04

Slices em Go

Transcrição

Para entendermos como funciona os Slices, essas abstrações do array em Go, vamos criar um, para aprendermos fazendo.

A primeira grande vantagem dele, é que, como foi falado no vídeo anterior, seu tamanho não é fixo, é dinâmico, indeterminado. Então, como exemplo, vamos criar a função `exibeNomes`, para testar a utilização dessa estrutura de dados.

Para criar um slice, podemos utilizar a declaração curta de variáveis. Sua declaração é bem parecida com a de um array:

```
func exibeNomes() {
    nomes := []string
}
```

Além disso, já na sua declaração, podemos preencher os seus dados, passando-os dentro de chaves, separados por vírgula:

```
func exibeNomes() {
    nomes := []string{"Douglas", "Daniel", "Bernardo"}
}
```

O slice infere o seu tamanho de acordo com a sua quantidade de elementos. Podemos verificar isso imprimindo a quantidade de itens contidos nele, através da função `len`:

```
func exibeNomes() {
    nomes := []string{"Douglas", "Daniel", "Bernardo"}
    fmt.Println(len(nomes))
}
```

Ao chamar essa função, vemos que o slice possui 3 itens.

Diferenças entre Array e Slice

Além do tamanho, que no array é fixo, e no slice não, quais são as outras diferenças entre eles? Na verdade, o slice nada mais é do que um array, é uma abstração que funciona acima do array. Quando criamos um array, ele é criado na memória com as suas posições, e com o tipo especificado. Quando atribuímos valores aos seus índices, eles vão sendo preenchidos, e os índices que não utilizarmos vão continuar em branco, esperando serem preenchidos.

Já no caso do slice, ele cria um array de acordo com os elementos que passamos para ele. Por exemplo, na função que criamos, um array de 3 índices foi criado, e cada índice foi preenchido com um nome.

Mas podemos adicionar itens no slice, através da função `append`, que recebe o slice e o item a ser adicionado. Vamos adicionar o retorno dessa função no próprio slice e imprimir novamente a quantidade de itens contidos no slice:

```
func exibeNomes() {
    nomes := []string{"Douglas", "Daniel", "Bernardo"}
```

```
    fmt.Println("O meu slice tem", len(nomes), "itens")

    nomes = append(nomes, "Aparecida")
    fmt.Println("O meu slice tem", len(nomes), "itens")
}
```

Mas a função `len` informa a **quantidade de itens** contidos nele, e não a sua **capacidade**. Para descobrir a sua capacidade, devemos utilizar a função `cap` :

```
func exibeNomes() {
    nomes := []string{"Douglas", "Daniel", "Bernardo"}
    fmt.Println("O meu slice tem", len(nomes), "itens")
    fmt.Println("O meu slice tem capacidade para", cap(nomes), "itens")

    nomes = append(nomes, "Aparecida")
    fmt.Println("O meu slice tem", len(nomes), "itens")
    fmt.Println("O meu slice tem capacidade para", cap(nomes), "itens")
}
```

Ao executar a função, temos a seguinte saída:

```
O meu slice tem 3 itens
O meu slice tem capacidade para 3 itens
O meu slice tem 4 itens
O meu slice tem capacidade para 6 itens
```

Ou seja, o slice dobrou de tamanho quando adicionamos um novo item! Então, sempre que estourarmos a capacidade máxima do slice, do array abaixo dele, ele dobra de tamanho.

Logo, o slice nada mais é do que o Go cuidando do array para nós, pois eles não funcionam de forma diferente. O slice é um array com algumas coisas abstraídas, evitando com que nos preocupemos com o tamanho e capacidade do array, focando somente em trabalhar com os dados.