Go 言語であそぼう

(第三回) Go の基礎

Yugo Okamoto a.k.a donabe8898

本講義のターゲット層

- サーバーに興味がある
- モダンな言語を習得したい

自己紹介

岡本 悠吾 - Yugo Okamoto

- T 学科 4 回生
- サークル内で Linux 関連の講義をやってます

欲しい物リスト

- 車（できれば MR）
- パソコン（そろそろ買い替えたい）

Section 8. Go とデータ

型

- int8/int16/int32/int64: 整数
- uint8/uint16/uint32/uint64: 非負整数
- int: int32 か int64（システムの bit により異なる）
- float32/float64: 浮動小数点数
- complex64/complex128: 虚数
- byte: 1 バイトデータ（=uint8）
- rune: 1 文字（=int32）
- uintptr: ポインタを表現するのに十分な非負整数
- string: 文字列
- bool: 真偽値

型変換

- 型名()で型変換を行うことができる

main.go

```
var hoge uint32 = 8110
fmt.Printf("%T\n", hoge) // uint32
var huga uint64 = uint64(hoge)
fmt.Printf("%T\n", huga) // uint64
```

Printf 関数

- `fmt.Printf` 関数では引数をフォーマットして文字列として出力する.
- 個人的によく使う形式は以下の通り

```
%v（デフォルト）, %s（文字列）, %e（浮動小数点数）, %f（小数）, %d（整数）, %T（型表示）, %p（ポインタ）
```

リテラル

データの表現方法を色々と工夫できる

`nil`	無しを表す値
`true`	真
`false`	偽
`1234, 1_064`	整数（_は無視されるのでカンマの代わりに）
`0b101001, 0B1101`	2 進数
`0777, 0o757, 0O775`	8 進数
`0xFFFF, 0XFF12`	16 進数
`3.14`	小数
`1.21e5, 3.141E6`	浮動小数点数
`1.33i`	虚数
`"Hello"`	文字列
`'E'`	文字（rune 型）

Section 9. Go とデータ構造

配列

コンパイル時に個数が決定されるものが**配列**

写経タイム

Go1.pdf

```
var arr1 = [3]string{} // 配列宣言
  arr1[0] = "りんご"
  arr1[1] = "バナナ"
  arr1[2] = "CL7 Honda ACCORD Euro R"
  arr2 := [3]int{3, 1, 8} // :=での宣言も可能

  // 出力
  for i := 0; i < 3; i++ {
    fmt.Printf("%s = %d\n", arr1[i], arr2[i])
  }
```


```go
// 宣言時に個数が決まっているのであれば[...]でも Ok
arr3 := [...]int64{1000, 1500, 1600, 1050, 1150, 1550, 1650, 1100}
fmt.Printf("%v\n", arr3)
```

スライス

個数を途中で変更可能なものをスライスと呼ぶ

配列との違い

- メモリ効率や速度が低下する
- 配列よりも便利

写経タイム

Go2.pdf

```
arr := []string{} // 個数未決定のスライス

arr = append(arr, "Microsoft Window")
arr = append(arr, "Apple macOS")
arr = append(arr, "GNU/Linux")
```


```
arr = append(arr, "FreeBSD")

fmt.Println("長さ: ", len(arr))
fmt.Println("キャパ: ", cap(arr))

mk := make([]byte, 0, 114514) // make()によるメモリ確保
fmt.Println("長さ: ", len(mk))
fmt.Println("キャパ: ", cap(mk))
```

マップ

- 所謂、連想配列・辞書型
  - (key, value)の形で保存されるスライス
  - key で検索して value を得る

写経タイム

Go3.pdf

```
// 今回はコメント書かなくて OK です
package main

import "fmt"

func main() {
  // 3 大都市の連想配列
  mp := map[string]bool{
```


```
    "Tokyo":  true,
    "Osaka":  true,
    "Nagoya": false,
}
fmt.Printf("%v\n", mp) // 一旦表示してみる

mp["Fukuoka"] = false // 福岡を追加
fmt.Printf("%v\n", mp)

delete(mp, "Nagoya") // 名古屋は 3 大都市引退したほうが良いと思う（爆）
fmt.Printf("%v\n", mp)

fmt.Println("長さ=", len(mp)) // map の長さを確認

_, ok := mp["Osaka"] // 大阪が存在するかどうか
if ok {
```


```
    fmt.Println("ok")
  } else {
    fmt.Println("Not found")
  }

  for k, v := range mp {
    fmt.Printf("%s=%t\n", k, v)
  }

}
```

構造体

- Go には **class が無い**
  - struct に対してメソッドの実装を行うことはできる（struct が class の一部機能を有している）