

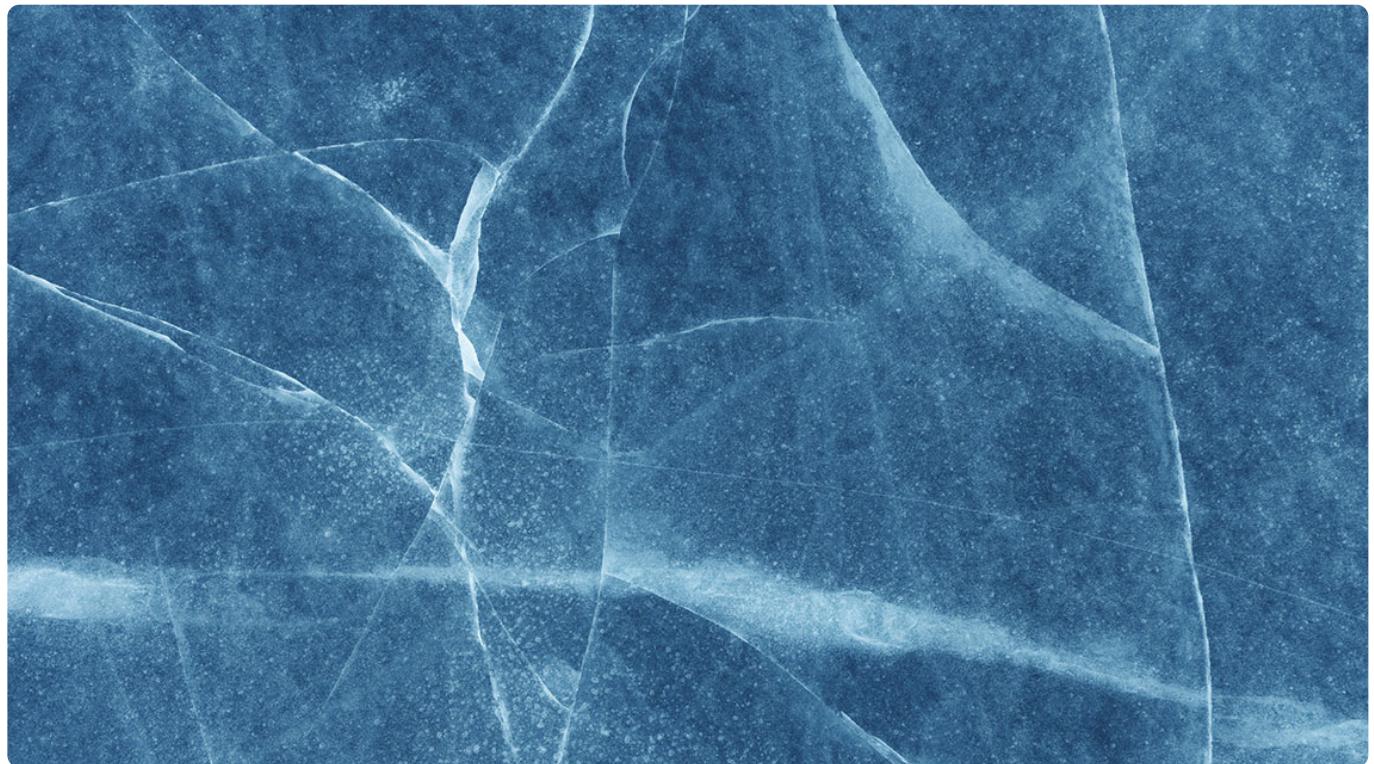


02 | 命令源码文件

2018-08-13 郝林

Go语言核心36讲

[进入课程 >](#)



讲述：黄洲君

时长 08:45 大小 8.02M



我们已经知道，环境变量 GOPATH 指向的是一个或多个工作区，每个工作区中都会有以代码包为基本组织形式的源码文件。

这里的源码文件又分为三种，即：命令源码文件、库源码文件和测试源码文件，它们都有着不同的用途和编写规则。（我在[“预习篇”的基础知识图](#)介绍过这三种文件的基本情况。）



(长按保存大图查看)

今天，我们就沿着**命令源码文件**的知识点，展开更深层级的学习。

一旦开始学习用编程语言编写程序，我们就一定希望在编码的过程中及时地得到反馈，只有这样才能清楚对错。实际上，我们的有效学习和进步，都是通过不断地接受反馈和执行修正实现的。

对于 Go 语言学习者来说，你在学习阶段中，也一定会经常编写可以直接运行的程序。这样的程序肯定会涉及命令源码文件的编写，而且，命令源码文件也可以很方便地用 go run 命令启动。

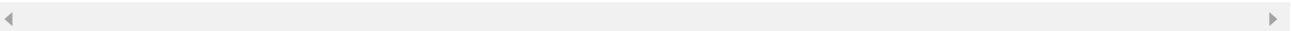
那么，我今天的问题就是：命令源码文件的用途是什么，怎样编写它？

这里，我给出你一个**参考的回答**：命令源码文件是程序的运行入口，是每个可独立运行的程序必须拥有的。我们可以通过构建或安装，生成与其对应的可执行文件，后者一般会与该命令源码文件的直接父目录同名。

如果一个源码文件声明属于main包，并且包含一个无参数声明且无结果声明的main函数，那么它就是命令源码文件。 就像下面这段代码：

 复制代码

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello, world!")
7 }
```



如果你把这段代码存成 demo1.go 文件，那么运行`go run demo1.go`命令后就会在屏幕（标准输出）中看到`Hello, world!`

当需要模块化编程时，我们往往会将代码拆分到多个文件，甚至拆分到不同的代码包中。但无论怎样，对于一个独立的程序来说，命令源码文件永远只会也只能有一个。如果有与命令源码文件同包的源码文件，那么它们也应该声明属于main包。

问题解析

命令源码文件如此重要，以至于它毫无疑问地成为了我们学习 Go 语言的第一助手。不过，只会打印`Hello, world`是远远不够的，咱们千万不要成为“Hello, world”党。既然决定学习 Go 语言，你就应该从每一个知识点深入下去。

无论是 Linux 还是 Windows，如果你用过命令行（command line）的话，肯定就会知道几乎所有命令（command）都是可以接收参数（argument）的。通过构建或安装命令源码文件，生成的可执行文件就可以被视为“命令”，既然是命令，那么就应该具备接收参数的能力。

下面，我就带你深入了解一下与命令参数的接收和解析有关的一系列问题。

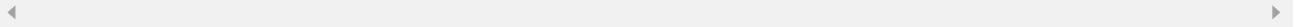
知识精讲

1. 命令源码文件怎样接收参数

我们先看一段不完整的代码：

 复制代码

```
1 package main
2
3 import (
4     // 需在此处添加代码。[1]
5     "fmt"
6 )
7
8 var name string
9
10 func init() {
11     // 需在此处添加代码。[2]
12 }
13
14 func main() {
15     // 需在此处添加代码。[3]
16     fmt.Printf("Hello, %s!\n", name)
17 }
18
```



如果邀请你帮助我，在注释处添加相应的代码，并让程序实现“根据运行程序时给定的参数问候某人”的功能，你会打算怎样做？

如果你知道做法，请现在就动手实现它。如果不知道也不要着急，咱们一起来搞定。

首先，Go 语言标准库中有一个代码包专门用于接收和解析命令参数。这个代码包的名字叫 `flag`。

我之前说过，如果想要在代码中使用某个包中的程序实体，那么应该先导入这个包。因此，我们需要在 [1] 处添加代码 "`flag`"。注意，这里应该在代码包导入路径的前后加上英文半角的引号。如此一来，上述代码导入了 `flag` 和 `fmt` 这两个包。

其次，人名肯定是由字符串代表的。所以我们要在 [2] 处添加调用 `flag` 包的 `StringVar` 函数的代码。就像这样：

 复制代码

```
1 flag.StringVar(&name, "name", "everyone", "The greeting object.")
```

函数`flag.StringVar`接受 4 个参数。

第 1 个参数是用于存储该命令参数值的地址，具体到这里就是在前面声明的变量`name`的地址了，由表达式`&name`表示。

第 2 个参数是为了指定该命令参数的名称，这里是`name`。

第 3 个参数是为了指定在未追加该命令参数时的默认值，这里是`everyone`。

至于第 4 个参数，即是该命令参数的简短说明了，这在打印命令说明时会用到。

顺便说一下，还有一个与`flag.StringVar`函数类似的函数，叫`flag.String`。这两个函数的区别是，后者会直接返回一个已经分配好的用于存储命令参数值的地址。如果使用它的话，我们就需要把

 复制代码

```
1 var name string
```

改为

 复制代码

```
1 var name = flag.String("name", "everyone", "The greeting object.")
```

所以，如果我们使用`flag.String`函数就需要改动原有的代码。这样并不符合上述问题的要求。

再说最后一个填空。我们需要在 [3] 处添加代码 `flag.Parse()`。函数 `flag.Parse` 用于真正解析命令参数，并把它们的值赋给相应的变量。

对该函数的调用必须在所有命令参数存储载体的声明（这里是 `name` 的声明）和设置（这里是在 [2] 处对 `flag.StringVar` 函数的调用）之后，并且在读取任何命令参数值之前进行。

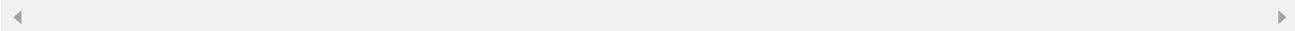
正因为如此，我们最好把 `flag.Parse()` 放在 `main` 函数的函数体的第一行。

2. 怎样在运行命令源码文件的时候传入参数，又怎样查看参数的使用说明

如果我们把上述代码存成名为 `demo2.go` 的文件，那么运行如下命令就可以为参数 `name` 传值：

 复制代码

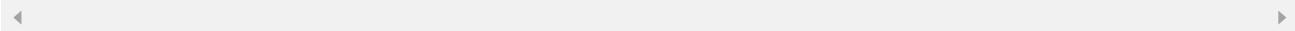
```
1 go run demo2.go -name="Robert"
2
```



运行后，打印到标准输出（`stdout`）的内容会是：

 复制代码

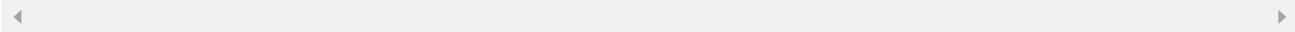
```
1 Hello, Robert!
```



另外，如果想查看该命令源码文件的参数说明，可以这样做：

 复制代码

```
1 $ go run demo2.go --help
```



其中的 `$` 表示我们是在命令提示符后运行 `go run` 命令的。运行后输出的内容会类似：

 复制代码

```
1 Usage of /var/folders/ts/7lg_t1_x2gd_k1lm5g_48c7w0000gn/T/go-build155438482/b001/exe/der
2 -name string
3     The greeting object. (default "everyone")
4 exit status 2
```

 复制代码

```
1 /var/folders/ts/7lg_t1_x2gd_k1lm5g_48c7w0000gn/T/go-build155438482/b001/exe/demo2
```

 复制代码

```
1 $ go build demo2.go
2 $ ./demo2 --help
```

那么输出就会是

 复制代码

```
1 Usage of ./demo2:
2 -name string
3     The greeting object. (default "everyone")
```

3. 怎样自定义命令源码文件的参数使用说明

这有很多种方式，最简单的一种方式就是对变量`flag.Usage`重新赋值。`flag.Usage`的类型是`func()`，即一种无参数声明且无结果声明的函数类型。

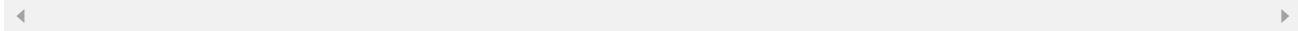
`flag.Usage`变量在声明时就已经被赋值了，所以我们才能够在运行命令`go run demo2.go --help`时看到正确的结果。

注意，对`flag.Usage`的赋值必须在调用`flag.Parse`函数之前。

现在，我们把`demo2.go`另存为`demo3.go`，然后在`main`函数体的开始处加入如下代码。

 复制代码

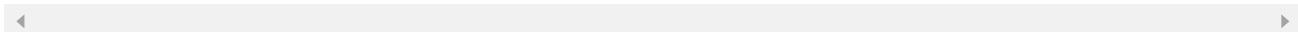
```
1 flag.Usage = func() {
2     fmt.Fprintf(os.Stderr, "Usage of %s:\n", "question")
3     flag.PrintDefaults()
4 }
```



那么当运行

 复制代码

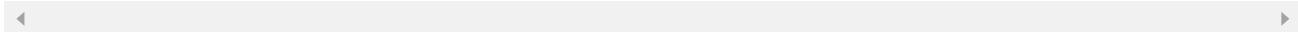
```
1 $ go run demo3.go --help
```



后，就会看到

 复制代码

```
1 Usage of question:
2   -name string
3       The greeting object. (default "everyone")
4 exit status 2
```



现在再深入一层，我们在调用`flag`包中的一些函数（比如`StringVar`、`Parse`等等）的时候，实际上是在调用`flag.CommandLine`变量的对应方法。

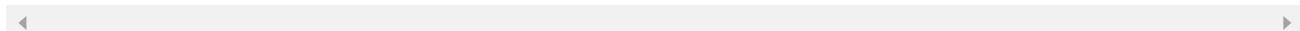
`flag.CommandLine`相当于默认情况下的命令参数容器。所以，通过对`flag.CommandLine`重新赋值，我们可以更深层次地定制当前命令源码文件的参数使用说

明。

现在我们把main函数体中的那条对flag.Usage变量的赋值语句注销掉，然后在init函数体的开始处添加如下代码：

 复制代码

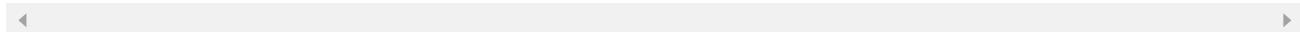
```
1 flag.CommandLine = flag.NewFlagSet("", flag.ExitOnError)
2 flag.CommandLine.Usage = func() {
3     fmt.Fprintf(os.Stderr, "Usage of %s:\n", "question")
4     flag.PrintDefaults()
5 }
```



再运行命令go run demo3.go --help后，其输出会与上一次的输出的一致。不过后面这种定制的方法更加灵活。比如，当我们把为flag.CommandLine赋值的那条语句改为

 复制代码

```
1 flag.CommandLine = flag.NewFlagSet("", flag.PanicOnError)
```



后，再运行go run demo3.go --help命令就会产生另一种输出效果。这是由于我们在这里传给flag.NewFlagSet函数的第二个参数值是flag.PanicOnError。

flag.PanicOnError和flag.ExitOnError都是预定义在flag包中的常量。

flag.ExitOnError的含义是，告诉命令参数容器，当命令后跟--help或者参数设置的不正确的时候，在打印命令参数使用说明后以状态码2结束当前程序。

状态码2代表用户错误地使用了命令，而flag.PanicOnError与之的区别是在最后抛出“运行时恐慌（panic）”。

上述两种情况都会在我们调用flag.Parse函数时被触发。顺便提一句，“运行时恐慌”是Go程序错误处理方面的概念。关于它的抛出和恢复方法，我在本专栏的后续部分中会讲到。

下面再进一步，我们索性不用全局的`flag.CommandLine`变量，转而自己创建一个私有的命令参数容器。我们在函数外再添加一个变量声明：

 复制代码

```
1 var cmdLine = flag.NewFlagSet("question", flag.ExitOnError)
```

然后，我们把对`flag.StringVar`的调用替换为对`cmdLine.StringVar`调用，再把`flag.Parse()`替换为`cmdLine.Parse(os.Args[1:])`。

其中的`os.Args[1:]`指的就是我们给定的那些命令参数。这样做就完全脱离了`flag.CommandLine`。`*flag.FlagSet`类型的变量`cmdLine`拥有很多有意思的方法。你可以去探索一下。我就不在这里一一讲述了。

这样做好处依然是更灵活地定制命令参数容器。但更重要的是，你的定制完全不会影响到那个全局变量`flag.CommandLine`。

总结

恭喜你！你现在已经走出了 Go 语言编程的第一步。你可以用 Go 编写命令，并可以让它们像众多操作系统命令那样被使用，甚至可以把它们嵌入到各种脚本中。

虽然我为你讲解了命令源码文件的基本编写方法，并且也谈到了为了让它接受参数而需要做的各种准备工作，但这并不是全部。

别担心，我在后面会经常提到它的。另外，如果你想详细了解`flag`包的用法，可以到[这个网址](#)查看文档。或者直接使用`godoc`命令在本地启动一个 Go 语言文档服务器。怎样使用`godoc`命令？你可以参看[这里](#)。

思考题

我们已经见识过为命令源码文件传入字符串类型的参数值的方法，那还可以传入别的吗？这就是今天我留下的思考题。

1. 默认情况下，我们可以让命令源码文件接受哪些类型的参数值？

2. 我们可以把自定义的数据类型作为参数值的类型吗？如果可以，怎样做？

你可以通过查阅文档获得第一个问题的答案。记住，快速查看和理解文档是一项必备的技能。

至于第二个问题，你回答起来可能会有些困难，因为这涉及了另一个问题：“怎样声明自己的数据类型？”这个问题我在专栏的后续部分中也会讲到。如果是这样，我希望你记下它和这里说的另一问题，并在能解决后者之后再来回答前者。

[戳此查看 Go 语言专栏文章配套详细代码。](#)

The banner features the '极客时间' logo at the top left. The main title 'GO语言核心36讲' is prominently displayed in large blue font. Below it, the subtitle '3个月带你通关 GO 语言' is shown. To the right of the text is a portrait photo of He Lin, a man with glasses and dark hair, wearing a blue shirt. At the bottom, there's a call-to-action: '新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。'.

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 01 | 工作区和GOPATH

下一篇 03 | 库源码文件

精选留言 (47)

写留言



雨生 置顶
2019-02-28



flag的讲解很棒，通过这个命令，我们就可以控制程序在不同环境的执行内容了，通过控制参数设置更多的内容！

展开 ▼



咖啡色的羊...

2018-08-13

64

看完本文，记住的两点：

- 1.源码文件分为三种：命令，库，测试。
- 2.编写命令源码文件的关键包：flag。

回答下问题：....

展开 ▼



Dragoonium

2018-08-13

36

我试着把参数增加到两个，然后试试运行结果

```
func init() {  
    flag.StringVar(&name, "name1", "ladies", "The greeting object 1")  
    flag.StringVar(&name, "name2", "gentlemen", "The greeting object 2")  
}...
```

展开 ▼



Abirdcfly

2018-08-14

13

解答一下 Dragoonium 同学的疑惑，在 flag 包的文档里第一个 example 里就有你提到的这种情况，注释已经说明白了。

我不太精确的翻译一下：

...

展开 ▼



javaadu

2018-08-13

9

喜欢这种重实践和编码的风格，便于上手

展开 ▾



Tron

2018-09-05

↑ 7

go语言ide还是推荐goland

展开 ▾



wjq310

2018-08-15

↑ 5

demo3之后，要import os

展开 ▾



云学

2018-08-13

↑ 4

init在main之前执行，go程序的执行顺序是否可以讲下

展开 ▾



吉祥

2018-08-13

↑ 3

undefined: os 怎么回事

展开 ▾

作者回复: 你好，请到GitHub上下载完整的源码文件。



alan

2018-08-13

↑ 3

感谢老师。感觉编码细节有些偏多了哈，希望多一些总结性和主观的内容。



zhaopan

2019-02-17

↑ 2

1. 会出现冲突

2. 导入包的几种方式

2.1 常规方式

```
import "your/lib"
```

通过包名lib调用SayHello方法。lib.SayHello() ...

展开 ▾



何何何何何...

2018-12-08

2

1. 默认情况下，我们可以让命令源码文件接受哪些类型的参数值？

答：前面讲过`flag`是专门用来处理命令行参数的包，所以我们只需要看`flag`这个包支持哪些数据结构就行了。结果如下：

- * int(int|int64|uint|uint64),
- * float(float|float64)...

展开 ▾



梦里追逐

2018-08-15

2

咱们用的都是哪个IDE？

展开 ▾

作者回复: 你好，我用的是goland，但是代码不会依赖于IDE的，只会依赖于Go语言本身。免费的编辑器推荐vs code。

成都福哥

2018-08-15

2

用自定义的cmdLine的时候，usage函数里的flag.PrintDefaults()应该相应的变成cmdLine.PrintDefaults()吧。

展开 ▾



丸子说

2018-08-14

2

从flag.StringVar(flag.String)到flag.CommandLine再到私有 cmdline 命令参数容器，循序渐进，由浅到深。



芒果

2018-08-14

2

关于变量以标准输入为准的问题，我个人认为init中的定义只是定义了解析规则，真正执行解析是flag.Parse()时开始，因此以标准输出为准。想想我们自己写的时候会怎么实现，先获得输入如：-name1=a，然后解析为key=name1和value=a，然后走一个if, else判断，如果key匹配则对其赋值。所以就很好解释了。个人感觉自己的理解还是比较靠谱的，虽然没有研究源码。欢迎大神们交流

展开 ▼



松烽

2018-08-14

2

自定义参数，还可以自己通过字符串转对象的方式实现

展开 ▼



飞吧蝴蝶

2018-10-12

1

问题1：通过flag库的提示，或者看flag包的用法，参数支持

Bool/Duration/Float64/Int/Int64/Uint/Uint64，也支持Float32，猜测考虑到精度问题，flag没有支持float32。

问题2：参数值的类型可以是自定义的数据类型，使用实现flag包里的Value接口，然后使用flag.Var()实现。（flag源码里有提示，Value is the interface to the dynamic value...）

展开 ▼



mayunian

2018-08-30

1

老师，今天试了一下类型转换。

为什么转换var x uint = uint(-1) 的时候会报错？

而var y int = -1

var x uint = uint(y)就不会报错呢？

作者回复: -1是负数，编译器看出来了，帮你挑出来。y是int类型的变量，编译器不知道里面存的是不是负数，没法帮你挑出来。转换会成功结果会不正确。



世风十三

2018-08-13

1

flag.Usage 那部分有个 os 变量是 undefined 啊 ?

展开 ▼