Dhanush Gopinath Follow
Co-founder & CTO @geektrust. Gopher. Polyglot.
Feb 1 · 2 min read

Concurrent HTTP downloads using Go

Recently for an implementation of feature in Geektrust, we had to download multiple images from the internet and use it. This had to be an efficient implementation. This post is a brief explanation of the simple implementation done using goroutines and channels.

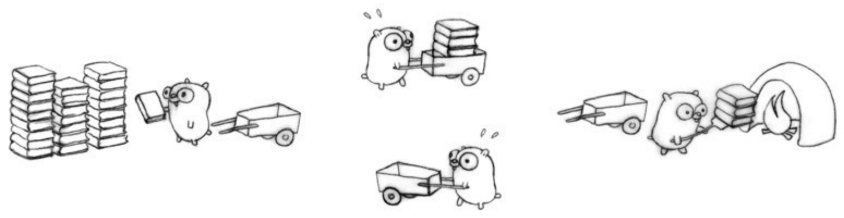

Image courtesy — https://hackernoon.com/why-we-love-concurrency-and-you-should-too-c64c2d08a059

Go has an http package which has simple APIs to do HTTP operations. For e.g. if you want to download a page all you need to do is this.

```
resp, err := http.Get("http://example.com/")
if err != nil {
        // handle error
}
defer resp.Body.Close()
body, err := ioutil.ReadAll(resp.Body)
```

Given below is the implementation of downloading a file from a public URL and using it as bytes.

```go
func downloadFile(URL string) ([]byte, error) {
        response, err := http.Get(URL)
        if err != nil {
                return nil, err
        }
        defer response.Body.Close()
        if response.StatusCode != http.StatusOK {
                return nil, errors.New(response.Status
        }
        var data bytes.Buffer
        _, err = io.Copy(&data, response.Body)
        if err != nil {
```

Download a file represented by a public url

For downloading multiple files concurrently you can use the help of goroutines and channels.

```go
func downloadMultipleFiles(urls []string) [][]byte {
        done := make(chan []byte, len(urls))
        errch := make(chan error, len(urls))
        for _, URL := range urls {
                go func(URL string) {
                        b, err := downloadFile(URL)
                        if err != nil {
                                errch <- err
                                done <- nil
                                return
                        }
                        done <- b
                        errch <- nil
                }(URL)
        }
        bytesArray := make([][]byte, 0)
        var errStr string
```

Download multiple files using goroutines & channels

In the above code snippet, we initialise two channels, one for sending the file downloaded as ***[]byte*** type and another for sending any errors that happened while downloading.

We then loop though all the urls and call the ***downloadFile*** method for each of them in a goroutine. These goroutines are executed concurrently and so the for loop is finished processing in no time. Inside the goroutine each file is downloaded and we get back bytes and error.

If there is an error in the download, then the error is pushed to the error channel and nil value is pushed to the done channel. If there is no error then the bytes are pushed to the done channel and a nil entry is pushed to the error channel.

An array of byte arrays is declared to collect the downloaded files back. We again loop through as many times as there are URLs to collect the byte array back from the done channel.

```
bytesArray = append(bytesArray, <-done)
if err := <-errch; err != nil {
   errStr = errStr + " " + err.Error()
}
```

These two lines inside the for loop wait for the done and the error channel to receive data. Once the data has arrived, it increments the loop count. Bytes are appended to the array of bytes while error string is appended, if there are any errors.

You have now downloaded the files concurrently and they are available as an array of bytes. All just using simple Go code.