

Go

- [return](#)
- [for loop](#)
- [Go 面试题](#)
- [Go tcp client 面试题](#)
- [defer](#)
- [面试题](#)

return

```
client, err := net.DialTimeout(netScheme, host, time.Duration(w.Timeout))

if err != nil {
    fmt.Printf("Error: %v\n", err)
} else {
    fmt.Printf("Connected to %v\n", client.RemoteAddr())
}
```

👉 👉,

```
// Java👉 👉 👉 👉 👉 👉 👉

// 👉 👉 👉 👉
class ConnectionResult {
    public Socket client;
    public Exception error;

    public ConnectionResult(Socket client, Exception error) {
        this.client = client;
        this.error = error;
    }
}

public ConnectionResult connect(String host, int timeout) {
    try {
        Socket client = new Socket();
        client.connect(new InetSocketAddress(host, timeout));
        return new ConnectionResult(client, null);
    } catch (IOException e) {
        return new ConnectionResult(null, e);
    }
}

// 👉 👉
ConnectionResult result = connect("example.com", 1000);
if (result.error != null) {
```

```
    System.out.println("Error: " + result.error.getMessage());  
} else {  
    System.out.println("Connected to " + result.client.getRemoteSocketAddress());  
}
```

for loop

go

- `_` 變數名稱 變數 變數 變數 變數.
- 變數 變數 變數 變數 變數
- `:=` 變數 變數 變數 變數

```
// w.hosts 變數 變數 變數
for _, host := range w.hosts {
    // 變數 host 變數 w.hosts 變數 變數.
    fmt.Println("Connecting to:", host)
}
```

java

- 變數 變數 for-each 變數 變數 變數 變數

```
// hosts 變數 變數 變數
List<String> hosts = Arrays.asList("host1", "host2", "host3");
for (String host : hosts) {
    // 變數 host 變數 hosts 變數 變數.
    System.out.println("Connecting to: " + host);
}
```

Go 学习

<https://go.dev/doc/tutorial>

Getting Started

- root 下 go.mod 文件
 - module 名称 与 仓库名称 一致 且 仓库名称 在 github/example
 - go mod init github/example
- 网络代理
 - 代理 地址 是什么?
 - Go Proxy
 - proxy.golang.org
 - 代理 地址 是什么
 - 代理 地址 是 \$GOPATH/pkg/mod
 - 代理 地址 是 `import "github.com/user/repo/package"`

Create a Go module

```
// 文件 1 和 文件 2
var message string
message = fmt.Sprintf("Hi, %v. Welcome!", name)

// 文件 1 和 文件 2
message := fmt.Sprintf("Hi, %v. Welcome!", name)
```

Call your code from another module

```
<home>/
|-- greetings/ (文件 1)
```

```
|-- hello/ (package main)
```

- `import "example.com/greetings"`
- `production` `go mod edit -replace example.com/greetings=../greetings` `go.mod`

```
module example.com/hello
```

```
go 1.22.5
```

```
replace example.com/greetings = >../greetings
```

- `go mod tidy` `../greetings`

Return and handle an error

- Go `2` `errors`
- `"errors"`

```
// name nil? Go string nil? "" string. , string
```

```
// string? nil? : string
```

..

```
1. bool
```

```
• bool: false
```

```
• string: "" ( )
```

```
• int, int8, int16, int32, int64, uint, uint8, uint16, uint32, uint64, uintptr: 0
```

```
• float32, float64: 0.0
```

```
• complex64, complex128: 0+0i
```

```
2. nil
```

```
• : []
```

```
• : nil
```

```
• : nil
```

```
• : nil
```

```
• : nil
```

•: nil

•: nil

•: 1 2 3 4 5 6 7 8

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var b bool
```

```
    var s string
```

```
    var i int
```

```
    var f float64
```

```
    var c complex128
```

```
    var a [3]int
```

```
    var p *int
```

```
    var sl []int
```

```
    var m map[string]int
```

```
    var ch chan int
```

```
    var fn func() int
```

```
    var iface interface{}
```

```
    fmt.Printf("bool: %v\n", b)
```

```
    fmt.Printf("string: %v\n", s)
```

```
    fmt.Printf("int: %v\n", i)
```

```
    fmt.Printf("float64: %v\n", f)
```

```
    fmt.Printf("complex128: %v\n", c)
```

```
    fmt.Printf("array: %v\n", a)
```

```
    fmt.Printf("pointer: %v\n", p)
```

```
    fmt.Printf("slice: %v\n", sl)
```

```
    fmt.Printf("map: %v\n", m)
```

```
    fmt.Printf("channel: %v\n", ch)
```

```
    fmt.Printf("function: %v\n", fn)
```

```
    fmt.Printf("interface: %v\n", iface)
```

```
}
```

bool: false

string:

int: 0

```
float64: 0
complex128: (0+0i)
array: [0 0 0]
pointer: <nil>
slice: []
map: map[]
channel: <nil>
function: <nil>
interface: <nil>
```

Return a random greeting

--	--

- 
- 

--	--	--	--

- 变量名 变量名 变量名 变量名 变量名 变量名(exported) 变量名
- 变量名, 变量名 变量名 变量名(unexported) 变量名
- 变量名 变量名 变量名, 变量名 变量名 go 变量名 randomFormat() 变量名

--	--	--	--	--	--	--

```
var arr [3]int // 3 int
arr = [3]int{1, 2, 3} // 
fmt.Println(arr) // : [1 2 3]

s := []int{1, 2, 3} // int 
s = append(s, 4) // 
fmt.Println(s) // : [1 2 3 4]
```


Return greetings for multiple people

📄

- 📄 map 📄

```
// 📄: Go 📄📄📄 ma messages := make(map[string]string)
```

```
// 📄 📄📄  
messages := make(map[string]string)  
  
// 📄📄📄 📄📄  
numbers := make([]int, 5) // 📄📄 5📄 📄📄📄 📄  
numbers := make([]int, 5, 10) // 📄📄 5📄 📄📄 10📄 📄📄📄 📄  
  
// 📄 📄 (📄📄 📄📄 📄📄 📄)  
ch := make(chan int)
```

Add a test

- 📄 [_test 📄📄📄 📄
- "testing" import 📄.
- 📄 📄📄 t.Fatalf() 📄

Go tcp client ☐☐ ☐☐

```
func SendCloudWatch(cloudwatch model.AwsRdsCloudWatch) {  
    // ... ☐☐  
  
    ☐conn, err := net.Dial("tcp", fmt.Sprintf("%s:%s", host, portStr))  
    ☐if err != nil {  
        ☐☐logger.Error("Failed to connect to TCP server:", err)  
        ☐☐return  
    ☐}  
    ☐defer conn.Close()  
  
    ☐if err := send(conn, data.ToByteArray()); err != nil {  
        ☐☐logger.Error("Failed to send data:", err)  
    ☐}  
}
```

“ [defer](#) ☐☐☐☐ ☐☐.

```
func send(conn net.Conn, data []byte) error {  
    ☐const writeTimeout = 5 * time.Second  
    ☐totalBytesSent := 0  
  
    ☐for totalBytesSent < len(data) {  
        ☐☐writeDeadline := time.Now().Add(writeTimeout)  
        ☐☐if err := conn.SetWriteDeadline(writeDeadline); err != nil {  
            ☐☐☐return fmt.Errorf("failed to set write deadline: %v", err)  
        ☐☐}  
        ☐☐bytesSent, err := conn.Write(data[totalBytesSent:])  
        ☐☐if err != nil {  
            ☐☐☐return fmt.Errorf("failed to send data: %v", err)  
        ☐☐}  
        ☐☐totalBytesSent += bytesSent  
    ☐}  
    ☐return nil
```


defer

Go[defer

- [illegible]

```
file, err := os.Open("example.txt")
if err != nil {
    // ...
}
defer file.Close() // ...
// ...
```

- ☐

--	--	--	--

--	--

 :

```
conn, err := net.Dial("tcp", "example.com:80")
if err != nil {
    // ...
}
defer conn.Close() // ...
// ...
```

- | | |
|--|--|
| | |
|--|--|

--	--

--	--

 :

--	--

--	--	--

--	--	--	--

--	--	--	--

--	--

--	--

--

--	--

--	--

--	--

.

-

- [defer] □ □, □ □ □ □ □ □ □ □:

```
defer fmt.Println("First")
defer fmt.Println("Second")
defer fmt.Println("Third")
// 03: Third, Second, First (03 03 03)
```

- **Go**
 - Go `defer`: `defer` 키워드로 리소스를 닫을 수 있다.
 - Java `try-with-resources`: `try` 블록 안에서 `AutoCloseable` 인터페이스를 구현한 객체를 선언하면, try 블록이 끝나면 자동으로 리소스를 닫아준다.
- **Python**
 - Go `defer`: `with` 문으로 리소스를 관리할 수 있다.
 - Java `try-with-resources`: `AutoCloseable` 인터페이스를 구현한 객체를 선언하면, try 블록이 끝나면 자동으로 리소스를 닫아준다.
- **Java**
 - Go: `defer` 키워드로 리소스를 닫을 수 있다.
 - Java: `try-with-resources` 키워드로 리소스를 닫을 수 있다.



• Go 예제

```
package main

import (
    "fmt"
    "os"
)

func main() {
    file, err := os.Open("example.txt")
    if err != nil {
        fmt.Println("Error:", err)
        return
    }
    defer file.Close() // 리소스 닫기

    // 리드
}
```

• Java 예제

```
import java.io.FileInputStream;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try (FileInputStream file = new FileInputStream("example.txt")) {
            // 리드
        } catch (IOException e) {
            // 예외 처리
        }
    }
}
```

```
        System.out.println("Error: " + e.getMessage());
    }

    // try to read file
}

}
```



```
cd path/to/go-project
```

```
# Go [ ] [ ] [ ] [ ] Amazon Linux 2023 [ ] ARM64 [ ] [ ]
```

```
GOOS=linux GOARCH=arm64 go build -o bootstrap main.go
```

```
# [ ] [ ] [ ] [ ]
```

```
chmod +x bootstrap
```

```
# [ ] [ ] [ ] [ ]
```

```
zip function.zip bootstrap
```

```
GOOS=linux GOARCH=amd64 go build -o main .
```