MAVENIR™
Trust the Future

Golang:Net package

24/01/2023

Golang: Important packages

- Crypto
- Compress
- Database
- Debug
- Encoding
- Syscall
- testing
- Net
- context

Golang: net package..

Working with net package

- net provides a portable interface for network I/O
  - Supports tcp/udp name resolution
  - Supports Unix domain sockets
- Package provides low level network primitives
- However, Go app developers need very basic interfaces
  - Dial
  - Listen
  - Accept

Golang: net package

Dial

- The Dial function connects to a server
  - func Dial(network, address string) (Conn, error) Conn
  - func DialTimeout(network, address string, timeout time.Duration) (Conn, error) Conn

- Dial, DialTimeOut can work with both tcp and udp

Golang: net package

Dial

```
conn, err := net.Dial("tcp", "127.0.0.1:1234")
if err != nil {
        // handle error
}

fmt.Fprintf(conn, "GET / HTTP/1.0\r\n\r\n")
status, err := bufio.NewReader(conn).ReadString('\n')
// ...
```

Golang: net package

Listen

- Listnen function listen for a network connection
  - func Listen(network, address string) (Listener, error)
  - Connects to address string and network protocol

Golang: net package

Listen

```
ln, err := net.Listen("tcp", ":8080")
if err != nil {
        // handle error
}
for {
        conn, err := ln.Accept()
        if err != nil {
                // handle error
        }
        go handleConnection(conn)
}
```

Golang: net package

Listen

- Listnen function listen for a network connection
  - func Listen(network, address string) (Listener, error)
  - Connects to address string and network protocol

Golang: net/http package

http

>Http is an application layer protocol and works in client and server mode

>Http server is a program running on a machine

>It listens and responds to HTTP requests on a specific ip and port

> Http works in request and response

Golang: net/http package

http

>Http server components

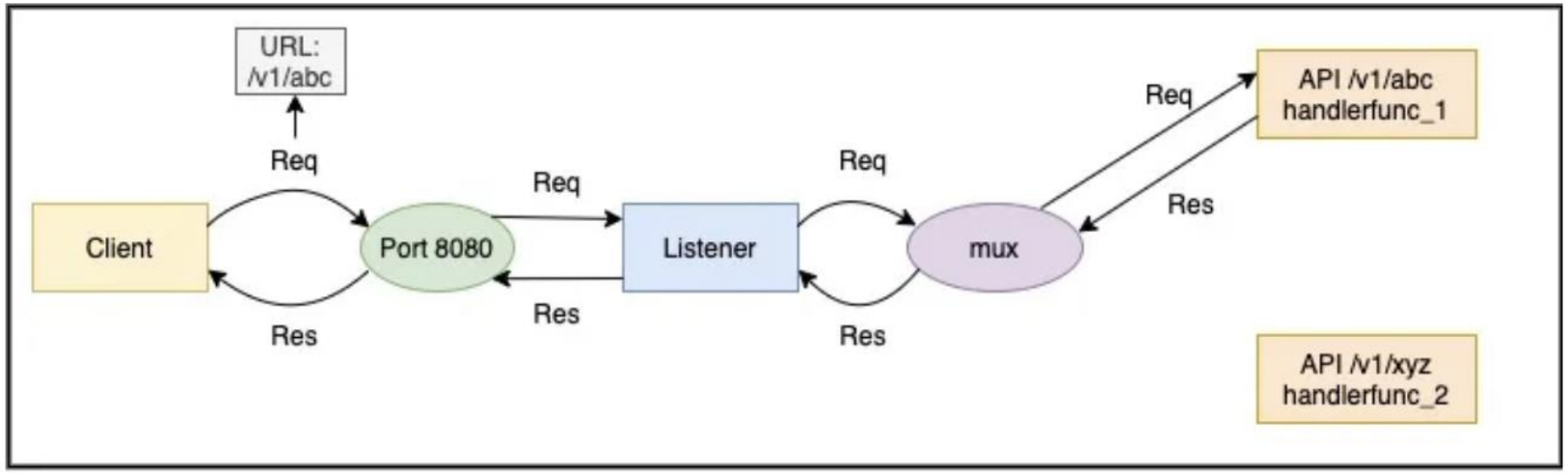
- > **Request –** it defines the request parameters i.e, Method, Api Signature, request headers, body, query params etc
- > **Response** – defines the response parameters i.e, Status code, response body, headers
- > Each API signature corresponds to a handler
- > Mux is router
- > **Listener –** It runs on the machine, which listens to a particular port.
- – On receiviing request on that port it forwards the request to the **mux**

MAVENIR

Golang: net/http package

http

> /v1/abc and handlerfunc_1 and /v1/xyz and handlerfunc_2



Golang: net/http package

Http request

- A request is represented by the **Request** Struct (https://golang.org/pkg/net/http/#Request)
- It contains the
  - request method,
  - Api Signature,
  - request headers,
  - body,
  - query params

Golang: net/http package

Http response

- A response is represented by the **ResponseWriter** Interface (https://golang.org/pkg/net/http/#ResponseWriter)
- ResponseWriter interface is used by an HTTP handler to construct an HTTP response
- It provides three functions to set the response parameters
  - Header – For writing response header
  - Write([]byte) – For writing response body
  - WriteHeader(statusCode int) – For writing the http status code

Golang: net/http package

API Signature and Handler

- API signature and its handler are paired
- Handler is called by the mux when it receives an API call matching the API signature
- A golang handler can be either a **function** or a **type**
  - type Handler interface {
  - ServeHTTP(ResponseWriter, *Request)
  - }
  - func(ResponseWriter, *Request)

Golang: net/http package

Mux

- mux route request to the registered handler based upon API signature
- If the signature and its handler is not registered with the mux, it raises a 404
- Go provides a default mux built in the language
  - There are mux available from third party as well
    - Gorilla mux and fiber
- Create Mux
  - mux := http.NewServeMux()
- mux.HandleFunc(pattern, handlerFunc)
- mux.Handle(pattern, handler)

Golang: net/http package

Mux

- mux route request to the registered handler based upon API signature
- If the signature and its handler is not registered with the mux, it raises a 404
- Go provides a default mux built in the language
  - There are mux available from third party as well
    - Gorilla mux and fiber
- Create Mux
  - mux := http.NewServeMux()
- mux.HandleFunc(pattern, handlerFunc)
- mux.Handle(pattern, handler)