常用命令

- **build** 用于编译指定的代码包或 Go 语言源码文件。命令源码文件会被编译成可执行文件，并存放到命令执行的目录或指定目录下。而库源码文件被编译后，则不会在非临时目录中留下任何文件。
- **clean** 用于清除因执行其他 go 命令而遗留下来的临时目录和文件。
- **doc** 用于显示打印 Go 语言代码包以及程序实体的文档。
- **env** 用于打印 Go 语言相关的环境信息。
- **fix** 用于修正指定代码的源码文件中包含的过时语法和代码调用。这使得我们在升级 Go 语言版本时，可以非常方便地同步升级程序。
- **fmt** 用于格式化指定代码包中的 Go 源码文件。实际上，它是通过执行 gofmt 命令来实现功能的。
- **generate** 用于识别指定代码中资源文件中的 “go:generate” 注释，并执行其携带的任意命令。该命令独立于 Go 语言标准的编译和安装体系。如果你有需要解析的 “go:generate” 注释，就单独运行它。这个命令非常有用，我常用它自动生成或改动 Go 源码文件。
- **get** 用于下载，编译并安装指定改动代码包及其依赖包。从我们自己的代码中转站或第三方代码库上自动拉取代码，就全靠它了。
- **install** 用于编译并安装指定的代码包及其依赖包。安装命令源码文件后，代码包所在的工作区目录的 bin 子目录，或者当前环境变量 GOBIN 指向的目录中会生成相应的可执行文件。安装源码文件后，会在代码包所在的工作目录的 pkg 子目录中生成相应的归档文件。
- **list** 用于显示指定代码包的信息，它可谓是代码包分析的一大便利工具。利用 Go 语言标准代码库代码包 “text/template” 中规定的模版语法，你可以非常灵活的控制输出信息。
- **run** 用于编译并运行指定的代码源码文件。当你想不生成可执行文件而直接运行命令源码文件时，就需要用到它。
- **test** 用于测试指定的代码包，前提是该代码包目录中必须存在测试源代码文件。
- **version**用于显示当前安装的 Go 语言的版本信息以及计算环境。
- **tool** 用于运行 Go 语言的特殊工具。
- **vet** 如果开发人员已经写了一些代码，vet 命令会帮开发人员检测代码的常见错误。让我们看看 vet 捕获哪些类型的错误。
    - 如Printf类函数调用时，类型匹配错误的参数。
    - 定义常用的方法时，方法签名的错误。
    - 错误的结构标签。
    - 没有指定字段名的结构字面量。

```
#用于检查指定代码包中的 Go 语言代码，并报告发现可疑代码问题。该命令提供了除编译之外的又一个程序检查方法，可用于找到程序中的潜在错误。
go vet main.go
```

附加参数

执行上述命令时，可以通过附加一些额外的标记来定制命令的执行过程。下面是一个比较通用的标记。

- **-n** 使命令仅打印其执行过程中用到的所有命令，而不真正执行它们。如果只想查看或验证命令的执行过程，而不想改变任何东西，使用它正合适。
- **-race** 用于检测并报告指定 Go 语言程序中存在的数据竞争问题。当用 Go 语言编写并发程序时，这是很重要的检测手段之一。
- **-v** 用于打印命令执行过程中涉及的代码包。这一定包括我们指定的目标代码包，并且有时还会包括该代码包直接或间接依赖的那些代码包。这会让你知道哪些代码包被命令处理过了。
- **-work** 用于打印命令执行时生成和使用的临时工作目录的名字，且命令执行完成后不删除它。这个目录下的文件可能会对你有用，也可以从侧面了解命令的执行过程。如果不添加此标记，那么临时工作目录会在命令执行完毕前删除。
- **-x：**使命令打印其执行过程中用到的所有命令，同时执行它们。

我们可以把这些标记看作命令的特殊参数，他们都可以添加到命令名称和命令的真正参数中间。用于编译， 安装，运行和测试 Go 语言代码包或源码文件的命令都支持它们。上面提到了 tool 这个子命令，它用来运行一些特殊的 Go 语言工具。直接执行 go tool 命令，可以看到这些特殊工具。它们有的是其他 Go 标准命令的底层支持，有的规则是可以独当一面的利器。其中有两个工具值得特别介绍一下。

pprof 用于以交互的方式访问一些性能概要文件。命令将会分析给定的概要文件，并根据要求提供高可读性的输出信息。这个工具可以分析的概要文件包括 CPU 概要文件，内存概要文件和程序阻塞概要文件。这些包含 Go 程序运行信息的概要文件，可以通过标准库代码 runtime 和runtime/pprof 中的程序来生成。

trace 用于读取 Go 程序踪迹文件，并以图形化的方式展现出来。它能够让我们深入了解 Go 程序在运行过程中的内部情况。比如，当前进程中堆的大小及使用情况。又比如，程序中的多个 goruntime 是怎样调度的，以及它们在某个时刻被调度的原因。Go 程序踪迹文件可以通过标准代码包 “runtime/trace” 和 “net/http/pprof” 中的程序来生成。

上述的两个特殊的工具对 Go 程序调优非常有用。如果想探究程序运行的过程，或者想让程序跑的更快，更稳定，那么这两个工具是必知必会的。另外，这两个工具都受到 go test 命令的直接支持。因此你可以很方便地把它们融入到程序测试当中。