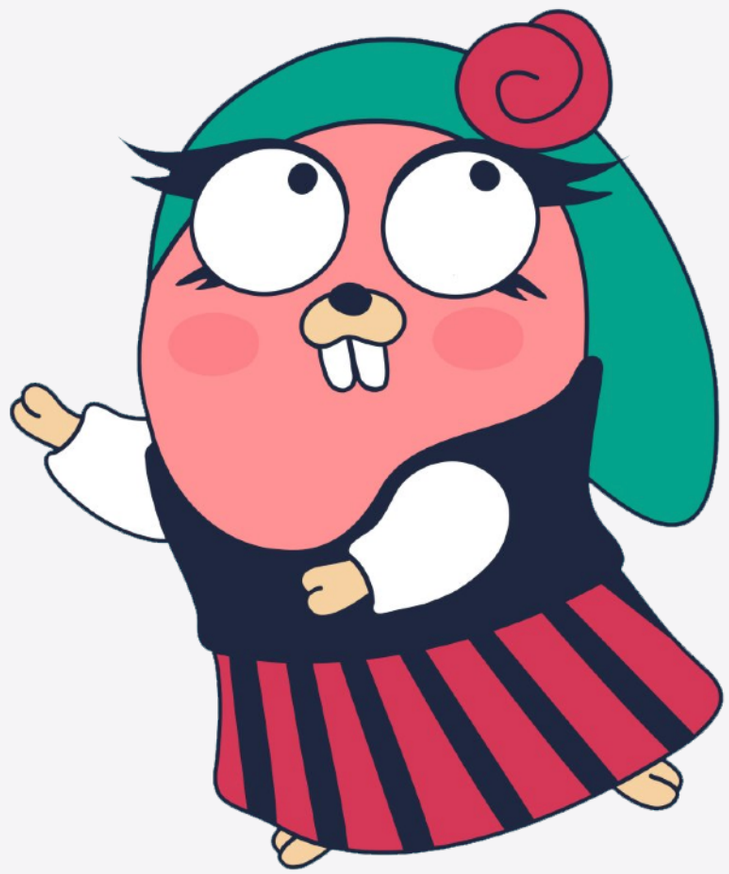
All you need is ~~love~~ io.Reader and io.Writer

Ferran Orriols - 28th of September 2017



· WHO AM I? ·

· Ferran Orriols

· Backend engineer

· Gopher since 1 year aprox

AD Secure®

trovit

What is this talk about?

Read and Write

The power of the interfaces

“The bigger the interface, the weaker the abstraction”

Rob Pike - Gopherfest - https://youtu.be/PAAkCSZUG1c?t=5m18s

io.Reader

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

Why not?

```
type Reader interface {
    Read() (p []byte, err error)
}
```

io.Writer

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

Reader examples

- os.Stdin
- os.File
- strings.NewReader(s **string**)
- bytes.Buffer
- etc.

Writer examples

- os.Stdout
- os.File
- bytes.Buffer
- http.ResponseWriter
- etc.

Use io.Reader and io.Writer whenever you deal with streams of data

```
func myPrint(s string)  {
     fmt.Print(s)
}

func main() {
     myPrint("hello world")
}
```

VS

```
func myPrint(r io.Reader)  {
     io.Copy(os.Stdout, r)
}

func main() {
     myPrint(strings.NewReader(“hello world”))
}
```

Simple usage

```
s := "longer than 8 characters"
r := strings.NewReader(s)
var txt []byte
for {
    b := make([]byte, 8)
    _, err := r.Read(b)
    if err == io.EOF {
      break
    }
    txt = append(txt, b...)
}

fmt.Println(string(txt))
//output: longer than 8 chracters
```

func ioutil.ReadAll

```
// ReadAll reads from r until an error
// or EOF and returns the data it read.
func ReadAll(r io.Reader) ([]byte, error{
   return readAll(r, bytes.MinRead)
}
```

Sometimes overused

```
s := "longer than 8 characters"
r := strings.NewReader(s)
txt, _ := ioutil.ReadAll(r)

fmt.Println(string(txt))
//output: longer than 8 chracters
```

io.Copy

```
// Copy copies from src to dst until either EOF is reached
// on src or an error occurs.
func Copy(dst Writer, src Reader) (written int64, err error)
```

```
io.Copy(os.Stdout, os.Stdin)
```

io.Copy

This function uses a **32KB** buffer to read from **src** and then write to **dst**. If any error besides io.EOF occurs in the read or write then the copy is stopped and the error is returned.

How to modify the buffer size?

```
// CopyBuffer can modify this size
func CopyBuffer(dst Writer, src Reader, buf []byte) (written int64, err error)
```

Concatenate Reader

```
r1 := strings.NewReader("golang")
r2 := strings.NewReader("bcn")

for _, r := range []io.Reader{r1, r2} {
    _, err := io.Copy(os.Stdout, r)
    if err != nil {
        panic(err)
    }
}

//output: golangbcn
```

Multiple Writer

```
w1 := os.Stdout
w2 := new(bytes.Buffer)

for _, w := range []io.Writer{w1, w2} {
  _, err := io.Copy(w, strings.NewReader("hi"))
  if err != nil {
    panic(err)
  }
}
fmt.Println(w2)

//output: hihi
```

io.MultiReader

```
r1 := strings.NewReader("golang")
r2 := strings.NewReader("bcn")

mr := io.MultiReader(r1,r2)
io.Copy(os.Stdout, mr)

//output: golangbcn
```

io.MultiWriter

```
w1 := os.Stdout
w2 := new(bytes.Buffer)

mw := io.MultiWriter(w1, w2)

fmt.Fprint(mw, "hi")
fmt.Print(w2)

//output: hihi
```

In Action

- Files
- Images
- Network connection
- Compressed data
- etc.

Decompressors

```
//$ cat hello.txt.gz | go run main.go

r, _ := gzip.NewReader(os.Stdin)
io.Copy(os.Stdout, r);
```

http.Response body is a reader

```
resp, _ := http.Get("https://www.google.com")
io.Copy(os.Stdout, resp.Body)
```

Working with images

```
func image.Decode(r io.Reader) (Image, string, error)
func jpeg.Encode(w io.Writer, m image.Image, o *Options) error
```

```
f, err := os.Create("foo.jpg")
if err != nil {
    panic(err)
}
defer f.Close()

img, _, err := image.Decode(os.Stdin)
if err != nil {
    panic(err)
}

err = jpeg.Encode(f, img, nil)
if err != nil {
    panic(err)
}
```

JSON Decoder

```
{
   "name": "Bob Smith",
   "location":"BCN",
   "age": 22
}
```

```
type person struct {
   Name      string      `json:"name"`
   Location  string      `json:"location"`
   Age       int         `json:"age"`
}
```

```
var p person
json.NewDecoder(os.Stdin).Decode(&p)
```

Limiting techniques

```
io.Copy(os.Stdout, io.LimitReader(os.Stdin, 8))
```

```
b := make([]byte, 8)
io.ReadFull(os.Stdin, b)
fmt.Fprintln(os.Stdout, string(b))
```

```
if _, err := io.CopyN(os.Stdout, os.Stdin, 27); err != nil {
    log.Fatal(err)
}
```

The TeeReader

```
func TeeReader(r Reader, w Writer) Reader
```

```
tr := io.TeeReader(strings.NewReader("hello"), os.Stdout)
f, err := os.Create("foo.txt")
defer f.Close()

if err != nil {
    panic(err)
}

io.Copy(f, tr)
```

Example usage of TeeReader

```
var buf bytes.Buffer
body := io.TeeReader(req.Body, &buf)

// ... process body ...

if err != nil {
    // inspect buf
    return err
}
```

Optimizing string writes

```
func io.WriteString(w Writer, s string) (n int, err error)
```

```
io.WriteString(os.Stdout, "hello world")
```

Demo time

Let’s write our own Reader and Writer

Links

https://golang.org/pkg/io/

https://github.com/miku/exploreio

https://github.com/campoy/justforfunc/tree/master/19-pipes

https://github.com/ardanlabs/gotraining/tree/master/topics/go/packages/io

https://medium.com/go-walkthrough/go-walkthrough-io-package-8ac5e95a9fbd

THANKS