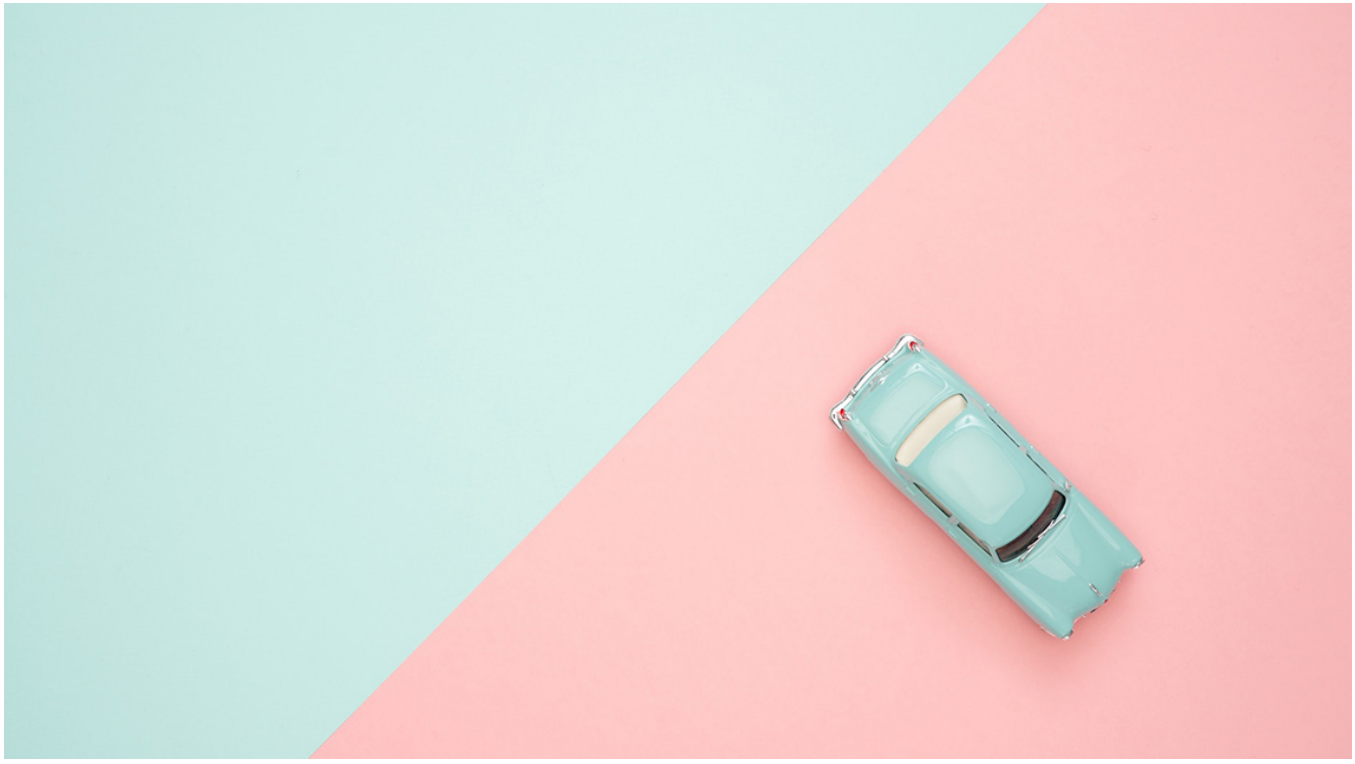
04 | 程序实体的那些事儿（上）

2018-08-17 郝林



04 | 程序实体的那...
朗读人：黄洲君　11′45″ | 8.07M　0:00 / 11:45

【Go 语言代码较多，建议配合文章收听音频。】

我已经为你打开了 Go 语言编程之门，并向你展示了程序从初建到拆分再到模块化的基本演化路径。

一个编程老手让程序完成这个基本演化可能也就需要几十分钟甚至十几分钟，因为他们一开始就会把车开到模块化编程的道路上。我相信，等你真正理解了这个过程之后也会驾轻就熟的。

上述套路是通用的，不是只适用于 Go 语言。但从本篇开始，我会开始向你介绍 Go 语言中的各种特性以及相应的编程方法和思想。

我在讲解那两种源码文件基本编写方法的时候，声明和使用了一些程序实体。你也许已经若有所觉，也许还在云里雾里。没关系，我现在就与你一起梳理这方面的重点。

还记得吗？Go 语言中的程序实体包括变量、常量、函数、结构体和接口。Go 语言是静态类型的编程语言，所以我们在声明变量或常量的时候都需要指定它们的类型，或者给予足够的信息以使 Go 语言能够推导出它们的类型。

在 Go 语言中，变量的类型可以是其预定义的那些类型，也可以是程序自定义的函数、结构体或接口。常量的合法类型不多，只能是那些 Go 语言预定义的基本类型。它的声明方式也更简单一些。

好了，下面这个简单的问题你需要了解一下。

问题：声明变量有几种方式？

先看段代码。

复制代码

```
package main

import (
        "flag"
        "fmt"
)

func main() {
        var name string // [1]
        flag.StringVar(&name, "name", "everyone", "The greeting object.") // [2]
        flag.Parse()
        fmt.Printf("Hello, %v!\n", name)
}
```

这是一个很简单的命令源码文件，我把它命名为 demo7.go。它是 demo2.go 的微调版。我只是把变量`name`的声明和对`flag.StringVar`函数的调用都移动到了`main`函数中，这分别对应代码中的注释`[1]`和`[2]`。

具体的问题是，除了`var name string`这种声明变量`name`的方式还有其他方式吗？你可以选择性地改动注释`[1]`和`[2]`处的代码。

典型回答

这有几种做法，我在这里只说最典型的两种。

第一种方式需要先对注释`[2]`处的代码稍作改动，把被调用的函数由`flag.StringVar`改为`flag.String`，传参的列表也需要随之修改，这是为了将`[1]`和`[2]`处的代码合并而做的准备工作。

复制代码

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

合并后的代码看起来更简洁一些。我把注释[1]处的代码中的string去掉了，右边添加了一个=，然后再拼接上经过修改的[2]处的代码。

我在调用flag.String函数的代码左边加了一个*，这是因为该函数返回的结果值的类型是*string而不是string。类型*string代表的是字符串的指针类型，而不是字符串类型。

关于 Go 语言中的指针，我在后面会有专门的介绍。你在这里只需要知道，我通过一个“*”把这个字符串的指针值指向的字符串值取出来了，然后通过赋值符号“=”把后者赋给了name变量。

好了，我想你已经基本理解了这行代码中的每一个部分。下面说第二种方式。第二种方式与第一种方式非常类似。基于第一种方式的代码，赋值符号=右边的代码不动，左边只留下name，再把=变成:=。

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

复制代码

问题解析

这个问题的基本考点有两个。一个是你要知道 Go 语言中的类型推断，以及它在代码中的基本体现，另一个是短变量声明的用法。

第一种方式中的代码在声明变量name的同时还为它赋了值，而这时声明中并没有显式指定name的类型。还记得吗？之前的变量声明语句是var name string。这里利用了 Go 语言自身的类型推断而省去了string。

简单地说，类型推断是一种编程语言在编译期自动解释表达式类型的能力。什么是表达式？详细的解释你可以参看 Go 语言规范中的表达式和表达式语句章节。我在这里就不赘述了。

你可以认为，表达式类型就是对表达式进行求值后得到结果的类型。Go 语言中的类型推断是很简约的，这也是 Go 语言整体的风格。

它只能用于对变量或常量的初始化，就像上述回答中描述的那样。对flag.String函数的调用其实就是一个调用表达式，而这个表达式的类型是*string，即字符串的指针类型。

这也是调用flag.String函数后得到结果的类型。随后，Go 语言把这个调用了flag.String函数的表达式类型直接作为了变量name的类型，这就是“推断”一词所指代的操作了。

至于第二种方式所用的短变量声明，实际上就是 Go 语言的类型推断再加上一点点语法糖。我们只能在函数体内部使用短变量声明。在编写`if`、`for`或`switch`语句的时候，我们经常把它安插在初始化子句中，并用来声明一些临时的变量。而相比之下，第一种方式更加通用，它可以被用在任何地方。

短变量声明还有其他的玩法，我稍后就会讲到。

知识扩展

1. Go 语言的类型推断可以带来哪些好处？

如果面试官问你这个问题，你应该怎样回答？

当然，在写代码时，我们通过使用 Go 语言的类型推断，而节省下来的键盘敲击次数几乎可以忽略不计。但它真正的好处往往会体现在我们写代码之后的那些事情上，比如代码重构。

为了更好的演示，我们先要做一点准备工作。我们依然通过调用一个函数在声明`name`变量的同时为它赋值，但是这个函数不是`flag.String`，而是其他的我们自己定义的某个函数，比如叫`getTheFlag`。

复制代码

```
package main

import (
        "flag"
        "fmt"
)

func main() {
        var name = getTheFlag()
        flag.Parse()
        fmt.Printf("Hello, %v!\n", *name)
}

func getTheFlag() *string {
        return flag.String("name", "everyone", "The greeting object.")
}
```

我们可以用`getTheFlag`函数包裹（或者说包装）那个对`flag.String`函数的调用，并把其结果直接作为`getTheFlag`函数的结果，结果的类型是`*string`。这样一来，`var name =`右边的表达式可以变为针对`getTheFlag`函数的调用表达式了。

这实际上是对声明并赋值`name`变量的那行代码的重构。我们通常把“不改变某个程序与外界的任何交互方式和规则，而只改变其内部实现”的代码修改方式，叫做对该程序的重构。

重构的对象可以是一行代码、一个函数、一个功能模块，甚至一个软件系统。

好了，在准备工作做完之后，你会发现，你可以随意改变getTheFlag函数的内部实现及其返回结果的类型，而不用修改main函数中的任何代码。

这个命令源码文件依然可以通过编译，并且构建和运行也都不会有问题。也许你能感觉得到，这是一个关于程序灵活性的质变。

我们不显式地指定变量name的类型，使得它可以被赋予任何类型的值。也就是说，变量name的类型可以在其初始化时由其他程序动态地确定。

在你改变getTheFlag函数的结果类型之后，Go 语言的编译器会在你再次构建该程序的时候，自动地更新变量name的类型。如果你使用过Python或Ruby这种动态类型的编程语言的话一定会觉得这情景似曾相识。

没错，通过这种类型推断你可以体验到动态类型编程语言所带来的一部分优势，即程序灵活性的明显提升。但在那些编程语言中，这种提升可以说是用程序的可维护性和运行效率换来的。

Go 语言是静态类型的，所以一旦在初始化变量时确定了它的类型，之后就不可能再改变。这就避免了在后面维护程序时的一些问题。另外，请记住，这种类型的确定是在编译期完成的，因此不会对程序的运行效率产生任何影响。

现在，你应该已经对这个问题有一个比较深刻的理解了。如果只用一两句话回答这个问题的话，我想可以是这样的：Go 语言的类型推断可以明显提升程序的灵活性，使得代码重构变得更加容易，同时又不会给代码的维护带来额外负担（实际上，它恰恰可以避免散弹式的代码修改），更不会损失程序的运行效率。

2. 变量的重声明是什么意思？

这涉及了短变量声明。通过使用它，我们可以对同一个代码块中的变量进行重声明。

既然说到了代码块，我先来解释一下它。在 Go 语言中，代码块一般就是一个由花括号括起来的区域，里面可以包含表达式和语句。Go 语言本身以及我们编写的代码共同形成了一个非常大的代码块，也叫全域代码块。

这主要体现在，只要是公开的全局变量，都可以被任何代码所使用。相对小一些的代码块是代码包，一个代码包可以包含许多子代码包，所以这样的代码块也可以很大。

接下来，每个源码文件也都是一个代码块，每个函数也是一个代码块，每个if语句、for语句、switch语句和select语句都是一个代码块。甚至，switch或select语句中的case子句也都是独立的代码块。

走个极端，我就在main函数中写一对紧挨着的花括号算不算一个代码块？当然也算，这甚至还有个名词，叫“空代码块”。

回到变量重声明的问题上。其含义是对已经声明过的变量再次声明。变量重声明的前提条件如下。

1. 由于变量的类型在其初始化时就已经确定了，所以对它再次声明时赋予的类型必须与其原本的类型相同，否则会产生编译错误。
2. 变量的重声明只可能发生在某一个代码块中。如果与当前的变量重名的是外层代码块中的变量，那么就是另外一种含义了，我在下一篇文章中会讲到。
3. 变量的重声明只有在使用短变量声明时才会发生，否则也无法通过编译。如果要在此处声明全新的变量，那么就应该使用包含关键字`var`的声明语句，但是这时就不能与同一个代码块中的任何变量有重名了╬。
4. 被“声明并赋值”的变量必须是多个，并且其中至少有一个是新的变量。这时我们才可以说对其中的旧变量进行了重声明。

这样来看，变量重声明其实算是一个语法糖（或者叫便利措施）。它允许我们在使用短变量声明时不用理会被赋值的多个变量中是否包含旧变量。可以想象，如果不这样会多写不少代码。

我把一个简单的例子写在了“Golang_Puzzlers”项目的`puzzlers/article4/q3`包中的 demo9.go 文件中，你可以去看一下。

这其中最重要的两行代码如下：

```
var err error
n, err := io.WriteString(os.Stdout, "Hello, everyone!\n")
```

复制代码

我使用短变量声明对新变量`n`和旧变量`err`进行了“声明并赋值”，这时也是对后者的重声明。

总结

在本篇中，我们聚焦于最基本的 Go 语言程序实体：变量。并详细解说了变量声明和赋值的基本方法及其背后的重要概念和知识。我们使用关键字`var`和短变量声明都可以实现对变量的“声明并赋值”。

这两种方式各有千秋，有着各自的特点和适用场景。前者可以被用在任何地方，而后者只能被用在函数或者其他更小的代码块中。

不过，通过前者我们无法对已有的变量进行重声明，也就是说它无法处理新旧变量混在一起的情况。不过它们也有一个很重要的共同点，即：基于类型推断，Go 语言的类型推断只应用在了对变量或常量的初始化方面。

思考题

本次的思考题只有一个：如果与当前的变量重名的是外层代码块中的变量，那么这意味着什么？

这道题对于你来说可能有些难，不过我鼓励你多做几次试验试试，你可以在代码中多写一些打印语句，然后运行它，并记录下每次试验的结果。如果有疑问也一定要写下来，答案将在下篇文章中揭晓。

戳此查看 Go 语言专栏文章配套详细代码。

写留言

精选留言

Shawn 14

当前变量覆盖外层变量

2018-08-17

玉明 7

应该是没影响的，不同栈上的变量

2018-08-17

Andy Chen 7

“你可以随意改变getTheFlag函数的内部实现及其返回结果的类型，而不用修改main函数中的任何代码。”这个说法只在你给定的例子下面成立，事实上main函数的代码已经假设getTheFlag会返回字符串，因为它在用返回值，如果getTheFlag一开始是返回某种结构体指针，main使用了这个指针指向的一系列成员，然后你再改getTheFlag返回类型看看。类型推断已经深入大多数语言，包括c++，C#，等等，但它没办法解决所谓的使用者不需要改变任何代码就能进行重构

2018-08-17

作者回复

重构有很多种，有大有小啊。

2018-08-17

xiaolonghuster 5

按照老师的demo，不能获取命令参数，只能得到默认值，改成下面这样可以:我用的是1.10.3版本，是不是版本问题

func main() {

var name = getTheFlag()

flag.Parse()
fmt.Printf("Hello, %v!\n", *name)
}

func getTheFlag() *string {

return flag.String("name", "everybody", "The greeting object.")
}

2018-08-17

作者回复

这里确实是写错了，你改的很对，谢谢指正！我想邀请你进入本专栏的微信讨论群。你知道入群方法吗？

2018-08-17

阿杜 5

go支持类型推断；
两种变量定义方式：var完整方式、:=短变量定义；
重声明只可以在短变量定义中出现，并且是在多个变量声明中出现（给新变量赋值，给旧变量赋新值）

2018-08-17

陈悬高 3

所谓“变量的重声明”容易引发歧义，而且也不容易理解。如果没有为变量分配一块新的内存区域，那么用声明是不恰当的。在《Go 语言圣经》一书中将短声明的这种特性称为赋值。个人总结如下：

在使用短变量声明的时候，你可能想要同时对一个已有的变量赋值，类似使用 `=` 进行多重赋值那样（如 `i, j = 2, 3`）。所以，Go 为短声明语法提供了一个语法糖（或者叫便利措施）：短变量声明不需要声明所有在左边的变量。如果多个变量在同一个词法块中声明，那么对于这些变量，短声明的行为等同于*赋值*。

比如，在下面的代码中，第一条语句声明了 `in` 和 `err`。第二条语句仅声明了 `out`，但向已有的 `err` 变量进行赋值。

```
in, err := os.Open(infile)
```

// ...
out, err := os.Create(outfile)
```

但是这种行为需要一些前提条件：

* 要赋值的变量必须声明在同一个词法块中。

如果两个变量位于不同的词法块中，短声明语法表示的仍然是“声明”而非“赋值”。此时它们就是重名的变量了，而且内层变量会“覆盖”外部变量。

* 必须至少声明一个新变量，否则代码将不能编译通过。

原因很简单，如果不用声明新变量而仅仅是为了赋值，那么直接使用赋值符 `=` 即可：

```
f, err := os.Open(infile)
// ...
// f, err := os.Create(outfile) // 编译错误：没有新变量
f, err = os.Create(outfile) // 使用普通的赋值语句即可
```

2018-10-06

慢熊胖胖跑 2

由于go是值传递，因此即使传入重名变量，一般在代码块中变量可以正常使用，但是值得改变不会引起变化，因为变量传入后，代码块中赋予了新的地址。 除非如同case3一样中传入变量的指针，然后才会使用相同的变量地址，修改变量的值。func reusevarnam1(var1 int) { var 1 = 3 + var1 fmt.Printf("Inside reusevarnam1,var1 is %d, address is %s\n", var1, &var 1)}
func reusevarnam2(var2 int) { for var2 := 1; var2 < 3; var2++ { fmt.Println("reusev arnam2 ...") fmt.Printf("Inside reusevarnam2,var2 is %d, address is %s\n", var2, &va r2) } fmt.Println("reusevarnam2")}
func reusevarnam3(var3 *int) { *var3 = *var3 + 100 fmt.Printf("Inside reusevarnam2, var3 is %d, address is %s\n", *var3, var3)}

2018-08-18

作者回复

你这一堆❀是啥？：）

2018-08-19

冰激凌的眼泪 1

var name = *flag.String("name", "everyone", "The greeting object.")
这一句是不是导致name是个副本，parse后也不会变？请郝老师确认一下

2018-08-18

作者回复

已经修正了。
```

2018-08-19

lfn 1

变量解析由内向外，内层找不到才回去外层找。

2018-08-17

objcoding 1

类型推断只能在局部生效，相当于Java的局部变量，而在函数体外的声明变量相当于Java的实例变量

2018-08-17

小木 0

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

这句代码不能正确打印 name 是为什么呢？

2018-10-10

虎虎❤ 0

重声明的目的是什么？可以带来什么好处呢？

是不是说只有在多个变量声明并赋初值的场景才有用？这样把已经声明过的变量重新利用，而不用再引入一个新的变量？

就比如你举的例子中，n, err := ...，如果不可以重声明，那么err写在 :=前面就不合适了。是这个目的吗？

2018-10-02

liyinda0000 0

在article4/q1/demo7.go中在使用{方式1}中，发现name传参未能正常打印，我使用的go版本1.9.4

解决办法： 13行 *flag.String改成flag.String；19行 fmt.Printf("Hello, %v!\n", name) 将name改成*name

问题思考：应该是文中未深入探讨‘指针变量’的问题，flag.String()得到的是指针变量，*代表取值符，*name将指针变量中的值取出（运行&name发现为内存地址，&代表取地址符），上述问题的出现原因请您指点？另如何加入微信学习群，望早日加入组织，哈哈

2018-09-07

作者回复

微信学习群已经满了，不过可以去开发者头条App搜索GoHackers并加入组织。这个组织是我个人在几年前发起的。

2018-09-08

charlesgogo01 0

name := *flag.string（ ）这儿为啥会有*，本来返回应该是个变量值，这样是传递地址吗？

2018-09-03

作者回复

Go里面没有传递地址这种说法。*在这里是取值操作符。

2018-09-04

$侯 0

内层和外层变量的作用域不一样，修改了内层的不会对外层重名的变量产生影响

2018-08-20

AJIE 0

思考题:

进入到一个代码块后，就进入了一个新的作用域。不会发生合法的重声明，也不会报错不合法的重声明。

2018-08-20

MKing 0

思考题: 如果内层变量使用:=去定义 会产生内部变量 但不影响外部变量 如果使用= 则会直接使用外部变量操作 会对外部变量造成修改 不知道对不对

2018-08-19

作者回复

对的，但是内层的赋值用=的时候不能加var，加了var还是新变量。

2018-08-20

张玉锡 0

demo9.go里的赋值，左边是两个变量n和err，一个是短变量，一个是声明变量，为啥都可以用:=的方式赋值？

2018-08-19

作者回复

这就是本篇文章所以说的变量重声明。

2018-08-20

张震 0

这个redefine跟用var声明后赋值没啥区别吧

2018-08-19

作者回复

var是没法参与变量重声明的，但是可以参与可重名变量的声明。

2018-08-19

runner 0

重构，赋值处是不用改，后面对变量处理还是要修改啊。

2018-08-17

作者回复

所以我是说对这一行的重构。重点是getTheFlag的结果类型改了之后，这段代码其他地方就不用改了。实际开发过程中这种便利是可以利用的，少改写代码。

2018-08-17

小小笑儿 0

思考题:

内部作用域的变量会覆盖外部作用域的变量，是声明一个新的变量而不是重声明，重声明只在同一个作用域生效。

可以使用go vet -shadow显示这一类错误。

2018-08-17

作者回复

vet会提示有隐患，重声明和可重名变量都容易让人产生迷惑，这也是也讲到它们的原因。哦，对了，课重名变量会在下一篇讲。

2018-08-17