💡 Empty slice vs nil slice in Go

September 2, 2021

introduction slice

Slices in Go can be represented by 3 elements:

- `ptr` - a pointer to the underlying array that contains data of the slice
- `len` - length, number of elements in the slice
- `cap` - capacity, number of elements in the underlying data array, starting from the element pointed by the `ptr`

A `nil` slice declared as `var s1 []string` has no underlying data array - it points to nothing. An empty slice, declared as `s2 := []string{}` or `s3 := make([]string, 0)` points to an empty, non-nil array.

> See also in detail what is the difference between length and capacity of slices

See the table to compare the properties of the `nil` and empty slices.

	ptr	len	cap
nil: `[]string`	0	0	0
empty: `[]string{}`	<addr>	0	0
empty: `make([]string, 0)`	<addr>	0	0

The `<addr>` is a non-zero address to an empty, non-nil array.

```
package main

import "fmt"

func main() {
    var s1 []string         // nil
    s2 := []string{}        // empty
    s3 := make([]string, 0) // empty, equivalent to s2 := []string{}

    fmt.Printf("s1 is nil: %t, len: %d, cap: %d\n", s1 == nil, len(s1), cap(s1))
    fmt.Printf("s2 is nil: %t, len: %d, cap: %d\n", s2 == nil, len(s2), cap(s2))
    fmt.Printf("s3 is nil: %t, len: %d, cap: %d\n", s3 == nil, len(s3), cap(s3))
}
```

Output:

```
s1 is nil: true, len: 0, cap: 0
s2 is nil: false, len: 0, cap: 0
s3 is nil: false, len: 0, cap: 0
```

Empty and `nil` slices behave in the same way that is the built-in functions like `len()`, `cap()`, `append()`, and `for .. range` loop return the same results. So, since the `nil` slice declaration is simpler, you should prefer it to creating an empty slice. However, there are some cases when you may need the empty, non-nil slice. For instance, when you want to return an empty JSON array `[]` as an HTTP response, you should create the empty slice (`[]string{}` or `make([]string, 0)`) because if you use a `nil` slice, you will get a `null` JSON array after encoding:

```
package main

import (
	"encoding/json"
	"fmt"
	"log"
)

func main() {
	var s1 []string         // nil
	s2 := []string{}        // empty
	s3 := make([]string, 0) // empty, equivalent to s2 := []string{}

	s1JSON, err := json.Marshal(s1)
	if err != nil {
		log.Fatal(err)
	}
	fmt.Printf("s1 JSON: %s\n", string(s1JSON))

	s2JSON, err := json.Marshal(s2)
	if err != nil {
		log.Fatal(err)
	}
	fmt.Printf("s2 JSON: %s\n", string(s2JSON))

	s3JSON, err := json.Marshal(s3)
	if err != nil {
		log.Fatal(err)
	}
	fmt.Printf("s3 JSON: %s\n", string(s3JSON))
}
```

Output:

```
s1 JSON: null
s2 JSON: []
s3 JSON: []
```


Related

👭 Copy a slice in Go

Learn how to make a deep copy of a slice

introduction slice

November 25, 2021

🔎 Check if the slice contains the given value in Go

Learn how to write a function that checks if a slice has a specific value

introduction slice

November 5, 2021

🖨️ Convert string to []byte or []byte to string in Go

Learn the difference between a string and a byte slice

introduction strings slice

October 13, 2021