



핵심! Go 프로그래밍

• 1일차

목차

- 배열
- 슬라이스
- 맵

배열?

- 같은 자료형을 특정 기준에 맞춰 원하는 만큼 담는 자료구조
- 각각의 value를 원하는 Index에 대응시키는 구조
- 순차적 특성을 가지고 있음
- Go 배열은 길이가 고정되어 있음
- Index는 0부터 시작
 - Index의 끝은 데이터 개수 -1

배열?

- Go 배열은 따로 init해주지 않으면, 0으로 초기화 됨

```
8 ▶ func main() {  
9  
10     var a [5]int  
11     a[2] = 7  
12     fmt.Println(a... "a : ", a)  
13  
14     var b = [5]int{1,2,3,4,5}  
15     fmt.Println(a... "b : ", b)  
16  
17     c := [5]int{6,7,8,9,10}  
18     fmt.Println(a... "c : ",c)  
19  
20 }  
21
```

```
8 ▶ func main() {  
9  
10     var a [5]int  
11     a[2] = 7  
12     fmt.Println(a... "a : ", a)  
13  
14     var b = [5]int{1,  
15         2,  
16         3,  
17         4,  
18         5,  
19     }  
20     fmt.Println(a... "b : ", b)  
21  
22     c := [...]int{6,7,8,9,10}  
23     fmt.Println(a... "c : ",c)  
24  
25 }
```

<4 go setup calls>

a :	[0 0 7 0 0]
b :	[1 2 3 4 5]
c :	[6 7 8 9 10]

배열?

- Go 배열은 따로 init해주지 않으면, 0으로 초기화 됨 (colloc)

```
8 ▶ func main() {  
9  
10     var a [5]int  
11     a[2] = 7  
12     fmt.Println(a... "a : ", a)  
13  
14     var b = [5]int{1,2,3,4,5}  
15     fmt.Println(a... "b : ", b)  
16  
17     c := [5]int{6,7,8,9,10}  
18     fmt.Println(a... "c : ",c)  
19  
20 }  
21
```

```
8 ▶ func main() {  
9  
10     var a [5]int  
11     a[2] = 7  
12     fmt.Println(a... "a : ", a)  
13  
14     var b = [5]int{1,  
15         2,  
16         3,  
17         4,  
18         5,  
19     }  
20     fmt.Println(a... "b : ", b)  
21  
22     c := [...]int{6,7,8,9,10}  
23     fmt.Println(a... "c : ",c)  
24  
25 }
```

<4 go setup calls>

a :	[0 0 7 0 0]
b :	[1 2 3 4 5]
c :	[6 7 8 9 10]

배열 순회

- 배열에 저장되어 있는 데이터를 순차적으로 사용

```
8 ▶ func main() {  
9     var a = [5]int{1,2,3,4,5}  
10    for i := 0 ; i < len(a) ; i++{  
11        fmt.Print(a[i], " ")  
12    }  
13  
14 }
```

```
8 ▶ func main() {  
9     const flag = 5  
10    var a = [flag]int{1,2,3,4,5}  
11    for i := 0 ; i < flag ; i++{  
12        fmt.Print(a[i], " ")  
13    }  
14  
15 }
```

```
8 ▶ func main() {  
9     const flag = 5  
10    var a = [flag]int{10,20,30,40,50}  
11    for index, value := range a{  
12        fmt.Println(index, value)  
13    }  
14  
15 }
```

```
8 ▶ func main() {  
9     const flag = 5  
10    var a = [flag]int{10,20,30,40,50}  
11    for index := range a{  
12        fmt.Println(index, a[index])  
13    }  
14  
15 }
```

```
8 ▶ func main() {  
9     const flag = 5  
10    var a = [flag]int{10,20,30,40,50}  
11    for _, value := range a{  
12        fmt.Println(value)  
13    }  
14  
15 }
```

배열 복사

- 배열에 저장되어 있는 데이터를 순차적으로 사용

```
8 func main() {
9     const flag = 5
10    var a = [flag]int{10,20,30,40,50}
11    var b [len(a)]int
12
13    for i := 0 ; i < len(a); i++{
14        b[i] = a[i]
15    }
16    fmt.Println(a...)
17    fmt.Println(b...)
18
19 }
```

go build 1day-3.go

<4 go setup calls>

a : [10 20 30 40 50]

b : [10 20 30 40 50]

```
8 func main() {
9     const flag = 5
10    var a = [flag]int{10,20,30,40,50}
11    b := a
12    b[2] = 111
13    fmt.Println(a...)
14    fmt.Println(b...)
15
16 }
```

go build 1day-3.go

<4 go setup calls>

a : [10 20 30 40 50]

b : [10 20 111 40 50]

슬라이스?

- 배열과 같은 구조를 가지고 있음
- 배열은 크기가 고정되어 있지만, 슬라이스는 크기가 동적임 (동적 할당)
- 슬라이스의 경우, call by reference 형태로 동작
- 실제 슬라이스 할당 시, 배열의 포인터를 할당하는 형식으로 진행 됨

슬라이스?

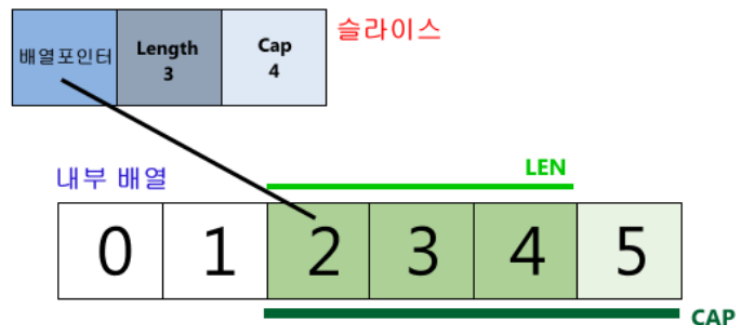
- 배열과 같은 구조를 가지고 있음
- 배열은 크기가 고정되어 있지만, 슬라이스는 크기가 동적임 (동적 할당)
- 슬라이스의 경우, call by reference 형태로 동작
- 실제 슬라이스 할당 시, 배열의 포인터를 할당하는 형식으로 진행 됨

슬라이스 구조

`S := []int { 0, 1, 2, 3, 4, 5 }` 실행시 슬라이스



`S := S[2:5]` 실행시 변경된 슬라이스



- 배열 동적 할당
- C언어의 Colloc과 비슷한 기능
- Make Rule
 - Type
 - Lan
 - Cap

출처: <http://golang.site/go/article/13-Go-%EC%BB%AC%EB%A0%89%EC%85%98---Slice>

슬라이스 선언

```
8 ▶ func main() {  
9  
10     var a []int = make([]int, 5)  
11     var b = make([]int, 5)  
12     c := make([]int, 5)  
13  
14     fmt.Println(a, b, c)  
15  
16 }
```

go build 1day-3.go ×

<4 go setup calls>
[0 0 0 0 0] [0 0 0 0 0] [0 0 0 0 0]

```
8 ▶ func main() {  
9  
10     a := make([]int, 0)  
11     fmt.Println(a, len(a), cap(a))  
12  
13     a = append(a, 1, 2, 3, 4, 5)  
14     fmt.Println(a, len(a), cap(a))  
15  
16     a = append(a, 6, 7)  
17     fmt.Println(a, len(a), cap(a))  
18  
19 }
```

go build 1day-3.go ×

<4 go setup calls>
[] 0 0
[1 2 3 4 5] 5 6
[1 2 3 4 5 6 7] 7 12

슬라이스 활용

```
8 ▶ func main() {  
9     var a []int = make([]int, 5)  
10    var b = make([]int, 5, 6)  
11    c := make([]int, 5, 10)  
12  
13    fmt.Println(a, b, c)  
14    fmt.Println(a, len(a), cap(a))  
15    fmt.Println(b, len(b), cap(b))  
16    fmt.Println(c, len(c), cap(c))  
17  
18    a = append(a, elems... 1, 2, 3)  
19    b = append(b, elems... 1, 2, 3)  
20    c = append(c, elems... 1, 2, 3)  
21  
22    fmt.Println(a, b, c)  
23    fmt.Println(a, len(a), cap(a))  
24    fmt.Println(b, len(b), cap(b))  
25    fmt.Println(c, len(c), cap(c))  
26  
27 }
```

```
go build 1day-3.go x  
↑ <4 go setup calls>  
↓  
[0 0 0 0 0] [0 0 0 0 0] [0 0 0 0 0]  
[0 0 0 0 0] 5 5  
[0 0 0 0 0] 5 6  
[0 0 0 0 0] 5 10  
[0 0 0 0 0 1 2 3] [0 0 0 0 0 1 2 3] [0 0 0 0 0 1 2 3]  
[0 0 0 0 0 1 2 3] 8 10  
[0 0 0 0 0 1 2 3] 8 12  
[0 0 0 0 0 1 2 3] 8 10
```

슬라이스 활용

```
8 func main() {
9     var a []int = make([]int, 0)
10    fmt.Println(a, len(a), cap(a))
11
12    a = append(a, elems... 1, 2, 3, 4, 5)
13    fmt.Println(a, len(a), cap(a))
14
15    a = append(a, elems... 6, 7)
16    fmt.Println(a, len(a), cap(a))
17
18
19 }
20
```

go build 1day-3.go x

<4 go setup calls>

[] 0 0

[1 2 3 4 5] 5 6

[1 2 3 4 5 6 7] 7 12

```
8 func main() {
9     var a []int = make([]int, 0, 8)
10    fmt.Println(a, len(a), cap(a))
11
12    a = append(a, elems... 1, 2, 3, 4, 5)
13    fmt.Println(a, len(a), cap(a))
14
15    a = append(a, elems... 6, 7)
16    fmt.Println(a, len(a), cap(a))
17
18
19 }
20
```

go build 1day-3.go x

<4 go setup calls>

[] 0 8

[1 2 3 4 5] 5 8

[1 2 3 4 5 6 7] 7 8

슬라이스 활용

```
8 ▶ func main() {  
9     var a []int = make([]int, 0, 8)  
10    fmt.Println(a, len(a), cap(a))  
11  
12    a = append(a, elems... 1, 2, 3, 4, 5)  
13    fmt.Println(a, len(a), cap(a))  
14  
15    fmt.Println(a[4])  
16  
17 }  
18
```

```
go build 1day-3.go x  
↑ <4 go setup calls>  
↓ [] 0 8  
  [1 2 3 4 5] 5 8  
  5
```

```
8 ▶ func main() {  
9     var a []int = make([]int, 0, 8)  
10    fmt.Println(a, len(a), cap(a))  
11  
12    a = append(a, elems... 1, 2, 3, 4, 5)  
13    fmt.Println(a, len(a), cap(a))  
14  
15    fmt.Println(a[6])  
16  
17 }
```

```
go build 1day-3.go x  
↑ <4 go setup calls>  
↓ [] 0 8  
  [1 2 3 4 5] 5 8  
  panic: runtime error: index out of range [6] with length 5  
  
goroutine 1 [running]:  
main.main()  
  C:/Users/holly/Desktop/workspace_go/src/main/1day-3.go:15 +0x25c
```

슬라이스 활용

```
8 ▶ func main() {  
9     a := make([]int, 0, 8)  
10    b := make([]int, 3)  
11    fmt.Println(a, len(a), cap(a))  
12  
13    a = append(a, b)  
14    fmt.Println(a, len(a), cap(a))  
15  
16 }
```

```
go build 1day-3.go ×  
↑ <3 go setup calls>  
# command-line-arguments  
↓  
= ./1day-3.go:13:12: cannot use b (type []int) as type int in append
```

```
8 ▶ func main() {  
9     a := make([]int, 0, 8)  
10    b := make([]int, 3)  
11    fmt.Println(a, len(a), cap(a))  
12  
13    a = append(a, b...)  
14    fmt.Println(a, len(a), cap(a))  
15  
16 }
```

```
go build 1day-3.go ×  
↑ <4 go setup calls>  
[] 0 8  
↓  
= [0 0 0] 3 8
```

슬라이스 활용

```
8 ▶ func main() {  
9     a := make([]int, 0, 8)  
10    b := make([]int, 3)  
11    fmt.Println(a, len(a), cap(a))  
12  
13    a = append(a, b)  
14    fmt.Println(a, len(a), cap(a))  
15  
16 }
```

```
go build 1day-3.go ×  
↑ <3 go setup calls>  
# command-line-arguments  
↓  
= ./1day-3.go:13:12: cannot use b (type []int) as type int in append
```

```
8 ▶ func main() {  
9     a := make([]int, 0, 8)  
10    b := make([]int, 3)  
11    fmt.Println(a, len(a), cap(a))  
12  
13    a = append(a, b...)  
14    fmt.Println(a, len(a), cap(a))  
15  
16 }
```

```
go build 1day-3.go ×  
↑ <4 go setup calls>  
[] 0 8  
↓  
= [0 0 0] 3 8
```


슬라이스 활용

```
8 ▶ func main() {  
9     a := make([]int, 0, 8)  
10    b := make([]int, 3)  
11    fmt.Println(a, len(a), cap(a))  
12  
13    a = append(a, b)  
14    fmt.Println(a, len(a), cap(a))  
15  
16 }
```

```
go build 1day-3.go ×  
↑ <3 go setup calls>  
# command-line-arguments  
.\1day-3.go:13:12: cannot use b (type []int) as type int in append
```

```
8 ▶ func main() {  
9     a := make([]int, 0, 8)  
10    b := make([]int, 3)  
11    fmt.Println(a, len(a), cap(a))  
12  
13    a = append(a, b...)  
14    fmt.Println(a, len(a), cap(a))  
15  
16 }
```

```
8 ▶ func main() {  
9     a := make([]int, 0, 8)  
10    b := make([]int, 3, 100)  
11    fmt.Println(a, len(a), cap(a))  
12  
13    a = append(a, b...)  
14    fmt.Println(a, len(a), cap(a))  
15  
16 }
```

```
go build 1day-3.go ×  
↑ <4 go setup calls>  
[] 0 8  
[0 0 0] 3 8
```

슬라이스 활용

```
8 func main() {  
9     a := make([]int, 0, 8)  
10    b := make([]int, 10, 100)  
11    fmt.Println(a, len(a), cap(a))  
12  
13    a = b  
14    b[0] = 10  
15    fmt.Println(a, len(a), cap(a))  
16  
17 }
```

go build 1day-3.go

<4 go setup calls>

[] 0 8

[10 0 0 0 0 0 0 0 0 0] 10 100

```
8 func main() {  
9     a := make([]int, 5, 5)  
10    b := make([]int, 10, 100)  
11    fmt.Println(a, len(a), cap(a))  
12  
13    for i := range b {  
14        b[i] = i  
15    }  
16    copy(a, b)  
17    fmt.Println(a, len(a), cap(a))  
18    b[0] = 10  
19    fmt.Println(a, len(a), cap(a))  
20 }
```

go build 1day-3.go

<4 go setup calls>

[0 0 0 0 0] 5 5

[0 1 2 3 4] 5 5

[0 1 2 3 4] 5 5

슬라이스 활용

```
8 ▶ func main() {  
9     a := make([][]string, 3)  
10    fmt.Println(a, len(a), cap(a))  
11 }
```

go build 1day-3.go ×
↑ <4 go setup calls>
↓
[[] [] [] 3 3

```
8 ▶ func main() {  
9     a := make([][]string, 3)  
10    for i := range a {  
11        a[i] = make([]string, 5)  
12    }  
13    fmt.Println(a, len(a), cap(a))  
14 }
```

go build 1day-3.go ×
↑ <4 go setup calls>
↓
[[] [] [] 3 3

```
8 ▶ func main() {  
9     a := make([][]string, 3)  
10    for i := range a {  
11        a[i] = make([]string, 5)  
12    }  
13    fmt.Println(a, len(a), cap(a))  
14  
15    b := []string{"ab", "ac"}  
16    c := []string{"bc", "bd", "be"}  
17    a = append(a, b)  
18    a[0] = c  
19    fmt.Println(a, len(a), cap(a))  
20    fmt.Println(a[0], len(a[0]), cap(a[0]))  
21    fmt.Println(a[1], len(a[1]), cap(a[1]))  
22 }
```

go build 1day-3.go ×
↑ <4 go setup calls>
↓
[[] [] [] 3 3
[[bc bd be] [] [] [ab, ac]] 4 6
[bc bd be] 3 3
[] 5 5

슬라이스 활용

```
8 ▶ func main() {  
9     a := make([][]string, 3)  
10    for i := range a {  
11        a[i] = make([]string, 5)  
12    }  
13    fmt.Println(a, len(a), cap(a))  
14  
15    b := []string{"ab", "ac"}  
16    c := []string{"bc", "bd", "be"}  
17    a = append(a, b)  
18    a[0] = c  
19    copy(a[1], c)  
20    c[0] = "changeVal"  
21    fmt.Println(a, len(a), cap(a))  
22    fmt.Println(a[0], len(a[0]), cap(a[0]))  
23    fmt.Println(a[1], len(a[1]), cap(a[1]))  
24 }
```

```
go build 1day-3.go ×  
↑ <4 go setup calls>  
↓  
[[ ] [ ] [ ]] 3 3  
[[changeVal bd be] [bc bd be ] [ ] [ab, ac]] 4 6  
[changeVal bd be] 3 3  
[bc bd be ] 5 5
```

맵?

- Map은 Key-Value 쌍을 가지고 있는 자료구조
- Golang에서의 Map은 Hash Table로 구현됨
- Key는 중복허용, Value는 중복허용 안됨, 순서x
- 내장 함수로서, 손쉽게 사용 가능
- Reference type

맵 선언

```
8 ▶ func main() {
9
10     var a map[string]string
11
12     a = map[string]string{
13         "a": "aaa",
14         "b": "bbb",
15         "c": "ccc",
16     }
17
18     fmt.Println(a, len(a))
19 }
```

go build 1day-3.go ×

<4 go setup calls>
map[a:aaa b:bbb c:ccc] 3

```
8 ▶ func main() {
9
10     var a map[string]string
11     var b = make(map[string]string)
12     c := make(map[string]string, 10)
13     a = map[string]string{
14         "a": "aaa",
15         "b": "bbb",
16         "c": "ccc",
17     }
18
19     fmt.Println(a, len(a))
20     fmt.Println(b, len(b))
21     fmt.Println(c, len(c))
22 }
```

go build 1day-3.go ×

<4 go setup calls>
map[a:aaa b:bbb c:ccc] 3
map[] 0
map[] 0

맵 활용

```
8 func main() {
9
10     var a map[string]string
11     a = map[string]string{
12         "a": "aaa",
13         "b": "bbb",
14         "c": "ccc",
15     }
16     fmt.Println(a, len(a))
17     a["d"] = "ddd"
18     fmt.Println(a, len(a))
19 }
```

go build 1day-3.go x

<4 go setup calls>
map[a:aaa b:bbb c:ccc] 3
map[a:aaa b:bbb c:ccc d:ddd] 4

```
8 func main() {
9
10     var a map[string]string
11     a = map[string]string{
12         "a": "aaa",
13         "b": "bbb",
14         "c": "ccc",
15     }
16     fmt.Println(a, len(a))
17     a["d"] = "ddd"
18     a["a"] = "apple"
19     fmt.Println(a, len(a))
20 }
```

go build 1day-3.go x

<4 go setup calls>
map[a:aaa b:bbb c:ccc] 3
map[a:apple b:bbb c:ccc d:ddd] 4

```
8 func main() {
9
10     var a map[string]string
11     a = map[string]string{
12         "a": "aaa",
13         "b": "bbb",
14         "c": "ccc",
15     }
16     fmt.Println(a, len(a))
17     a["d"] = "ddd"
18     a["a"] = "apple"
19     delete(a, "b")
20     fmt.Println(a, len(a))
21 }
```

go build 1day-3.go x

<4 go setup calls>
map[a:aaa b:bbb c:ccc] 3
map[a:apple c:ccc d:ddd] 3

맵 활용

```
8 ▶ func main() {  
9  
10     var a map[string]string  
11     a = map[string]string{  
12         "a": "aaa",  
13         "b": "bbb",  
14         "c": "ccc",  
15     }  
16     fmt.Println(a, len(a))  
17  
18     for key, value := range a{  
19         fmt.Println(a... "key : ", key, " value : ", value)  
20     }  
21 }
```

a ...interface{}

```
go build 1day-3.go ×  
↑ <4 go setup calls>  
map[a:aaa b:bbb c:ccc] 3  
↓  
key : a value : aaa  
key : b value : bbb  
key : c value : ccc
```

```
8 ▶ func main() {  
9  
10     var a map[string]string  
11     a = map[string]string{  
12         "a": "aaa",  
13         "b": "bbb",  
14         "c": "ccc",  
15     }  
16     fmt.Println(a, len(a))  
17  
18     val, flag := a["a"]  
19     fmt.Println(a... "val : ", val, " flag : ", flag)  
20     if flag {  
21         fmt.Println(a... "hit")  
22     } else{  
23         fmt.Println(a... "no Hit")  
24     }  
25 }
```

```
go build 1day-3.go ×  
↑ <4 go setup calls>  
map[a:aaa b:bbb c:ccc] 3  
↓  
val : aaa flag : true  
hit
```


맵 활용

```
8 ▶ func main() {
9
10     var a map[string]string
11     var b = make(map[string]string)
12     c := make(map[string]string)
13     a = map[string]string{
14         "a": "aaa",
15         "b": "bbb",
16         "c": "ccc",
17     }
18     b = a
19     a["a"] = "apple"
20     fmt.Print(a, "\n", b, "\n", c, "\n")
21 }
```

go build 1day-3.go ×

<4 go setup calls>

- map[a:apple b:bbb c:ccc]
- map[a:apple b:bbb c:ccc]
- map[]

```
8 ▶ func main() {
9
10     var a map[string]string
11     var b = make(map[string]string)
12     c := make(map[string]string)
13     a = map[string]string{
14         "a": "aaa",
15         "b": "bbb",
16         "c": "ccc",
17     }
18     for key, val := range a{
19         b[key] = val
20     }
21     a["a"] = "apple"
22     fmt.Print(a, "\n", b, "\n", c, "\n")
23 }
```

go build 1day-3.go ×

<4 go setup calls>

- map[a:apple b:bbb c:ccc]
- map[a:aaa b:bbb c:ccc]
- map[]