Installing External Packages

This lesson discusses how Go let users install external packages using go get and go install. In the later part, the difference between them is discussed.

WE'LL COVER THE FOLLOWING

Introduction

If you need one or more external packages in your application, you will first have to install them locally on your machine with the `go get` command. Suppose you want to use a (fictional) package which can be downloaded from the *not-existent* URL http://codesite.ext/author/goExample/goex, where *codesite* could be **googlecode**, **github**, **bitbucket**, **launchpad** or others. Where *ext* is the extension. The correct name for this is **TLD** (**T**op-**L**evel **D**omain) like *.com* or *.org*. The `goex` is the package name (often these start with go, but this is by no means a necessary convention). You install this with the command:

```
go get codesite.ext/author/goExample/goex
```

This installs the code in the folder **codesite.ext/author/goExample/goex** under **$GOPATH/src/**. Once installed, to import it in your code, use:

```
import goex "codesite.ext/author/goExample/goex"
```

The import path will be the web-accessible URL for your project's root followed by the subdirectory. The documentation for go install lists the import paths for a number of widely used code repositories on the web.

Explanation

Explanation

`go get` is Go's automatic package installation tool. Downloading them from remote repositories over the internet if needed, on the local machine, which includes checkout, compile and install in one go. It installs each of the packages given on the command line. Moreover, it installs a package's *prerequisites* before trying to install the package itself and handles the dependencies automatically. The dependencies of the packages residing in subfolders are also installed but documentation or examples are not. The dependencies are browsable on each package's website, though. Again the variable GOPATH is used (see Appendix).

Here is a concrete example. Suppose we want to use `groupcache-db` (from this link) in our Go project. This is a software solution to enhance the responsiveness of a slow database using front-ends and a cache server. How do we install this? Use the command:

```
go get -v github.com/capotej/groupcache-db-experiment/...
```

This command produces the following terminal output(verbose because of the `-v` flag):

```
github.com/golang/groupcache (download)
github.com/golang/protobuf (download)
github.com/capotej/groupcache-db-experiment/api
github.com/golang/groupcache/lru
github.com/golang/groupcache/singleflight
github.com/golang/groupcache/consistenthash
github.com/capotej/groupcache-db-experiment/slowdb
github.com/golang/protobuf/proto
github.com/capotej/groupcache-db-experiment/client
github.com/capotej/groupcache-db-experiment/dbserver
github.com/capotej/groupcache-db-experiment/cli
github.com/golang/groupcache/groupcachepb
github.com/golang/groupcache
github.com/capotej/groupcache-db-experiment/frontend
```

(The `-v` flag produces verbose output) This puts the Go source-files of this project in **$GOPATH/src/github.com/capotej/groupcache-db-experiment**

At the same time, the project is compiled to .a files (on a Windows 64-bit

machine) in

$GOPATH/pkg/windows_amd64/github.com/capotej/groupcache-db-experiment.

The executables for the projects containing `main()` points in **$GOPATH/bin**. From then on the functionality of the package `groupcache-db` can be used in Go code by importing:

```
import groupcache "github.com/capotej/groupcache-db-experiment"
```

If you only want to update an external package, use the `-u` flag:

```
go get -u package(s)
```

Difference between `go install` and `go get`

`go get` verifies if there are packages that need to be downloaded. If yes, then packages are downloaded and compiled. `go install` doesn't do a download. It only compiles and throws an error if any packages are missing locally.

More info can be found here.

Packages provide great flexibility when writing code whether it be a package from the standard library, a custom package or an external package. That's it about the packages in Go. The next chapter brings new concepts to learn about structs and methods.