01 | 工作区和GOPATH

2018-08-10 郝林

【Go语言代码较多，建议配合文章收听音频。】

你好，我是郝林。从今天开始，我将和你一起梳理Go语言的整个知识体系。

在过去的几年里，我与广大爱好者一起见证了Go语言的崛起。

从Go 1.5版本的自举（即用Go语言编写程序来实现Go语言自身），到Go 1.7版本的极速GC（也称垃圾回收器），再到2018年2月发布的Go 1.10版本对其自带工具的全面升级，以及可预见的后续版本关键特性（比如用来做程序依赖管理的`go mod`命令），这一切都令我们欢欣鼓舞。Go语言在一步步走向辉煌的同时，显然已经成为软件工程师们最喜爱的编程语言之一。

我开办这个专栏的主要目的，是要与你一起探索Go语言的奥秘，并帮助你在学习和实践的过程中获取更多。

我假设本专栏的读者已经具备了一定的计算机基础，比如，你要知道操作系统是什么、环境变量怎么设置、怎样正确使用命令行，等等。

当然了，如果你已经有了编程经验，尤其是一点点Go语言编程经验，那就更好了，毕竟我想教给你的，都是Go语言中非常核心的技术。

如果你对Go语言中最基本的概念和语法还不够了解，那么可能需要在学习本专栏的过程中去查阅Go语言规范文档，也可以把预习篇的基础知识图拿出来好好研究一下。

最后，我来说一下专栏的讲述模式。我总会以一道Go语言的面试题开始，针对它进行解答，我会告诉你为什么我要关注这道题，这道题的背后隐藏着哪些知识，并且，我会对这部分的内容，进行相关的知识扩展。

好了，准备就绪，我们一起开始。

我们学习Go语言时，要做的第一件事，都是根据自己电脑的计算架构（比如，是32位的计算机还是64位的计算机）以及操作系统（比如，是Windows还是Linux），从Go语言官网下载对应的二进制包，也就是可以拿来即用的安装包。

随后，我们会解压缩安装包、放置到某个目录、配置环境变量，并通过在命令行中输入`go version`来验证是否安装成功。

在这个过程中，我们还需要配置3个环境变量，也就是GOROOT、GOPATH和GOBIN。这里我可以简单介绍一下。

- GOROOT：Go语言安装根目录的路径，也就是GO语言的安装路径。
- GOPATH：若干工作区目录的路径。是我们自己定义的工作空间。
- GOBIN：GO程序生成的可执行文件（executable file）的路径。

其中，GOPATH背后的概念是最多的，也是最重要的。那么，**今天我们的面试问题是：你知道设置GOPATH有什么意义吗？**

关于这个问题，它的**典型回答**是这样的：

你可以把GOPATH简单理解成Go语言的工作目录，它的值是一个目录的路径，也可以是多个目录路径，每个目录都代表Go语言的一个工作区（workspace）。

我们需要利于这些工作区，去放置Go语言的源码文件（source file），以及安装（install）后的归档文件（archive file，也就是以“.a”为扩展名的文件）和可执行文件（executable file）。

事实上，由于Go语言项目在其生命周期内的所有操作（编码、依赖管理、构建、测试、安装等）基本上都是围绕着GOPATH和工作区进行的。所以，它的背后至少有3个知识点，分别是：

1. Go语言源码的组织方式是怎样的；

2.你是否了解源码安装后的结果（只有在安装后，Go语言源码才能被我们或其他代码使用）；

3.你是否理解构建和安装Go程序的过程（这在开发程序以及查找程序问题的时候都很有用，否则你很可能会走弯路）。

下面我就重点来聊一聊这些内容。

知识扩展

1. Go语言源码的组织方式

与许多编程语言一样，Go语言的源码也是以代码包为基本组织单位的。在文件系统中，这些代码包其实是与目录一一对应的。由于目录可以有子目录，所以代码包也可以有子包。

一个代码包中可以包含任意个以.go为扩展名的源码文件，这些源码文件都需要被声明属于同一个代码包。

代码包的名称一般会与源码文件所在的目录同名。如果不同名，那么在构建、安装的过程中会以代码包名称为准。

每个代码包都会有导入路径。代码包的导入路径是其他代码在使用该包中的程序实体时，需要引入的路径。在实际使用程序实体之前，我们必须先导入其所在的代码包。具体的方式就是`import`该代码包的导入路径。就像这样：

```
import "github.com/labstack/echo"
```

在工作区中，一个代码包的导入路径实际上就是从src子目录，到该包的实际存储位置的相对路径。

所以说，Go语言源码的组织方式就是以环境变量GOPATH、工作区、src目录和代码包为主线的。一般情况下，Go语言的源码文件都需要被存放在环境变量GOPATH包含的某个工作区（目录）中的src目录下的某个代码包（目录）中。

2. 了解源码安装后的结果

了解了Go语言源码的组织方式后，我们很有必要知道Go语言源码在安装后会产生怎样的结果。

源码文件以及安装后的结果文件都会放到哪里呢？我们都知道，源码文件通常会被放在某个工作区的src子目录下。

那么在安装后如果产生了归档文件（以“.a”为扩展名的文件），就会放进该工作区的pkg子目录；如果产生了可执行文件，就可能会放进该工作区的bin子目录。

我再讲一下归档文件存放的具体位置和规则。

源码文件会以代码包的形式组织起来，一个代码包其实就对应一个目录。安装某个代码包而产生的归档文件是与这个代码包同名的。

放置它的相对目录就是该代码包的导入路径的直接父级。比如，一个已存在的代码包的导入路径是

```
github.com/labstack/echo，
```

那么执行命令

```
go install github.com/labstack/echo
```

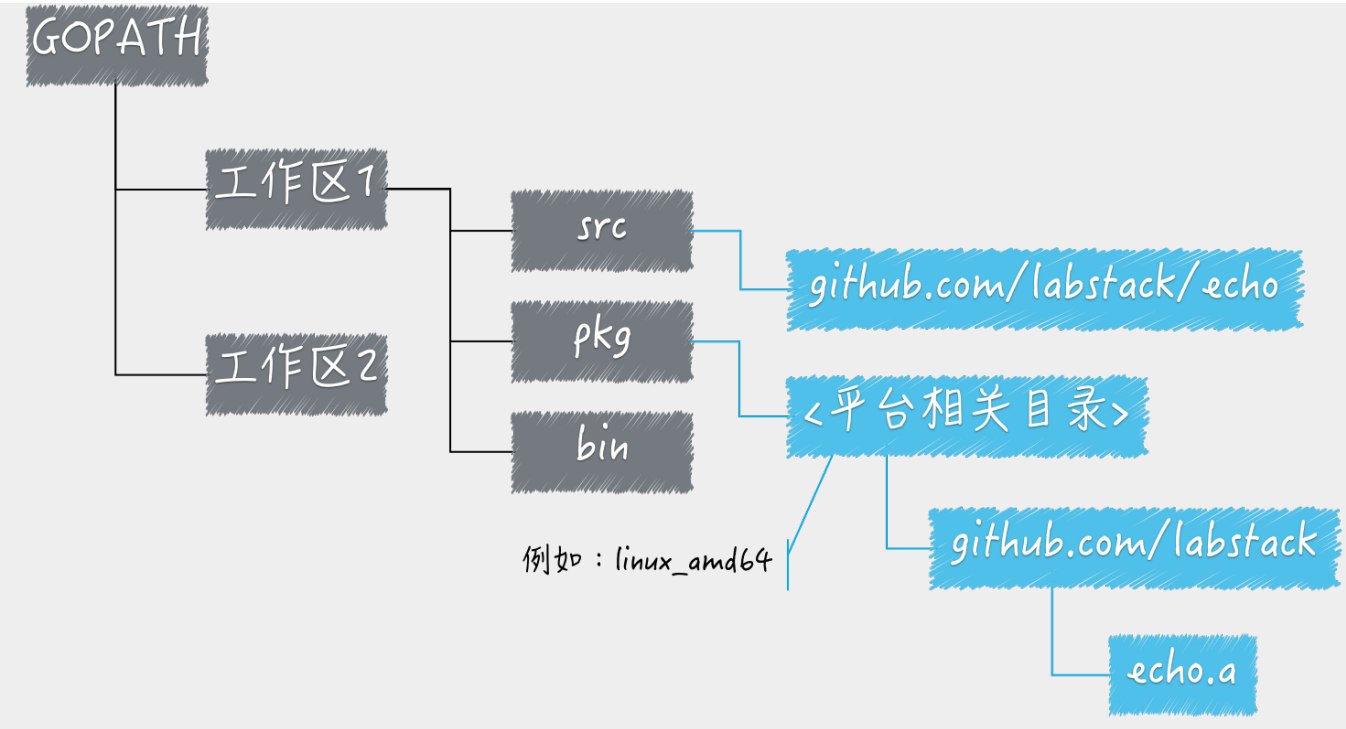
生成的归档文件的相对目录就是 github.com/labstack， 文件名为echo.a。

顺便说一下，上面这个代码包导入路径还有另外一层含义，那就是：该代码包的源码文件存在于GitHub网站的labstack组的代码仓库echo中。

再说回来，归档文件的相对目录与pkg目录之间还有一级目录，叫做平台相关目录。平台相关目录的名称是由build（也称“构建”）的目标操作系统、下划线和目标计算架构的代号组成的。

比如，构建某个代码包时的目标操作系统是Linux，目标计算架构是64位的，那么对应的平台相关目录就是linux_amd64。

因此，上述代码包的归档文件就会被放置在当前工作区的子目录pkg/linux_amd64/github.com/labstack中。



（GOPATH与工作区）

总之，你需要记住的是，某个工作区的src子目录下的源码文件在安装后一般会被放置到当前工作区的pkg子目录下对应的目录中，或者被直接放置到该工作区的bin子目录中。

3. 理解构建和安装Go程序的过程

我们再来说说构建和安装Go程序的过程都是怎样的，以及它们的异同点。

构建使用命令`go build`，安装使用命令`go install`。构建和安装代码包的时候都会执行编译、打包等操作，并且，这些操作生成的任何文件都会先被保存到某个临时的目录中。

如果构建的是库源码文件，那么操作后产生的结果文件只会存在于临时目录中。这里的构建的主要意义在于检查和验证。

如果构建的是命令源码文件，那么操作的结果文件会被搬运到源码文件所在的目录中。（这里讲到的两种源码文件我在“预习篇”的基础知识图中提到过，在后面的文章中我也会带你详细了解。）

安装操作会先执行构建，然后还会进行链接操作，并且把结果文件搬运到指定目录。

进一步说，如果安装的是库源码文件，那么结果文件会被搬运到它所在工作区的pkg目录下的某个子目录中。

如果安装的是命令源码文件，那么结果文件会被搬运到它所在工作区的bin目录中，或者环境变量`GOBIN`指向的目录中。

这里你需要记住的是，构建和安装的不同之处，以及执行相应命令后得到的结果文件都会出现在哪里。

总结

工作区和GOPATH的概念和含义是每个Go工程师都需要了解的。虽然它们都比较简单，但是说它们是Go程序开发的核心知识并不为过。

然而，我在招聘面试的过程中仍然发现有人忽略掉了它们。Go语言提供的很多工具都是在GOPATH和工作区的基础上运行的，比如上面提到的`go build`、`go install`和`go get`，这三个命令也是我们最常用到的。

思考题

说到Go程序中的依赖管理，其实还有很多问题值得我们探索。我在这里留下两个问题供你进一步思考。

1. Go语言在多个工作区中查找依赖包的时候是以怎样的顺序进行的？
2. 如果在多个工作区中都存在导入路径相同的代码包会产生冲突吗？

这两个问题之间其实是有一些关联的。答案并不复杂，你做几个试验几乎就可以找到它了。你也可以看一下Go语言标准库中go build包及其子包的源码。那里面的宝藏也很多，可以助你深刻理解Go程序的构建过程。

补充阅读

go build命令一些可选项的用途和用法

在运行go build命令的时候，默认不会编译目标代码包所依赖的那些代码包。当然，如果被依赖的代码包的归档文件不存在，或者源码文件有了变化，那它还是会被编译。

如果要强制编译它们，可以在执行命令的时候加入标记-a。此时，不但目标代码包总是会被编译，它依赖的代码包也总会被编译，即使依赖的是标准库中的代码包也是如此。

另外，如果不但要编译依赖的代码包，还要安装它们的归档文件，那么可以加入标记-i。

那么我们怎么确定哪些代码包被编译了呢？有两种方法。

1. 运行go build命令时加入标记-x，这样可以看到go build命令具体都执行了哪些操作。另外也可以加入标记-n，这样可以只查看具体操作而不执行它们。
2. 运行go build命令时加入标记-v，这样可以看到go build命令编译的代码包的名称。它在与-a标记搭配使用时很有用。

下面再说一说与Go源码的安装联系很紧密的一个命令：go get。

命令go get会自动从一些主流公用代码仓库（比如GitHub）下载目标代码包，并把它们安装到环境变量GOPATH包含的第1工作区的相应目录中。如果存在环境变量GOBIN，那么仅包含命令源码文件的代码包会被安装到GOBIN指向的那个目录。

最常用的几个标记有下面几种。

- -u：下载并安装代码包，不论工作区中是否已存在它们。
- -d：只下载代码包，不安装代码包。
- -fix：在下载代码包后先运行一个用于根据当前Go语言版本修正代码的工具，然后再安装代码包。
- -t：同时下载测试所需的代码包。
- -insecure：允许通过非安全的网络协议下载和安装代码包。HTTP就是这样的协议。

Go语言官方提供的go get命令是比较基础的，其中并没有提供依赖管理的功能。目前GitHub上有很多提供这类功能的第三方工具，比如glide、gb以及官方出品的dep、vgo等等，它们在内部大都会直接使用go get。

有时候，我们可能会出于某种目的变更存储源码的代码仓库或者代码包的相对路径。这时，为了让代码包的远程导入路径不受此类变更的影响，我们会使用自定义的代码包导入路径。

对代码包的远程导入路径进行自定义的方法是：在该代码包中的库源码文件的包声明语句的右边加入导入注释，像这样：

```
package semaphore // import "golang.org/x/sync/semaphore"
```

这个代码包原本的完整导入路径是`github.com/golang/sync/semaphore`。这与实际存储它的网络地址对应的。该代码包的源码实际存在GitHub网站的golang组的sync代码仓库的semaphore目录下。而加入导入注释之后，用以下命令即可下载并安装该代码包了：

```
go get golang.org/x/sync/semaphore
```

而Go语言官网golang.org下的路径/x/sync/semaphore并不是存放`semaphore`包的真实地址。我们称之为代码包的自定义导入路径。

不过，这还需要在golang.org这个域名背后的服务端程序上，添加一些支持才能使这条命令成功。

关于自定义代码包导入路径的完整说明可以参看这里。

好了，对于`go build`命令和`go get`命令的简短介绍就到这里。如果你想查阅更详细的文档，那么可以访问Go语言官方的命令文档页面，或者在命令行下输入诸如`go help build`这类的命令。

戳此查看Go语言专栏文章配套详细代码。

极客时间

GO语言核心36讲

3个月带你通关GO语言

郝林

《Go 并发编程实战》作者
GoHackers 技术社群发起人
前轻松筹大数据负责人