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

```go
func main() {

    var a [5]int
    a[2] = 7
    fmt.Println( a...: "a : ", a)

    var b = [5]int{1,2,3,4,5}
    fmt.Println( a...: "b : ", b)

    c := [5]int{6,7,8,9,10}
    fmt.Println( a...: "c : ",c)

}
```

```go
func main() {

    var a [5]int
    a[2] = 7
    fmt.Println( a...: "a : ", a)

    var b = [5]int{1,
        2,
        3,
        4,
        5,
    }
    fmt.Println( a...: "b : ", b)

    c := [...]int{6,7,8,9,10}
    fmt.Println( a...: "c : ",c)

}
```

```
<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)

```go
func main() {

    var a [5]int
    a[2] = 7
    fmt.Println( a...: "a : ", a)

    var b = [5]int{1,2,3,4,5}
    fmt.Println( a...: "b : ", b)

    c := [5]int{6,7,8,9,10}
    fmt.Println( a...: "c : ",c)

}
```

```go
func main() {

    var a [5]int
    a[2] = 7
    fmt.Println( a...: "a : ", a)

    var b = [5]int{1,
        2,
        3,
        4,
        5,
    }
    fmt.Println( a...: "b : ", b)

    c := [...]int{6,7,8,9,10}
    fmt.Println( a...: "c : ",c)

}
```

```
<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

```go
func main() {
    var a = [5]int{1,2,3,4,5}
    for i := 0 ; i < len(a) ; i++{
        fmt.Print(a[i], " ")
    }

}
```

```go
func main() {
    const flag = 5
    var a = [flag]int{1,2,3,4,5}
    for i := 0 ; i < flag ; i++{
        fmt.Print(a[i], " ")
    }

}
```

```go
func main() {
    const flag = 5
    var a = [flag]int{10,20,30,40,50}
    for index, value := range a{
        fmt.Println(index, value)
    }

}
```

```go
func main() {
    const flag = 5
    var a = [flag]int{10,20,30,40,50}
    for index := range a{
        fmt.Println(index, a[index])
    }

}
```

```go
func main() {
    const flag = 5
    var a = [flag]int{10,20,30,40,50}
    for _, value := range a{
        fmt.Println(value)
    }

}
```

배열 복사

- 배열에 저장되어 있는 데이터를 순차적으로 사용

```go
func main() {
    const flag = 5
    var a = [flag]int{10,20,30,40,50}
    var b [len(a)]int

    for i := 0 ; i < len(a); i++{
        b[i] = a[i]
    }
    fmt.Println( a...: "a : ", a)
    fmt.Println( a...: "b : ", b)

}
```

```
go build 1day-3.go
<4 go setup calls>
a :  [10 20 30 40 50]
b :  [10 20 30 40 50]
```

```go
func main() {
    const flag = 5
    var a = [flag]int{10,20,30,40,50}
    b := a
    b[2] = 111
    fmt.Println( a...: "a : ", a)
    fmt.Println( a...: "b : ", b)

}
```

```
go build 1day-3.go
<4 go setup calls>
a :  [10 20 30 40 50]
b :  [10 20 111 40 50]
```

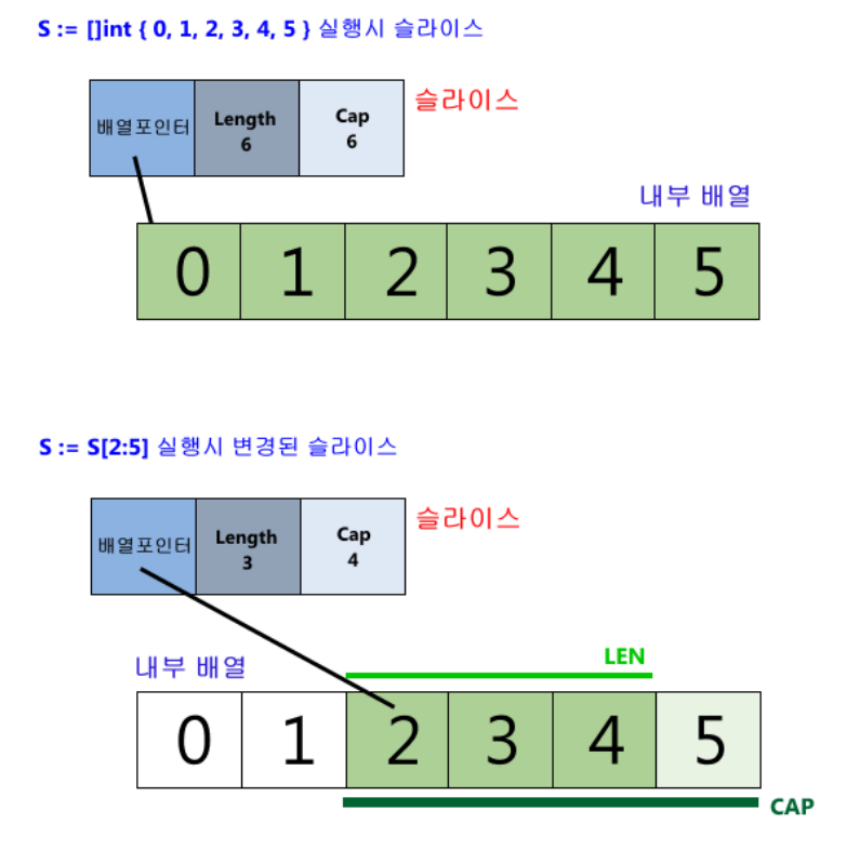
슬라이스?

- 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



- 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

```go
func main() {

    var a []int = make([]int, 5)
    var b = make([]int, 5)
    c := make([]int, 5)

    fmt.Println(a, b, c)

}
```

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

```go
func main() {

    a := make([]int, 0)
    fmt.Println(a, len(a), cap(a))

    a = append(a, elems...: 1, 2, 3, 4, 5)
    fmt.Println(a, len(a), cap(a))

    a = append(a, elems...: 6, 7)
    fmt.Println(a, len(a), cap(a))

}
```

```
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

```go
func main() {
    var a []int = make([]int, 5)
    var b = make([]int, 5, 6)
    c := make([]int, 5, 10)

    fmt.Println(a, b, c)
    fmt.Println(a, len(a), cap(a))
    fmt.Println(b, len(b), cap(b))
    fmt.Println(c, len(c), cap(c))

    a = append(a, elems...: 1, 2, 3)
    b = append(b, elems...: 1, 2, 3)
    c = append(c, elems...: 1, 2, 3)

    fmt.Println(a, b, c)
    fmt.Println(a, len(a), cap(a))
    fmt.Println(b, len(b), cap(b))
    fmt.Println(c, len(c), cap(c))

}
```

```
go build 1day-3.go
<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

```go
func main() {
    var a []int = make([]int, 0)
    fmt.Println(a, len(a), cap(a))

    a = append(a, elems...: 1, 2, 3, 4, 5)
    fmt.Println(a, len(a), cap(a))

    a = append(a, elems...: 6, 7)
    fmt.Println(a, len(a), cap(a))


}
```

```
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
```

```go
func main() {
    var a []int = make([]int, 0, 8)
    fmt.Println(a, len(a), cap(a))

    a = append(a, elems...: 1, 2, 3, 4, 5)
    fmt.Println(a, len(a), cap(a))

    a = append(a, elems...: 6, 7)
    fmt.Println(a, len(a), cap(a))


}
```

```
go build 1day-3.go
<4 go setup calls>
[] 0 8
[1 2 3 4 5] 5 8
[1 2 3 4 5 6 7] 7 8
```

슬라이스 활용

```go
func main() {
    var a []int = make([]int, 0, 8)
    fmt.Println(a, len(a), cap(a))

    a = append(a, elems...: 1, 2, 3, 4, 5)
    fmt.Println(a, len(a), cap(a))

    fmt.Println(a[4])

}
```

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

```go
func main() {
    var a []int = make([]int, 0, 8)
    fmt.Println(a, len(a), cap(a))

    a = append(a, elems...: 1, 2, 3, 4, 5)
    fmt.Println(a, len(a), cap(a))

    fmt.Println(a[6])

}
```

```
go build 1day-3.go
<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

```go
func main() {
    a := make([]int, 0, 8)
    b := make([]int, 3)
    fmt.Println(a, len(a), cap(a))

    a = append(a, b)
    fmt.Println(a, len(a), cap(a))

}
```

```
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
```

```go
func main() {
    a := make([]int, 0, 8)
    b := make([]int, 3)
    fmt.Println(a, len(a), cap(a))

    a = append(a, b...)
    fmt.Println(a, len(a), cap(a))

}
```

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

슬라이스 활용

```go
func main() {
    a := make([]int, 0, 8)
    b := make([]int, 3)
    fmt.Println(a, len(a), cap(a))

    a = append(a, b)
    fmt.Println(a, len(a), cap(a))

}
```

```
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
```

```go
func main() {
    a := make([]int, 0, 8)
    b := make([]int, 3)
    fmt.Println(a, len(a), cap(a))

    a = append(a, b...)
    fmt.Println(a, len(a), cap(a))

}
```

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

슬라이스 활용

```go
func main() {
    a := make([]int, 0, 8)
    b := make([]int, 3)
    fmt.Println(a, len(a), cap(a))

    a = append(a, b)
    fmt.Println(a, len(a), cap(a))

}
```

```
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
```

```go
func main() {
    a := make([]int, 0, 8)
    b := make([]int, 3)
    fmt.Println(a, len(a), cap(a))

    a = append(a, b...)
    fmt.Println(a, len(a), cap(a))

}
```

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

```go
func main() {
    a := make([]int, 0, 8)
    b := make([]int, 3, 100)
    fmt.Println(a, len(a), cap(a))

    a = append(a, b...)
    fmt.Println(a, len(a), cap(a))

}
```

슬라이스 활용

```go
func main() {
    a := make([]int, 0, 8)
    b := make([]int, 10, 100)
    fmt.Println(a, len(a), cap(a))

    a = b
    b[0] = 10
    fmt.Println(a, len(a), cap(a))

}
```

```
go build 1day-3.go
<4 go setup calls>
[] 0 8
[10 0 0 0 0 0 0 0 0 0] 10 100
```

```go
func main() {
    a := make([]int, 5, 5)
    b := make([]int, 10, 100)
    fmt.Println(a, len(a), cap(a))

    for i := range b{
        b[i] = i
    }
    copy(a, b)
    fmt.Println(a, len(a), cap(a))
    b[0] = 10
    fmt.Println(a, len(a), cap(a))
}
```

```
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

```go
func main() {
    a := make([][]string, 3)
    fmt.Println(a, len(a), cap(a))
}
```

```
go build 1day-3.go
<4 go setup calls>
[[] [] []] 3 3
```

```go
func main() {
    a := make([][]string, 3)
    for i := range a {
        a[i] = make([]string, 5)
    }
    fmt.Println(a, len(a), cap(a))
}
```

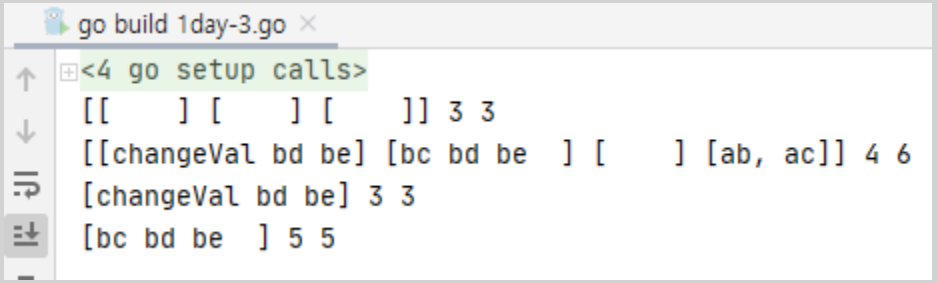
```
go build 1day-3.go
<4 go setup calls>
[[    ] [    ] [    ]] 3 3
```

```go
func main() {
    a := make([][]string, 3)
    for i := range a {
        a[i] = make([]string, 5)
    }
    fmt.Println(a, len(a), cap(a))

    b := []string{"ab, ac"}
    c := []string{"bc", "bd", "be"}
    a = append(a, b)
    a[0] = c
    fmt.Println(a, len(a), cap(a))
    fmt.Println(a[0], len(a[0]), cap(a[0]))
    fmt.Println(a[1], len(a[1]), cap(a[1]))
}
```

```
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

```go
func main() {
    a := make([][]string, 3)
    for i := range a {
        a[i] = make([]string, 5)
    }
    fmt.Println(a, len(a), cap(a))

    b := []string{"ab, ac"}
    c := []string{"bc", "bd", "be"}
    a = append(a, b)
    a[0] = c
    copy(a[1], c)
    c[0] = "changeVal"
    fmt.Println(a, len(a), cap(a))
    fmt.Println(a[0], len(a[0]), cap(a[0]))
    fmt.Println(a[1], len(a[1]), cap(a[1]))
}
```



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

```go
func main() {

    var a map[string]string

    a = map[string]string{
        "a": "aaa",
        "b": "bbb",
        "c": "ccc",
    }

    fmt.Println(a, len(a))
}
```

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

```go
func main() {

    var a map[string]string
    var b = make(map[string]string)
    c := make(map[string]string, 10)
    a = map[string]string{
        "a": "aaa",
        "b": "bbb",
        "c": "ccc",
    }

    fmt.Println(a, len(a))
    fmt.Println(b, len(b))
    fmt.Println(c, len(c))
}
```

```
go build 1day-3.go
<4 go setup calls>
map[a:aaa b:bbb c:ccc] 3
map[] 0
map[] 0
```

맵 활용

```go
func main() {

    var a map[string]string
    a = map[string]string{
        "a": "aaa",
        "b": "bbb",
        "c": "ccc",
    }
    fmt.Println(a, len(a))
    a["d"] = "ddd"
    fmt.Println(a, len(a))
}
```

```
go build 1day-3.go
<4 go setup calls>
map[a:aaa b:bbb c:ccc] 3
map[a:aaa b:bbb c:ccc d:ddd] 4
```

```go
func main() {

    var a map[string]string
    a = map[string]string{
        "a": "aaa",
        "b": "bbb",
        "c": "ccc",
    }
    fmt.Println(a, len(a))
    a["d"] = "ddd"
    a["a"] = "apple"
    fmt.Println(a, len(a))
}
```

```
go build 1day-3.go
<4 go setup calls>
map[a:aaa b:bbb c:ccc] 3
map[a:apple b:bbb c:ccc d:ddd] 4
```

```go
func main() {

    var a map[string]string
    a = map[string]string{
        "a": "aaa",
        "b": "bbb",
        "c": "ccc",
    }
    fmt.Println(a, len(a))
    a["d"] = "ddd"
    a["a"] = "apple"
    delete(a, key: "b")
    fmt.Println(a, len(a))
}
```

```
go build 1day-3.go
<4 go setup calls>
map[a:aaa b:bbb c:ccc] 3
map[a:apple c:ccc d:ddd] 3
```

맵 활용

```go
func main() {

    var a map[string]string
    a = map[string]string{
        "a": "aaa",
        "b": "bbb",
        "c": "ccc",
    }
    fmt.Println(a, len(a))

    for key, value := range a{
        fmt.Println( a...: "key : ", key, " value : ", value)
    }
}
```

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
```

```go
func main() {

    var a map[string]string
    a = map[string]string{
        "a": "aaa",
        "b": "bbb",
        "c": "ccc",
    }
    fmt.Println(a, len(a))

    val, flag := a["a"]
    fmt.Println( a...: "val : ", val, " flag : ", flag)
    if flag {
        fmt.Println( a...: "hit")
    } else{
        fmt.Println( a...: "no Hit")
    }
}
```

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

맵 활용

```go
func main() {

    var a map[string]string
    var b = make(map[string]string)
    c := make(map[string]string)
    a = map[string]string{
        "a": "aaa",
        "b": "bbb",
        "c": "ccc",
    }
    b = a
    a["a"] = "apple"
    fmt.Print(a, "\n", b, "\n", c, "\n")
}
```

```
go build 1day-3.go
<4 go setup calls>
map[a:apple b:bbb c:ccc]
map[a:apple b:bbb c:ccc]
map[]
```

```go
func main() {

    var a map[string]string
    var b = make(map[string]string)
    c := make(map[string]string)
    a = map[string]string{
        "a": "aaa",
        "b": "bbb",
        "c": "ccc",
    }
    for key, val := range a{
        b[key] = val
    }
    a["a"] = "apple"
    fmt.Print(a, "\n", b, "\n", c, "\n")
}
```

```
go build 1day-3.go
<4 go setup calls>
map[a:apple b:bbb c:ccc]
map[a:aaa b:bbb c:ccc]
map[]
```