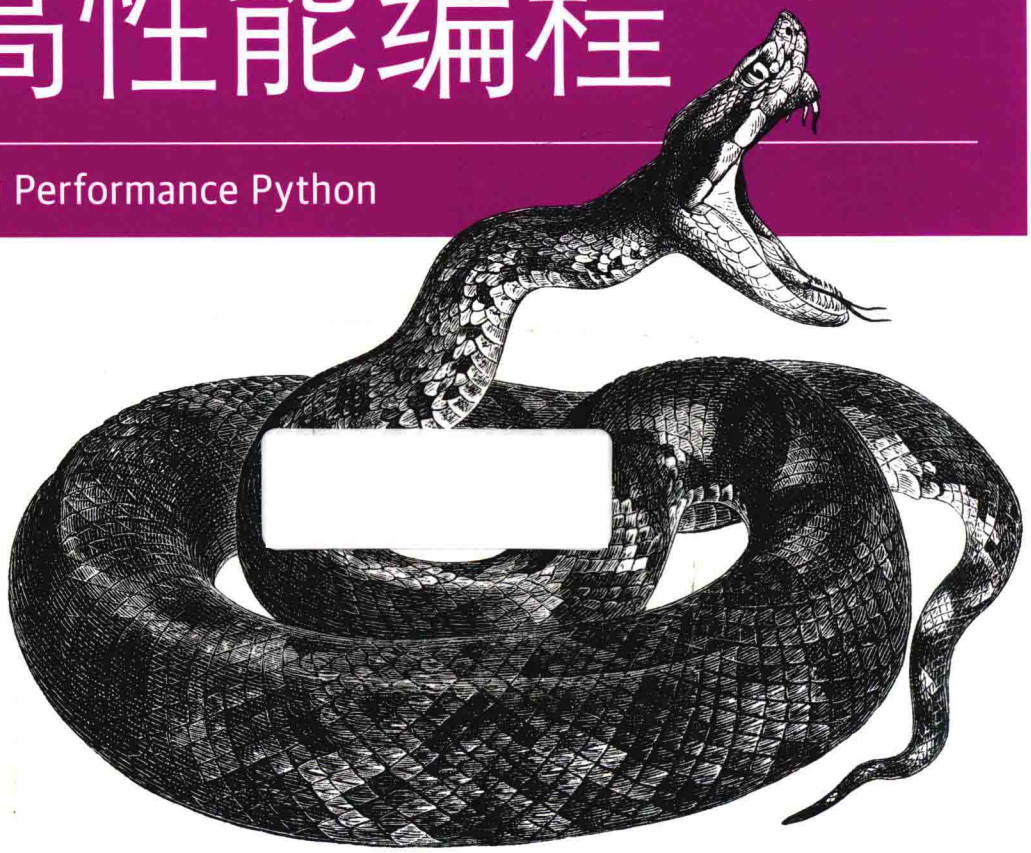
O'REILLY®

异步图书
www.epubit.com.cn

Python 高性能编程

High Performance Python



[美] Micha Gorelick Ian Ozsvald 著
胡世杰 徐旭彬 译

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

O'REILLY®

Python 高性能编程

[美] Micha Gorelick
Ian Ozsvald 著
胡世杰 徐旭彬 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Python高性能编程 / (美) 戈雷利克 (Micha Gorelick), (美) 欧日沃尔德 (Ian Ozsvald) 著 ; 胡世杰, 徐旭彬译. -- 北京 : 人民邮电出版社, 2017.7
ISBN 978-7-115-45489-8

Ⅰ. ①P… Ⅱ. ①戈… ②欧… ③胡… ④徐… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆CIP数据核字(2017)第114298号

版权声明

◆ 著 [美] Micha Gorelick Ian Ozsvald
译 胡世杰 徐旭彬
责任编辑 陈冀康
责任印制 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
三河市海波印务有限公司印刷

◆ 开本：800×1000 1/16
印张：22
字数：415 千字 2017 年 7 月第 1 版
印数：1－3 000 册 2017 年 7 月河北第 1 次印刷

著作权合同登记号 图字：01-2013-9303 号

定价：79.00 元

读者服务热线：(010)81055410 印装质量热线：(010)81055316
反盗版热线：(010)81055315

前言

Python 很容易学。你之所以阅读本书可能是因为你的代码现在能够正确运行，而你希望它能跑得更快。你可以很轻松地修改代码，反复地实现你的想法，你对这一点很满意。但*能够轻松实现*和*代码跑得够快*之间的取舍却是一个世人皆知且令人惋惜的现象。而这个问题其实是可以解决的。

有些人想要让顺序执行的过程跑得更快。有些人需要利用多核架构、集群，或者图形处理单元的优势来解决他们的问题。有些人需要可伸缩系统在保证可靠性的前提下酌情或根据资金多少处理更多或更少的工作。有些人意识到他们的编程技巧，通常是来自其他语言，可能不如别人的自然。

我们会在本书中覆盖所有这些主题，给出明智的指导去了解瓶颈并提出效率更高、伸缩性更好的解决方案。我们也会在本书中包含那些来自前人的战场故事，让你可以避免重蹈覆辙。

Python 很适合快速开发、生产环境部署，以及可伸缩系统。Python 的生态系统里到处都是帮你解决伸缩性的人，让你有更多时间处理那些更有挑战性的工作。

本书适合哪些人

你使用 Python 的时间已经足够长，了解为什么某些代码会跑得慢，也见过以本书讨论的 Cython、numpy 以及 PyPy 等技术作为解决方案。你可能还有其他语言的编程经验，因此知道解决性能问题的路不止一条。

本书主要的目标读者是那些需要解决 CPU 密集型问题的人，同时我们也会关注数据传输以及内存密集型问题。科学家、工程师、数据分析专家、学者通常会面临这些问题。

我们还会关注网页开发者可能面临的问题，包括数据的移动以及为了快速提升性能而使用 PyPy 这样的即时（JIT）编译器。

如果你有一个 C（或 C++，或 Java）的背景可能会有帮助，但这不是必须的。Python 最常用的解释器（CPython——你在命令行输入 `python` 时启动的标准解释器）是用 C 写的，所以各种钩子和库全都血淋淋地暴露了内部的 C 机制。但我们也会谈到对 C 一无所知的人也能使用的许多其他技术。

你可能还具有 CPU、内存架构和数据总线的底层知识，还是那句话，这些也不完全是必须的。

本书不适合哪些人

本书适用于中高级 Python 程序员。积极的 Python 初学者可能可以跟上，但我们建议要具有坚实的 Python 基础。

我们不会探讨存储系统优化。如果你有一个 SQL 或 NoSQL 瓶颈，本书可能帮不了你。

你会学到什么

我们两位作者在业界和学术界工作了很多年，专门应对大数据应用、处理*我需要更快得到答案!* 之类的请求、可伸缩架构等需求。我们会将自己经历千辛万苦获得的经验传授于你，让你免于重蹈覆辙。

在每一章开头，我们会列出问题，并在后续的文字中回答（如果没有回答，告诉我们，我们会在下一个版本中修正!）。

我们会覆盖下面这些主题：

- 计算机内部结构的背景知识，让你知道在底层发生了什么。
- 列表和元组——在这些基本数据结构中细微的语义和速度区别。
- 字典和集合——在这些重要数据结构中的内存分配策略和访问算法。
- 迭代器——Python 风格的代码应该怎样写，用迭代打开无限数据流的大门。
- 纯 Python 方法——如何高效使用 Python 及其模块。
- 使用 `numpy` 的矩阵——像一头野兽一样使用心爱的 `numpy` 库。
- 编译和即时计算——编译成机器码可以跑得更快，让性能分析的结果指引你。
- 并发——高效移动数据的方法。
- `multiprocessing`——使用内建 `multiprocessing` 库进行并行计算的各种方式，高效共享 `numpy` 矩阵、进程间通信（IPC）的代价和收益。
- 集群计算——将你的 `multiprocessing` 代码转换成在研究系统以及生产系统的本地集群或远程集群上运行的代码。
- 使用更少的 RAM——不需要购买大型机就能解决大型问题的方法。
- 现场教训——来自前人的战场故事，让你可以避免重蹈覆辙。

Python 2.7

Python 2.7 在科学和工程计算中是占主导地位的 Python 版本。在*nix 环境(通常是 Linux 或 Mac）下，64 位的版本占了主导地位。64 位让你能够拥有更宽广的 RAM 寻址范围。*nix 让你构建出的应用程序的行为、部署和配置方法都可以很容易地被别人所理解。

如果你是一个 Windows 用户，那么你就要系好安全带了。我们展示的大多数代码都可以正常工作，但有些东西是针对特定操作系统的，你将不得不研究 Windows 下的解决方案。Windows 用户可能面临的最大的困难是模块的安装：搜索 StackOverflow 等站点应该可以帮助你找到你需要的答案。如果你正在使用 Windows，那么使用一台安装了 Linux 的虚拟机（比如 VirtualBox）可能可以帮助你更自由地进行实验。

Windows 用户绝对应该看看那些通过 Anaconda、Canopy、Python(x,y)或 Sage 等 Python 发行版提供的打包的解决方案。这些发行版也会让 Linux 和 Mac 用户的生活简单许多。

迁移至 Python 3

Python 3 是 Python 的未来，每一个人都在迁移过去。尽管 Python 2.7 还将在接下来的很多年里面继续跟我们做伴（有些安装版仍在使用 2004 年的 Python 2.4），它的退役日期已经被定在 2020 年了。

升级到 Python 3.3+让 Python 库的开发者伤透了脑筋，人们移植代码的速度一直都很慢(这是有原因的)，所以人们转用 Python 3 的速度也很慢。这主要是因为要把一个混用了 Python 2 的 string 和 Unicode 数据类型的应用程序切换成 Python 3 的 Unicode 和 byte 实在太过复杂了。

通常来说，当你需要重现基于一批值得信任的库的结果时，你不会想要站在危险的技术前沿。高性能 Python 的开发者更有可能在接下来的几年里使用和信任 Python 2.7。

本书的大多数代码只需要稍做修改就能运行于 Python 3.3+（最明显的修改是 `print` 从一个语句变成了一个函数)。在一些地方，我们将特地关注 Python 3.3+带来的性能提升。一个可能需要你关注的地方是在 Python 2.7 中 / 表示 *integer* 的除法，而在 Python 3 中它变成了 *float* 的除法。当然，作为一个好的开发者，你精心编写的单元测试应该已经在测试你的关键代码路径了，所以如果你的代码需要关注这点，那么你应该已经收到来自你单元测试的警告了。

`scipy` 和 `numpy` 从 2010 年开始就已经兼容 Python 3 了。`matplotlib` 从 2012 年开始兼容，`scikit-learn` 是 2013，`NLTK` 是 2014，Django 是 2013。这些库的迁移备忘录可以在它们各自的代码库和新闻组里查看。如果你也有旧代码需要移植到 Python 3，那么就值得回顾一下这些库移植的过程。

我们鼓励你用 Python 3.3+进行新项目的开发，但你要当心那些最近刚刚移植还没有多少用户的库——追踪 bug 将会更困难。比较明智的做法是让你的代码可以兼容 Python 3.3+（学习一下__future__模块的导入），这样未来的升级就会更简单。

有两本参考手册不错：《把 Python 2 的代码移植到 Python 3》和《移植到 Python 3：深度指南》。Anaconda 或 Canopy 这样的发行版让你可以同时运行 Python 2 和 Python 3——这会让你的移植变得简单一些。

版权声明

如何引用

如果你需要使用本书，知识共享许可证要求你署名。署名意味着你需要写一些东西让其他人能够找到本书。下面是一个不错的例子：“High Performance Python by Micha Gorelick and Ian Ozsvald (O'Reilly). Copyright 2014 Micha Gorelick and Ian Ozsvald, 978-1-449-36159-4.”

勘误和反馈

我们鼓励你在 Amazon 这样的公开网站上评论本书——请帮助其他人了解他们是否能从本书中受益！你也可以发 E-mail 给我们：feedback@highperformancepython.com。

我们特别希望听到您指出本书的错误，本书帮到你的成功案例，以及我们应该在下一版本加上的高性能技术。你可以通过 O'Reilly 官网给我们留言。

至于抱怨，欢迎你使用即时抱怨传输服务> `/dev/null`。

排版约定

本书采用下列排版约定：

斜体

表示新词、E-mail 地址、文件名，以及文件扩展名。

等宽

用于程序列印，以及在文字中表示命令、模块和程序元素，如变量或函数名、数据库、数据类型、环境变量、语句和关键字。

等宽加粗

表示命令或其他需要用户原封不动输入的文字。

等宽斜体

表示需要被替换成用户指定的值或根据上下文决定的值。

问题
这个记号表示一个问题或练习。

备忘
这个记号表示一个备忘。

警告
这个记号表示一个警告或注意。

使用示例代码

补充材料（示例代码、练习等）可以通过 GitHub 下载。

本书是为了帮你搞定你的问题。通常来说，只要是本书提供的示例代码，你就可以在你的程序和文档中使用。你不需要联系我们获得许可，除非你需要对很大一部分代码进行转载。比如，写一个使用了好几段本书代码的程序不需要许可。以 CD-ROM 的形式销售或分发 O'Reilly 图书中的示例需要许可。引用本书文字和示例代码回答问题不需要许可。在你的产品文档中合并大量本书示例代码需要许可。

如果你觉得你对示例代码的使用超出了上述的许可范围，请通过 permissions@oreilly.com 联系我们。

Safari®在线图书

*Safari 在线图书*是一个应需数字图书馆，它以书和视频的形式提供来自全球顶尖作者的技术和商业内容。

技术专家、软件开发者、网页设计者，以及商业和创新人员将 Safari 在线图书当成他们研究、解决问题、学习和资格认证训练的主要资源。

Safari 在线图书为企业，政府，教育和个人提供了各种收费标准。

其成员可以通过全文搜索数据库访问成千上万的图书，训练视频，以及还未正式出版的手稿。它们来自 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology，以及其他几百个出版社。更多信息请在线访问 https://www.safaribooksonline.com/。

联系我们

请将关于本书的评论和问题发给本书出版社：

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)

707-829-0515 (international or local)

707-829-0104 (fax)

你也可以发送 E-mail 到 bookquestions@oreilly.com 对本书进行评论或询问技术问题。

关于我们的图书、课程、会议和新闻等更多信息请访问我们的网站。

我们的 Facebook：http://facebook.com/oreilly

Twitter：http://twitter.com/oreillymedia

YouTube：http://www.youtube.com/oreillymedia

致谢

感谢来自 Jake Vanderplas、Brian Granger、Dan Foreman-Mackey、Kyran Dale、John Montgomery、Jamie Matthews、Calvin Giles、William Winter、Christian Schou Oxvig、

Balthazar Rouberol、Matt “snakes” Reiferson、Patrick Cooper 和 Michael Skirpan 的反馈和贡献。Ian 感谢他的妻子 Emily 让他消失 10 个月之久来完成本书（她真的太善解人意了）。Micha 感谢 Elaine 及其他朋友和他的家庭能够耐心等待他学习写书。O’Reilly 也是很好的合作伙伴。

第 12 章的提供者非常亲切地共享了他们的时间和宝贵的经验。我们感谢 Ben Jackson、Radim Řehůřek、Sebastjan Trebca、Alex Kelly、Marko Tasic 和 Andrew Godwin 花费的时间和精力。

目录

第 1 章　理解高性能 Python ····· 1
1.1　基本的计算机系统 ····· 1
1.1.1　计算单元 ····· 2
1.1.2　存储单元 ····· 5
1.1.3　通信层 ····· 6
1.2　将基本的元素组装到一起 ····· 8
1.3　为什么使用 Python ····· 12
第 2 章　通过性能分析找到瓶颈 ····· 15
2.1　高效地分析性能 ····· 16
2.2　Julia 集合的介绍 ····· 17
2.3　计算完整的 Julia 集合 ····· 20
2.4　计时的简单方法——打印和修饰 ····· 24
2.5　用 UNIX 的 time 命令进行简单的计时 ····· 27
2.6　使用 cProfile 模块 ····· 28
2.7　用 runsnakerun 对 cProfile 的输出进行可视化 ····· 33
2.8　用 line_profiler 进行逐行分析 ····· 34
2.9　用 memory_profiler 诊断内存的用量 ····· 39
2.10　用 heapy 调查堆上的对象 ····· 45
2.11　用 dowser 实时画出变量的实例 ····· 47
2.12　用 dis 模块检查 CPython 字节码 ····· 49
2.13　在优化期间进行单元测试保持代码的正确性 ····· 53
2.14　确保性能分析成功的策略 ····· 56
2.15　小结 ····· 57
第 3 章　列表和元组 ····· 58
3.1　一个更有效的搜索 ····· 61
3.2　列表和元组 ····· 63
3.2.1　动态数组：列表 ····· 64
3.2.2　静态数组：元组 ····· 67
3.3　小结 ····· 68

第 4 章　字典和集合 ……69
4.1　字典和集合如何工作 ……72
4.1.1　插入和获取 ……73
4.1.2　删除 ……76
4.1.3　改变大小 ……76
4.1.4　散列函数和熵 ……76
4.2　字典和命名空间 ……80
4.3　小结 ……83
第 5 章　迭代器和生成器 ……84
5.1　无穷数列的迭代器 ……87
5.2　生成器的延迟估值 ……89
5.3　小结 ……93
第 6 章　矩阵和矢量计算 ……94
6.1　问题介绍 ……95
6.2　Python 列表还不够吗 ……99
6.3　内存碎片 ……103
6.3.1　理解 perf ……105
6.3.2　根据 perf 输出做出抉择 ……106
6.3.3　使用 numpy ……107
6.4　用 numpy 解决扩散问题 ……110
6.4.1　内存分配和就地操作 ……113
6.4.2　选择优化点：找到需要被修正的地方 ……116
6.5　numexpr：让就地操作更快更简单 ……120
6.6　告诫故事：验证你的“优化”（scipy） ……121
6.7　小结 ……123
第 7 章　编译成 C ……126
7.1　可能获得哪种类型的速度提升 ……127
7.2　JIT 和 AOT 编译器的对比 ……129
7.3　为什么类型检查有助代码更快运行 ……129
7.4　使用 C 编译器 ……130
7.5　复习 Julia 集的例子 ……131
7.6　Cython ……131
7.6.1　使用 Cython 编译纯 Python 版本 ……132
7.6.2　Cython 注解来分析代码块 ……134
7.6.3　增加一些类型注解 ……136

7.7 Shed Skin ····· 140
7.7.1 构建扩展模块 ····· 141
7.7.2 内存拷贝的开销 ····· 144
7.8 Cython 和 numpy ····· 144
7.9 Numba ····· 148
7.10 Pythran ····· 149
7.11 PyPy ····· 151
7.11.1 垃圾收集的差异 ····· 152
7.11.2 运行 PyPy 并安装模块 ····· 152
7.12 什么时候使用每种工具 ····· 154
7.12.1 其他即将出现的项目 ····· 155
7.12.2 一个图像处理单元（GPU）的注意点 ····· 156
7.12.3 一个对未来编译器项目的展望 ····· 157
7.13 外部函数接口 ····· 157
7.13.1 ctypes ····· 158
7.13.2 cffi ····· 160
7.13.3 f2py ····· 163
7.13.4 CPython 模块 ····· 166
7.14 小结 ····· 170
第 8 章 并发 ····· 171
8.1 异步编程介绍 ····· 172
8.2 串行爬虫 ····· 175
8.3 gevent ····· 177
8.4 tornado ····· 182
8.5 AsyncIO ····· 185
8.6 数据库的例子 ····· 188
8.7 小结 ····· 191
第 9 章 multiprocessing 模块 ····· 193
9.1 multiprocessing 模块综述 ····· 196
9.2 使用蒙特卡罗方法来估算 pi ····· 198
9.3 使用多进程和多线程来估算 pi ····· 199
9.3.1 使用 Python 对象 ····· 200
9.3.2 并行系统中的随机数 ····· 207
9.3.3 使用 numpy ····· 207
9.4 寻找素数 ····· 210

9.5 使用进程间通信来验证素数……221
9.5.1 串行解决方案……225
9.5.2 Naïve Pool 解决方案……225
9.5.3 Less Naïve Pool 解决方案……226
9.5.4 使用 Manager.Value 作为一个标记……227
9.5.5 使用 Redis 作为一个标记……229
9.5.6 使用 RawValue 作为一个标记……232
9.5.7 使用 mmap 作为一个标记……232
9.5.8 使用 mmap 作为一个标记的终极效果……234
9.6 用 multiprocessing 来共享 numpy 数据……236
9.7 同步文件和变量访问……243
9.7.1 文件锁……243
9.7.2 给 Value 加锁……247
9.8 小结……249
第 10 章 集群和工作队列……251
10.1 集群的益处……252
10.2 集群的缺陷……253
10.2.1 糟糕的集群升级策略造成华尔街损失 4.62 亿美元……254
10.2.2 Skype 的 24 小时全球中断……255
10.3 通用的集群设计……255
10.4 怎样启动一个集群化的解决方案……256
10.5 使用集群时避免痛苦的方法……257
10.6 三个集群化解决方案……258
10.6.1 为简单的本地集群使用 Parallel Python 模块……259
10.6.2 使用 IPython Parallel 来支持研究……260
10.7 为鲁棒生产集群的 NSQ……265
10.7.1 队列……265
10.7.2 发布者/订阅者……266
10.7.3 分布式素数计算器……268
10.8 看一下其他的集群化工具……271
10.9 小结……272
第 11 章 使用更少的 RAM……273
11.1 基础类型的对象开销高……274
11.2 理解集合中的 RAM 使用……278
11.3 字节和 Unicode 的对比……280

11.4 高效地在 RAM 中存储许多文本 ····· 281
11.5 使用更少 RAM 的窍门 ····· 290
11.6 概率数据结构 ····· 291
11.6.1 使用 1 字节的 Morris 计数器来做近似计数 ····· 292
11.6.2 K 最小值 ····· 295
11.6.3 布隆过滤器 ····· 298
11.6.4 LogLog 计数器 ····· 303
11.6.5 真实世界的例子 ····· 307
第 12 章 现场教训 ····· 311
12.1 自适应实验室（Adaptive Lab）的社交媒体分析（SoMA） ····· 311
12.1.1 自适应实验室（Adaptive Lab）使用的 Python ····· 312
12.1.2 SoMA 的设计 ····· 312
12.1.3 我们的开发方法论 ····· 313
12.1.4 维护 SoMA ····· 313
12.1.5 对工程师同行的建议 ····· 313
12.2 使用 RadimRehurek.com 让深度学习飞翔 ····· 314
12.2.1 最佳时机 ····· 314
12.2.2 优化方面的教训 ····· 316
12.2.3 总结 ····· 318
12.3 在 Lyst.com 的大规模产品化的机器学习 ····· 318
12.3.1 Python 在 Lyst 的地位 ····· 319
12.3.2 集群设计 ····· 319
12.3.3 在快速前进的初创公司中做代码评估 ····· 319
12.3.4 构建推荐引擎 ····· 319
12.3.5 报告和监控 ····· 320
12.3.6 一些建议 ····· 320
12.4 在 Smesh 的大规模社交媒体分析 ····· 321
12.4.1 Python 在 Smesh 中的角色 ····· 321
12.4.2 平台 ····· 321
12.4.3 高性能的实时字符串匹配 ····· 322
12.4.4 报告、监控、调试和部署 ····· 323
12.5 PyPy 促成了成功的 Web 和数据处理系统 ····· 324
12.5.1 先决条件 ····· 325
12.5.2 数据库 ····· 325
12.5.3 Web 应用 ····· 326

12.5.4 OCR 和翻译 ……326
12.5.5 任务分发和工作者 ……327
12.5.6 结论 ……327
12.6 在 Lanyrd.com 中的任务队列 ……327
12.6.1 Python 在 Lanyrd 中的角色 ……328
12.6.2 使任务队列变高性能 ……328
12.6.3 报告、监控、调试和部署 ……328
12.6.4 对开发者同行的建议 ……329

第 1 章

理解高性能 Python

读完本章之后你将能够回答下列问题

- 计算机架构有哪些元素？
- 常见的计算机架构有哪些？
- 计算机架构在 Python 中的抽象表达是什么？
- 实现高性能 Python 代码的障碍在哪里？
- 性能问题有哪些种类？

计算机编程可以被认为是以特定的方式进行数据的移动和转换来得到某种结果。然而这些操作有时间上的开销。因此，高性能编程可以被认为是通过降低开销（比如撰写更高效的代码）或改变操作方式（比如寻找一种更合适的算法）来让这些操作的代价最小化。

数据的移动发生在实际的硬件上，我们可以通过降低代码开销的方式来了解更多硬件方面的细节。这样的练习看上去可能没什么用，因为 Python 做了很多工作将我们对硬件的直接操作抽象出来。然而，通过理解数据在硬件层面的移动方式以及 Python 在抽象层面移动数据的方式，你会学到一些编写高性能 Python 程序的知识。

1.1 基本的计算机系统

一台计算机的底层组件可被分为三大基本部分：计算单元，存储单元，以及两者之间的连接。除此之外，这些单元还具有多种属性帮助我们了解它们。计算单元有一个属性告诉我们它每秒能够进行多少次计算，存储单元有一个属性告诉我们它能保