External Packages, Projects and Libraries

This lesson discusses the Go's flexibility to import a package, library or a project from the outer world.

WE'LL COVER THE FOLLOWING

Go external packages and projects

We now know how to use Go and its standard library, but the Go-ecosystem is bigger than this. When embarking on our Go-projects, it is best to first search if we can use some existing third party Go package(s) or project(s). Most of these can be installed with the `go get` tool. The place to look is here. It provides a list of indexes and search engines specific for Go packages and projects.

The index is further classified by categories such as Astronomy, Build Tools, Compression, Data Structures, Databases and Storage, Development Tools and so on. It contains a wealth of more than 500 projects, giving for each its name, a short description and a download link. Use any of the available filters to refine your search.

There is a wealth of many great external libraries, for example:

• **MySQL** (GoMySQL)

• **PostgreSQL** (go-pgsql)

• **MongoDB** (mgo, gomongo)

• **CouchDB** (couch-go)

• **ODBC** (godbcl)

• **Redis**(redis.go)

• **SQLite3** (gosqlite) database drivers

• **SDL bindings**

• **Google's Protocol Buffers** (goprotobuf)

• **XML-RPC** (go-xmlrpc)

• **Twitter** (twitterstream)

• **OAuth libraries**, (GOAuth)

and many more.

Using an external library in a Go program

When starting a new project or adding new functionalities to an existing project, you could save development time by incorporating an already existing Go-library into your application. In order to do that, you have to understand the **API** (**A**pplication **P**rogramming **I**nterface) of the library which contains the methods you can call in this library and how to call them. Remember that you can only use the *exported* methods, whose name starts with a capital letter. Only these will usually be documented. You might not have the source of this library, but the makers will surely have documented the API and details of how to use it.

How do you use a *local* library you have created yourself in a subdirectory `pack1` in a directory `mypackage`? This is very simple, just do:

```
import "mypackage/pack1"
```

and use the exported methods of `pack1`.

Using an external library is very simple. Once you know its API, you just call the library's functions and work with its results. We'll use a little library developed by **Inanc Gumus**, which you can find here. This library allows us to easily GET HTTP resources from a specified URL with timeout support. This means you can set a timeout in seconds for every GET request you want to

execute.

Downloading and installing the external library

Downloading and installing the external library will be accomplished with the `go get` tool. First, verify that the GOPATH variable is set in your environment because the external source code will be downloaded into the directory **$GOPATH/src**.

The packages will also be installed under the **$GOPATH/pkg/"machine_arch"**/. Then we install the library by invoking the following command in the console:

```
go get github.com/inancgumus/myhttp
```

`go get` takes the URL path without **https://** as argument, and will download the source code, compile it and install the package.

How to use the external library

Of course, we need to import this package with:

```
import "github.com/inancgumus/myhttp"
```

The docs tell us that the `New` and `Get` methods form the API of this very simple library:

- First use the `New` function on the package's name (myhttp) to create an instance `mh`. This sets the timeout (here 1 second) for the Get request.
- Call the Get method on `mh`. This returns an *http Response* object, from which we print out the StatusCode value.

Here is the complete code:

```
package main
import (
"fmt"
"time"
"github.com/inancgumus/myhttp"
)

func main() {
```

```
  mh := myhttp.New(time.Second)
  response, _ := mh.Get("https://jsonip.com/")

  fmt.Println("HTTP status code: ", response.StatusCode)
}
```

For brevity of code the error returns were not checked, but this should be done for a real production application!

That's it about the introduction to external packages and the libraries. In the next lesson, we'll see how to install such packages to make them work.