**01.05.2023 Golang Lesson Instruction**

1. **go doc**

Go doc prints the documentation for a package, const, func, type, var, or method

* Go doc accepts zero, one, or two arguments.
* **ZERO**
* Prints package documentation for the package in the current directory
  + **go doc**
* **ONE**
* Argument Go-syntax-like representation of item to be documented
* Fyi: <sym> also known as “identifier”
  + **go doc <pkg>**
  + **go doc <sym>[.<method>]**
  + **go doc [<pkg>.]<sym>[<method>]**
  + **go doc [<pkg>.][<sym>.]<method>**
* The first item in this list that succeeds is the one whose documentation is printed. If there is symbol but no package, the package in the current directory is chosen. However, if the argument begins with a capital letter it is always assumed to be a symbol in the current directory
* **TWO**
* First argument must be a full package path
  + **go doc <pkg> <sym>[,<method>]**

Example:

**go doc fmt, go doc os, go doc Sum**, **go doc fmt.Printf, go doc json, go doc, go doc cmd/doc**

**2. godoc**

1. **install godoc with following command:**

*go get golang.org/x/tools/cmd/godoc* ***not working***

go install golang.org/x/tools/cmd/godoc

1. **Start godoc server:**

*godoc -http=:6060*

**we can see all of golang documentation without http and internet connection. It workd on our *localhost:6060*.**

**3.Writing documentation**

Documentation is a huge of making software accessible and maintainable. Of course it must be well-written and accurate, but it also must be easy to write and maintain.

**4.Writer**

The io.Writer interface it’s one of Go’s very small interfaces. It has only one method. The Write method. The io.Writer interface is used by many packages in the Go standard library and it represents the ability to write a byte slice into a stream of data. More generically allows you to write data into something that implements the io.Writer interface. Here’s the io.Writer interface definition

package main

import (

    "fmt"

    "os"

)

func main() {

    f, err := os.OpenFile("123.txt", os.O\_WRONLY|os.O\_CREATE|os.O\_APPEND, 0600)

    if err != nil {

        panic(err)

    }

    defer f.Close()

    n, err := f.Write([]byte("I am being written to txt file!"))

    if err != nil {

        panic(err)

    }

    fmt.Println("wrote %d bytes", n)

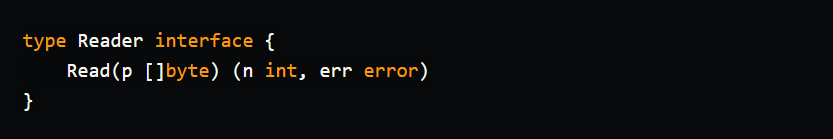
}

io.Writer with json encode

In the Go standard library there are a few implementations and examples of the io.Writer interface. One of them is the json/encoding’s NewEncoder method which takes a io.Writer as input and writes the json encoded output into the underlying stream of data that implements the io.Writer.

5. **Reader**

If you write Go programs, you'll come across the io.Reader interface quite a bit. It's used all over the place to represent reading data from lots of different sources, but its type definition is actually pretty humble.



Reader is a little interface with a strong abstraction! What is that abstraction exactly? The idea with the Read method is that it represents reading bytes of data from **some source**, so that we can then use those bytes in our code. That source could be files, cameras, network connections, or just a plain old string. For example, if we're reading data from files, the io.Reader we would use is a \*os.File.