06

Monitorando os nossos sites

Transcrição

Agora que entendemos a questão das funções com múltiplos retornos, podemos ver o conteúdo da resposta da nossa requisição. Ao executar o programa e digitar o comando 1, temos uma resposta semelhante a essa:

```
&{200 OK 200 HTTP/2.0 2 0 map[X-Cloud-Trace-Context:[6f3fa7e590ac68bd43d76c82a67df476] Date:[Tue, 1
GMT] Server:[Google Frontend] X-Ua-Compatible:[IE=edge,chrome=1] Expires:[Tue, 13 Jun 2017 21:50:36
text/html] Cache-Control:[public, max-age=1800] Age:[1298] X-Dns-Prefetch-Control:[on]] 0xc4200d490
ap[] 0xc42000a800 0xc4203a3080}
```

Nessa resposta, temos o status da requisição, a data, os cabeçalhos, entre outras informações. O que nos informa se um site carregou com sucesso, ou teve algum problema, é o **status da requisição**. Quando temos um status 200, significa que o site foi carregado perfeitamente. Então, se obtivermos algum status diferente de 200, significa que o nosso site está com problema.

Para saber o status da resposta, podemos acessar a sua propriedade `StatusCode`. Logo, podemos fazer um `if`, se o status for 200, nós imprimimos uma mensagem de sucesso, mas se não for, nós imprimimos uma mensagem dizendo que o site está com problema, imprimindo o status em seguida:

```
// restante do código omitido

func iniciarMonitoramento() {
    fmt.Println("Monitorando...")
    site := "https://www.alura.com.br"
    resp, _ := http.Get(site)

    if resp.StatusCode == 200 {
        fmt.Println("Site:", site, "foi carregado com sucesso!")
    } else {
        fmt.Println("Site:", site, "está com problemas. Status Code:", resp.StatusCode)

    }
}
```

Ao executar o programa, recebemos uma mensagem de sucesso. Para não termos que ficar dependendo do site da Alura cair, podemos inventar uma URL inexistente, por exemplo:

```
// restante do código omitido

func iniciarMonitoramento() {
    fmt.Println("Monitorando...")
    // site com URL inexistente
    site := "https://www.alura.com.br/askjdbahsbciahsbca"
    resp, _ := http.Get(site)

    if resp.StatusCode == 200 {
        fmt.Println("Site:", site, "foi carregado com sucesso!")
    } else {
        fmt.Println("Site:", site, "está com problemas. Status Code:", resp.StatusCode)
```

```
        fmt.Println("Site:", site, "está com problemas. Status Code:", resp.StatusCode)
    }
}
```

Assim recebemos um status code 404. Ou podemos utilizar o site https://random-status-code.herokuapp.com/ (https://random-status-code.herokuapp.com/), que nos retorna um status diferente à cada requisição.

Colocando o nosso programa em loop

Conseguimos monitorar o site, mas assim que monitoramos, o programa é encerrado. O ideal é que o programa fique rodando em loop, eternamente, e só pare de rodar quando nós quisermos.

Em outras linguagens de programação, poderíamos utilizar o `while`, mas ele não existe no Go! Para isso, vamos utilizar a instrução `for`, sem passar nada para ela, pois assim ela ficará em loop eternamente:

```
package main

import (
    "fmt"
    "net/http"
    "os"
)

func main() {
    exibeIntroducao()

    for {
        exibeMenu()
        comando := leComando()

        switch comando {
        case 1:
            iniciarMonitoramento()
        case 2:
            fmt.Println("Exibindo Logs...")
        case 0:
            fmt.Println("Saindo do programa")
            os.Exit(0)
        default:
            fmt.Println("Não conheço este comando")
            os.Exit(-1)
        }
    }
}

// outras funções omitidas
```

Agora, ao rodar o nosso programa, vemos que após digitar o comando e ter o seu código executado, o menu volta a ser exibido, logo o nosso código está em loop. Para sair do loop, basta que digitemos a opção 0.

Com isso, avançamos mais na nossa aplicação, nos próximos capítulos colocaremos mais sites para o programa monitorar, escrever no log, entre outras funcionalidades!