05

Lendo de um arquivo

Transcrição

Se o erro que está ocorrendo é que o sites.txt não existe, então vamos criá-lo dentro da pasta hello, com o conteúdo que estava no nosso slice:

```
https://random-status-code.herokuapp.com/
https://www.alura.com.br
https://www.caelum.com.br
```

Agora, ao executar o programa e iniciar o monitoramento, vemos um código estranho sendo impresso, isso acontece pois o que a função `Open` nos retorna nada mais é do que um ponteiro para o arquivo em si.

Então, precisamos ler o arquivo de outro jeito, e há diversas maneiras para isso. Uma delas, um jeito fácil e rápido para ler um arquivo inteiro, sem ter que ler linha a linha, é utilizar o pacote `io/ioutil` .

Lendo os dados de um arquivo

Através do pacote `io/ioutil` , chamamos a função `ReadFile` , para ler o arquivo passado:

```
// restante do código omitido

func leSitesDoArquivo() []string {

    var sites []string
    arquivo, err := ioutil.ReadFile("sites.txt")

    if err != nil {
        fmt.Println("Ocorreu um erro:", err)
    }

    fmt.Println(arquivo)

    return sites
}
```

Essa função nos retorna o arquivo em um array de bytes, então basta convertê-los para uma string através da função `string` :

```
// restante do código omitido

func leSitesDoArquivo() []string {

    var sites []string
    arquivo, err := ioutil.ReadFile("sites.txt")

    if err != nil {
        fmt.Println("Ocorreu um erro:", err)
    }
```

```
    fmt.Println(string(arquivo))

    return sites
}
```

Ao executar o programa, iniciar o monitoramento, vemos o conteúdo do arquivo sendo impresso no console! Mas esse é um jeito bom para quem quer exibir o conteúdo todo do arquivo, o que não é muito útil para nós, já que queremos ler cada site de uma vez, para salvar cada um dentro do slice.

Lendo a primeira linha do arquivo

Para ler o arquivo linha a linha, vamos ver uma terceira forma de ler arquivos em Go, utilizando o pacote `bufio`. Para isso, vamos voltar à primeira leitura do arquivo, utilizando `os.Open`:

```
// restante do código omitido

func leSitesDoArquivo() []string {

    var sites []string
    arquivo, err := os.Open("sites.txt")

    if err != nil {
        fmt.Println("Ocorreu um erro:", err)
    }

    return sites
}
```

E com o `bufio`, nós criamos um leitor através da função `NewReader`, que recebe o arquivo a ser lido:

```
// restante do código omitido

func leSitesDoArquivo() []string {

    var sites []string
    arquivo, err := os.Open("sites.txt")

    if err != nil {
        fmt.Println("Ocorreu um erro:", err)
    }

    leitor := bufio.NewReader(arquivo)

    return sites
}
```

Com esse leitor, temos funções que nos ajudam a ler o arquivo, lendo linha a linha, como por exemplo a `ReadString`, que lê uma linha do arquivo e nos retorna em forma de string. Essa função deve receber um byte delimitador, para saber até onde queremos ler a linha.

No nosso caso, queremos ler a linha inteiro, ou seja, até a quebra de linha, que é representada pelo `\n`. Como queremos representar um byte, utilizaremos aspas simples:

```
// restante do código omitido

func leSitesDoArquivo() []string {

    var sites []string
    arquivo, err := os.Open("sites.txt")

    if err != nil {
        fmt.Println("Ocorreu um erro:", err)
    }

    leitor := bufio.NewReader(arquivo)

    leitor.ReadString('\n')

    return sites
}
```

Essa função nos retorna a linha e um possível erro, que vamos detectar. Além disso, vamos imprimir a linha:

```
// restante do código omitido

func leSitesDoArquivo() []string {

    var sites []string

    arquivo, err := os.Open("sites.txt")
    if err != nil {
        fmt.Println("Ocorreu um erro:", err)
    }

    leitor := bufio.NewReader(arquivo)

    linha, err := leitor.ReadString('\n')
    if err != nil {
        fmt.Println("Ocorreu um erro:", err)
    }

    fmt.Println(linha)

    return sites
}
```

Quando iniciamos o monitoramento, a primeira linha do arquivo é impressa. Mas queremos ler todas as linhas do arquivo, então como fazemos isso? Veremos no próximo vídeo.