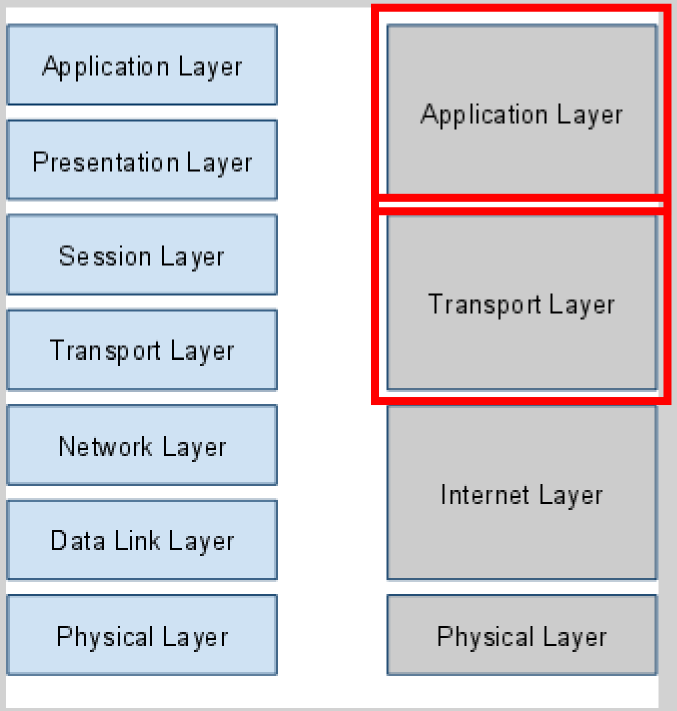
핵심! Go 프로그래밍

- 3일차

목차

통신



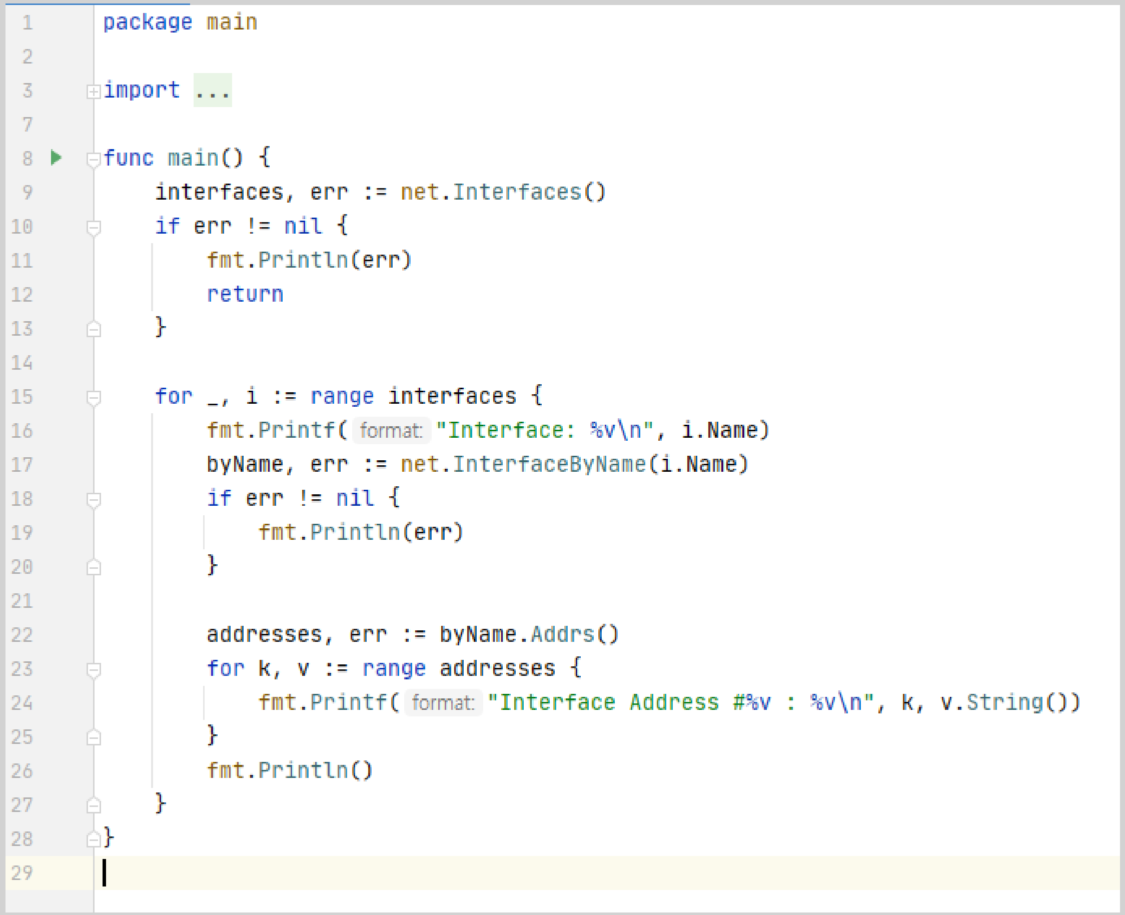
- Application Layer
  - 네트워크를 사용하는 응용프로그램
  - Http, FTP, Telnet, SMTP 등…
- Transport Layer
  - TCP, UDP 등…

출처: https://www.joinc.co.kr/w/Site/Network_Programing/Documents/IntroTCPIP

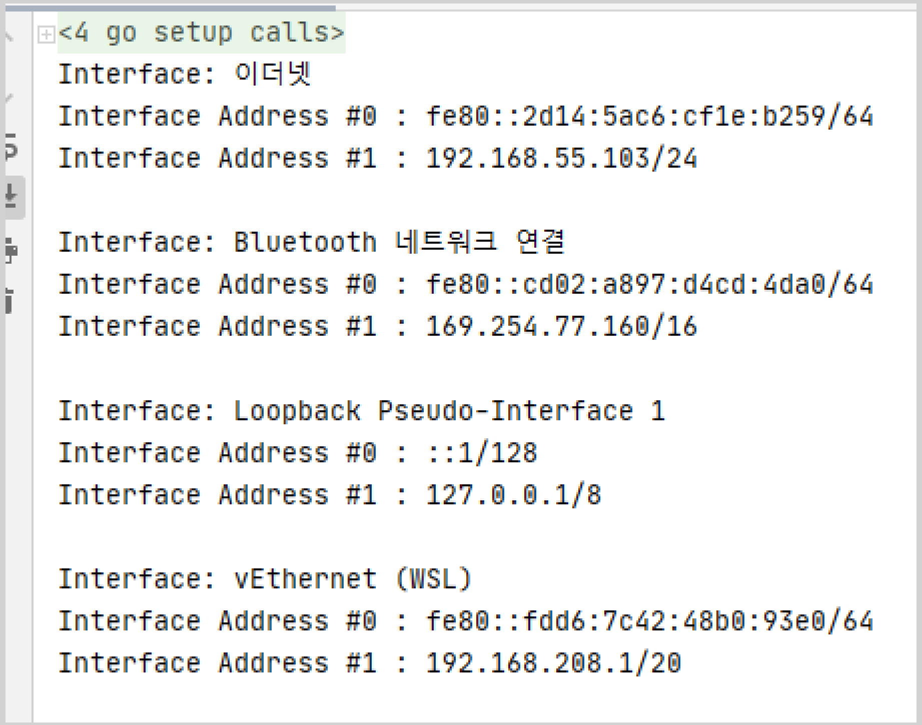
TCP/IP

- TCP (Transmission Control Protocol)
  - 세그먼트(Segment) 단위 데이터 전송 => TCP 패킷(Packet)
  - 풀 듀플렉스(Full Duplex) 방식의 가상 회로 생성을 통한 정확성 보장
  - Connect에 문제가 있는 경우, 이를 감지해 프로세서에 전달
- IP (Internet Protocol)
  - 신뢰성을 보장하지 않음
  - 네트워크서 사용되는 데이터를 캡슐화 해 Source에서 Destination으로 패킷을 전달하는 역할
  - 라우터가 필요함 (보통 TCP/IP를 지원하는 장치는 라우팅 기능을 수행할 수 있음)

TCP

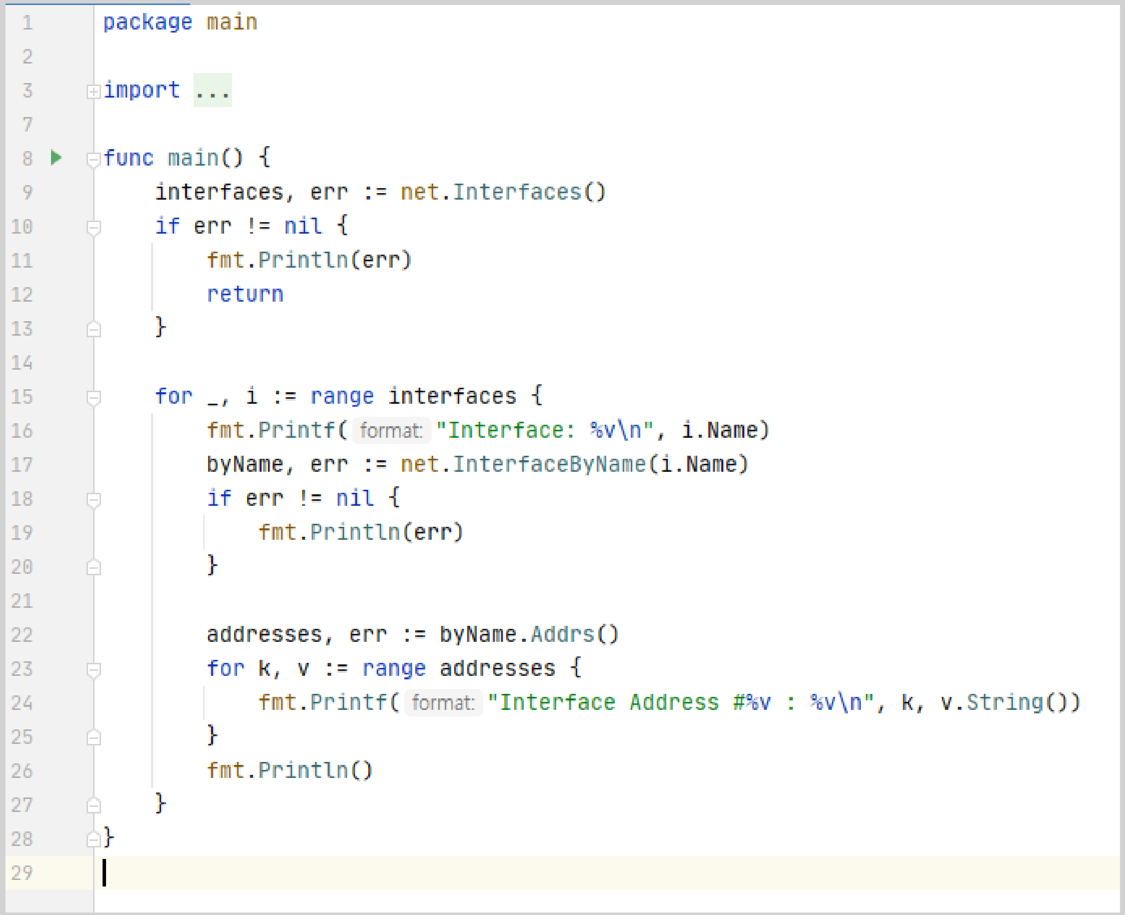


```go
package main

import ...

func main() {
	interfaces, err := net.Interfaces()
	if err != nil {
		fmt.Println(err)
		return
	}

	for _, i := range interfaces {
		fmt.Printf( format: "Interface: %v\n", i.Name)
		byName, err := net.InterfaceByName(i.Name)
		if err != nil {
			fmt.Println(err)
		}

		addresses, err := byName.Addrs()
		for k, v := range addresses {
			fmt.Printf( format: "Interface Address #%v : %v\n", k, v.String())
		}
		fmt.Println()
	}
}
```

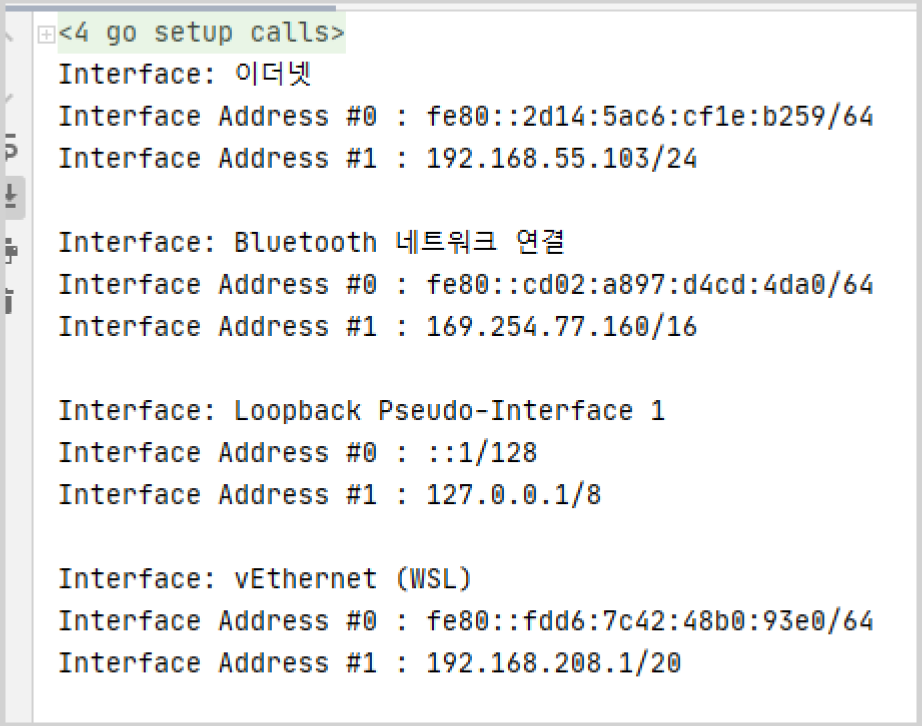


```
<4 go setup calls>
Interface: 이더넷
Interface Address #0 : fe80::2d14:5ac6:cf1e:b259/64
Interface Address #1 : 192.168.55.103/24

Interface: Bluetooth 네트워크 연결
Interface Address #0 : fe80::cd02:a897:d4cd:4da0/64
Interface Address #1 : 169.254.77.160/16

Interface: Loopback Pseudo-Interface 1
Interface Address #0 : ::1/128
Interface Address #1 : 127.0.0.1/8

Interface: vEthernet (WSL)
Interface Address #0 : fe80::fdd6:7c42:48b0:93e0/64
Interface Address #1 : 192.168.208.1/20
```

TCP

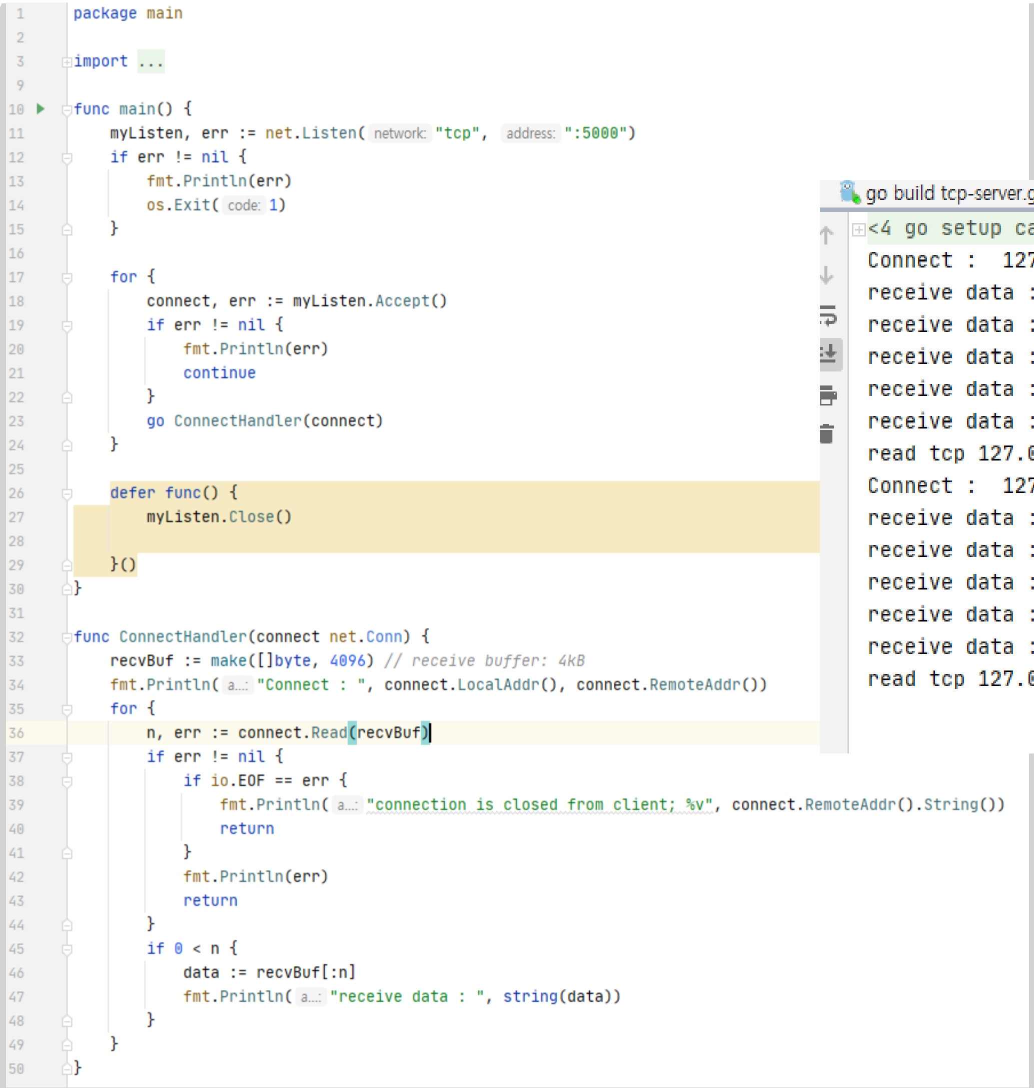


```go
package main

import ...

func main() {
	interfaces, err := net.Interfaces()
	if err != nil {
		fmt.Println(err)
		return
	}

	for _, i := range interfaces {
		fmt.Printf( format: "Interface: %v\n", i.Name)
		byName, err := net.InterfaceByName(i.Name)
		if err != nil {
			fmt.Println(err)
		}

		addresses, err := byName.Addrs()
		for k, v := range addresses {
			fmt.Printf( format: "Interface Address #%v : %v\n", k, v.String())
		}
		fmt.Println()
	}
}
```

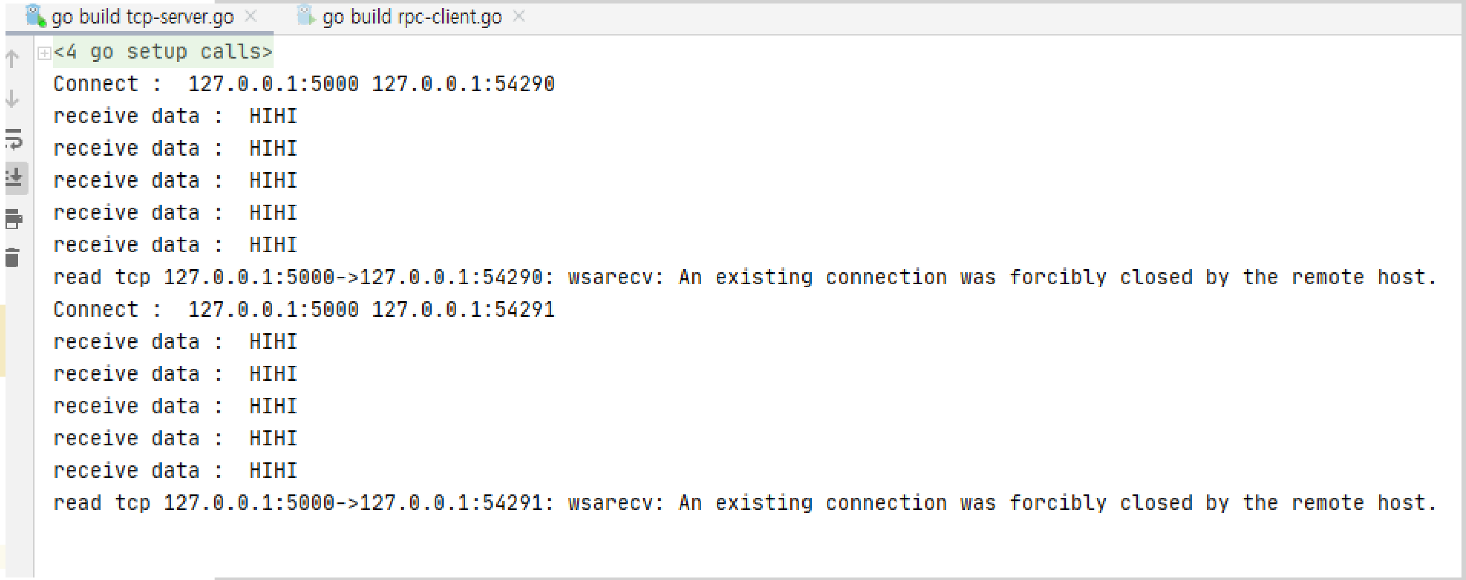


```
<4 go setup calls>
Interface: 이더넷
Interface Address #0 : fe80::2d14:5ac6:cf1e:b259/64
Interface Address #1 : 192.168.55.103/24

Interface: Bluetooth 네트워크 연결
Interface Address #0 : fe80::cd02:a897:d4cd:4da0/64
Interface Address #1 : 169.254.77.160/16

Interface: Loopback Pseudo-Interface 1
Interface Address #0 : ::1/128
Interface Address #1 : 127.0.0.1/8

Interface: vEthernet (WSL)
Interface Address #0 : fe80::fdd6:7c42:48b0:93e0/64
Interface Address #1 : 192.168.208.1/20
```

TCP



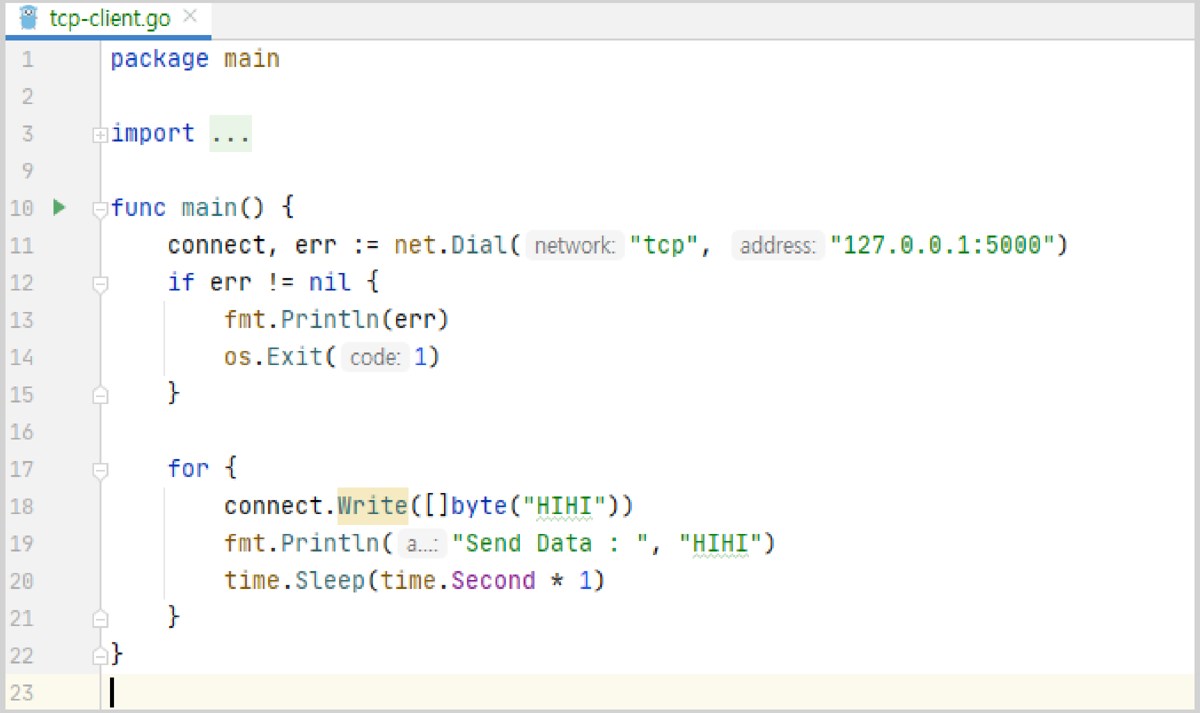
```go
package main

import ...

func main() {
    myListen, err := net.Listen( network: "tcp",  address: ":5000")
    if err != nil {
        fmt.Println(err)
        os.Exit( code: 1)
    }

    for {
        connect, err := myListen.Accept()
        if err != nil {
            fmt.Println(err)
            continue
        }
        go ConnectHandler(connect)
    }

    defer func() {
        myListen.Close()

    }()
}

func ConnectHandler(connect net.Conn) {
    recvBuf := make([]byte, 4096) // receive buffer: 4kB
    fmt.Println( a...: "Connect : ", connect.LocalAddr(), connect.RemoteAddr())
    for {
        n, err := connect.Read(recvBuf)
        if err != nil {
            if io.EOF == err {
                fmt.Println( a...: "connection is closed from client; %v", connect.RemoteAddr().String())
                return
            }
            fmt.Println(err)
            return
        }
        if 0 < n {
            data := recvBuf[:n]
            fmt.Println( a...: "receive data : ", string(data))
        }
    }
}
```



go build tcp-server.go   go build rpc-client.go

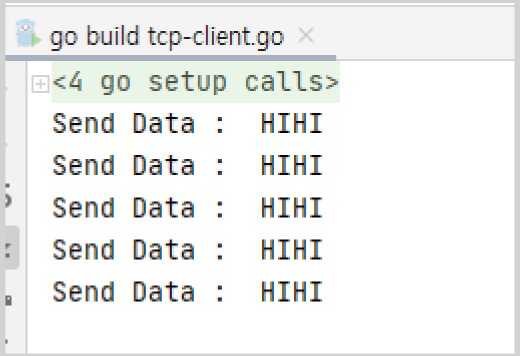
```
<4 go setup calls>
Connect :  127.0.0.1:5000 127.0.0.1:54290
receive data :  HIHI
receive data :  HIHI
receive data :  HIHI
receive data :  HIHI
receive data :  HIHI
read tcp 127.0.0.1:5000->127.0.0.1:54290: wsarecv: An existing connection was forcibly closed by the remote host.
Connect :  127.0.0.1:5000 127.0.0.1:54291
receive data :  HIHI
receive data :  HIHI
receive data :  HIHI
receive data :  HIHI
receive data :  HIHI
read tcp 127.0.0.1:5000->127.0.0.1:54291: wsarecv: An existing connection was forcibly closed by the remote host.
```

TCP



tcp-client.go

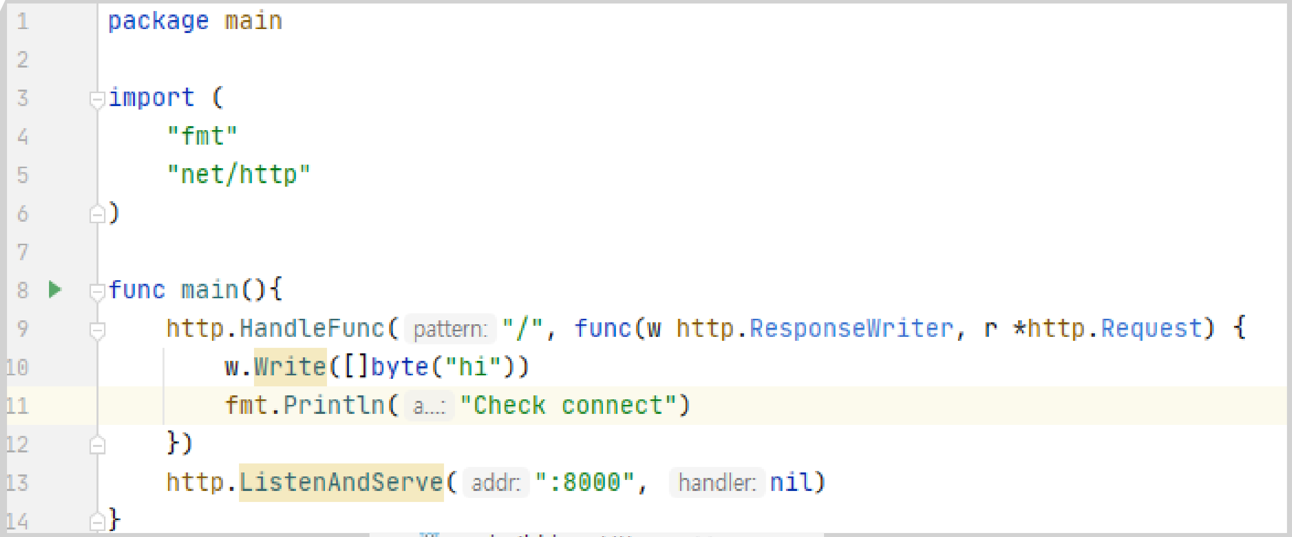
```go
package main

import ...

func main() {
	connect, err := net.Dial( network: "tcp",  address: "127.0.0.1:5000")
	if err != nil {
		fmt.Println(err)
		os.Exit( code: 1)
	}

	for {
		connect.Write([]byte("HIHI"))
		fmt.Println( a...: "Send Data : ", "HIHI")
		time.Sleep(time.Second * 1)
	}
}
```



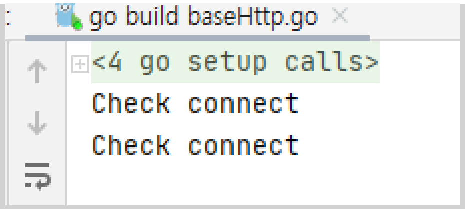
go build tcp-client.go

```
<4 go setup calls>
Send Data :  HIHI
Send Data :  HIHI
Send Data :  HIHI
Send Data :  HIHI
Send Data :  HIHI
```

HTTP

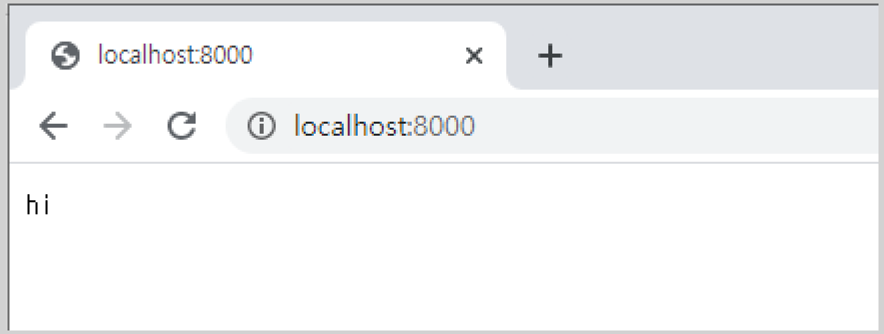


```go
package main

import (
	"fmt"
	"net/http"
)

func main(){
	http.HandleFunc( pattern: "/", func(w http.ResponseWriter, r *http.Request) {
		w.Write([]byte("hi"))
		fmt.Println( a...: "Check connect")
	})
	http.ListenAndServe( addr: ":8000",  handler: nil)
}
```



go build baseHttp.go

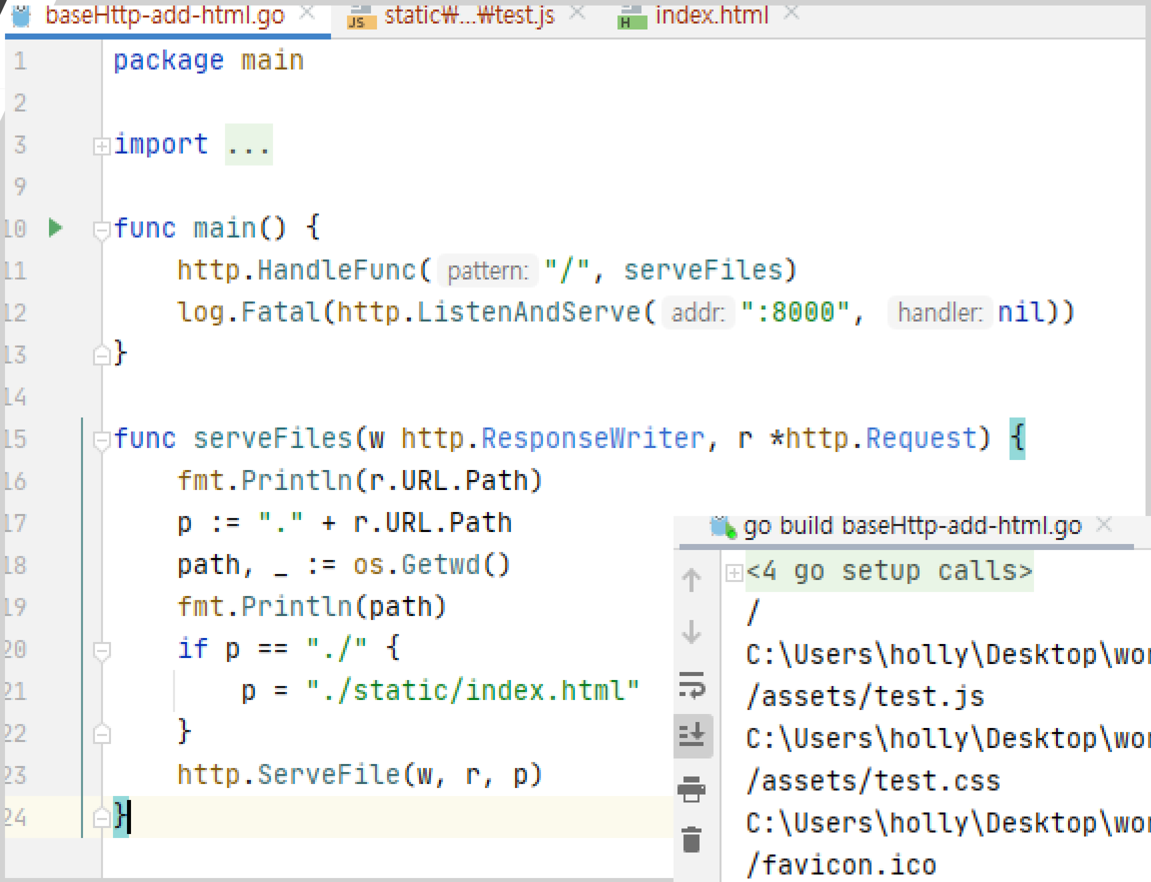
```
<4 go setup calls>
Check connect
Check connect
```



localhost:8000

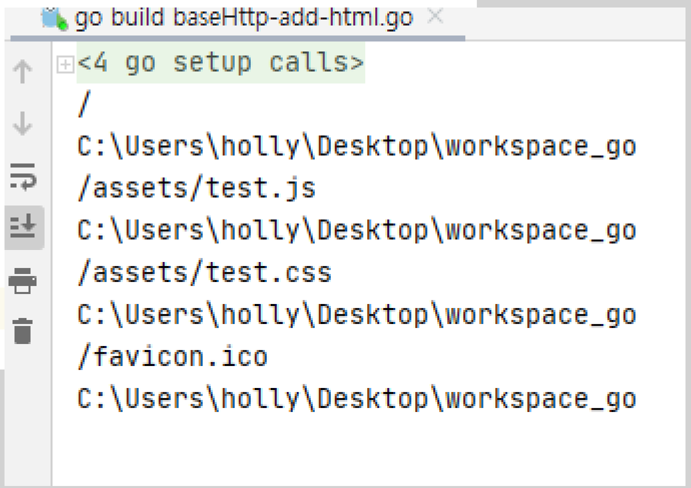
```
hi
```

HTTP



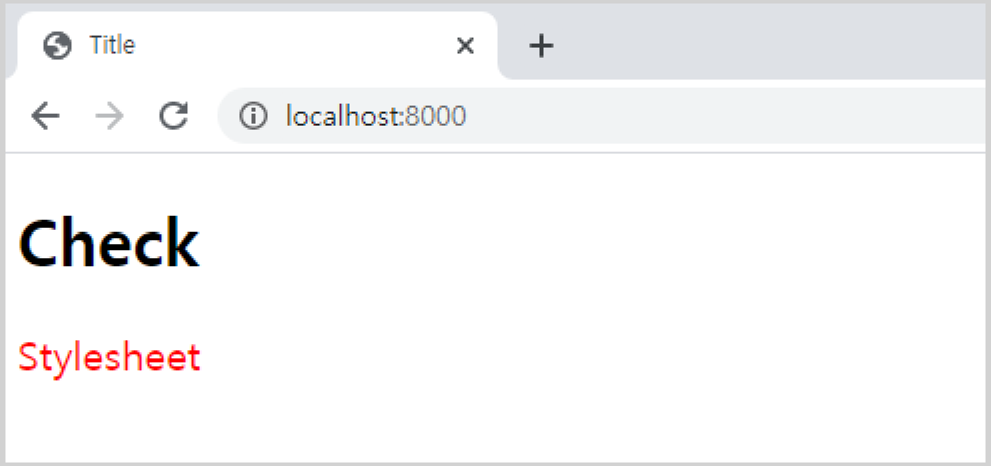
baseHttp-add-html.go × static₩...₩test.js × index.html ×

```go
package main

import ...

func main() {
	http.HandleFunc( pattern: "/", serveFiles)
	log.Fatal(http.ListenAndServe( addr: ":8000",  handler: nil))
}

func serveFiles(w http.ResponseWriter, r *http.Request) {
	fmt.Println(r.URL.Path)
	p := "." + r.URL.Path
	path, _ := os.Getwd()
	fmt.Println(path)
	if p == "./" {
		p = "./static/index.html"
	}
	http.ServeFile(w, r, p)
}
```



go build baseHttp-add-html.go ×

```
<4 go setup calls>
/
C:\Users\holly\Desktop\workspace_go
/assets/test.js
C:\Users\holly\Desktop\workspace_go
/assets/test.css
C:\Users\holly\Desktop\workspace_go
/favicon.ico
C:\Users\holly\Desktop\workspace_go
```



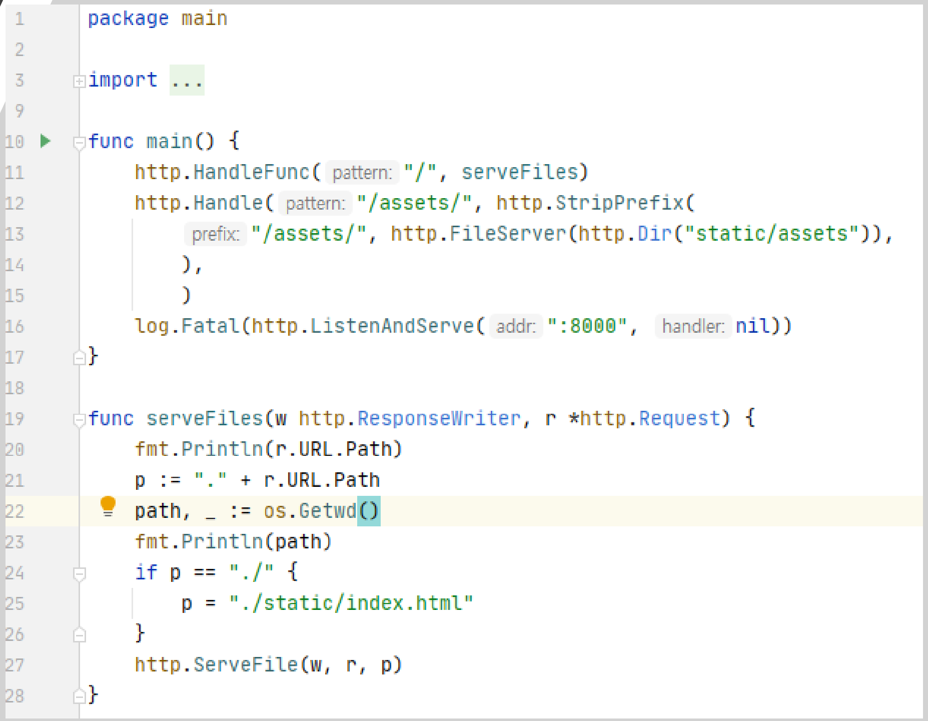
Title ×

localhost:8000

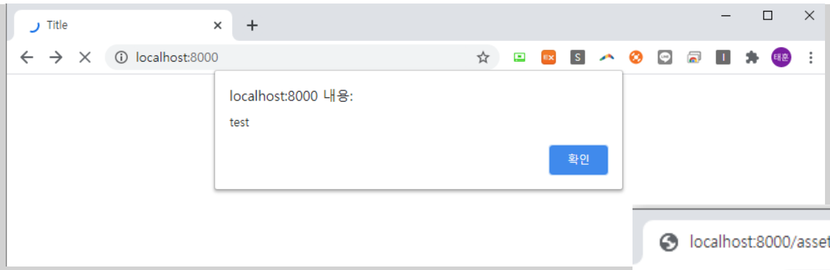
Check

Stylesheet

HTTP

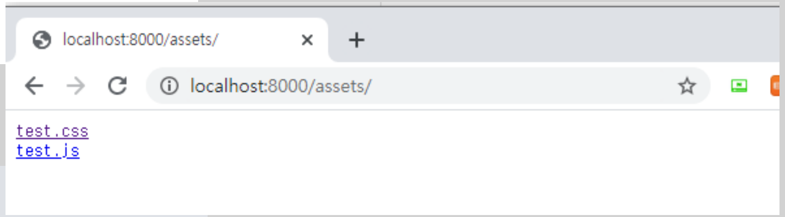


```go
package main

import ...

func main() {
	http.HandleFunc( pattern: "/", serveFiles)
	http.Handle( pattern: "/assets/", http.StripPrefix(
		 prefix: "/assets/", http.FileServer(http.Dir("static/assets")),
		),
		)
	log.Fatal(http.ListenAndServe( addr: ":8000",  handler: nil))
}

func serveFiles(w http.ResponseWriter, r *http.Request) {
	fmt.Println(r.URL.Path)
	p := "." + r.URL.Path
	path, _ := os.Getwd()
	fmt.Println(path)
	if p == "./" {
		p = "./static/index.html"
	}
	http.ServeFile(w, r, p)
}
```

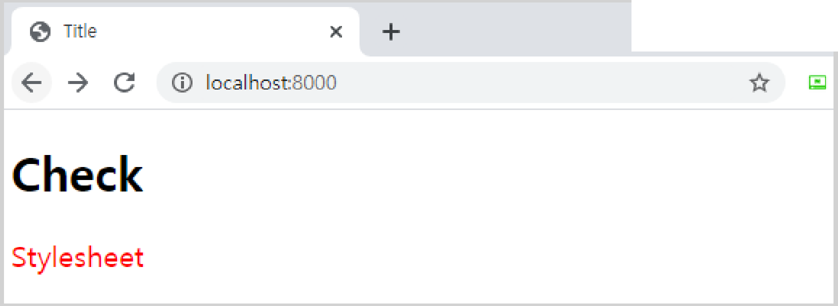


localhost:8000 내용:

test

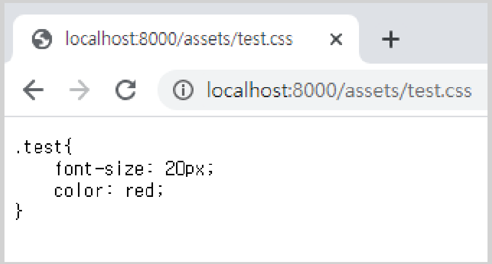


test.css
test.js



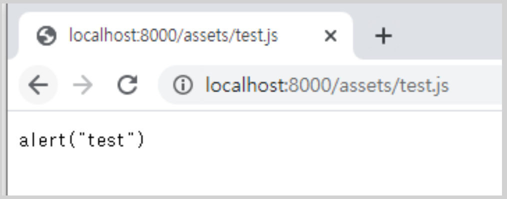
Check

Stylesheet



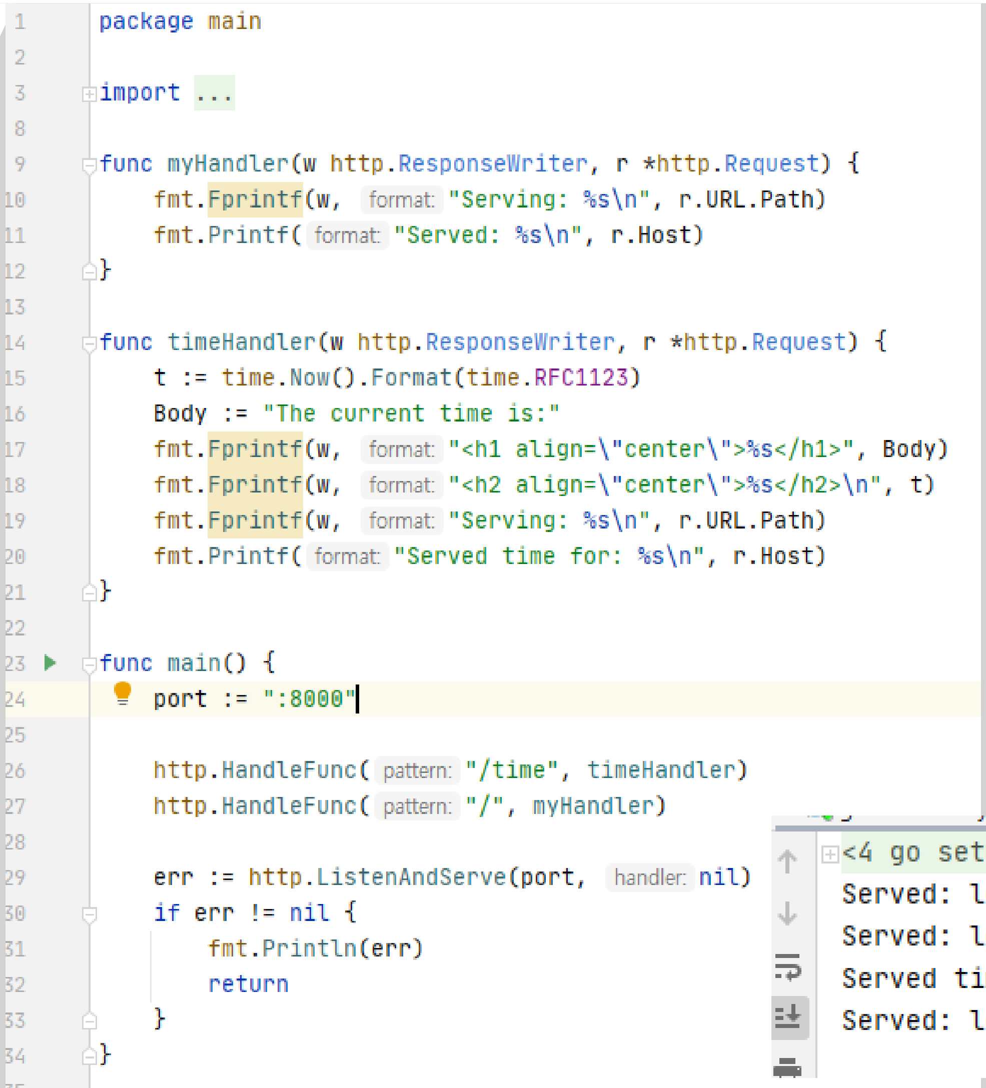
localhost:8000/assets/test.css
localhost:8000/assets/test.css

```css
.test{
    font-size: 20px;
    color: red;
}
```

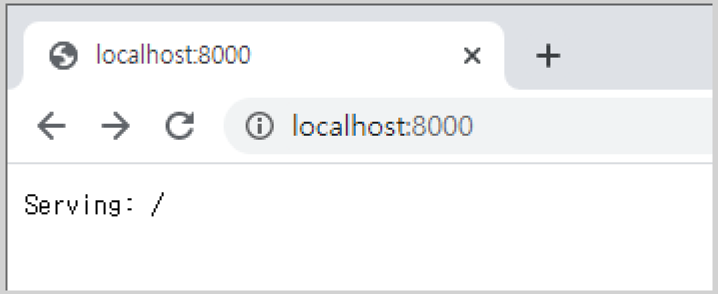


```js
alert("test")
```

HTTP

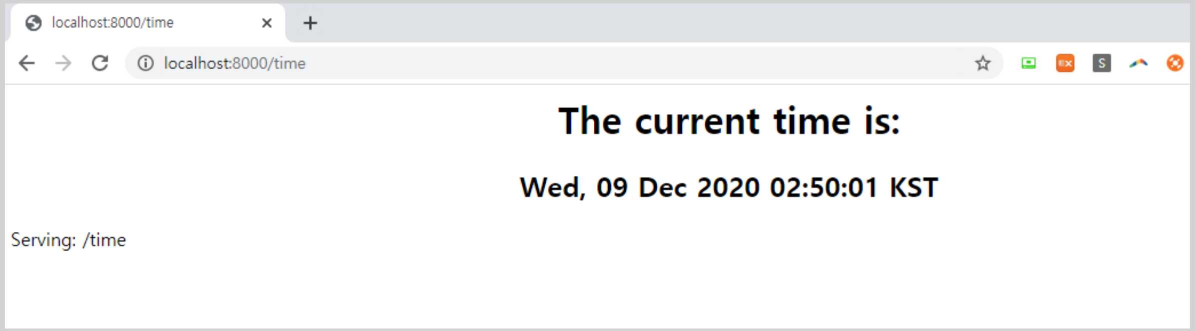


```go
package main

import ...

func myHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, format: "Serving: %s\n", r.URL.Path)
    fmt.Printf( format: "Served: %s\n", r.Host)
}

func timeHandler(w http.ResponseWriter, r *http.Request) {
    t := time.Now().Format(time.RFC1123)
    Body := "The current time is:"
    fmt.Fprintf(w, format: "<h1 align=\"center\">%s</h1>", Body)
    fmt.Fprintf(w, format: "<h2 align=\"center\">%s</h2>\n", t)
    fmt.Fprintf(w, format: "Serving: %s\n", r.URL.Path)
    fmt.Printf( format: "Served time for: %s\n", r.Host)
}

func main() {
    port := ":8000"

    http.HandleFunc( pattern: "/time", timeHandler)
    http.HandleFunc( pattern: "/", myHandler)

    err := http.ListenAndServe(port,  handler: nil)
    if err != nil {
        fmt.Println(err)
        return
    }
}
```



localhost:8000

```
Serving: /
```

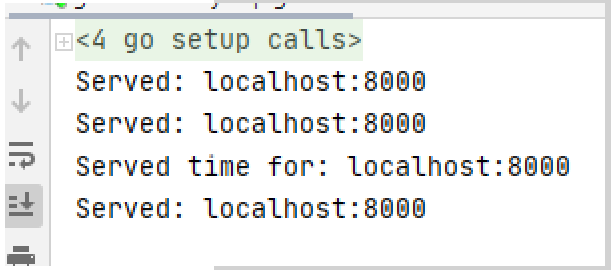


localhost:8000/time

The current time is:

Wed, 09 Dec 2020 02:50:01 KST

Serving: /time

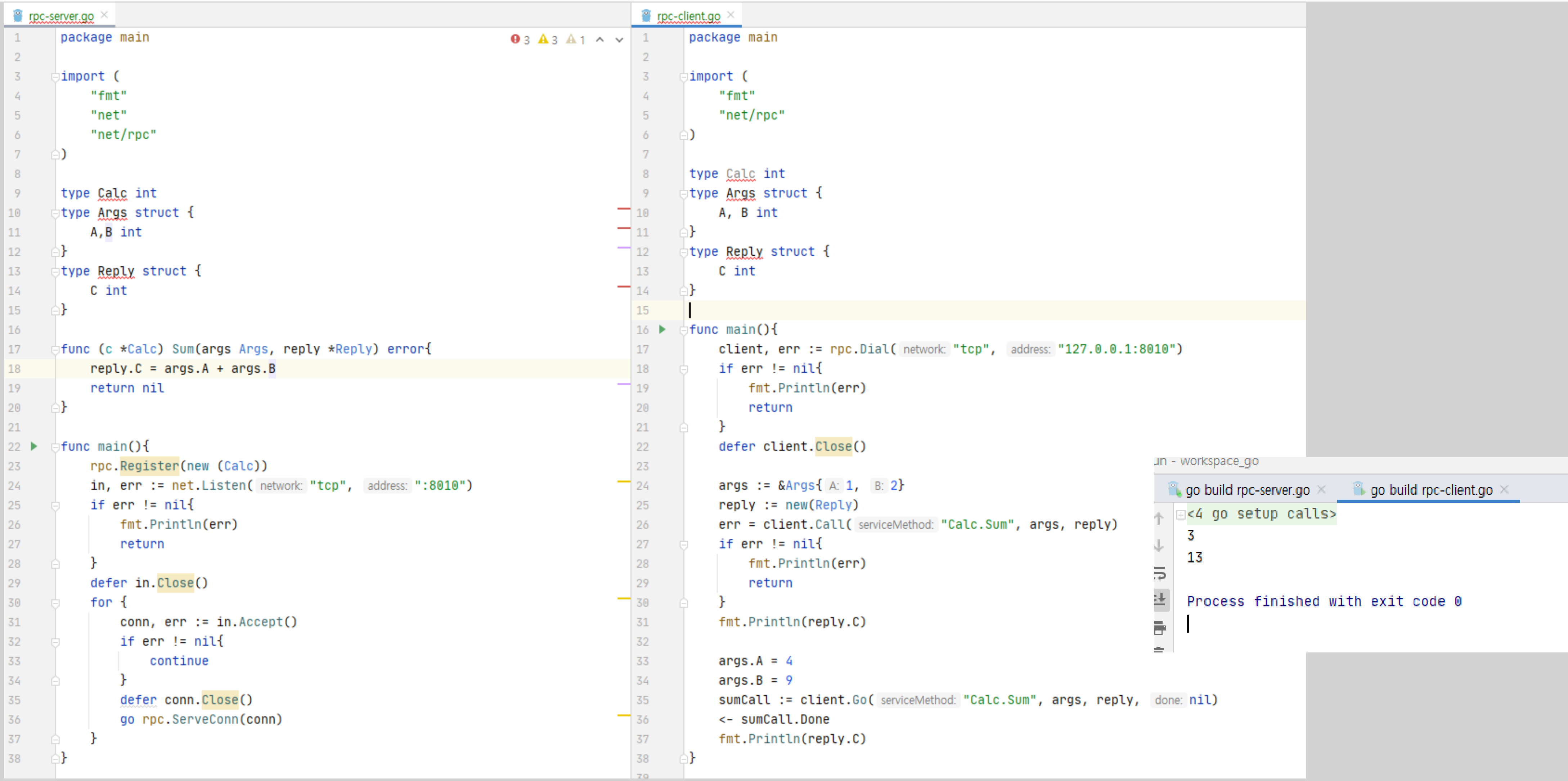


```
<4 go setup calls>
Served: localhost:8000
Served: localhost:8000
Served time for: localhost:8000
Served: localhost:8000
```

RPC

- RPC (Remote Procedure Call)
  - 별도의 원격 제어를 위한 코딩 없이 다른 주소 공간에서 리모트의 함수나 프로시저를 실행 할 수 있게 해주는 프로세스간 통신
  - 위치에 상관없이 RPC를 통해 서버에 있는 함수를 사용할 수 있음
  - IPC (Inter-Process Communication)의 한 종류
- 특징
  - OS, 언어에 영향을 받지 않고 사용할 수 있음
  - 분산 환경에서 많이 사용됨

RPC



rpc-server.go

```go
package main

import (
	"fmt"
	"net"
	"net/rpc"
)

type Calc int
type Args struct {
	A,B int
}
type Reply struct {
	C int
}

func (c *Calc) Sum(args Args, reply *Reply) error{
	reply.C = args.A + args.B
	return nil
}

func main(){
	rpc.Register(new (Calc))
	in, err := net.Listen( network: "tcp",  address: ":8010")
	if err != nil{
		fmt.Println(err)
		return
	}
	defer in.Close()
	for {
		conn, err := in.Accept()
		if err != nil{
			continue
		}
		defer conn.Close()
		go rpc.ServeConn(conn)
	}
}
```

rpc-client.go

```go
package main

import (
	"fmt"
	"net/rpc"
)

type Calc int
type Args struct {
	A, B int
}
type Reply struct {
	C int
}

func main(){
	client, err := rpc.Dial( network: "tcp",  address: "127.0.0.1:8010")
	if err != nil{
		fmt.Println(err)
		return
	}
	defer client.Close()

	args := &Args{ A: 1,  B: 2}
	reply := new(Reply)
	err = client.Call( serviceMethod: "Calc.Sum", args, reply)
	if err != nil{
		fmt.Println(err)
		return
	}
	fmt.Println(reply.C)

	args.A = 4
	args.B = 9
	sumCall := client.Go( serviceMethod: "Calc.Sum", args, reply,  done: nil)
	<- sumCall.Done
	fmt.Println(reply.C)
}
```

un - workspace_go

go build rpc-server.go | go build rpc-client.go

```
<4 go setup calls>
3
13

Process finished with exit code 0
```