的困扰，这本书正是为了解决这个问题而出现：它从改善工作环境，调整饮食结构，预防头痛眼痛，以及进行室内/室外锻炼等方面出发，给出了一套全面且可行的程序员健康改善计划，以帮助程序员打造健康的身体。

2. 思维方式



作为程序员，我们需要不断地学习——既要学习新技术，也要学习如何解决各种领域的问题。为了提升学习效率，我们需要学习**如何学习**。程序员的思维修炼正是这样一本讲如何学习的书，它集合了认知科学，神经学，以及行为理论的最新研究成果，并系统的介绍了大脑的工作机制。通过这本书，你将学会如何高效的使用自己的大脑，从而提高思考能力，改善学习效率。



> *Mastery is not about perfection. It's about a process, a journey. The master is the one who stays on the path day after day, year after year. The master is*

> *the one who is willing to try, and fail, and try again, for as long as he or she lives.*

为什么同样资质的人，大多数人会碌碌无为，而只有极少数能做到登峰造极？如何在领域内做到顶尖？如何克服通往顶尖之路上的重重险阻？如何把事情做到最好回答了这些问题，并极具哲理的指出登峰造极并不是结果，而是一段永不停止的旅程。阅读这本书不会让你立刻脱胎换骨，但它会指引你走向正确的道路——通往登峰造极之路。

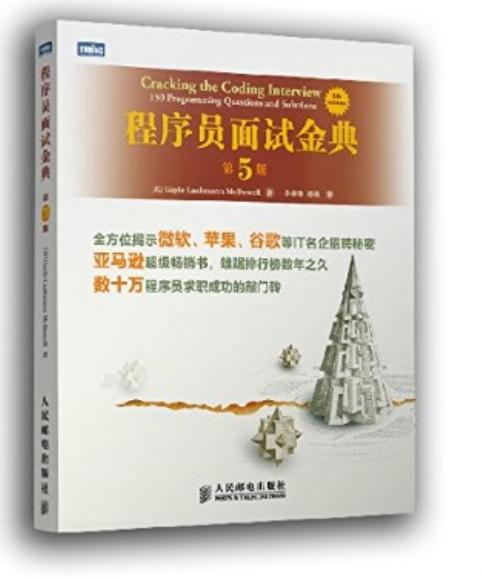
延伸阅读：

- 怎样解题：数学思维的新方法：不要被标题中的“数学思维”吓到，它并不仅仅只是一本数学解题书，它所提出的四步解题法（理解题目->拟定方案->执行计划->总结反思）适用于任何领域；
- 暗时间：刘未鹏所写的关于学习思维方法的文章集，既包含了他对学习方法的思考，也包含了大量进一步阅读的资源；
- 批判性思维：带你走出思维的误区：这本书系统的分析了人类思维的常见误区，并针对各个误区给出了解决方案，从而帮助程序员养成严谨正确的思考方式；
- Conceptual Blockbusting: A Guide to Better Ideas：与批判性思维相反，这本书专注于创造性思维（Creative Thinking），它分析了阻碍创造性思维的常见思维障碍（Blockbuster）以及这些思维障碍背后的成因，并给出了各种方法以破除这些障碍。

3. 求职面试

知己知彼，百战不殆。金领简历：敲开苹果微软谷歌的大门是程序员求职的必读书籍，它覆盖了程序员求职的方方面面：从开始准备到编写简历，从技术面试到薪酬谈判。由于该书作者曾在Google，微软，和苹果任职并进行过技术招聘，因此这本书的内容非常实用。

顺便吐个槽：这本书翻译的还不错，但我实在无法理解封面上的“进入顶级科技公司的葵花宝典”这段文字——找个工作而已，用不着切JJ这么凶残吧。-_-#

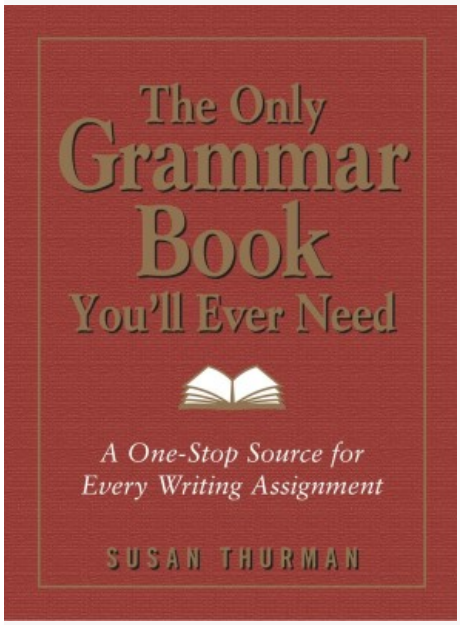


同样是来自金领简历作者的作品，程序员面试金典（第5版）专注于技术面试题，它既包含了IT企业（诸如微软，Google，和苹果）的面试流程以及如何准备技术面试，也包含了大量（超过200道）常见技术面试题题目以及解题思路。无论你打算进入国内企业还是外企，你都应该把这本书的题目练一遍，以找到技术面试的感觉（我在求职时就曾经专门搞了一块白板，然后每二十分钟一道题的练习，效果很不错）。

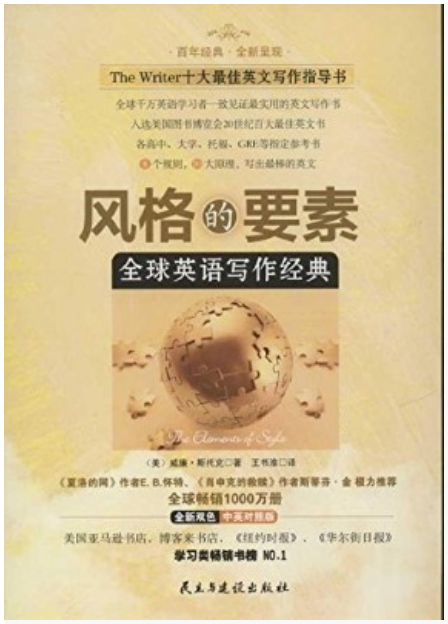
延伸阅读：

- 编程之美：微软技术面试心得：恐怕是国内技术面试第一书，这本书里面的多数题目都曾经是国内IT企业面试的必问题目。这本书的缺点是它太旧而且被用滥了（以至于一些企业开始避免使用这本书上的题目）——但你可以把它当成一本算法趣题来读；
- 剑指Offer：名企面试官精讲典型编程题：相对于东拼西凑的XX面试宝典，剑指Offer是一本少见的国产精品技术面试书籍，尽管这本书的技术面试题目不多（60余道），但作者为大多数题目都给出了不同方式的解法，并分析了这些解法之间的优劣，此外作者还以面试官的视角分析了技术面试的各个环节，从而帮助读者把握技术面试；
- 人人都有好工作：IT行业求职面试必读：可以把它看做金领简历的补充阅读——这本书的特点在于它给出了非常详细的简历/求职信/电子邮件编写技巧，而这正是不少国内程序员所缺乏的。

4. 英语写作



词汇量决定阅读能力，语法决定写作能力。计算机专业词汇并不多，但精确性非常重要，因此每个程序员都应具备良好的英语语法，但程序员并不需要过于专业的英语语法——掌握常用语法并把它用对就可以。The Only Grammar Book You'll Ever Need正好可以满足这个需求，尽管它篇幅不大（不足200页），却覆盖了英语中的关键语法以及常见错误。把这本书读两遍，它会大幅度提高你的英语写作能力。



既是最畅销的英语写作书籍，也是计算机书籍中引用最多的非计算机书籍。风格的要素用极其简练的语言讲述了如何进行**严肃**，**精确**，**清楚**的英语写作。从这本书中，你不仅可以学到英语写作，更可以学到一种严谨至简的处事态度，而这正是专业开发所必需的。

延伸阅读：

- 牛津英语用法指南（第3版）：全面且权威的英语用法指南，它覆盖语法，词汇，发音，以及修辞等方面，并兼顾口语和书面语，以帮助读者掌握合理的英语用法（Proper English Usage）。不要被这本书的篇幅（1000多页）吓到——原书并没有这么厚，因为这本书被翻译成中文但又得保留原有的英文内容，所以它的篇幅几乎翻了一倍。考虑到这本书使用的词汇都很基础，所以我认为具有英语基础的读者直接阅读原版（Practical English Usage）会更合适；
- 写作法宝：非虚构写作指南（30周年纪念版）：详尽的非虚构（Non-Fiction）写作指南，无论你要写地方，技术，商务，运动，艺术，还是自传，你都可以从这本书中找到珍贵的建议；
- 中式英语之鉴：中国人使用英语最大的问题就是会把中式思维掺杂其中，从而形成啰里啰嗦不伦不类的中式英语（Chinglish）。中式英语之鉴系统的探讨了中式英语以及其成因，然后根据成因对中式英语进行归类，并对每个类别给出了大量的实际案例以及修改建议。如果你想摆脱中式英语，那么这本书是绝好的起点。

如何使用这个书单

> *学而不思则罔，思而不学则殆。*
>
> *不愤不启，不悱不发。举一隅不以三隅反，则不复也。*
>
> *不闻不若闻之，闻之不若见之，见之不若知之，知之不若行之，学至于行之而止矣。*

来自他人的书单

它山之石，可以攻玉。我在本文最后给出其他中外优秀程序员的书单，以便参考&补充。

刘未鹏（暗时间作者）

以下同一条目下用“/”隔开的表示任选，当然也可以都读。

1. 编码：隐匿在计算机软硬件背后的语言
2. 深入理解计算机系统 / Windows核心编程 / 程序员的自我修养
3. 代码大全 / 程序员修炼之道
4. 编程珠玑 / 算法概论 / 算法设计 / 编程之美
5. C程序设计语言
6. C++程序设计语言 / C++程序设计原理与实践 / Accelerated C++
7. 计算机程序的构造与解释

8. 代码整洁之道 / 实现模式
9. 设计模式 / 敏捷软件开发（原则模式与实践）
10. 重构

云风（中国游戏编程先行者，前网易游戏部门资深程序员，简悦创始人）

1. C++编程思想
2. Effective C++
3. 深度探索C++对象模型
4. C++语言的设计与演化
5. C专家编程
6. C陷阱与缺陷
7. C语言接口与实现
8. Lua程序设计
9. 链接器和加载器
10. COM本质论
11. Windows核心编程
12. 深入解析Windows操作系统
13. 程序员修炼之道
14. 代码大全
15. UNIX编程艺术
16. 设计模式
17. 代码优化：有效使用内存
18. 深入理解计算机系统
19. 深入理解LINUX内核
20. TCP/IP详解

洪强宁（豆瓣技术总监）

1. 代码大全
2. 人月神话
3. 编码：隐匿在计算机软硬件背后的语言
4. 计算机程序设计艺术
5. 程序员修炼之道
6. 设计模式
7. 计算机程序的构造与解释
8. 重构
9. C程序设计语言
10. 算法导论

陈皓（CoolShell博主）

1. 点石成金：访客至上的Web和移动可用性设计秘笈
2. 重来：更为简单有效的商业思维
3. 黑客与画家
4. 清醒思考的艺术
5. TCP/IP详解
6. UNIX环境高级编程
7. UNIX网络编程

张峥（微软亚洲研究院副院长）

1. 算法概论
2. Data Structure and Algorithms
3. C程序设计语言
4. UNIX操作系统设计
5. 编译原理
6. 计算机体系结构：量化研究方法
7. 当下的幸福
8. 异类：不一样的成功启示录

Jeff Atwood（Stackoverflow联合创始人）

1. 代码大全
2. 人月神话
3. 点石成金：访客至上的Web和移动可用性设计秘笈
4. 快速软件开发
5. 人件
6. The Design of Everyday Things
7. 交互设计精髓
8. The Inmates Are Running the Asylum
9. GUI设计禁忌 2.0
10. 编程珠玑
11. 程序员修炼之道
12. 精通正则表达式

Joel Spolsky（Stackoverflow联合创始人）

软件项目管理

1. 人件
2. 人月神话
3. 快速软件开发

编程技艺

1. 代码大全
2. 程序员修炼之道

编程哲学

1. 禅与摩托车维修艺术
2. 哥德尔、艾舍尔、巴赫：集异璧之大成
3. 建筑模式语言

界面设计

1. 点石成金：访客至上的Web和移动可用性设计秘笈
2. 交互设计精髓
3. The Design of Everyday Things

资本运作

1. 漫步华尔街

图形设计

1. 写给大家看的设计书

思维方式

1. 影响力
2. Helplessness On Depression, Development and Death

编程入门

1. 编码：隐匿在计算机软硬件背后的语言
2. C程序设计语言

DHH（Ruby on Rails创始人）

1. Smalltalk Best Practice Patterns
2. 重构
3. 企业应用架构模式
4. 领域驱动设计
5. 你的灯亮着吗？发现问题的真正所在

参考

1. 怎样花两年时间去面试一个人
2. What is the single most influential book every programmer should read?
3. Recommended Reading for Developers
4. Book Reviews — Joel Spolsky
5. The five programming books that meant most to me