Advanced Patterns with io.ReadWriter

18 Aug 2016

Paul Bellamy
Software Engineer, Weaveworks

Advanced Patterns with io.ReadWriter

18 Aug 2016

Paul Bellamy
Software Engineer, Weaveworks

- Goal: After this talk you'll be able to say "That's not advanced, that's obvious!”
- We’ll start with some basics, and build up.
- There will be quite a lot of code. We’ll go through stuff fairly quick
- Slides linked at end

- A simple, portable, and reliable way to network and manage containers and microservices.

- Not this kind of reader

Basics - What's Available in io, bufio, & ioutil?

- Simply put? Building blocks, and great examples
- Tools that can help your design without compromising performance

Basics - What's Available in io, bufio, & ioutil?

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

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

- ReadWriter is one of the most powerful abstractions available in Go.
- Simple, Expressive, Composable, Interchangeable

Basics - What's Available in io, bufio, & ioutil?

Tools for working with ReadWriters:

- io.Copy
- io.LimitReader
- io.TeeReader
- bufio.Scanner
- ioutil.Discard

Loads more!

- Analogy to unix pipes
- Loads more in standard library

Basics - Composition

- Building our own abstractions around Readers is really easy
- And composing them
- Can be interchanged

Basics - Composition

Setup:

```
var r io.Reader
r = strings.NewReader("1234567890")
r = io.LimitReader(r, 5)
```

Output:

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

- LimitedReader limits the amount of data returned to N bytes, like “head -n5”
- After N bytes have been read, it returns io.EOF, the signal for "no more data"
- Bytes will be left in the strings.NewReader
- LimitReader source is very readable
- Wraps our number stream, Still fulfills same interface
- io.Copy is Backwards

Basics - Composition

Setup:

```
var r io.Reader
r = strings.NewReader("1234567890")
r = io.LimitReader(r, 5)
```

Output:

```
io.Copy(gzip.NewWriter(os.Stdout), r)
```

- gzip writer example
- use this a lot handling http
- free performance

Example 1 - HTTP Chunking

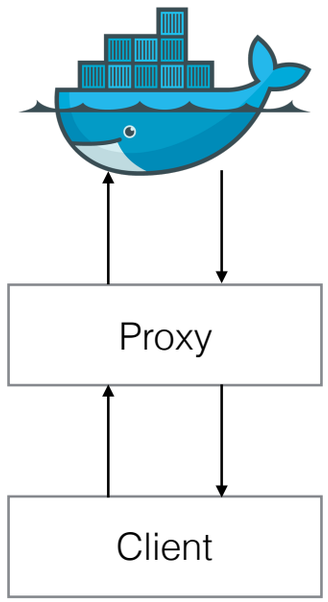
Example 1 - HTTP Chunking

Let's transparently proxy a chunked HTTP in a stream.

- What does transparently mean?
- What does HTTP chunked encoding look like?

Example 1 - HTTP Chunking

Let's transparently proxy a chunked HTTP in a stream.



- Friendly ocean wildlife
- No relation to sf tech companies real or fictitious

Example 1 - HTTP Chunking

```
4\r\n
Wiki\r\n
5\r\n
pedia\r\n
E\r\n
 in\r\n
\r\n
chunks.\r\n
0\r\n
Date: Sun, 06 Nov 1994 08:49:37 GMT\r\n
Content-MD5: 1B2M2Y8AsgTpgAmY7PhCfg==\r\n
\r\n
```

- Body is the interesting part here.
- Each chunk has a hex length, then carriage-return-linefeed
- E chunk is 14 characters long.
- Body ends with 0-length chunk
- Note the two trailers, Date, and Content-MD5

Example 1 - HTTP Chunking

```
Wikipedia in

chunks.
```

- Client should only care about this
- Some expect json objects to align

Solution 1

Obvious solution with io.Copy


```
io.Copy(clientWriter, responseBody)
```

Solution 1

Obvious solution with io.Copy


```
io.Copy(clientWriter, responseBody)
```

Problems:

- Won't let us process the Trailers separately
- We need to stop after the body

Solution 2

Better solution with httputil.ChunkedReader


```
io.Copy(
  clientWriter,
  httputil.NewChunkedReader(responseBody),
)
parseTrailers(responseBody)
```

- Dig around in httputil, and find httputil.ChunkedReader
- Is great, and probably good enough for most of the time
- But, has some issues

Solution 2

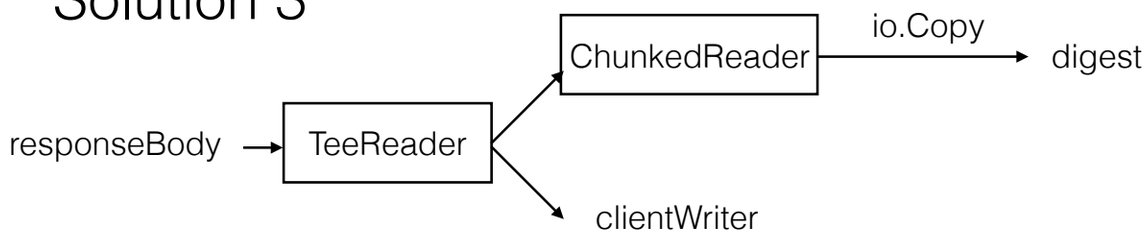
Better solution with httputil.ChunkedReader

Problems:

- Strips out the chunking data
- Can’t validate the MD5

- Turns it into one continuous byte-stream
- Breaks clients relying on specific chunks

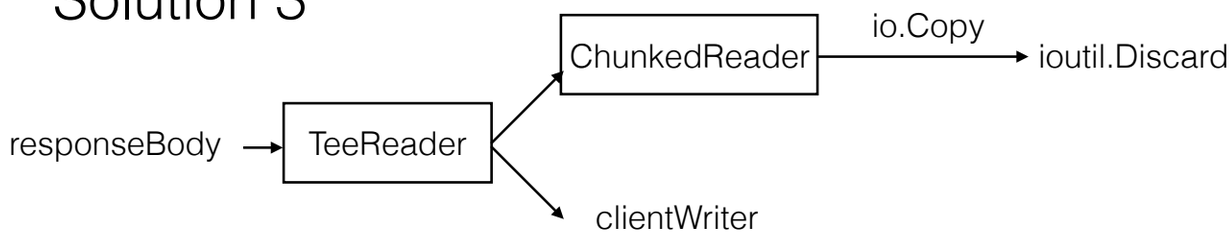
Solution 3




```
digest := md5.New()
io.Copy(digest,
  httputil.NewChunkedReader(
    io.TeeReader(
      responseBody,
      clientWriter,
    ),
  ),
)
parseTrailers(responseBody)
```

- Every read from TeeReader writes the raw data to clientWriter
- ChunkedReader to detect the end of the body
- Like a transistor
- What if we don’t care about the digest

Solution 3




```
io.Copy(ioutil.Discard,
  httputil.NewChunkedReader(
    io.TeeReader(
      responseBody,
      clientWriter,
    ),
  ),
)
parseTrailers(responseBody)
```

- If we don’t care about the md5
- Still need to perform the io.Copy
- 1. Could we implement this in a single reader? Yes
- 2. Would that be more performant, or less? Depends on number of copies

Example 2 - Multiplexing

- Something a bit lighter
- Doing some build stuff for weave

Example 2 - Multiplexing

Run subcommands, multiplex the output, give each a unique prefix.

```
echo “Hello\nWorld!”
make all
git status
```

```
[echo] Hello,
[make] make: *** No rule to make target.
[echo] World!
[git] On branch master
[git] Your branch is up-to-date
[git] nothing to commit
```

- Running a bunch of subcommands
- Multiplexing the streaming output
- Can’t just do strings.Split

Example 2 - Multiplexing

	Output	Prefixed	Merged
echo	Hello, World!	[echo] Hello, [echo] World!	[echo] Hello, [echo] World! [git] Branch: master [git] Up-to-date ...
git	Branch: master Up-to-date ...	[git] Branch: master [git] Up-to-date ...	

- Example pipeline

Example 2 - Multiplexing

Starting up the commands

```
commands := [][]string{
  {"echo", "Hello,\nWorld!"},
  {"make", "all"},
  {"git", "status"},
}

for _, cmd := range commands {
  c := exec.Command(cmd[0], cmd[1])

  // Wire up our prefix writer
  prefixedOutput := prefixingWriter(cmd[0], os.Stdout)
  c.Stdout = prefixedOutput
  c.Stderr = prefixedOutput

  // Start each in a goroutine
  go func() { done <- c.Run() }()
}
```

- Take list of commands
- Set up each one
- Create prefixingWriter for each (We'll implement that)
- Run them async

Example 2 - Multiplexing

prefixingWriter

```
func prefixingWriter(prefix string, output io.Writer) io.Writer
```

- Type signature
- Takes a prefix, and an output source
- Returns a new writer
- Any data written to this will have prefix on each line

Example 2 - Multiplexing

Splitting streams into sections with bufio.Scanner

```
input := ...

// Iterate over each line
scanner := bufio.NewScanner(input)
scanner.SplitFunc(bufio.ScanLines)

for scanner.Scan() {

  // Write the prefix into the output
  fmt.Fprintf(output, "[%s] ", prefix)

  // Copy the line
  output.Write(scanner.Bytes())

  // Re-add a newline (scanner removes it)
  fmt.Fprint(output, "\n")

}
```

- A scanner turns streams of bytes into streams of tokens
- Default is split by line, others are utf-8 runes, words, or custom.

Example 2 - Multiplexing

Splitting streams into sections with bufio.Scanner

```
input := ...

// Iterate over each line
scanner := bufio.NewScanner(input)
scanner.SplitFunc(bufio.ScanLines)

for scanner.Scan() {

  // Write the prefix into the output
  fmt.Fprintf(output, "[%s] ", prefix)

  // Copy the line
  output.Write(scanner.Bytes())

  // Re-add a newline (scanner removes it)
  fmt.Fprint(output, "\n")

}
```

- This lets us iterate over the lines in a reader

Example 2 - Multiplexing

Splitting streams into sections with bufio.Scanner

```
input := ...

// Iterate over each line
scanner := bufio.NewScanner(input)
scanner.SplitFunc(bufio.ScanLines)

for scanner.Scan() {

  // Write the prefix into the output
  fmt.Fprintf(output, "[%s] ", prefix)

  // Copy the line
  output.Write(scanner.Bytes())

  // Re-add a newline (scanner removes it)
  fmt.Fprint(output, "\n")

}
```

- Add our prefix

Example 2 - Multiplexing

Splitting streams into sections with bufio.Scanner

```
input := ...

// Iterate over each line
scanner := bufio.NewScanner(input)
scanner.SplitFunc(bufio.ScanLines)

for scanner.Scan() {

  // Write the prefix into the output
  fmt.Fprintf(output, "[%s] ", prefix)

  // Copy the line
  output.Write(scanner.Bytes())

  // Re-add a newline (scanner removes it)
  fmt.Fprint(output, "\n")

}
```

- Copy the original bytes across

Example 2 - Multiplexing

Splitting streams into sections with bufio.Scanner

```
input := ...

// Iterate over each line
scanner := bufio.NewScanner(input)
scanner.SplitFunc(bufio.ScanLines)

for scanner.Scan() {

  // Write the prefix into the output
  fmt.Fprintf(output, "[%s] ", prefix)

  // Copy the line
  output.Write(scanner.Bytes())

  // Re-add a newline (scanner removes it)
  fmt.Fprint(output, "\n")

}
```

- Add back a newline
- Gotcha with scanners

Example 2 - Multiplexing

Splitting streams into sections with bufio.Scanner

```
input := ...

// Iterate over each line
scanner := bufio.NewScanner(input)
scanner.SplitFunc(bufio.ScanLines)

for scanner.Scan() {

  // Write the prefix into the output
  fmt.Fprintf(output, "[%s] ", prefix)

  // Copy the line
  output.Write(scanner.Bytes())

  // Re-add a newline (scanner removes it)
  fmt.Fprint(output, "\n")

}
```

- Not instantiating string for performance (allocation, unicode handling)
- mention BufferedReader and Fscanf

Example 2 - Multiplexing

Splitting streams into sections with bufio.Scanner

```
input := ...

// Iterate over each line
scanner := bufio.NewScanner(input)
scanner.SplitFunc(bufio.ScanLines)

for scanner.Scan() {

  // Write the prefix into the output
  fmt.Fprintf(output, "[%s] ", prefix)

  // Copy the line
  output.Write(scanner.Bytes())

  // Re-add a newline (scanner removes it)
  fmt.Fprint(output, "\n")

}
```

- What is this input thing up top?
- This is what we'll return from PrefixingWriter

Example 2 - Multiplexing

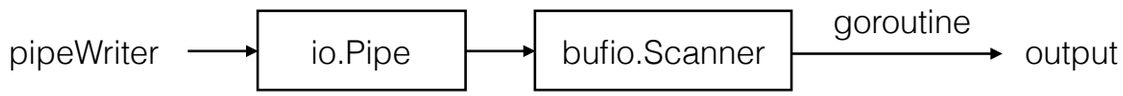
What is input?

- bufio.Scanner expects a io.Reader
- exec.Cmd expects a io.Writer

- Input needs to connect a Writer to a Reader

Example 2 - Multiplexing

io.Pipe to the rescue



- io.Pipe two synchronous halves, similar to a chan

Example 2 - Multiplexing

```
func prefixingWriter(prefix string, output io.Writer) io.Writer {
    pipeReader, pipeWriter := io.Pipe()

    scanner := bufio.NewScanner(pipeReader)
    scanner.SplitFunc(bufio.ScanLines)

    go func() {
        for scanner.Scan() {
            fmt.Fprintf(output, "[%s] ", prefix)
            output.Write(scanner.Bytes())
            fmt.Fprint(output, "\n")
        }
    }()

    return pipeWriter
}
```

- Pipe to connect the reader and writer, Scanner to tokenize into lines, Goroutine prefixing and merging into output
- 1. Could this leak goroutines? Yes, if output blocks indefinitely
- 2. Avoid leaking goroutines? Don’t use a goroutine. Or make output a WriteCloser, to guarantee cleanup
- 3. How to save the output until the process exited? bytes.Buffer

Example 2 - Multiplexing

```
func prefixingWriter(prefix string, output io.Writer) io.Writer {
    pipeReader, pipeWriter := io.Pipe()

    scanner := bufio.NewScanner(pipeReader)
    scanner.SplitFunc(bufio.ScanLines)

    go func() {
        for scanner.Scan() {
            fmt.Fprintf(output, "[%s] ", prefix)
            output.Write(scanner.Bytes())
            fmt.Fprint(output, "\n")
        }
    }()

    return pipeWriter
}
```

- Pipe to connect the reader and writer, Scanner to tokenize into lines, Goroutine prefixing and merging into output

Example 2 - Multiplexing

```
func prefixingWriter(prefix string, output io.Writer) io.Writer {
    pipeReader, pipeWriter := io.Pipe()

    scanner := bufio.NewScanner(pipeReader)
    scanner.SplitFunc(bufio.ScanLines)

    go func() {
        for scanner.Scan() {
            fmt.Fprintf(output, "[%s] ", prefix)
            output.Write(scanner.Bytes())
            fmt.Fprint(output, "\n")
        }
    }()

    return pipeWriter
}
```

- Scanner to tokenize into lines, as before

Example 2 - Multiplexing

```
func prefixingWriter(prefix string, output io.Writer) io.Writer {
    pipeReader, pipeWriter := io.Pipe()

    scanner := bufio.NewScanner(pipeReader)
    scanner.SplitFunc(bufio.ScanLines)

    go func() {
        for scanner.Scan() {
            fmt.Fprintf(output, "[%s] ", prefix)
            output.Write(scanner.Bytes())
            fmt.Fprint(output, "\n")
        }
    }()

    return pipeWriter
}
```

- Note, new goroutine
- Other ways to do this, avoiding a grouting

Example 2 - Multiplexing

```
func prefixingWriter(prefix string, output io.Writer) io.Writer {
    pipeReader, pipeWriter := io.Pipe()

    scanner := bufio.NewScanner(pipeReader)
    scanner.SplitFunc(bufio.ScanLines)

    go func() {
        for scanner.Scan() {
            fmt.Fprintf(output, "[%s] ", prefix)
            output.Write(scanner.Bytes())
            fmt.Fprint(output, "\n")
        }
    }()

    return pipeWriter
}
```

- 1. Could this leak goroutines? Yes, if output blocks indefinitely
- 2. Avoid leaking goroutines? Don’t use a goroutine. Or make output a WriteCloser, to guarantee cleanup
- 3. How to save the output until the process exited? bytes.Buffer

Thank you

18 Aug 2016

Paul Bellamy

Software Engineer, Weaveworks

paul@weave.works

github.com/paulbellamy/golanguk2016

@pyrhho

- Questions: Find me in the break room after
- Slides will be at this github repo