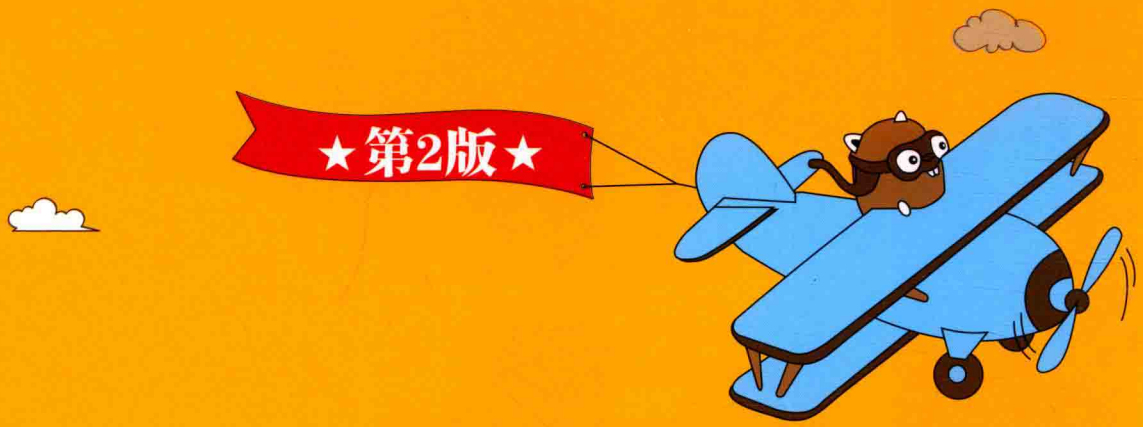
TURING 图灵原创



GO

并发编程实战

郝林◎著

中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

站　在　巨　人　肩　上

Standing on Shoulders of Giants

图书在版编目（CIP）数据

Go并发编程实战 / 郝林著. -- 2版. -- 北京 : 人民邮电出版社, 2017.4
（图灵原创）
ISBN 978-7-115-45251-1

Ⅰ. ①G… Ⅱ. ①郝… Ⅲ. ①程序语言－程序设计
Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2017)第052901号

内 容 提 要

本书首先介绍了Go语言的优秀特性、安装设置方法、工程结构、标准命令和工具、语法基础、数据类型以及流程控制方法，接着阐述了与多进程编程和多线程编程有关的知识，然后重点介绍了goroutine、channel以及Go提供的传统同步方法，最后通过一个完整实例——网络爬虫框架进一步阐述Go语言的哲学和理念，同时分享作者在多年编程生涯中的一些见解和感悟。

与上一版相比，本书不仅基于Go 1.8进行了全面更新，而且更深入地描绘了Go运行时系统的内部机理，并且大幅改进了示例代码。

本书适用于有一定计算机编程基础的从业者以及对Go语言编程感兴趣的爱好者，非常适合作为Go语言编程进阶教程。

◆ 著　　郝　林
责任编辑　王军花
责任印制　彭志环
◆ 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：23.75
字数：472千字　　2017年4月第2版
印数：8 501－12 500册　　2017年4月河北第1次印刷

定价：79.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广字第8052号

序

Go 是年轻而有活力的语言。

它最初于 2007 年由 Robert Griesemer、Rob Pike 和 Ken Thompson 在 Google 开始开发，2009 年正式发布。作者们希望 Go 能使复杂、高效系统的编写工作变得简单、可靠；同时，也希望 Go 能成为一个相对通用的编程环境，适应诸如桌面应用、移动应用、数值计算等。

Go 的设计理念充分体现了这些设计目标。它是极简化语言的代表，推崇少即是多。为了避免复杂、不可读的代码，Go 限制了语言功能与语法特性。Go 的可读性在众多编程语言中是独树一帜的。另外，为了减轻使用者编写高性能应用的负担，它也引入了 runtime，提供了诸如协程、垃圾回收等功能。runtime 虽然使语言本身的实现更复杂，但它让使用者获得了更简单易用的编程环境。

Go 语言是极易掌握的语言，它与 C 语言十分相近。熟悉 C、C++等语言的编程人员可以在短时间内掌握 Go 语言来编写简单、高效的应用。它只有 20 多个语言关键词。作为初学者，它也是相对容易入门的语言。

国内的 Go 语言社区十分活跃，这得益于致力推广 Go 的技术精英们。我认识本书作者郝林，也是源于他组织的 Go 语言北京交流会。利用业余时间，他广泛推广普及 Go 语言，组织、邀请技术专家参与交流会。他坚持不懈两年有余，取得了显著的成绩。郝林对 Go 社区建设的执着与热情令人敬佩。我相信，本书也是凝集了他对技术推广的一腔热情，希望让 Go 语言的初学者、工程师们能更快捷、深入地理解 Go 语言，以促进整个技术领域的发展。

Go 语言方面的图书对培养高素质的业余爱好者、从业人员起到了至关重要的作用。本书在各种 Go 语言图书中也是特点鲜明。本书首先介绍了 Go 语言的基础知识，对初学者有所铺垫。书中大量篇幅覆盖了 Go 语言的并发特性，详细讲解了其中的哲学、原理与实现。我相信很多像我这样，每天都沉浸在 Go 语言的从业人员，也并不完全知道 Go 内部实现的奥妙。每天花上一些时间来读此书，即便对有经验的 Go 从业人员来说，也会有所帮助。

在翻读本书时，我也深深体会到了作者写作的用心之处，每章不光有概念的讲解，还有

实现实例和经典案例。这些细心之处，让这样一本严肃的技术书读起来并不枯燥、乏味。书末更有独立的一章来介绍用 Go 语言实现的一个爬虫系统。相信很多读者都会迫不及待地跟着作者一起动起手来，实践书中的知识与概念。

最后，作为 Go 社区和开源社区的一员，我希望读者们能够在享受 Go 开发带来的乐趣与收获的同时，能够回馈、融入社区。你们的每一个建议与意见，每一个问题反馈与代码补丁，都会促进和推动开源社区，以及整个计算机产业的发展。我想这也是郝林如此用心编写此书的一个初衷。

李响，CoreOS 分布式系统组主管（Head of distributed systems）
2017 年 3 月 5 日，于美国加利福尼亚州

前　言

很高兴你能选择这本已经过大量改进的书，希望本书能够让你成为真正的 Go 粉。很多 Go 语言爱好者都喜欢称自己是 Gopher，这是一个来自官方的传统，希望你也能这样称呼自己。

Go 编程语言（或称 Golang，以下简称 Go 语言）是云计算时代的 C 语言。7 年过去了，它渐渐向世人证明此言不虚。如果你关注 TIOBE 的编程语言排行榜就会发现，Go 语言从前些年的第 50 多位，经过多次上窜已经跃居第 13 位，跻身绝对主流的编程语言行列！同时，它还被评为 2016 年的年度语言！经过了数年的不断改进，Go 语言在开发效率和程序运行效率方面又上了数个台阶。

下面我简单列举一下它在最近两年比较显而易见的变化。

- **本身的自举**。也就是说，Go 语言几乎完全用 Go 语言程序重写了自己，仅留有一些汇编程序。Go 语言的自举非常彻底，包括了最核心的编译器、链接器、运行时系统等。现在任何学习 Go 语言的人都可以直接读它的源代码了。此变化也使 Go 程序的跨平台编译变得轻而易举。
- **运行时系统的改进**。这主要体现在更高效的调度器、内存管理以及垃圾回收方面。调度器已能让 goroutine 更及时地获得运行时机。运行时系统对内存的利用和控制也更加精细了。因垃圾回收而产生的调度停顿时间已经小于原来的 1‰。另外，最大 P 数量的默认值由原先的 1 变为与当前计算机的 CPU 核心数相同。
- **标准工具的增强**。在 Go 1.4 加入 go generate 之后，一个惊艳的程序调试工具 go tool trace 也被添加进来了。另外，go tool compile、go tool asm 和 go tool link 等工具也已到位；一旦你安装好 Go，就可以直接使用它们。同时，几乎所有的标准工具和命令都得到了不同程度的改进。
- **访问控制的细化**。这种细化始于 Go 1.4，正式支持始于 Go 1.5，至今已被广泛应用。经过细化，对于 Go 程序中的程序实体，除了原先的两种访问控制级别（公开和包级私有）之外，又多了一种——模块级私有。这是通过把名称首字母大写的程序实体放入 internal 代码包实现的。

- **vendor 机制的支持**。自 Go 1.5 之后，一个特殊的目录——vendor——被逐渐启用。它用于存放其父目录中的代码包所依赖的那些代码包。在程序被编译时，编译器会优先引用存于其中的代码包。这为固化程序的依赖代码迈出了很重要的一步。在 Go 1.7 中，vendor 目录以及背后的机制被正式支持。

当然，上述变化并不是全部。它的标准库也经历了超多的功能和性能改进。如果你是在本书的第 1 版面市时开始学习 Go 语言的，那么一定能感受到这些变化带来的巨大红利。

从本书第 1 版出版至今，Go 语言的版本号已经从 1.4 升至 1.8，本书第 2 版就是基于 Go 1.8 写成的。

本书第 2 版根据 Go 语言本身的变化以及第 1 版读者的大量反馈做了很多改进，也重写了非常多的内容。你的第一感觉肯定是书变薄、变轻了！没错，最直观的改进就是书中几乎每个章节都更为精炼。在讲 Go 编程基础的时候，我只说明重点，并尽量用代码和表格代替文字，同时让它们变得易于速查。因此，本书在这方面的篇幅由 5 章缩减为了 2 章。然而，讲并发编程理念、方法和实战的部分却得到了适当的扩充。你可能已经发现，后面几章的内容非常多，这是因为我更加细致地讲解了那些核心知识。

同时，本书的所有示例程序都重新编排，变得更加有条理，更加容易查找。当然，我对示例程序本身也做了相当大的改造。首先是充分使用 Go 语言的新特性。比如，有的代码包拥有了自己的 internal 包。又比如，示例项目本身也用到了 vendor 目录。其次，更多的优质代码包被引入进来并充分利用，比如 io/ioutil、context 等。另外，还对一些关键示例进行了脱胎换骨的改写。比如，完全重写了 ConcurrentMap，它的性能比原先的版本高出数倍。又比如，我对网络爬虫框架做了大幅更新，既包括较为底层的数据结构，也包括调度器以及一些模块的接口和实现。这使得它更易用、扩展性更好，同时代码中体现的技巧和理念也更多、更突出。因此，本书最后一章也几乎完全重写了。

本书结构

本书共分为 6 章。

第 1 章，快速介绍了 Go 语言的优秀特性、安装设置方法、工程结构以及标准命令和工具。

第 2 章，讲述了 Go 语言的语法基础、数据类型以及流程控制方法。

第 3 章，主要阐述了与多进程编程和多线程编程有关的知识。这些知识作为理解 Go 语言并发编程模型的先导内容。看过以后，你就可以对并发编程有一个比较清晰的理解。之后，本章还简要剖析了多核时代（基于多 CPU 核心的计算时代）的并发编程需求。

第 4 章，在深入展示和说明 Go 语言的并发编程模型之后，本章讲解 Go 特有的编程要素——goroutine（也可称为 Go 例程）的用法以及背后的运作机理。此外，本章还会对 Go 并发编程中另一个不可或缺的部分——channel 进行重点介绍，包括概念、使用规则以及应用技巧。

第 5 章，会对 Go 语言提供的传统同步方法进行介绍。这包括互斥锁、条件变量、原子操作、WaitGroup、临时对象池等，这些同步方法大多是标准库代码包 sync 中的一员。虽然 Go 语言官方并不建议优先使用这些方式来保障程序的并发安全性，但不容忽视的是，它们在一些应用场景中确实简单有效。

第 6 章，包括一个基本囊括了本书所有概念和知识的完整示例——网络爬虫框架。我会带你逐步编写这个示例，并进一步阐述 Go 语言的哲学和理念，同时分享我在多年编程生涯中的一些见解和感悟。你可以通过这个示例来巩固前面学到的 Go 语言知识，并加深对 Go 并发编程的理解。

附录 A，简单介绍目前在国内外比较活跃的一部分 Go 语言开源项目和 Go 语言社区，这会使你学习 Go 语言的道路变得更加顺畅，也有利于你找到志同道合的朋友。

目标读者

原则上来讲，任何对计算机编程和 Go 语言感兴趣的人都可以阅读本书。但是，当你学习一门编程语言的时候，往往还是需要有一些基础的。比如，怎样使用文本编辑器、怎样在相应的操作系统中安装软件，等等。而要想成为一名高级的 Go 软件工程师，你需要了解的知识可能比表面看上去的多很多。这就像摘苹果一样，如果要摘到苹果，就需要徒手爬上果树，或者找到足够高的梯子；如果想摘到果树顶端最甜的那个苹果，就需要花费更多的时间和精力，爬过更多的枝叶。希望本书能成为帮助你摘苹果的梯子。但是，在想有所收获之前，请先潜心学习和积累。

当然，你在阅读本书的过程中边看边学也完全没问题，甚至可以看完本书再去学习相关知识。采用哪种学习方式，这完全取决于你自己。

关于示例代码

我会把本书涉及的示例代码[①]都放到一个名为 example.v2 的项目中，你可以访问

① 本书源代码也可去图灵社区（iTuring.cn）本书主页免费注册下载。

https://github.com/gopcp/example.v2 查看或下载。你存放该项目目录的绝对路径应该包含在环境变量 GOPATH 中。如果你不了解 Git（一款代码版本控制工具），请在网上搜索“git”并详细了解。

关于勘误

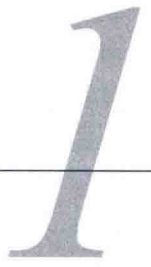
由于作者水平和时间有限，书中难免会有一些纰漏和错误，欢迎读者及时指正，本书后续印次或版本中将加以改正。非常希望和大家一起学习和讨论 Go 语言，并共同推动 Go 语言在中国的发展。你可以通过电子邮件（hypermind@outlook.com）联系我，也可以到图灵社区（iTuring.cn）本书主页上发表评论。

致谢

写书是一项需要大量精力和毅力的工作，尤其是编写技术图书，更需要作者对相关知识进行深入的梳理和系统的整合，还需要制作各种图表，编写各种示例，工作量确实不小。但是，这个写作过程也很有趣，我通过写作也收获了很多。当然，很多收获来自他人的传授。其中，图灵公司的编辑王军花和傅志红老师都给予了我很大的帮助，尤其是在写作技巧和图书结构方面。当然，还有目前中国业内公认的 Go 语言专家们，我在编写本书的时候经常向他们讨教。在此，谨对帮助过我的所有人表示由衷的感谢。同时也要感谢我的家人，没有他们的支持和理解，我不可能在有限的业余时间里完成本书。

目　录

第 1 章 初识 Go 语言



在讲解怎样用 Go 语言之前，我们先介绍 Go 语言的特性、基础概念和标准命令。

1.1 语言特性

我们可以用几个关键词或短语来概括 Go 语言的主要特性。

- **开放源代码**。这显示了Go作者开放的态度以及营造语言生态的决心。顺便说一句，Go本身就是用Go语言编写的。
- **静态类型和编译型**。在Go中，每个变量或常量都必须在声明时指定类型，且不可改变。另外，程序必须通过编译生成归档文件或可执行文件，而后才能被使用或执行。不过，其语法非常简洁，就像一些解释型脚本语言那样，易学易用。
- **跨平台**。这主要是指跨计算架构和操作系统。目前，它已经支持绝大部分主流的计算架构和操作系统，并且这个范围还在不断扩大。只要下载与之对应的Go语言安装包，并且经过简单的安装和设置，就可以使Go就绪了。除此之外，在编写Go语言程序的过程中，我们几乎感觉不到不同平台的差异。
- **自动垃圾回收**。程序在运行过程中的垃圾回收工作一般由Go运行时系统全权负责。不过，Go也允许我们对此项工作进行干预。
- **原生的并发编程**。拥有自己的并发编程模型，其主要组成部分有goroutine（也可称为Go例程）和channel（也可称为通道）。另外，还拥有一个特殊的关键字go。
- **完善的构建工具**。它自带了很多强大的命令和工具，通过它们，可以很轻松地完成Go程序的获取、编译、测试、安装、运行、分析等一系列工作。
- **多编程范式**。Go支持函数式编程。函数类型为第一等类型，可以方便地传递和赋值。此外，它还支持面向对象编程，有接口类型与实现类型的概念，但用嵌入替代了继承。
- **代码风格强制统一**。Go安装包中有自己的代码格式化工具，可以用来统一程序的

编码风格。

- **高效的编程和运行**。Go简洁、直接的语法使我们可以快速编写程序。加之它强大的运行时系统，程序可以充分利用计算环境飞快运行。
- **丰富的标准库**。Go是通用的编程语言，其标准库中有很多开箱即用的API。尤其是在编写诸如系统级程序、Web程序和分布式程序时，我们几乎无需依赖第三方库。

看到Go如此多的先进特性后，你是否已经心动了？反正我已经为此折服并感到非常激动。这正是我迫切需要的语言！这也正是我迫不及待地深入研究它，并通过一本书把我知道的所有细节分享给大家的原因。

1.2　安装和设置

安装Go相当简单，只要你的操作系统被Go语言支持即可。Go语言官方网站放出的每一个版本都会有主流平台的二进制安装包以及源码包，你可以自行挑选对应的文件下载。Go的下载地址为：https://golang.org/dl。

本节，我们以64位的Linux操作系统为计算环境，简要描述安装和设置过程。

假设已经从Go的下载地址中挑选了go1.8.linux-amd64.tar.gz文件并下载。注意，这个文件的名称中有两个关键词，一个是linux，一个是amd64。前者表示该文件对应的操作系统种类，其他的同类词有darwin、freebsd、windows等。后者表示该文件对应的计算架构种类，其中amd64表示64位的计算架构；另一个同类词是386，表示32位的计算架构。实际上，这里的计算架构和操作系统可以统称为Go的计算平台或计算环境。

解压该文件时会得到一个名为go的文件夹，其中包含所有的Go语言相关文件。该文件夹下还有很多文件夹和文件，下面简要说明其中主要文件夹的功用。

- **api文件夹**。用于存放依照Go版本顺序的API增量列表文件。这里所说的API包含公开的变量、常量、函数等。这些API增量列表文件用于Go语言API检查。
- **bin文件夹**。用于存放主要的标准命令文件，包括go、godoc和gofmt。
- **blog文件夹**。用于存放官方博客中的所有文章，这些文章都是Markdown格式的。
- **doc文件夹**。用于存放标准库的HTML格式的程序文档。我们可以通过godoc命令启动一个Web程序展现这些文档。
- **lib文件夹**。用于存放一些特殊的库文件。
- **misc文件夹**。用于存放一些辅助类的说明和工具。

- pkg文件夹。用于存放安装Go标准库后的所有归档文件。注意，你会发现其中有名称为linux_amd64的文件夹，我们称为平台相关目录。可以看到，这类文件夹的名称由对应的操作系统和计算架构的名称组合而成。通过go install命令，Go程序（这里是标准库中的程序）会被编译成平台相关的归档文件并存放到其中。另外，pkg/tool/linux_amd64文件夹存放了使用Go制作软件时用到的很多强大命令和工具。
- src文件夹。用于存放Go自身、Go标准工具以及标准库的所有源码文件。深入探究Go，就靠它了。
- test文件夹。存放用来测试和验证Go本身的所有相关文件。

现在，你已经大致了解了 go 文件夹中的目录结构及其用途。下面我们把这个 go 文件夹放到一个合适的目录中。对于 Linux 操作系统，Go 官方推荐的目录是/usr/local。

在这之后，我们需要设置一个环境变量，即：GOROOT。GOROOT 的值应该是 Go 的根目录。这里是/usr/local/go。另外，环境变量中 PATH 中也应该增加一项，即$GOROOT/bin。这样就可以在任意目录下使用那几个 Go 命令了。

至此，Go 已经安装好了。下面重点了解一下 Go 中的一些基础概念。

1.3 工程结构

Go 是一门推崇软件工程理念的编程语言，它为开发周期的每个环节都提供了完备的工具和支持。Go 语言高度强调代码和项目的规范和统一，这集中体现在工程结构或者说代码体制的细节之处。Go 也是一门开放的语言，它本身就是开源软件。更重要的是，它让开发人员很容易通过 go get 命令从各种公共代码库（比如著名的代码托管网站 GitHub）中下载开源代码并使用。这除了得益于 Go 自带命令的强大之外，还应该归功于 Go 工程结构的严谨和完善。本节中，我们详述 Go 的工程结构。

1.3.1 工作区

一般情况下，Go 源码文件必须放在工作区中。但是对于命令源码文件来说，这不是必需的。工作区其实就是一个对应于特定工程的目录，它应包含 3 个子目录：src 目录、pkg 目录和 bin 目录，下面逐一说明。

- src目录。用于以代码包的形式组织并保存Go源码文件，这里的代码包与src下的子目录一一对应。例如，若一个源码文件被声明属于代码包log，那么它就应当保

存在src/log目录中。当然，你也可以把Go源码文件直接放在src目录下，但这样的Go源码文件就只能被声明属于main代码包了。除非用于临时测试或演示，一般还是建议把Go源码文件放入特定的代码包中。

- **pkg目录**。用于存放通过go install命令安装后的代码包的归档文件。前提是代码包中必须包含Go库源码文件。归档文件是指那些名称以“.a”结尾的文件。该目录与GOROOT目录下的pkg目录功能类似。区别在于，工作区中的pkg目录专门用来存放用户代码的归档文件。编译和安装用户代码的过程一般会以代码包为单位进行。比如log包被编译安装后，将生成一个名为log.a的归档文件，并存放在当前工作区的pkg目录下的平台相关目录中。
- **bin目录**。与pkg目录类似，在通过go install命令完成安装后，保存由Go命令源码文件生成的可执行文件。在类Unix操作系统下，这个可执行文件一般来说名称与源码文件的主文件名相同。而在Windows操作系统下，这个可执行文件的名称则是源码文件主文件名加.exe后缀。

注意　这里有必要明确一下Go语言的命令源码文件和库源码文件的区别。所谓命令源码文件，就是声明属于main代码包并且包含无参数声明和结果声明的main函数的源码文件。这类源码文件是程序的入口，它们可以独立运行（使用go run命令），也可以通过go build或go install命令得到相应的可执行文件。所谓库源码文件，则是指存在于某个代码包中的普通源码文件。

1.3.2　GOPATH

我们需要将工作区的目录路径添加到环境变量GOPATH中。否则，即使处于同一工作区（事实上，未被加入GOPATH中的目录不应该称为工作区），代码之间也无法通过绝对代码包路径调用。在实际开发环境中，工作区可以只有一个，也可以有多个，这些工作区的目录路径都需要添加到GOPATH中。与GOROOT一样，我们应该确保GOPATH一直有效。

注意

- GOPATH中不要包含Go语言的根目录，以便将Go语言本身的工作区同用户工作区严格分开。
- 通过Go工具中的代码获取命令go get，可将指定项目的源码下载到我们在GOPATH中设定的第一个工作区中，并在其中完成编译和安装。

本书约定，示例项目会被存放在$HOME/golang/example.v2目录下，并且在该目录

的 src/gopcp.v2 子目录中存放示例代码。而示例代码依赖的第三方代码包则会被放入 $HOME/ golang/example.v2/src/gopcp.v2/vendor 目录。vendor 目录是一个特殊的目录，一般用于存放其他代码包目录中的源码文件依赖的那些代码包。vendor 目录及其机制并不复杂，它们的说明详见 https://docs.google.com/document/d/1Bz5-UB7g2uPBdOx-rw5t9MxJwkfpx90cqG9AFL0JAYo 和 https://golang.org/doc/go1.6#go_command。它在 Go 1.5 版本时被试验性地添加进 Go 命令，并在 Go 1.6 版本中变成默认选项。而到了 Go 1.7 版本，它已经成为 Go 命令的正式成员。

注意，你需要把 example.v2 所在的目录当成一个工作区目录，即需要把这个目录添加到 GOPATH 中。

1.3.3 源码文件

我为本书涉及的示例程序专门建立了项目 example.v2，可以访问 https://github.com/gopcp/example.v2 下载。不过，无论从哪里得到示例项目，你都需要将 example.v2 文件夹放到$HOME/golang 目录下，并设置好环境变量 GOPATH。

example.v2 项目包含了 Go 源码文件的所有 3 个种类，即命令源码文件、库源码文件和测试源码文件，下面详细说明这 3 类源码文件。

1. 命令源码文件

如果一个源码文件被声明属于 main 代码包，且该文件代码中包含无参数声明和结果声明的 main 函数，则它就是命令源码文件。命令源码文件可通过 go run 命令直接启动运行。

同一个代码包中的所有源码文件，其所属代码包的名称必须一致。如果命令源码文件和库源码文件处于同一个代码包中，那么在该包中就无法正确执行 go build 和 go install 命令。换句话说，这些源码文件将无法通过常规方法编译和安装。因此，命令源码文件通常会单独放在一个代码包中。这是合理且必要的，因为通常一个程序模块或软件的启动入口只有一个。

同一个代码包中可以有多个命令源码文件，可通过 go run 命令分别运行，但这会使 go build 和 go install 命令无法编译和安装该代码包。所以，我们也不应该把多个命令源码文件放在同一个代码包中。

当代码包中有且仅有一个命令源码文件时，在文件所在目录中执行 go build 命令，

即可在该目录下生成一个与目录同名的可执行文件；而若使用 go install 命令，则可在当前工作区的 bin 目录下生成相应的可执行文件。例如，代码包 gopcp.v2/helper/ds 中只有一个源码文件 showds.go，且它是命令源码文件，则相关操作和结果如下：

```
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/ds$ ls
showds.go
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/ds$ go build
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/ds$ ls
ds  showds.go
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/ds$ go install
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/ds$ ls ../../../../bin
ds
```

需要特别注意，只有当环境变量 GOPATH 中只包含一个工作区的目录路径时，go install 命令才会把命令源码文件安装到当前工作区的 bin 目录下；否则，像这样执行 go install 命令就会失败。此时必须设置环境变量 GOBIN，该环境变量的值是一个目录的路径，该目录用于存放所有因安装 Go 命令源码文件而生成的可执行文件。

2. 库源码文件

通常，库源码文件声明的包名会与它直接所属的代码包（目录）名一致，且库源码文件中不包含无参数声明和无结果声明的 main 函数。下面来安装（其中包含编译）gopcp.v2/helper/log 包，其中含有若干库源码文件：

```
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/log$ ls
base          logger.go      logger_test.go logrus
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/log$ go install
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/log$ ls ../../../../pkg
linux_amd64
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/log$ ls ../../../../pkg/linux_amd64/gopcp.v2/helper
log   log.a
hc@ubt:~/golang/example.v2/src/gopcp.v2/helper/log$ ls ../../../../pkg/linux_amd64/gopcp.v2/helper
/log
base.a   field.a  logrus.a
```

这里，我们通过在 gopcp.v2/helper/log 代码包的目录下执行 go install 命令，成功安装了该包并生成了若干归档文件。这些归档文件的存放目录由以下规则产生。

- 安装库源码文件时所生成的归档文件会被存放到当前工作区的pkg目录中。example.v2项目的gopcp.v2/helper/log包所属工作区的根目录是~/golang/example.v2。因此，上面所说的pkg目录即~/golang/example.v2/pkg。
- 根据被编译时的目标计算环境，归档文件会被放在该pkg目录下的平台相关目录中。例如，我是在64位的Linux计算环境下安装的，对应的平台相关目录就是linux_amd64，那么归档文件一定会被存放到~/golang/example.v2/pkg/linux_amd64