HACKTIV8

Go Programming Microservice Into The Specialization

Sesi 1 : Defer & Exit, Error Handling

Defer & Exit

Defer

Defer merupakan sebuah keyword pada bahasa Go yang digunakan untuk memanggil sebuah *function* yang dimana proses eksekusi akan di tahan hingga block sebuah *function* selesai.
Agar lebih mudah dipahami, mari perhatikan pada gambar disebelah kanan.

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      defer fmt.Println("defer function starts to execute")
7      fmt.Println("Hai everyone")
8      fmt.Println("Welcome back to Go learning center")
9  }
```

Terdapat 3 buat function *fmt.Println* yang akan menampilkan text yang berbeda-beda. Namun function *fmt.Println* pertama dipanggil dengan menggunakan *defer*.

```
Hai everyone
Welcome back to Go learning center
defer function starts to execute
```

Jika dijalankan maka hasilnya akan seperti pada gambar kedua. Perhatikan bahwa text yang berada pada posisi paling bawah gambar kedua merupakan text yang ditampilkan menggunakan *defer* pada line 6 di gambar pertama.

Defer

Perhatikan pada gambar di sebelah kanan. Terdapat sebuah *function* bernama *callDeferFunc* yang digunakan untuk memanggil function *deferFunc* dengan keyword *defer*. Kemudian function *callDeferFunc* dipanggil di baris paling atas function *main* yang kemudian diikuti sebuah pemanggilan function *fmt.Println.*

Jika kita jalankan pada terminal, maka hasilnya akan seperti pada gambar kedua. Perhatikan bahwa tulisan **Defer func starts to execute** yang berasal dari function *deferFunc* dipanggil menggunakan keyword *defer,* ditampilkan lebih dulu daripada tulisan **Hai everyone!!** yang berasal dari function *fmt.Println.* Ini terjadi karena *deferFunc* dipanggil didalam block function *callDeferFunc,* bukan didalam block function *main*. Maka dari itu pemanggilan function *deferFunc* akan di eksekusi setelah function *callDeferFunc* telah mengeksekusi seluruh statement atau syntax nya.

```go
13  package main
14
15  import "fmt"
16
17  func main() {
18      callDeferFunc()
19      fmt.Println("Hai everyone!!")
20  }
21
22  func callDeferFunc() {
23      defer deferFunc()
24  }
25
26  func deferFunc() {
27      fmt.Println("Defer func starts to execute")
28  }
```

```
Defer func starts to execute
Hai everyone!!
```

Exit

Funcion *exit* yang yang berasal dari package *os* berguna untuk menghentikan suatu program secara paksa. Perhatikan hasil eksekusi syntax dari gambar pertama pada gambar kedua.

Tulisan **Invoke with defer** tidak ditampilkan. Ini karena setelah function *fmt.PrintIn* pada line 44 dijalankan, program di hentikan secara paksa oleh function *os.Exit* ditengah jalan. Function *os.Exit* menerima satu parameter berupa nilai dengan tipe data *int* yang digunakan sebagai status exit nya.

```
35  package main
36
37  import (
38    "fmt"
39    "os"
40  )
41
42  func main() {
43    defer fmt.Println("Invoke with defer")
44    fmt.Println("Before Exiting")
45    os.Exit(1)
46  }
47
```

```
Defer func starts to execute
Hai everyone!!
```

HACKTIV8

Error, Panic & Recover

HACKTIV8

Error

Error merupakan sebuah tipe data pada bahasa Go yang digunakan untuk me-return sebuah error jika ada error yang terjadi. Umumya, nilai *error* akan di-return pada pada posisi terakhir dari nilai-nilai return sebuah *function*. Contohnya seperti pada function *strconv.Atoi* yang digunakan untuk mengkonversi tipe data *string* menjadi tipe data *int*. Function *strconv.Atoi* akan me-return *error* jika nilai dengan tipe data *string* tersebut tidak bisa di konversi menjadi sebuah tipe data *int*, seperti contohnya nilai string yang mengandung nilai diluar angka **"123GH"**. Perhatikan pada gambar disebelah kanan.

Terdapat 2 variable bernama *number* dan *err*. Variable *number* memiliki tipe data *int* dan variable *err* memiliki tipe data *error*. Lalu kedua variable tersebut di reassign untuk menampung hasil dari function *strconv.Atoi*.

Karena argumen yang kita berikan pada function *strconv.Atoi* mengandung huruf, maka function *strconv.Atoi* akan me-return nilai dengan tipe data *error* yang kemudian akan ditampung oleh variable *err*. Ketika tidak ada error yang di return maka variable err akan menjadi *nil* karena zero value dari tipe data *error* adalah *nil*. Maka dari itu kita bisa menggunakan kondisional untuk memerika jika variable *err* sama dengan *nil* maka variable number dapat ditampilkan, dan jika tidak maka variable *err* akan ditampilkan. Lalu kita kita juga menggunakan method *Error* untuk menampilkan pesan jika terjadi sebuah error. Method *Error* berasal dari tipe data *error*.

```go
package main

import (
  "fmt"
  "strconv"
)

func main() {
  var number int
  var err error

  number, err = strconv.Atoi("123GH")

  if err == nil {
    fmt.Println(number)
  } else {
    fmt.Println(err.Error())
  }

  number, err = strconv.Atoi("123")

  if err == nil {
    fmt.Println(number)
  } else {
    fmt.Println(err.Error())
  }

}
```

Error

Kemudian pada line 20, karena argumen yang kita berikan pada function *strconv.Atoi* tersebut adalah angka semua, maka tidak akan ada error yang di return dan line 23 dapat ditampilkan.

Jika kita jalankan pada terminal kita, maka hasilnya akan seperti pada gambar kedua.

```
strconv.Atoi: parsing "123GH": invalid syntax
123
```

```go
package main

import (
  "fmt"
  "strconv"
)

func main() {
  var number int
  var err error

  number, err = strconv.Atoi("123GH")

  if err == nil {
    fmt.Println(number)
  } else {
    fmt.Println(err.Error())
  }

  number, err = strconv.Atoi("123")

  if err == nil {
    fmt.Println(number)
  } else {
    fmt.Println(err.Error())
  }

}
```

Error (custom error)

Kita juga dapat membuat pesan error kita sendiri dengan menggunakan method *New* yang merupakan sebuah method miliki tipe data *error*. Contohnya seperti pada gambar pertama di sebelah kanan.

Function *fmt.Scanln* digunakan untuk menangkap input yang kita tuliskan pada terminal. Lalu kita memberikan tanda memberikan nilai *pointer* dari variable *password* pada argumennya agar inputan kita dapat di simpan pada variable *password*.

Kemudian kita membuat function *validPassword* yang digunakan untuk memeriksa apakah password yang kita input pada terminal memiliki panjang karaketer lebih dari 4. Jika memang lebih dari 4 karakter maka function *validPassword* akan mereturn sebuah pesan berupa “Valid password” dan nilai *error* yang direturn berupa *nil*. Namun jika terjadi sebaliknya, maka function *validPassword* akan me-return akan mereturn sebuah empty *string* dan sebuah pesan error yang dibuat menggunakan method *errors.New*.

```go
package main

import (
  "errors"
  "fmt"
)

func main() {
  var password string

  fmt.Scanln(&password)

  if valid, err := validPassword(password); err != nil {
    fmt.Println(err.Error())
  } else {
    fmt.Println(valid)
  }

}

func validPassword(password string) (string, error) {
  pl := len(password)

  if pl < 5 {
    return "", errors.New("password has to have more than 4 characters")
  }

  return "Valid password", nil
}
```

HACKTIV8

Error (custom error)

Pada line 44 hiingga 48, terdapat sebuah kondisional if else untuk memeriksa hasil dari function validPassword. Dan jika kita jalankan pada terminal kita diikuti dengan mengetikkan input, maka hasilnya akan seperti pada gambar kedua.

Bisa dilihat, pesan yang ditunjukkan pada gambar kedua merupakan pesan error yang dibuat menggunakan method *errors.New*. Pesan error ditampilkan karena kita menginput **asd** yang dimana panjangnya tidak lebih dari 4 karakter.

Namun misalkan kita menginput lebih dari 4 karakter, maka hasilnya akan seperti pada gambar ketiga.

```
asd
password has to have more than 4 characters
```

```
asdfgh
Valid password
```

```go
package main

import (
  "errors"
  "fmt"
)

func main() {
  var password string

  fmt.Scanln(&password)

  if valid, err := validPassword(password); err != nil {
    fmt.Println(err.Error())
  } else {
    fmt.Println(valid)
  }

}

func validPassword(password string) (string, error) {
  pl := len(password)

  if pl < 5 {
    return "", errors.New("password has to have more than 4 characters")
  }

  return "Valid password", nil
}
```

HACKTIV8

Panic

Panic digunakan untuk menampilkan stack trace error sekaligus menghentikan flow goroutine (karena main() juga merupakan goroutine, maka behaviour yang sama juga berlaku). Setelah ada panic, proses akan terhenti, apapun setelah tidak di-eksekusi kecuali proses yang sudah di-defer sebelumnya (akan muncul sebelum panic error).

Panic menampilkan pesan error di console, sama seperti fmt.Println(). Informasi error yang ditampilkan adalah stack trace error, jadi sangat mendetail dan heboh.

Perhatikan gambar pertama di sebelah kanan. Kita mengganti function *fmt.Println* pada line 76 menjadi function *Panic*. Lalu mari kita jalankan terminal kita dan mengetikan sebuah input kurang dari 5 karakter agar menghasilkan error. Hasilnya akan seperti pada gambar kedua.

```
70  func main() {
71    var password string
72
73    fmt.Scanln(&password)
74
75    if valid, err := validPassword(password); err != nil {
76      panic(err.Error())
77    } else {
78      fmt.Println(valid)
79    }
80
81  }
82
83  func validPassword(password string) (string, error) {
84    pl := len(password)
85
86    if pl < 5 {
87      return "", errors.New("password has to have more than 4 characters")
88    }
89
90    return "Valid password", nil
91  }
```

```
qq
panic: password has to have more than 4 characters

goroutine 1 [running]:
main.main()
        /Users/sofyan/go/src/GLNG-Materi-Baru-PTP-2021/Sesi5/panic/error.go:76 +0x199
exit status 2
```

HACKTIV8

Panic

Function *recover* digunakan untuk menangkap error yang terjadi pada sebuah *Goroutine*. Sekarang kita akan menambahkan sebuah function bernama *catchErr* pada codingan kita sebelumnya yang akan menangkap error panic nya menggunakan function *recover*. Dan perlu diingat bahwa kita perlu memanggil function *catchErr* dengan keyword *defer*.

Jika kita jalankan, maka hasilnya akan seperti pada gambar kedua. Bisa kita lihat, pesan dari panic error tidak ditampilkan karena telah di tangkap menggunakan function *recover* yang terdapat didalam function *catchErr*.

```go
func main() {
  defer catchErr()

  var password string

  fmt.Scanln(&password)

  if valid, err := validPassword(password); err != nil {
    panic(err.Error())
  } else {
    fmt.Println(valid)
  }

}

func catchErr() {
  if r := recover(); r != nil {
    fmt.Println("Error occured:", r)
  } else {
    fmt.Println("Application running perfectly")
  }
}

func validPassword(password string) (string, error) {
  pl := len(password)

  if pl < 5 {
    return "", errors.New("password has to have more than 4 characters")
  }

  return "Valid password", nil
}
```

```
123
Error occured: password has to have more than 4 characters
```