

Beautiful (and strange) I/O

Lightning Talk, Go and Cloud Native Leipzig

<https://golangleipzig.space>

Martin Czygan

@BasislagerCo, 2019-06-14, 19:00

Go Proverb

- The bigger the interface, the weaker the abstraction

Exemplified in package io

Generic I/O with `io.Reader` and `io.Writer`. A few other 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.WriterSeeker		x		x
io.ReadWriteCloser	x	x	x	
io.ReadWriteSeeker	x	x		x

Missing things

Libraries might implement missing pieces, e.g.

- [ReadSeekCloser](#), [ReaderAtCloser](#)

From: github.com/go4org/go4.

IO interface list

- `io.ReaderAt (offset)`
- `io.ReaderFrom`
- `io.WriterAt (offset)`
- `io.WriterTo`

Use cases | io.ReaderAt

- io.ReaderAt, io.WriterAt -- (parallel writes) with offset

Sidenote: For filesystems, there is a [pread\(2\) system call](#) in Linux

read from or write to a file descriptor at a given offset ...

The pread() and pwrite() system calls are especially useful in **multithreaded applications**. They allow multiple threads to perform I/O on the **same file descriptor** without being affected by changes to the file offset by other threads.

- HTTP [range request example](#) (on zip without download)

Use cases | io.ReaderFrom

- Optimizing Copy

To avoid using an intermediate buffer entirely, types can implement interfaces to read and write directly. When implemented, the Copy() function will avoid the intermediate buffer and use these implementations directly.

- maybe: io.ReaderFrom — a data structure, that know how to deserialize itself

Example: different JSON API structs, but each of them implements io.ReaderFrom, so the data fetch can be separated --`fetchLocation(location string, r io.ReaderFrom)`

io.ReaderFrom is an optional interface

- Enabling optional optimizations/features

Readers for various types

Rune

- `io.RuneReader` (read a rune)
- `io.RuneScanner` (support for rewind)

Byte

- `io.ByteReader` (read a byte)
- `io.ByteScanner` (support for rewind)
- `io.ByteWriter`

String

- `io.StringWriter` (new in 1.12)

Who implements these interfaces?

- files, atomic files
- buffered io
- network connections
- response bodies
- compression algorithms
- hash sums
- images
- JSON and XML encoders and decoders
- utilities like counters, test data generators, stream splitters, mutli-readers, ... and more

A simple interface

Reader and Writer are single method interfaces.

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

Examples

Few examples for usage and custom implementations.

Empty reader and Discard

- [Empty](#)
- [Discard](#)

The standard library implementation of [ioutil.Discard](#).

Example: multireader

- MultiReader

Example: Embedding a reader

- [Embedding a reader](#)

This is also part of the Go Tour, currently in exercise [methods/23](#).

Example: Endless stream

- Endless stream

Example: Blackout

- Censoring reader

Example: stickyErrWriter

- [stickyErrWriter](#)

From [live hacking](#) with Brad and Andrew.

I am a collector of implementations

If you happen to come across an interesting implementation, please let me know - E-Mail, via issue on [exploreio](#), [@cvvfj](#), ...

Links:

- <https://golang.org/pkg/io/> (docs)
- <https://www.datadoghq.com/blog/crossing-streams-love-letter-gos-io-reader/> (love letter)
- <https://medium.com/go-walkthrough/go-walkthrough-io-package-8ac5e95a9fbd> (walkthrough)
- <https://www.youtube.com/watch?v=PAAkCSZUG1c> (Go Proverbs, 2015)
- <https://github.com/miku/exploreio> (example implementations)