PEARSON
HZ BOOKS
华章教育

经 典 原 版 书 库

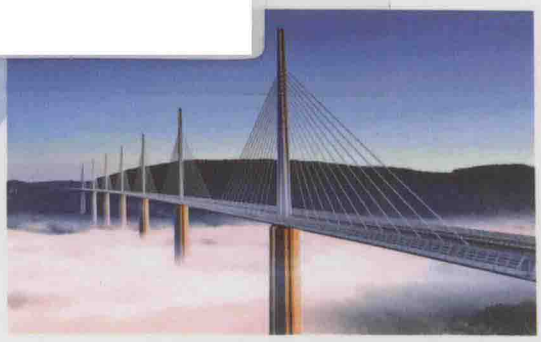
Go程序设计语言

[美] 艾伦 A. A. 多诺万 Alan A. A. Donovan
布莱恩 W. 柯尼汉 Brian W. Kernighan 著

（英文版）

The Go Programming Language

Alan A. A. Donovan
Brian W. Kernighan



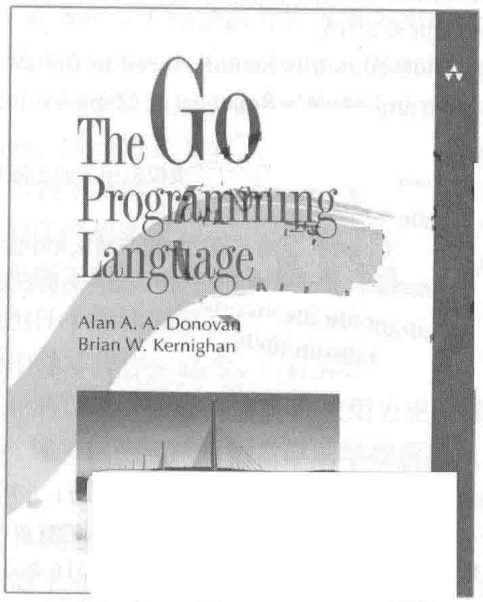
机械工业出版社
China Machine Press

经 典 原 版 书 库

Go程序设计语言

（英文版）

The Go Programming Language



[美] 艾伦 A. A. 多诺万 Alan A. A. Donovan 布莱恩 W. 柯尼汉 Brian W. Kernighan 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Go 程序设计语言（英文版）/（美）多诺万（Donovan, A. A. A.），（美）柯尼汉（Kernighan, B. W.）著. —北京：机械工业出版社，2016.1
（经典原版书库）
书名原文：The Go Programming Language

ISBN 978-7-111-52628-5

I. G… II. ①多… ②柯… III. C 语言－程序设计－英文 IV. TP312

中国版本图书馆 CIP 数据核字（2015）第 314265 号

本书版权登记号：图字：01-2015-8400

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：识振春　　责任校对：董纪丽
印　　刷：三河市宏图印务有限公司　　版　　次：2016 年 1 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：23.75
书　　号：ISBN 978-7-111-52628-5　　定　　价：79.00 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

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com

电子邮件：hzjsj@hzbook.com

联系电话：（010）88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037

华章科技图书出版中心

前　言

“Go 是一种开放源代码的程序设计语言，它意在使得人们能够方便地构建简单、可靠、高效的软件。”（来自 Go 官网 golang.org）

Go 形成构想是在 2007 年 9 月，并于 2009 年 11 月发布，其发明人是 Robert Griesemer、Rob Pike 和 Ken Thompson，这几位都任职于 Google。该语言及其配套工具集意在使得编译和执行都能够富有表达力和效率，而且使得程序员能够轻松写出可靠、健壮的程序。

Go 和 C 从表面上看起来相似，而且和 C 一样，它也是一种专业程序员使用的工具，兼有事半功倍之效。但是 Go 远不止是 C 的一种升级版本。它从多种其他语言中借用和改造了不少好的思想，还避开了那些导致复杂和脆弱的语言特性。它为并发提供的设施是全新的、高效的，它实现数据抽象和面向对象的途径是极其灵活的。它还提供了自动化的内存管理，或称为垃圾收集。

Go 特别适用于构建基础设施类软件（如网络服务器），以及程序员使用的工具和系统等。但它的的确确是一种通用语言，而且在诸多领域（如图像处理、移动应用和机器学习）中都能发现它的身影。它在很多场合下被用于替换无类型的脚本语言，这是由于它平衡了表达力和安全性：Go 程序通常比动态语言程序运行速度要快，遭遇意料之外的类型错误而导致的崩溃更是少得多。

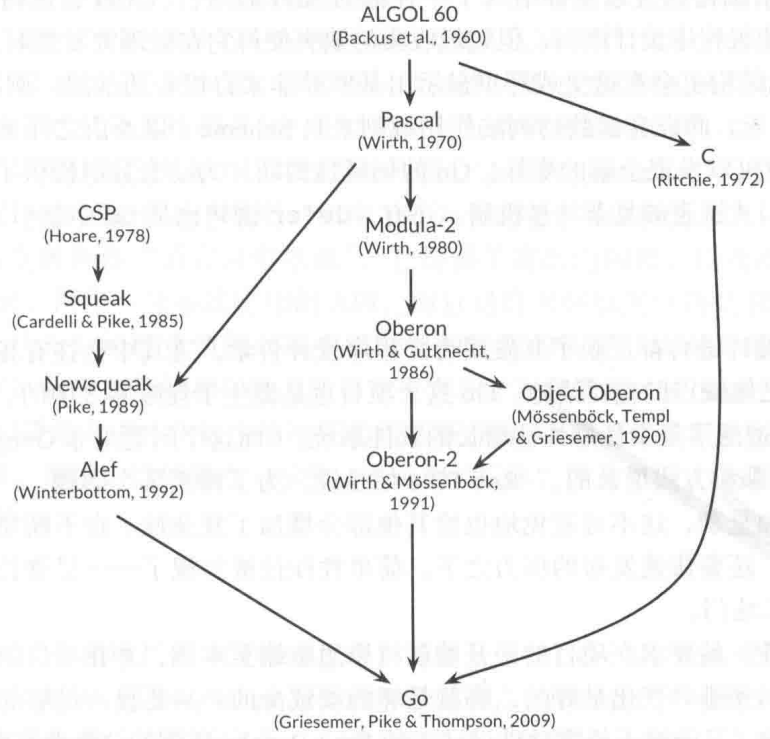
Go 是个开源项目，所以其编译器、库和工具的源代码是所有人皆可免费取得的。来自全世界的社区都在积极地向这个项目贡献代码。Go 的运行环境包括类 UNIX 系统——Linux、FreeBSD、OpenBSD 和 Mac OS X，还有 Plan 9 和 Microsoft Windows。只要在这些环境之一中写了一个程序，那么基本上不加修改就可以运行在其他环境中。

本书旨在帮助你立刻开始使用 Go，以及良好地掌握这门语言，并充分地利用 Go 的语言特性和标准库来撰写清晰的、符合习惯用法的、高效的程序。

Go 的起源

和生物学物种一样，成功的语言会繁衍后代，这些后代语言会从它们的祖先那里汲取各种优点；有时候，语言间的混血会产生异常强大的力量；在一些罕见情况下，某个重大的语言特性也可能凭空出现而并无先例。通过考察语言间的影响，我们可以学得不少知识，比如语言为什么会变成这个样子，以及它适配过哪些环境，等等。

下图展示了更早出现的程序设计语言对 Go 产生的最重要影响。



Go 有时会被称为“类 C 语言”或“21 世纪的 C”。从 C 中，Go 继承了表达式语法、控制流语句、基本数据类型、按值调用的形参传递、指针，但比这些更重要的是，继承了 C 所强调的程序要编译成高效的机器码，并自然地与所处的操作系统提供的抽象机制相配合。

可是，Go 的家谱中还有其他祖先。产生主要影响的是来自 Niklaus Wirth 设计的、以 Pascal 为发端的一个语言支流。Modula-2 启发了包概念。Oberon 消除了模块接口文件和模块实现文件的区隔。Oberon-2 影响了包、导入和声明语法。Object Oberon 则提供了方法声明语法。

另一支 Go 的世系祖先，也是使得 Go 相对于当下的程序设计显得卓然不群者，是一族名不见经传的、在贝尔实验室开发的研究用语言。这些语言都受到了 Tony Hoare 于 1978 年发表的关于并发性基础的开创性论文所提出的通信顺序进程（Communicating Sequential Process，CSP）的启发。在 CSP 中，程序就是一组无共享状态进程的并行组合，进程间的通信和同步采用信道完成。不过，Hoare 提出的 CSP 是一种形式语言，仅用以描述并发性的基本概念，并不是一种用来撰写可执行程序的程序设计语言。

Rob Pike 等人开始动手做一些实验，尝试把 CSP 实现为真正的语言。第一种这样的语言称为 Squeak（“和鼠类沟通的语言”）[⊖]，它是一种用以处理鼠标和键盘事件的语言。紧接着它的是 Newsqueak，它带来的是类 C 的语句和表达式语法，以及类 Pascal 的类型记法。它是一种纯粹的函数式语言，带有垃圾收集，同样也以管理键盘、鼠标和窗口事件为目标。信道成了一等值（first-class value），它可以动态创建并采用变量存储。

⊖ 该单词直译为（老鼠的）吱吱叫声，是为隐喻和双关。——译者注

Plan 9 操作系统将这些思想都纳入了一种称为 Alef 的语言。Alef 尝试将 Newsqueak 改造成一种可用的系统级程序设计语言，但垃圾收集的缺失使得它在处理并发性时太过痛苦了。

Go 中的其他结构也会在这里或那里显示出某些并非来自祖先的基因。例如，`iota` 多多少少有点 APL 的影子，而嵌套函数的词法作用域则来自 Scheme（以及由之而来的大部分语言）。在 Go 语言中，也可以发现全新的变异。Go 的创新性的切片为动态数组提供了高效的随机访问的同时，也允许旧式链表的复杂共享机制。还有，`defer` 语句也是 Go 中新引入的。

Go 项目

所有的程序设计语言都反映了其发明者的程序设计哲学，这其中往往有相当大的一部分是对于此前的语言已知缺点的应对措施。Go 这个项目也是诞生于挫败感之中的，这种挫败感来源于 Google 的若干遭遇了复杂性爆炸性增长的软件系统。（而这个问题绝非 Google 所独有的。）

“复杂性是以乘积方式增长的。”Rob Pike 如是说。为了修复某个问题，一点点地将系统的某个部分变得更加复杂，这不可避免地也给其他部分增加了复杂性。在不断要求增加系统功能和选项以及配置，还要快速发布的压力之下，简单性往往被忽视了——尽管长期来看，简单性才是好软件的不二法门。

要实现简单性，就要求在项目的一开始就将思想浓缩至本质，并在项目的整个生命期多保持一些自律，认识到哪些变化是好的，哪些是坏的或致命的。只要投入足够的努力，好的变化就既可以实现目的，又能够不损害软件设计上的 Fred Brooks 所谓的“概念完整性”[⊖]。坏的变化就做不到这一点，而致命的变化则会牺牲“简单性”而去换得其浅薄的兄弟概念——“方便性”。但是，只有通过设计上的简单性，系统才能在增长过程中保持稳定、安全和自洽。

Go 项目不仅仅包括语言本身、它的工具和标准库，还有决不能忽视的一点，就是它保持极端简单性的行为文化。在高级语言中，Go 出现得较晚，因而有一定后发优势，它的基础部分实现得不错：有垃圾收集、包系统、一等函数、词法作用域、系统调用接口，还有不可变的、默认用 UTF-8 编码的字符串。但相对来说，它的语言特性不多，而且不太会增加新特性了。比如说，它没有隐式数值类型强制转换，没有构造或析构函数，没有运算符重载，没有形参默认值，没有继承，没有泛型，没有异常，没有宏，没有函数注记，没有线程局部存储。这门语言成熟而稳定，并且保证兼容更早版本：在旧版本的 Go 语言中写的程序，可以在新版本的编译器和标准库下编译和运行。

Go 的类型系统足可以使程序员避免在动态语言中会无意犯下的绝大多数错误，但相对而言，它在带类型的语言中又算是类型系统比较简单的。其实现手法有时候会导致类型框架林立却彼此孤立的“无类型”程序设计风格，并且 Go 程序员不会在类型方面走到 C++ 或 Haskell 程序员的那一步——重度地表达类型安全性作为语言具备类型基础的证据。但在实际工作中，Go 却能为程序员提供具备相当强类型的系统才能实现的安全性和运行时性能，而不让程序员承担这种系统的复杂性。

Go 提倡充分利用当代计算机系统设计，尤其强调局部性的重要。其内建数据类型和大多

⊖ 见《人月神话》。——译者注

数库数据结构都经过仔细设计，力求以自然方式工作，而不要求显式的初始化或隐式的构造函数。这么一来，隐藏在代码中的内存分配和内存写入就大大减少了。Go 中的聚合类型（结构体和数组）都是以直接方式持有其元素，需要更少的存储以及比使用间接域的语言还少的分配动作和间接指针。由于现代计算机都是并行工作的，正如前面提到的那样，Go 有着基于 CSP 的并行特性。Go 还提供了变长栈来运行其轻量级线程，或称为 goroutine。这个栈初始化时非常小，所以创建一个 goroutine 的成本极低，创建 100 万个也完全可以接受。

Go 标准库常常被称作“语言自带电池”，它提供了清晰的构件，以及用于 I/O、文本处理、图像、加解密、网络、分布式应用的 API，而且对许多标准文件格式和协议都提供了支持。Go 的库和工具充分地尊重惯例，减少了配置和解释的需要，从而简化了程序逻辑，提高了多种多样的 Go 程序之间的相似性，使得它更容易学习和掌握。采用 `go` 工具构建的项目，仅使用文件和标识符的名字（在极少情况下使用特殊注释），就可以推断出一个项目使用的所有库、可执行文件、测试、性能基准、示例、平台相关变体，以及文档。Go 的源代码中就包含了构建的规格说明。

全书组织

我们假定你用过一两种其他语言编过程序，可能是像 C、C++ 或 Java 那样的编译型语言，也可能是像 Python、Ruby 或 JavaScript 那样的解释型语言，所以我们不会像对一个没有任何基础的初学者那样事无巨细地讲述所有事。表面上的语法大体雷同，变量、常量、表达式、控制流和函数也一样。

第 1 章是 Go 的基础结构的综述，通过十几个完成日常任务（包括读写文件、格式化文本、创建图像，以及在 Internet 客户端和服务器之间通信）的程序来介绍这门语言。

第 2 章讲述 Go 程序的组成元素——声明、变量、新类型、包和文件，以及作用域。第 3 章讨论数值、布尔量、字符串、常量，还解释如何处理 Unicode。第 4 章描述组合类型，即使用简单类型构造的类型，形式有数组、键值对、结构体，还有切片这种 Go 中动态表的实现。第 5 章的内容是函数，另外还讨论了错误处理、崩溃和恢复，以及 `defer` 语句。

可以看出，第 1 章到第 5 章是基础性的，其内容是任何命令式语言都有的。Go 的语法和风格可能与其他语言有所不同，但大多数程序员都能很快掌握。余下的章节所关注的则是 Go 语言中与惯常做法有一定区别的内容，包括方法、接口、并发、包、测试和反射。

Go 以一种不同寻常的方式来诠释面向对象程序设计。它没有类继承，甚至没有类。较复杂的对象行为是通过较简单的对象组合而非继承完成的。方法可以关联到任何用户自定义的类型，而不一定是结构体。具体类型和抽象类型（即接口）之间的关系是隐式的，所以一个具体类型可能会满足该类型设计者没有意识到其存在的接口。方法在第 6 章讲述，接口在第 7 章讲述。

第 8 章内容是 Go 的并发性处理途径，它基于 CSP 思想，采用 goroutine 和信道实现。第 9 章则讨论并发性中较传统的基于共享变量的一些方面的话题。

第 10 章讨论包，也就是组织库的机制。本章也说明如何高效地利用 `go` 工具，仅仅这个工具，就提供了编译、测试、性能基准测定、程序格式化、文档，以及完成许多其他任务的功能。

第 11 章讨论测试，在这里 Go 采取了显著的轻量级途径，避免了重重抽象的框架，转而使用简单的库和工具。测试库提供了一个基础，在其之上如有必要就可以构建更复杂的抽象了。

第 12 章讨论反射，即程序在执行期间考察自身表示的能力。反射是一种强大的工具，不过得小心使用它，本章通过演示如何用它来实现某些重要的 Go 库，解释了如何找到适当的平衡。第 13 章解释低阶程序设计的细节，运用 unsafe 包来绕过 Go 的类型系统，以及什么时候这样做是合适的。

每章都配以一定数量的练习，你可以用来测试对 Go 的理解，或是探索对书中示例的扩展和变形。

除了最简单的那些以外，所有书中的示例代码都可以从位于 gopl.io 的公开 Git 仓库下载。每个示例由其包的导入路径识别，从而能够方便地使用 go get 命令获取、构建和安装。你需要选取一个目录作为你的 Go 工作空间，并将 GOPATH 环境变量指向之。在必要时，go 工具会创建该目录。例如：

```
$ export GOPATH=$HOME/gobook          # choose workspace directory
$ go get gopl.io/ch1/helloworld       # fetch, build, install
$ $GOPATH/bin/helloworld              # run
Hello, 世界
```

欲运行这些例子，你至少需要使用 1.5 版本的 Go 语言。

```
$ go version
go version go1.5 linux/amd64
```

如果你的计算机上的 go 工具太旧或是缺失，请按 https://golang.org/doc/install 的指令操作。

更多信息来源

关于 Go 的更多信息，最好的来源就是 Go 的官方网站：https://golang.org，它上面列出了文档供读者访问，包括 Go 程序设计语言规范、标准包，诸如此类。它上面还列出语言综述，指导如何撰写 Go 程序，以及如何撰写好的 Go 程序，还有范围广泛的在线文本和视频资源，这些都是对本书的有价值的补充。位于 blog.golang.org 的 Go 博客发表的是关于 Go 的最好文章，以及该语言当下状态、未来计划、会议报告，还有范围广泛的 Go 相关话题的深度解读。

Go 官网在线访问的一个最有用的方面（这也是纸质书的一个令人遗憾的限制），就是它提供了从描述 Go 程序的网页上直接运行 Go 程序的能力。这种功能由位于 play.golang.org 的 Go 训练场提供，也有嵌入其他页面的，比如位于 golang.org 的首页，或是由 godoc 工具提供的文档页面。

训练场为执行简单的实验，撰写短小的程序来检验自己对语法、语义和库包的理解提供了方便，它在很多方面取代了其他语言中的*读取 – 求值 – 输出循环*（Read-Eval-Print Loop，REPL）。它的永久 URL 对于共享使用 Go 写成的小段代码、报告缺陷或提出建议都是很有用的。

在训练场的基础之上，位于 tour.golang.org 的 Go 综述就是一系列的小型交互式课程，内容是 Go 语言的基础思想和结构，是整门语言的有序攻略。

训练场和综述的主要缺点在于它只允许导入标准库，还有很多库特性（比如网络库）都出于可操作性或安全原因加以限制了。而要编译和运行每个程序，都要求 Internet 连接。所以，欲进行更详尽的实验，需要在本机上运行 Go 程序。幸运的是，下载过程相当直截了当，从 `golang.org` 获取 Go 的分发程序并开始撰写和运行你自己的 Go 程序，用不了几分钟。

由于 Go 是个开源项目，你可以从 `https://golang.org/pkg` 上在线读取标准库中的任何类型或函数的代码，每个供下载的分发都同样包含这些代码。请使用这些代码来弄明白某些程序的运行原理、回答关于程序细节的问题，也可以用它们来学一学专家是如何写出一流的 Go 代码的。

致谢

来自 Go 团队的核心成员 Rob Pike 和 Russ Cox 仔细通读了初稿数次，他们对于从词句的斟酌到整体结构和组织的建议都极富价值。在准备本书的日语版时，柴田芳树所做的贡献大大超过了他负担的义务，他的火眼金睛发现了大量英语文本中的不一致性，以及代码中的错误。我们向 Brian Goetz、Corey Kosak、Arnold Robbins、Josh Bleecher Snyder 以及 Peter Weinberger 对全书初稿所进行的彻底审查和批评建议深表敬意。

我们感激 Sameer Ajmani、Ittai Balaban、David Crawshaw、Billy Donohue、Jonathan Feinberg、Andrew Gerrand、Robert Griesemer、John Linderman、Minux Ma、Bryan Mills、Bala Natarajan、Cosmos Nicolaou、Paul Staniforth、Nigel Tao 以及 Howard Trickey 提供的诸多有用建议。我们也感谢 David Brailsford 和 Raph Levien 的排版建议。

我们的 Addison-Wesley 编辑 Greg Doench 最初催生了本书，而且一直不断地给予帮助。Addison-Wesley 的制作团队——John Fuller、Dayna Isley、Julie Nahil、Chuti Prasertsith 以及 Barbara Wood——非常杰出，作者得到了一流的支持。

Alan Donovan 想要感谢 Google 的 Sameer Ajmani、Chris Demetriou、Walt Drummond 以及 Reid Tatge 让他有时间来写作这本书，还有 Stephen Donovan 的建议和及时的鼓励。最重要的是他的妻子 Leila Kazemi，她为写作项目提供了毫不犹豫的热情和毫不动摇的支持，即使写作造成了很多家庭生活的分心和缺席。

Brian Kernighan 对他的朋友和同事深表谢意，他们对 Kernighan 花费了很长时间才达成对语言的理解表示了耐心和理解。尤其是他的妻子 Meg，她为 Kernighan 的写作以及太多的其他事务提供了不间断的支持。

纽约

2015 年 10 月

Contents

目　录

1 Tutorial

This chapter is a tour of the basic components of Go. We hope to provide enough information and examples to get you off the ground and doing useful things as quickly as possible. The examples here, and indeed in the whole book, are aimed at tasks that you might have to do in the real world. In this chapter we'll try to give you a taste of the diversity of programs that one might write in Go, ranging from simple file processing and a bit of graphics to concurrent Internet clients and servers. We certainly won't explain everything in the first chapter, but studying such programs in a new language can be an effective way to get started.

When you're learning a new language, there's a natural tendency to write code as you would have written it in a language you already know. Be aware of this bias as you learn Go and try to avoid it. We've tried to illustrate and explain how to write good Go, so use the code here as a guide when you're writing your own.

1.1. Hello, World

We'll start with the now-traditional "hello, world" example, which appears at the beginning of *The C Programming Language*, published in 1978. C is one of the most direct influences on Go, and "hello, world" illustrates a number of central ideas.

gopl.io/ch1/helloworld

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, 世界")
}
```

Go is a compiled language. The Go toolchain converts a source program and the things it depends on into instructions in the native machine language of a computer. These tools are accessed through a single command called `go` that has a number of subcommands. The simplest of these subcommands is `run`, which compiles the source code from one or more source files whose names end in `.go`, links it with libraries, then runs the resulting executable file. (We will use `$` as the command prompt throughout the book.)

```
$ go run helloworld.go
```

Not surprisingly, this prints

```
Hello, 世界
```

Go natively handles Unicode, so it can process text in all the world's languages.

If the program is more than a one-shot experiment, it's likely that you would want to compile it once and save the compiled result for later use. That is done with `go build`:

```
$ go build helloworld.go
```

This creates an executable binary file called `helloworld` that can be run any time without further processing:

```
$ ./helloworld
Hello, 世界
```

We have labeled each significant example as a reminder that you can obtain the code from the book's source code repository at `gopl.io`:

```
gopl.io/ch1/helloworld
```

If you run `go get gopl.io/ch1/helloworld`, it will fetch the source code and place it in the corresponding directory. There's more about this topic in Section 2.6 and Section 10.7.

Let's now talk about the program itself. Go code is organized into packages, which are similar to libraries or modules in other languages. A package consists of one or more `.go` source files in a single directory that define what the package does. Each source file begins with a `package` declaration, here `package main`, that states which package the file belongs to, followed by a list of other packages that it imports, and then the declarations of the program that are stored in that file.

The Go standard library has over 100 packages for common tasks like input and output, sorting, and text manipulation. For instance, the `fmt` package contains functions for printing formatted output and scanning input. `Println` is one of the basic output functions in `fmt`; it prints one or more values, separated by spaces, with a newline character at the end so that the values appear as a single line of output.

Package `main` is special. It defines a standalone executable program, not a library. Within package `main` the *function* `main` is also special—it's where execution of the program begins. Whatever `main` does is what the program does. Of course, `main` will normally call upon functions in other packages to do much of the work, such as the function `fmt.Println`.