Workspaces and Packages in Go

Code Organization and Sharing

Ian Harris

December 14, 2018

1 Code Organization in Go

Right now, we're going to talk about how code is organized in Go. First, we'll start with a workspace.

1.1 Workspace Directory

So there's this idea of a workspace, and it's basically a directory where your Go stuff will go, so your Go files. Your Go source files and other files will go in this workspace directory. Typically, there's a actually a hierarchy of directories within your workspace where you will store the different types of Go files that you're working with. Now, the reason why we're doing this, why the Go language defines this hierarchy of directories is because common organization is good for sharing.

1.1.1 Sharing

So, a big motivation behind Go, the Go language is for people to work together easily. So, remember that when you're programming, not necessarily in this class, in this class you're working on learning the language, the different aspects of the language, but when you get outside and you're working in a company or something like this, it's never one person alone, it's always a big group. You're working with people all over the place. They have to be able to work with your code, look at your code, merge it with their code, link it to their code that sort of thing. So, there's always this sharing going on. Maybe you want to upload to GitHub and have a communal group of people working on code together. So for that, it is nice to have a common standardized organization of your files, right? It makes it easier to share because then everybody knows where everything is. Tools know where things are and stuff like that.

1.1.2 Hierarchy of Subdirectories

So, inside your workspace directory, what is recommended are these three subdirectories. The *source directory*, it contains the source files, your source code

your Go code. *Package directory* contains packages, the other packages that you're going to link in that you need; and then the *bin directory* that contains all your executables, your compiled executables.

1.1.3 One Workspace — Many Projects

Now, the programmer typically has one workspace for many projects. So, I typically use my one workspace directory and I can have 20 projects, 20 different Go projects I'm working on in the same workspace directory. That's common. You don't have to do that but that's common. So, one thing to remember about these directory hierarchy is that it's recommended but it's not enforced. So, this idea of having the source subdirectory, the bin subdirectory, and the package subdirectory, that's not enforced. So, for instance, you can have an executable in the source directory if you want. It is not neat and it's harder for people to share, but it's going to run. You can compile it and put it anywhere you want and run the executable. So, it's not enforced is just a recommendation to make it easy to share with other people.

1.1.4 GOPATH Environment Variable

So, the workspace directory, you do have this one workspace directory though, and this workspace directory is defined by the GOPATH environment variable. Now, the GOPATH environment variable depends on, how you set environment variables is going to depend on your operating system. Normally, what happens is like on my Windows machine, but the Chaperone, Linux and OS X machine two, is that the GOPATH directory is set for you automatically during the installation process. So, that wizard, the install wizard, it should define the GOPATH environment variable. Certainly, on a Windows machine, the default directory where it sticks it, where it puts it is `C:\Users\yourname`. So, for me, `\user\Ian\go`. It sets that as your workspace directory. Now, I noticed that when I installed everything, that was my GOPATH. What you see up there . But it actually didn't create a Go directory. So, there was . I had to create the directory myself, which is fine. But I had to make that directory and put my stuff in there. But understand that that's the default workspace. You can change that and you can go to your GOPATH environment variable and change your environment variable in your operating system if you want to, but for now I'm just assuming that we're using the default Gopath. So, with Go tools, I'll assume that the code is inside the GOPATH somewhere.

1.2 Packages

Now, there is this other concept of packages. Your code is organized into packages. *A package is a group of related source code files.* Each package can be imported by other packages. So, this is the use for this.

1.2.1 Software Reuse

The main use for this is when you're working with other people, other groups of people in other places, you write all your code in one package, they write all their code in another package, and then if you need to use their code, you can use their code, you can import their package. So, it's good for software reuse that's the main goal.

1.2.2 `Package` Keyword

The first line of the file names the package. So, what I'm showing here in the picture, you can see these two pink boxes up here. These are two packages that are defined and you can see the first line of, so those are two different files, different source code files. You can see the package names are listed at the top, package package. There's a bunch of code in there and they're associated with that package name. Then in blue, I have some other piece of code in a different source file and it needs to use the packages from the other two people.

1.2.3 `Import` Statement

So, I have an input statement at the top of my blue file and I give the package names that I want to import. So, I can use these other two packages in my code if I want to. So, this is how packages get connected to each other. It's very convenient if you're working with somebody remotely or somewhere else that you can clean separation of the code.

1.2.4 Package `main`

Now, there always has to be one package called `main`, and that's where execution starts. So, there's got to be one package called `main` and you'll note that in the code that we're working on in this course, we just have one package and it is called `main`. Because we're not making such big code, we have different groups of people working together with different packages right now. We're just writing one package called `main`. But there must be one package called `main` and when you build the `main` package when you compile it, it makes an executable one. So, note that when you build another package on the non-`main` packages, then it doesn't make it executable for those, or not a running executable because it's not going to be executed directly. It will be incorporated into some other package. But the `main` package, that's what's going to be run, so when you compile that, when you build it, buildyou get an executable file.

1.2.5 `main()` Function

So, the `main` package needs to have a function called `main`. `main` is where code execution starts. So, you can see the example code right here. It's just printing, "Hello world". If we say "package main, import format", so that `import` right there is importing a package. "Format" it's not a package that I wrote. "Format"

is one of the packages that comes with the Go tool. So, when you download the Go tools you get all these standard packages including `fmt`. The "format" package has a lot of functions in it. We'll talk more about it later. But one of the functions that it has this print statement, so `Printf()` is included in the `fmt` package, so we have to import that package, and then we make our function `main` and in there it just says `fmt.Printf hello world`. So, pretty straightforward.