Chapter1. 마이크로서비스 소개

패키지를 사용해서
웹 서버 만들기, Go 구조체를 Json으로 마샬링하기

net/http로 간단한 웹 서버 만들기

정적 텍스트를 리턴하는 하나의 엔드포인트를 가진 HTTP 서버 만들기

net/http 패키지

- HTTP 프로토콜을 사용해서 통신하는 다른 서버에 요청을 보냄

  또는

- 요청을 별도의 Go 함수에서 처리
- 정적 파일을 제공하는 등
- HTTP 서버를 실행할 수 있는 기능 제공

net/http로 간단한 웹 서버 만들기

basic_http_example.go

```
package main

import (
"fmt"
"log"
"net/http"
)

func main() {
port := 8080

http.HandleFunc("/helloworld", helloWorldHandler)

        log.Printf("Server starting on port %v\n", port)
        log.Fatal(http.ListenAndServe(fmt.Sprintf(":%v", port), nil))
}

func helloWorldHandler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprint(w, "Hello World\n")
}
```

HandleFunc 메서드

- “모든IP주소:8080/helloword”에 서버를 바인딩
- helloWorldHandler 함수 호출

helloWorldHandler 함수

- Hello World 찍기

기대하는 바

이 코드를 실행하면

1. **"Server starting on port:8080”** 프린트문 확인
2. “모든IP주소:8080/helloword”에 접속
3. helloWorldHandler 함수의 반환값 확인 가능

net/http로 간단한 웹 서버 만들기

basic_http_example.go

```
package main

import (
"fmt"
"log"
"net/http"
)

func main() {
port := 8080

http.HandleFunc("/helloworld", helloWorldHandler)

        log.Printf("Server starting on port %v\n", port)
        log.Fatal(http.ListenAndServe(fmt.Sprintf(":%v", port), nil))
}

func helloWorldHandler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprint(w, "Hello World\n")
}
```

실제 실행

1.명령어 입력
$ go run ./basic_http_example.go
2018/11/10 12:59:52 Server starting on port 8080

2. **http://localhost:8080/helloworld 로 접속**
Hello World 라는 응답확인 가능

net/http로 간단한 웹 서버 만들기

포트 중복 문제

예시)

2018/11/10 13:11:04 Server starting on port 8080
2018/11/10 13:11:04 listen tcp :8080: bind: address already in use
exit status 1

해결방법

1. 터미널 열기
2. ps aux | grep ‘go run’
3. 현재 동작하고 있는 프로세스 리스트 확인
4. kill -9 “프로세스 id”
5. 해당하는 프로세스 종료됨

JSON 읽기 및 쓰기

표준 Go 구조체를 JSON 문자열로 변환

encoding/json 패키지

- 내장 표준 라이브러리
- JSON <-> Go 인코딩 및 디코딩 빠르고 쉽게
- **마샬**: 프로그램상의 데이터 구조를 바이너리로 표현하는 직렬화
- **언마샬**: 바이너리로 표현된 데이터를 프로그램상의 데이터구조로 역지렬화
- 인코더 및 디코더 타입도 제공

Go 구조체를 JSON으로 마샬링하기

마샬링 함수

func Marshal(v interface{}) ([]bytes, error)

- interface는 Go에서 모든 타입을 나타낸다. -> 모든 타입을 매개변수로 사용 가능
- (byte(리턴타입), error(에러))의 튜플 리턴
- 동작이 성공하면 에러는 nil이다.

Go는 다른 언어의 예외처리와 달리 **패닉(panic)과 복구(recover)**로 구현됨

- 런타임 패닉으로 인해 주어진 객체에 대해 JSON으로 인코딩된 바이트 배열을 생성할 수 없는 경우,
- 이를 포착해 문제를 자세히 설명하는 에러 객체를 호출한 함수로 리턴

Go 구조체를 JSON으로 마샬링하기

```
package main

import (
  "encoding/json"
  "fmt"
 "log"
 "net/http"
)

type helloWorldResponse struct {
        Message string
}

func main() {
        port := 8080
        http.HandleFunc("/helloworld", helloWorldHandler)
        log.Printf("Server starting on port %v\n", port)
        log.Fatal(http.ListenAndServe(fmt.Sprintf(":%v", port), nil))
}

func helloWorldHandler(w http.ResponseWriter, r *http.Request) {
         response := helloWorldResponse{Message: "Hello World"}
        data, err := json.Marshal(response)
         if err != nil {
         panic("Ooops")
         }
  fmt.Fprint(w, string(data))
}
```

helloWorldHandler 핸들러

1. 객체의 인스턴스를 만든다
2. 메세지를 설정한다
3. Marshal 함수를 사용해 문자열로 인코딩
4. 리턴

프로그램 실행시

{“Message”:”Hello World”}라는
유효한 JSON으로 렌더링 된 출력을
브라우저에서 볼 수 있다.

{“message”:”Hello World”}의 출력을 보고 싶다면?
helloWorldResponse 구조체의 Message를
message로 바꾸면 될까?
안된다. Go에서는 소문자로 된 프로퍼티는 내보내지 않는다.
그렇다면? 구조체 필드 태그 사용

Go 구조체를 JSON으로 마샬링하기

```
package main

import (
  "encoding/json"
  "fmt"
 "log"
 "net/http"
)

type helloWorldResponse struct {
        Message string `json:"message"`
}

func main() {
        port := 8080
        http.HandleFunc("/helloworld", helloWorldHandler)
        log.Printf("Server starting on port %v\n", port)
        log.Fatal(http.ListenAndServe(fmt.Sprintf(":%v", port), nil))
}

func helloWorldHandler(w http.ResponseWriter, r *http.Request) {
         response := helloWorldResponse{Message: "Hello World"}
        data, err := json.Marshal(response)
         if err != nil {
         panic("Ooops")
         }
  fmt.Fprint(w, string(data))
}
```

구조체 필드 태그 사용

출력이 어떻게 표시되는 지 더 잘 제어할 수 있다.

다음 코드의 출력 결과

{“message”:”Hello World”}

그렇다면 JSON으로 인코딩 할 수 없는 것들이 있을까?

1. 채널, 복소수 및 함수 -> UnsupportedTypeError
2. 순환 데이터 구조 -> 무한 재귀

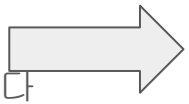
Go 구조체를 JSON으로 마샬링하기

들여쓰기

- 깔끔하게 양식을 지정한 JSON으로 내보내기 위해 **MarshalIndent함수** 사용

func MarshalIndent(v interface{}. prefix, indent string) ([]byte, error)

- 구조체를 바이트 배열로 디코딩
- 디코딩된 구조체를 응답 스트림에 쓴다



비효율

효율 : 데이터를 리턴하기 전에 임시 객체에 마샬링하지않고 바로 출력 스트림으로 데이터를 보낼 수 있는 방법

Go에서 스트림에 직접 쓸 수 있는 **효율적인 인코더 및 디코더** 제공

Go 구조체를 JSON으로 마샬링하기

1.ResponseWriter 인터페이스

인터페이스 구조 : Header()/Write()/WriteHeader()

fmt.Fprint()와 함께 사용하려면?

- fmt.Fprint(w io.Writer, a … interface {})
- w는 Writer 인터페이스
- RespinseWriter은 Write라는 메서드를 구현하므로 Writer 인터페이스 충족

따라서 ResponseWriter 인터페이스를 구현하는 모든 객체는 Writer를 매개 변수로 요구하는 모든 함수에 전달할 수 있다.

2.NewEncoder 함수 from encoding/json 패키지

return: 열려있는 writer에 JSON을 바로 쓸 수 있는 encoder 객체

func NewEncoder(w io.Writer) * Encoder

따라서 Marshsal의 결과를 바이트 배열에 저장하는 대신 HTTP 응답에 바로 쓸수있다.

예시)

```
func helloWorldHandler(w http.ResponseWriter, r *http.Request) {
  response := helloWorldResponse{Message: "HelloWorld"}
  encoder := json.NewEncoder(w)
  encoder.Encode(response)
}
```

Go 구조체를 JSON으로 마샬링하기

성능측정

func BenchmarkHelloHandlerVariable(b *testing.B)
{b.ResetTimer()
 	
var writer = ioutil.Discard
response := Response{Message: "Hello World"}
 	
for i := 0; i < b.N; i++ {
data, _ := json.Marshal(response)
fmt.Fprint(writer, string(data))
}
}
	func BenchmarkHelloHandlerEncoder(b *testing.B)
	{b.ResetTimer()
	
	var writer = ioutil.Discard
	response := Response{Message: "Hello World"}
	
	for i := 0; i < b.N; i++ {
	encoder := json.NewEncoder(writer)
	encoder.Encode(response)
	}
	}

511 ns/op	328 ns/op

바이트 배열로 마샬링하는 것보다 Encoder를 사용하는 것이 약 50%정도 더 빠르다.

Go 구조체를 JSON으로 마샬링하기

이전 장에 설명된 두 함수의 비교는

표준 패키지의 동작 방식을 이해하고

요구사항에 맞는 올바른 옵션을 선택하는 것으로

성능 튜닝이 아니라 **프레임 워크의 이해**이다.

이어서 JSON을 Go 구조체로 언마샬링하기 ->