Nama : Taufik Hidayat

Program : Golang for Back End Programmer

Summary Self Paced Learning: Golang Tingkat Lanjut

Function

Golang adalah bahasa pemrograman yang sangat mudah dipahami serta sangat cepat dalam performance. Golang juga merupakan Compiled Language, memiliki Garbage Collection, dan Concurrency.

Function adalah kumpulan instruksi kode yang disimpan pada suatu block agar dapat di-invoke berulang kali, function dapat memiliki input yang namanya parameter, function juga bisa me-return value bisa juga tidak. Argumen adalah value yang digunakan saat menginvoke sebuah function.

Alasan untuk menggunakan function ialah:

- Hanya perlu dideklarasikan sekali
- Baik untuk operasi yang umum digunakan
- Rapih
- Abstration membuat detail pada function tersembunyi. Detail pada sebuah function bisa disembunyikan
- Kita hanya perlu mengetahui perilaku dari input dan output
- Menambah understandability

Function Params dan Return Values

Function membutuhkan data dari user untuk melakukan operasinya. Jika function yg kita buat tidak memerlukan paramater, maka kita bisa mengosongkan parameternya, akan tetapi tanda kurung `()` harus tetap ditulis. Kita bisa mengelompokkan parameter yang memiliki tipe data yang sama seperti ini:

```
func kode(x, y int) {

}
```

Function juga bisa me-return value sebagai hasilnya, tipe data dari return value dideklarasikan setelah parameter, contohnya sebagai berikut:

```
func kode(x int) int {
        return x + 1
}
y := kode(1)
```

Kita juga bisa me-return lebih dari 1 value, dengan dipisahkan dengan tanda koma:

```
func kode2(x) (int, int) {
        return x, x + 1
}
a, b := kode2(3)
```

Call by Value dan Reference

Call by value yaitu argumen yang diberikan akan disalin ke dalam parameter, sehingga tidak berdampak dengan variable di luar function. Keunggulannya adalah adanya enkapsulasi data, kekurangannya nya adalah dibutuhkan lebih banyak waktu untuk copying data. Call by reference yaitu pointer dipassing sebagai argumen. Keuntungannya copying time nya lebih cepat. Kekurangannya karena tidak ada enkapsulasi data, jadi mungkin saja data parameter nya berubah di function lain

Menulis Function dengan Benar

Mengapa kita harus menulis function dengan benar:

- Agar dapat dipahami dengan mudah
  - Kode adalah function dan data
  - Find a feature, kita dapat lebih mudah dimana kode untuk suatu fitur ditulis
  - Where data is used, dan kita bisa dapat dengan mudah mencari di mana suatu data digunakan di dalam kode
- Debugging Principles
  - Kode crash di dalam function
  - Function ditulis secara tidak benar
  - Data yang digunakan tidak benar
- Support debugging
  - Function harus mudah dimengerti
  - Data harus bisa dilacak

Passing Array dan Slices

Kita bisa juga me-passing array, array pointer, dan slice ke dalam parameter function, seperti contoh berikut:

```
func kode(x [3]int) int {
        return x[0]
}

func kode2(x *[3]int) int {
        (*x)[0] = (*x)[0] + 1
```

```
}

func kode3(sli []int) int {
        sli[0] = sli[0] + 1
}

func main() {
        a := [3]int{1, 2, 3}
        fmt.Println(kode(a))

        b := [3]int{4, 5, 6}
        fmt.Println(kode2(b))

        c := []int{7, 8, 9}
        fmt.Println(kode3(c))
}
```

Function: First Class Values

Variabel bisa dideklarasikan sebagai function. Function juga bisa di-passing sebagai parameter dan return value. Kita juga bisa membuat function tanpa nama yang biasa disebut juga sebagai anonymous function.

Function: Returning Function

Kita juga bisa me-return sebuah function di dalam function lain sehingga hasil return dari function itu bisa di-invoke layaknya function biasa.

Function: Variadic dan Deferred

Function bisa menerima banyak argument, variadic argument dituliskan menggunakan tanda ellipsis `...` dan akan diperlakukan sebagai slice. Pemanggilan sebuah function bisa ditunda (defer) sampai kode setelahnya selesai. Parameter variadic dituliskan seperti ini:

```
func getMax(vals ...int) int {
        maxV := -1
        for _, v := range vals {
                if v > maxV {
                        maxV = v
                }
        }
        return maxV
}
```

Pemanggilan function secara defer dilakukan seperti ini:

```
func main() {
        defer fmt.Println("Bye, kode!")
        fmt.Println("Halo, kode!")
        fmt.Println("Halo, kode dua!")
}
```

Class dan Encapsulation

Class mendeskripsikan sebuah object, class bisa mendeskripsikan atribut dan behavior dari object tersebut. Encapsulation berguna untuk menjaga atribut agar tidak bisa diakses dari luar class.

Point Receivers

Kita bisa menyematkan sebuah function ke dalam sebuah tipe data agar saat kita me-running function tersebut kita tidak perlu mengcopy parameternya secara terus-menerus. Hal ini dapat membuat program kita berjalan lebih cepat dan memory-efficient.

Classes Support

Karena pada go tidak ada keyword class, maka pada golang kita membuat OOP dengan cara yang berbeda. Untuk membuat class pada go, kita bisa mengasosiasikan function/method terhadap sebuah tipe data menggunakan point receiver yang ditulis sebelum nama function.

```
type MyInt int
func (mi MyInt) Double() int {
        return int(mi*2)
}
```

Agar data field pada sebuah struct tidak dapat diakses dari luar, maka kita bisa menyembunyikan data pada struct hanya dengan menuliskan nama datanya berawalan dengan huruf kecil.

Encapsulation

Untuk mengontrol akses sebuah data dalam sebuah struct secara publik, kita bisa menuliskan nama variable, function, struct, field dengan huruf dengan kapital agar datanya bisa diakses dari package lain.

Referencing dan Dereferencing

Saat kita memanggil method dari pointer receiver, kita tidak perlu melakukan dereferencing, karena hal ini sudah dihandle secara otomatis oleh golang.

Referencing dan Dereferencing

Polymorhism adalah satu kemampuan dari objek untuk mempunyai bentuk yang lain tergantung konteksnya. Pada Inheritance/pewarisan, subclass akan mewarisi sifat dari super class.

Interface

Interface adalah suatu kumpulan method signature untuk sebuah object tetapi implementasinya tidak dituliskan.

Interface dan Concrete Types

Concrete types menspesifikasikan implementasi dari data. Interface types hanya menspesifikasikan signature dari method. Interface value pada golang bisa digunakan untuk membuat dynamic type, dan bisa juga untuk membuat contract signature.

Assertions

Type Assertions biasa digunakan untuk menentukan dan mengekstraksi concrete type yang ada di dalam sebuah function. Contoh type assertions pada golang adalah sebagai berikut:

```
func DrawShape(s Shape2D) bool {
  rect, ok := s.(Rectangle)
}
```

Error Handling

Kebanyakan program pada go akan me-return error interface pada objek untuk mengindikasikan adanya error. Untuk menghandle error, kita bisa mengecek apakah error nya sama dengan `nil` atau tidak