Google I/O Extended

Go's Byte Bridge: io.Reader and io.Writer deep dive

Miriah Peterson

Google I/O Extended

Never have I ever?

- Written a handleFunc() for an http.Server{}?
- Created an http.Request{} for an http.Client{}
- Read from a sq.DB?
- Written to or read from a file?
- Accessed an env variable?
- Accepted a CLI flag or argument?

What is I/O?

> In computing, input/output (I/O, i/o, or informally io or IO) is the communication between an information processing system, such as a computer, and the outside world, such as another computer system, peripherals, or a human operator (https://en.wikipedia.org/wiki/Input/output)

What is I/O?

In the go ecosystem we are mostly talking about from inside the go app's memory to outside the go apps memory.

Readers

What is a reader?

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

Readers

> The io package specifies the io.Reader interface, which represents the read end of a stream of data.
> (https://go.dev/tour/methods/21)

Readers

It takes the data from one location and ingests it to another location inside the go app

Writers

What is a writer?

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

It takes data from inside the fo app and sends it to another location outside the go app.

Inhereting Readers/Writers

```
// ReadWriter is the interface that groups the basic Read and Write methods.
type ReadWriter interface {
        Reader
        Writer
}

// ReadCloser is the interface that groups the basic Read and Close methods.
type ReadCloser interface {
        Reader
        Closer
}
```

src/io/io.go

How do I implement Readers and Writers?

How many times are the io.Reader and io.Writer interfaces implemented inside the standard lib?

- Std lib Reader
- Std lib Writer

Google I/O Extended

I/O Patterns

When choosing to write to and from memory, there are two major type of operations buffered and non-buffered I/o operations

Google I/O Extended

I/O Patterns

Who can tell me the difference?

File I/O

```
func readAllFile() {
        f, err := os.Open("twitchChat.txt")
        if err != nil {
                log.Fatal(err)
        }
        defer f.Close()
        _, err = io.ReadAll(f)
        if err != nil {
                log.Fatal(err)
        }
}
```

Google I/O Extended

File I/O

When to use classic I/O patterns

- Does memory matter?
- High Synchronization
- Realtime events

Buffered I/O

Buffered I/O in Go is any read write operation in the bufio module. These operators store the data in memory before it sends the data outside the api.

Buffered I/O

When to Use buffered I/O

- Optimized Throughput
- Convenient Error Handling

Buffered I/O (generated example)

```
func readFileBuf() {
        file, err := os.Open("twitchChat.txt")
        if err != nil {
                log.Fatal(err)
        }
        defer file.Close()
        scanner := bufio.NewScanner(file)
        for scanner.Scan() {
                line := scanner.Text()
                fmt.Println(line)
        }
        if err := scanner.Err(); err != nil {
                log.Fatal(err)
        }
}
```

Database I/O

Database drivers implement the io.Readers/Writers to manage the data stream between the data store and the go software app.

- These implementation happen in the drivers.
- sqlite and duckdb-go use cgo to manage their file writes and the io module
- establishing a new db connection every time you need to access the db is very inefficient, for efficient db write use a connection pool.

Database I/O

Google I/O Extended

```
func readfromdb() {
        filename := "twitchchat.db"
        db, err := sql.open("sqlite3", filename)
        if err != nil {
                log.fatal(err)
        }
        defer db.close()

        rows, err := db.query("select * from chat")
        if err != nil {
                log.fatal(err)
        }
        for rows.next() {
                var id int
                var message string
                rows.scan(&id, &message)
        }
}
```

I/O over the internet

In go, when we are making calls over ip, we are still using the io.reader/writer. all data is sent as a stream, so the chucks of bytes instead of being directly written are just sent as data in a packet.

Google I/O Extended

Network I/O

```
func messageHandler(w http.ResponseWriter, r *http.Request) {
        w.Header().Set("Content-Type", "text/plain")

        _, err := fmt.Fprintf(w, testString)
        if err != nil {
                log.Println("Error writing message:", err)
        }
}
```

Google I/O Extended

Demo

code

Google I/O Extended

In Summary

- Every Go App leverages I/O operations
- There are lots of options for implementations of io.Reader and io.Writer
- What kinds of operations Readers and Writers have
- How to evaluate our Readers and Writers

Google I/O Extended

Contact

- Twitter @captainnobody1
- GitHub soypete
- LinkedIn
- twitch