Go-Lang JSON

Eko Kurniawan Khannedy

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 10+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow

Eko Kurniawan Khannedy

- Telegram : @khannedy
- Facebook : fb.com/ProgrammerZamanNow
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Telegram Channel : t.me/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com

Sebelum Belajar

- Go-Lang Dasar
- Go-Lang Modules
- Go-Lang Unit Test

Agenda

- Pengenalan Package json
- Encode dan Decode JSON
- Stream Encoder dan Decoder

Pengenalan Package json

Pengenalan JSON

- JSON singkatan dari JavaScript Object Notation, merupakan struktur format data yang bentuknya seperti Object di JavaScript
- JSON merupakan struktur format data yang paling banyak digunakan saat kita membuat RESTful API
- Dan pate kelas ini kita akan menggunakan JSON juga
- https://www.json.org/json-en.html

Kode : Contoh JSON

```json
{
  "firstName": "Eko",
  "middleName": "Kurniawan",
  "lastName": "Khannedy",
  "website": {
    "title": "Programmer Zaman Now",
    "url": "https://www.programmerzamannow.com"
  },
  "hobbies": [
    "Coding",
    "Reading",
    "Gaming"
  ]
}
```

Package json

- Go-Lang sudah menyediakan package json, dimana kita bisa menggunakan package ini untuk melakukan konversi data ke JSON (encode) atau sebaliknya (decode)
- https://pkg.go.dev/encoding/json

Encode JSON

Encode JSON

- Go-Lang telah menyediakan function untuk melakukan konversi data ke JSON, yaitu menggunakan function json.Marshal(interface{})
- Karena parameter nya adalah interface{}, maka kita bisa masukan tipe data apapun ke dalam function Marshal

Kode : Encode JSON

```go
func logJson(data interface{}) {
    bytes, err := json.Marshal(data)
    if err != nil {
        panic(err)
    }
    fmt.Println(string(bytes))
}
```

```go
func TestMarshal(t *testing.T) {
    logJson("Eko")
    logJson(1)
    logJson(true)
    logJson([]string{"Eko", "Kurniawan", "Khannedy"})
}
```

JSON Object

JSON Object

- Pada materi sebelumnya kita melakukan encode data seperti string, number, boolean, dan tipe data primitif lainnya
- Walaupun memang bisa dilakukan, karena sesuai dengan kontrak interface{}, namun tidak sesuai dengan kontrak JSON
- Jika mengikuti kontrak json.org, data JSON bentuknya adalah Object dan Array
- Sedangkan value nya baru berupa

Kode : Contoh Object JSON

```
{
  "firstName": "Eko",
  "middleName": "Kurniawan",
  "lastName": "Khannedy"
}
```

Struct

- JSON Object di Go-Lang direpresentasikan dengan tipe data Struct
- Dimana tiap attribute di JSON Object merupakan attribute di Struct

Kode : Encode Struct ke JSON (1)

```
type Customer struct {
    FirstName  string
    MiddleName string
    LastName   string
}
```

Kode : Encode Struct ke JSON (2)

```
customer := Customer{
    FirstName:  "Eko",
    MiddleName: "Kurniawan",
    LastName:   "Khannedy",
}

bytes, _ := json.Marshal(customer)
fmt.Println(string(bytes))
```

Decode JSON

Decode JSON

- Sekarang kita sudah tahu bagaimana caranya melakukan encode dari tipe data di Go-Lang ke JSON
- Namun bagaimana jika kebalikannya?
- Untuk melakukan konversi dari JSON ke tipe data di Go-Lang (Decode), kita bisa menggunakan function json.Unmarshal(byte[], interface{})
- Dimana byte[] adalah data JSON nya, sedangkan interface{} adalah tempat menyimpan hasil konversi, biasa berupa pointer

Kode : Decode JSON

```
jsonRequest := `{"FirstName":"Eko","MiddleName":"Kurniawan","LastName":"Khannedy"}`
jsonBytes := []byte(jsonRequest)

customer := &Customer{}
json.Unmarshal(jsonBytes, customer)

fmt.Println(customer)
```

JSON Array

JSON Array

- Selain tipe dalam bentuk Object, biasanya dalam JSON, kita kadang menggunakan tipe data Array
- Array di JSON mirip dengan Array di JavaScript, dia bisa berisikan tipe data primitif, atau tipe data kompleks (Object atau Array)
- Di Go-Lang, JSON Array direpresentasikan dalam bentuk slice
- Konversi dari JSON atau ke JSON dilakukan secara otomatis oleh package json menggunakan tipe data slice

Kode : JSON Array Primitive (1)

```
type Customer struct {
    FirstName  string
    MiddleName string
    LastName   string
    Hobbies    []string
}
```

Kode : JSON Array Primitive (2)

```
customer := Customer{
    FirstName:  "Eko",
    MiddleName: "Kurniawan",
    LastName:   "Khannedy",
    Hobbies:    []string{"Gaming", "Reading", "Coding"},
}

bytes, _ := json.Marshal(customer)
fmt.Println(string(bytes))
```

Kode : JSON Array Complex (1)

```
type Address struct {
    Street     string
    Country    string
    PostalCode string
}
```

```
type Customer struct {
    FirstName  string
    MiddleName string
    LastName   string
    Hobbies    []string
    Addresses  []Address
}
```

Kode : JSON Array Complex (2)

```
customer := Customer{
    FirstName:  "Eko",
    MiddleName: "Kurniawan",
    LastName:   "Khannedy",
    Hobbies: []string{
        "Gaming", "Reading", "Coding",
    },
    Addresses: []Address{
        {
            Street:     "Jalan Belum Jadi",
            Country:    "Indonesia",
            PostalCode: "41111",
        },
    },
}
```

```
bytes, _ := json.Marshal(customer)
fmt.Println(string(bytes))
```

Decode JSON Array

- Selain menggunakan Array pada attribute di Object
- Kita juga bisa melakukan encode atau decode langsung JSON Array nya
- Encode dan Decode JSON Array bisa menggunakan tipe data Slice

Kode : Decode JSON Array

```
jsonArray := `[{"Street":"Jalan Belum Jadi","Country":"Indonesia","PostalCode":"41111"}]`
jsonBytes := []byte(jsonArray)

addresses := &[]Address{}
json.Unmarshal(jsonBytes, addresses)

fmt.Println(addresses)
```

JSON Tag

JSON Tag

- Secara default atribut yang terdapat di Struct dan JSON akan di mapping sesuai dengan nama atribut yang sama (case sensitive)
- Kadang ada style yang berbeda antara penamaan atribute di Struct dan di JSON, misal di JSON kita ingin menggunakan snake_case, tapi di Struct, kita ingin menggunakan PascalCase
- Untungnya, package json mendukun Tag Reflection
- Kita bisa menambahkan tag reflection dengan nama json, lalu diikuti dengan atribut yang kita inginkan ketika konversi dari atau ke JSON

Kode : JSON Tag

```go
type Product struct {
    Id       string `json:"id"`
    Name     string `json:"name"`
    Price    int64  `json:"price"`
    ImageURL string `json:"image_url"`
}
```

Map

Map

- Saat menggunakan JSON, kadang mungkin kita menemukan kasus data JSON nya dynamic
- Artinya atribut nya tidak menentu, bisa bertambah, bisa berkurang, dan tidak tetap
- Pada kasus seperti itu, menggunakan Struct akan menyulitkan, karena pada Struct, kita harus menentukan semua atribut nya
- Untuk kasus seperti ini, kita bisa menggunakan tipe data map[string]interface{}
- Secara otomatis, atribut akan menjadi key di map, dan value menjadi value di map
- Namun karena value berupa interface{}, maka kita harus lakukan konversi secara manual jika ingin mengambil value nya
- Dan tipe data Map tidak mendukung JSON Tag lagi

Kode : Map

```
jsonRequest := `{"id":"12345","name":"Apple iPhone","price":20000000}`
jsonBytes := []byte(jsonRequest)

var result map[string]interface{}
_ = json.Unmarshal(jsonBytes, &result)

fmt.Println(result)
fmt.Println(result["id"])
fmt.Println(result["name"])
fmt.Println(result["price"])
```

Streaming Decoder

Streaming Decoder

- Sebelumnya kita belajar package json dengan melakukan konversi data JSON yang sudah dalam bentuk variable dan data string atau []byte
- Pada kenyataanya, kadang data JSON nya berasal dari Input berupa io.Reader (File, Network, Request Body)
- Kita bisa saja membaca semua datanya terlebih dahulu, lalu simpan di variable, baru lakukan konversi dari JSON, namun hal ini sebenarnya tidak perlu dilakukan, karena package json memiliki fitur untuk membaca data dari Stream

json.Decoder

- Untuk membuat json Decoder, kita bisa menggunakan function json.NewDecoder(reader)
- Selanjutnya untuk membaca isi input reader dan konversikan secara langsung ke data di Go-Lang, cukup gunakan function Decode(interface{})

Kode : Streaming Decoder

```
reader, _ := os.Open("sample.json")
decoder := json.NewDecoder(reader)

customer := Customer{}
_ = decoder.Decode(&customer)

fmt.Println(customer)
```

Streaming Encoder

Streaming Encoder

- Selain decoder, package json juga mendukung membuat Encoder yang bisa digunakan untuk menulis langsung JSON nya ke io.Writer
- Dengan begitu, kita tidak perlu menyimpan JSON datanya terlebih dahulu ke dalam variable string atau []byte, kita bisa langsung tulis ke io.Writer

json.Encoder

- Untuk membuat Encoder, kita bisa menggunakan function json.NewEncoder(writer)
- Dan untuk menulis data sebagai JSON langsung ke writer, kita bisa gunakan function Encode(interface{})

Kode : Streaming Encoder

```
writer, _ := os.Create("sample_output.json")
encoder := json.NewEncoder(writer)

customer := Customer{
    FirstName:  "Eko",
    MiddleName: "Kurniawan",
    LastName:   "Khannedy",
}
_ = encoder.Encode(customer)

fmt.Println(customer)
```

Materi Selanjutnya

Materi Selanjutnya

- Go-Lang RESTful API