Kiểu Slice trong Go

`Slice` là một tham chiếu đến Array, nó mô tả một phần (hoặc toàn bộ) Array. Nó có kích thước động nên thường được sử dụng nhiều hơn Array.

Xem thêm về Array

1. Khởi tạo Slice

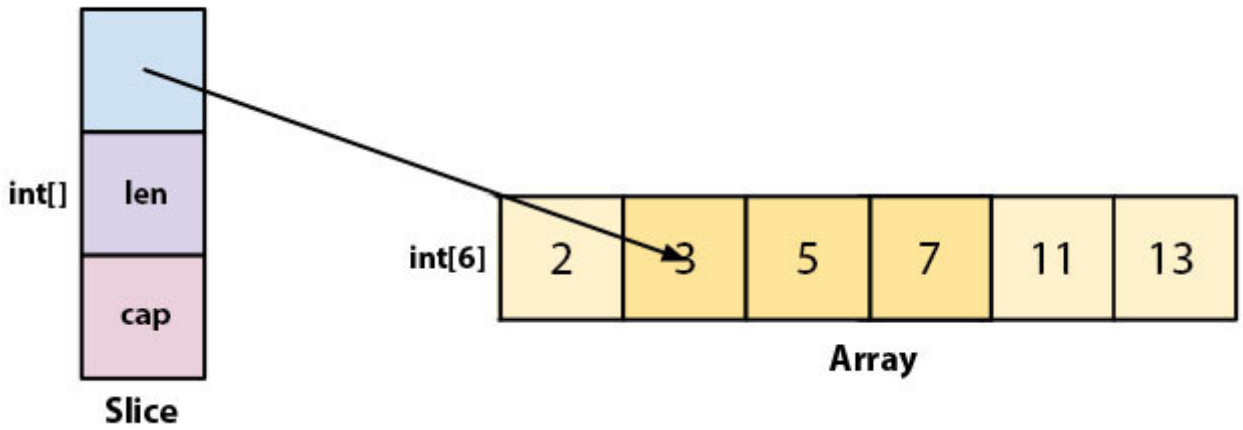
- Slice có thể tạo ra từ một Array bằng cách cung cấp 2 chỉ số (low và high) xác định vị trí phần tử trong Array.

Ví dụ:

```
// Khởi tạo Array primes
primes := [6]int{2, 3, 5, 7, 11, 13}

// Khởi tạo Slice s bằng cách cắt từ phần tử ở vị trí 1 (low) đến phần tử ở vị trí 3 (high - 1) c
Array primes
var s []int = primes[1:4]

// In ra giá trị của Slice s
fmt.Println(s) // Giá trị của s là [3, 5, 7]
```



- Một Slice sẽ có 2 thuộc tính là length (len) và capacity (cap). Length là số phần tử chứa trong Slice, còn capacity là số phần tử chứa trong Array mà Slice tham chiếu đến (bắt đầu tính từ phần tử đầu tiên của Slice). Để lấy ra length của Slice ta dùng hàm len(), còn để lấy ra capacity thì ta dùng hàm cap().

Ví dụ:

```
s := []int{2, 3, 5, 7, 11, 13}

s = s[0:0] // s = [], len(s) = 0, cap(s) = 6
s = s[0:4] // s = [2, 3, 5, 7], len(s) = 4, cap(s) = 6
s = s[2:4] // s = [5, 7], len(s) = 2, cap(s) = 4, cap được tính từ vị trí s[2] trở đi
s = s[0:4] // s = [5, 7, 11, 13], len(s) = 4, cap(s) = 4
```

- Khi tạo Slice ta có thể bỏ qua các chỉ số low và high, khi đó Go sẽ tự sử dụng giá trị mặc định: 0 cho low và length của Slice cho high.

Ví dụ:

```
s := []int{2, 3, 5, 7, 11, 13}

s = s[:0] // s = [0:0]
s = s[:4] // s = [0:4]
s = s[2:] // s = [2:len(s)] => s = [2:4]
s = s[:4] // s = [0:4]
```

- Ngoài việc tạo Slice như trên, chúng ta có thể tạo theo các cách sau:

- Khai báo như một mảng nhưng không chỉ ra kích thước mảng:

```
q := []int{2, 3, 5, 7, 11, 13}
```

- Sử dụng hàm make() với công thức sau:

```
func make([]T, len, cap) []T

a := make([]int, 5) // len(a)=5
b := make([]int, 0, 5) // len(b)=0, cap(b)=5
```

- Slice có zero value là nil (length = 0 và capacity = 0), nil tương đương với giá trị null trong các ngôn ngữ lập trình khác.

- Do Slice chỉ là tham chiếu đến Array, do đó thay đổi giá trị của Slice sẽ làm thay đổi giá trị của Array mà nó tham chiếu đến. Nếu có nhiều Slice cùng tham chiếu đến một Array thì khi thay đổi giá trị một Slice có thể làm thay đổi giá trị các Slice khác.

Ví dụ:

```
numbers := [4]int{1, 2, 3, 4}

a := numbers[0:2] // a = [1, 2]
b := numbers[1:3] // b = [2, 3]

b[0] = 5 // Thay đổi giá trị phần tử đầu tiên của Slice b

fmt.Println(a, b) // a = [1, 5], b = [5, 3]
fmt.Println(numbers) // numbers = [1, 5, 3, 4]
```

2. Append Slice

- Để bổ sung thêm phần tử cho slice, ta dùng hàm append() với công thức sau:

```
func append(s []T, vs ...T) []T
```

- Hàm này sẽ trả về một slice có chứa toàn bộ các phần tử của slice ban đầu và các phần tử mới thêm vào. Trong trường hợp slice ban đầu có sức chứa nhỏ (Array mà nó tham chiếu đến có size nhỏ), một Array mới có kích thước lớn hơn sẽ được tạo ra và slice mới sẽ tham chiếu đến Array đó.

```
var s []int

// Append có thể hoạt động với nil slice.
s = append(s, 0) // s = [0]

// Append thêm một phần tử vào slice.
s = append(s, 1) // s = [0, 1]

// Append thêm nhiều phần tử vào slice.
s = append(s, 2, 3, 4) // s = [0, 1, 2, 3, 4]
```

3. Range Slide

- Range là một hình thức của vòng lặp for dùng để duyệt qua một slice hoặc map (sẽ nhắc đến ở phần sau). Mỗi một vòng lặp sẽ trả về 2 giá trị: Giá trị đầu tiên là chỉ số (vị trí) của phần tử, và giá trị thứ hai là bản sao của phần tử đó (cùng giá trị).

Ví dụ:

```
var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}
for i, v := range pow {
	fmt.Printf("i = %d, v = %d \n", i, v)
```

```
}
```

- Trong trường hợp khi lặp chỉ sử dụng 1 trong 2 giá trị trả về thì ta sẽ bỏ qua giá trị còn lại bằng cách thay tên biến bằng ký tự gạch dưới (vì nếu không thì khi biên dịch sẽ báo lỗi biến được định nghĩa mà không sử dụng).

Ví dụ:

```
var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}
for _, v := range pow {
	fmt.Printf("v = %d \n", v)
}
```

Revision #1
Created 5 March 2022 01:50:53 by Laptrinh.vn
Updated 5 March 2022 01:58:12 by Laptrinh.vn