Beautiful IO

> A tour through standard library pkg/io and various implementations of its interfaces.

Golab 2019, 2019–10–21, Florence

Martin Czygan

About me

SWE @ubleipzig working mostly with Python and Go.

Taming data – open source – writing.

> Explore IO workshop at Golab 2017.

Background

- Go Proverbs (2015)

> The bigger the interface, the weaker the abstraction.

Prominent examples are `io.Reader` and `io.Writer`.

The IO package

- contains basic, widely used interfaces (within and outside standard library)
- utility functions

Why beautiful?

> La bellezza è negli occhi di chi guarda

- small, versatile interfaces
- composable

Praise and love

> This article aims to convince you to use io.Reader in your own code wherever you can. -- @matryer

> "Crossing Streams: a love letter to Go io.Reader" -- @jmoiron

> Which brings me to io.Reader, easily my favourite Go interface. -- @davecheney

What's in pkg/io?

- 25 types
- 21/25 are interfaces
- 12 functions, 3 constants, 6 errors

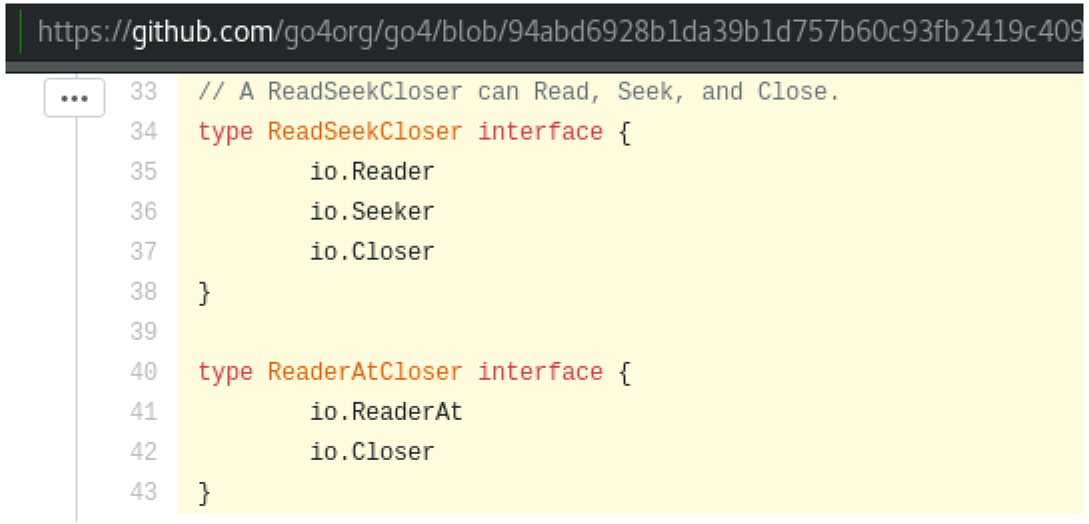
The concrete types are: `LimitedReader`, `PipeReader`, `PipeWriter`, `SectionReader`; functions: `Copy`, `CopyN`, `CopyBuffer`, `Pipe`, `ReadAtLeast`, `ReadFull`, `WriteString`, `LimitReader`, `MultiReader`, `TeeReader`, `NewSectionReader`, `MultiWriter`

A few Interfaces

	R	W	C	S
io.Reader	x			
io.Writer		x		
io.Closer			x	
io.Seeker				x
io.ReadWriter	x	x		
io.ReadCloser	x		x	
io.ReadSeeker	x			x
io.WriteCloser		x	x	
io.WriteSeeker		x		x
io.ReadWriteCloser	x	x	x	
io.ReadWriteSeeker	x	x		x

Missing interfaces

You might find some missing pieces elsewhere.



https://github.com/go4org/go4/blob/94abd6928b1da39b1d757b60c93fb2419c409

```go
// A ReadSeekCloser can Read, Seek, and Close.
type ReadSeekCloser interface {
	io.Reader
	io.Seeker
	io.Closer
}

type ReaderAtCloser interface {
	io.ReaderAt
	io.Closer
}
```

How many readers, writers are there?

```
$ guru -json implements /usr/local/go/src/io/io.go:#3309
```

I counted over 280 implementations of io.Reader and X of io.Writer.

What is a Reader?

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

The reader implementation will populate a given byte slice.

- at most `len(p)` bytes are read
- to signal the end of a stream, return `io.EOF`

There is some flexibility around the end of a stream.

> Callers should always process the n > 0 bytes returned before considering the error err. Doing so correctly handles I/O errors that happen after reading some bytes and also both of the allowed EOF behaviors.

Notes

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

- The byte slice is under the control of the caller.

> Implementations must not retain p.

This hints at the streaming nature of this interface.

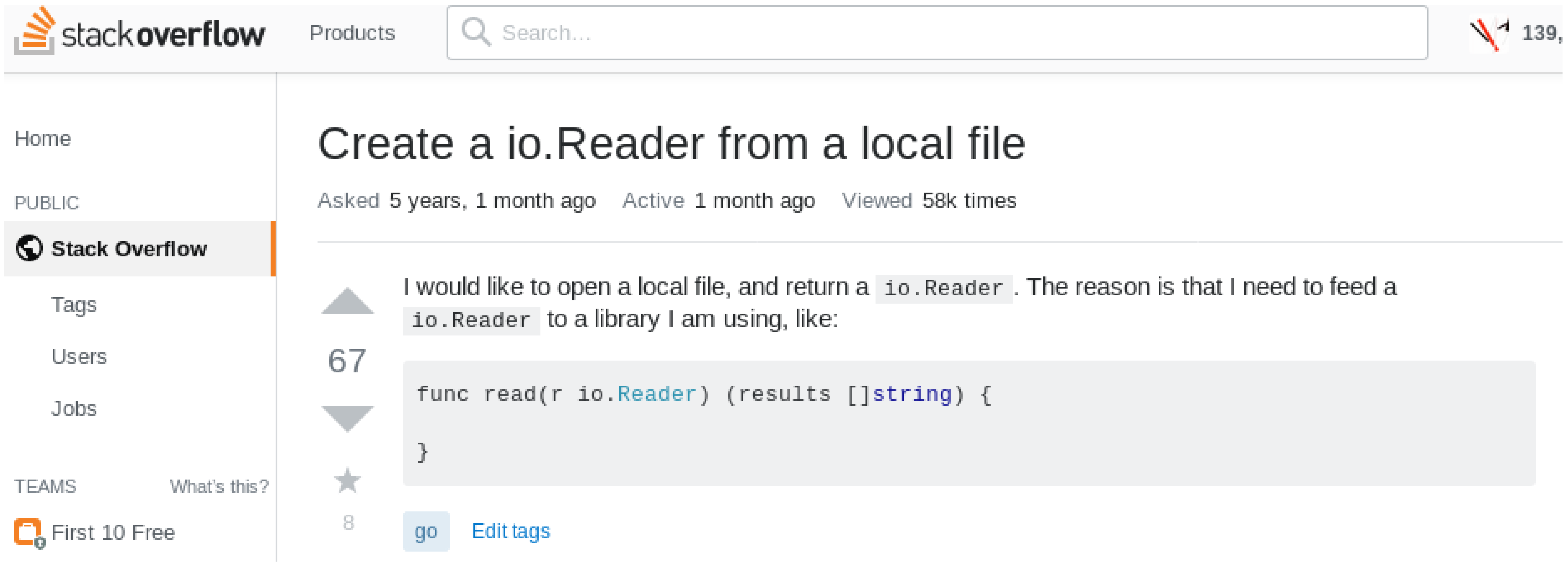
Implementations

- files
- network connections
- standard input and output
- compression
- hashing
- encoding
- formatting
- ...

Many uses in testing as well.

Conversions

Are not required.



Create a io.Reader from a local file

Asked 5 years, 1 month ago Active 1 month ago Viewed 58k times

67

8

I would like to open a local file, and return a `io.Reader`. The reason is that I need to feed a `io.Reader` to a library I am using, like:

```
func read(r io.Reader) (results []string) {

}
```

go Edit tags