

Go 程式設計課 04

Go 與作業系統



大綱

- ❖ 後端與作業系統
- ❖ Linux 作業系統
- ❖ Golang 三大組件
- ❖ Golang GC
- ❖ Golang runtime scheduler
- ❖ (D)DOS Attack By Golang

runtime
compiler → obj file
linker → exe



後端與作業系統

後端工程師

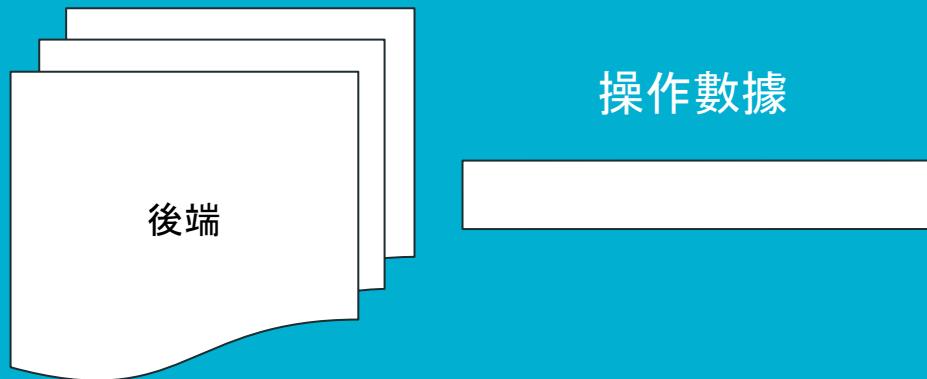
優化+串接底層API(DB)

提供介面, 轉成內部介面

儲存數據

數據介面

1. 網頁後端
2. APP 後端
3. 遊戲後端
4. IOT 後端
5.



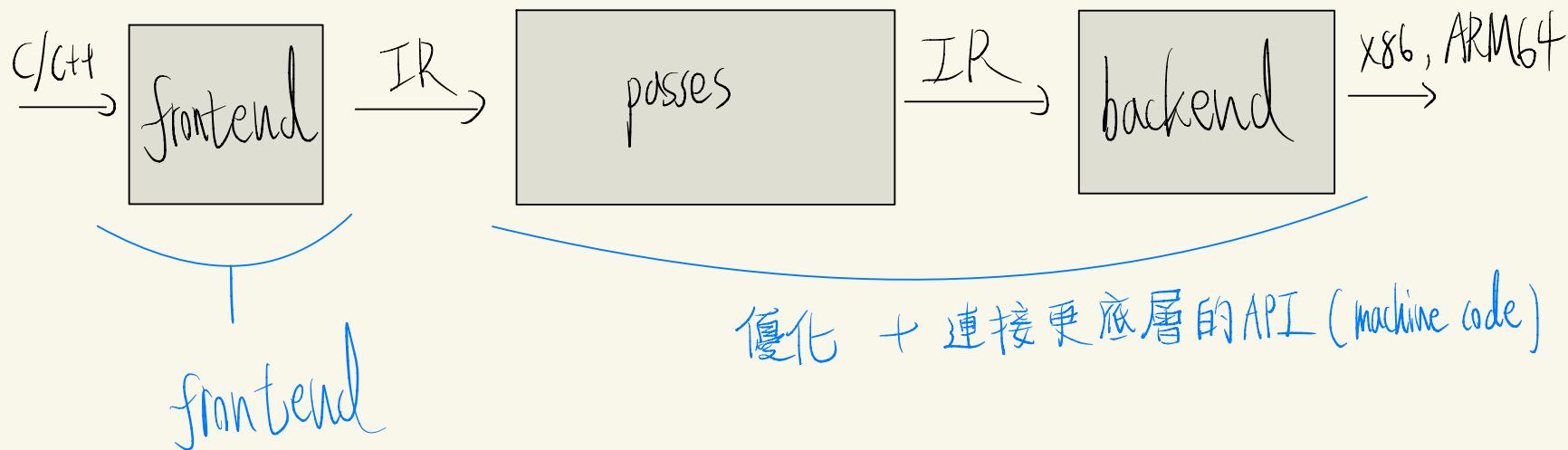
開發和維護API(Application Programming Interface),讓前端和其他服務能夠與後端系統進行溝通。



compiler backend

calculate ($0.5 \times 1.2 \times \text{distance}$)
↓ ↓ ↓
新 customer 長距 基數

優化 IR



多種語言 → 中間語言(介面)

SSD

NVMe

SATA

OS driver



LB + GC + err handle



nand API



公定介面 → 內部FTL介面

frontend

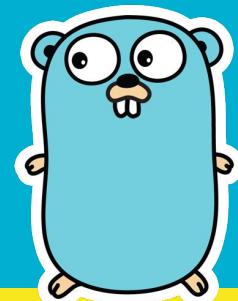
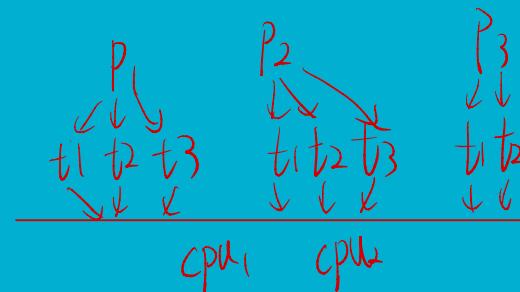
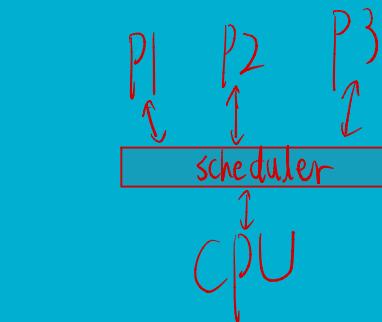
優化對 NAND 的操作手段

+
串接對 NAND 顆粒的 API

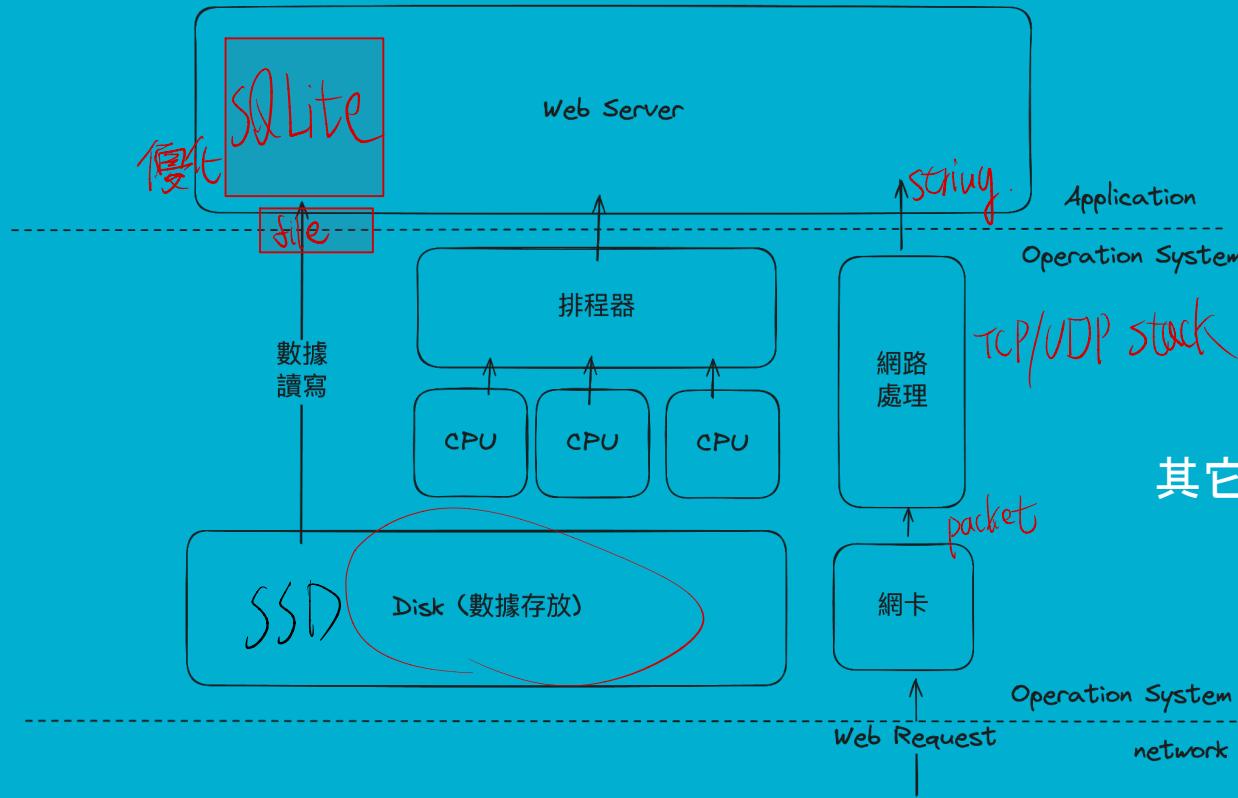
作業系統

管理電腦硬體、軟體資源並為使用者和應用程式提供服務的系統軟體。
它是電腦系統的核心,負責控制和協調各個元件的運作。

- ❖ 資源管理: IO / CPU / Mem
- ❖ 程序管理: Schedule / thread / process
- ❖ 檔案管理: file operation *run program*
- ❖ windows / windows server
- ❖ mac
- ❖ Linux (ubuntu, centOS, Debian,)



網頁伺服器與作業系統



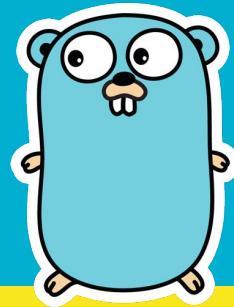
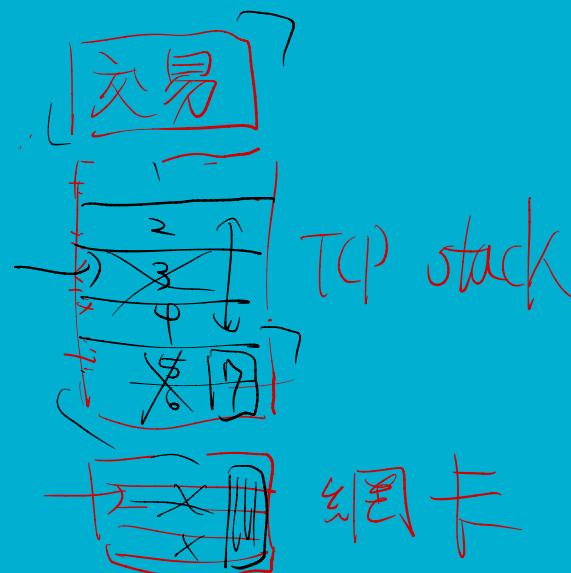
Linux 作業系統

何為 Linux ?

- <https://zh.wikipedia.org/zh-tw/Linux>
- <https://github.com/torvalds/linux>

一個開源的作業系統

1. 便宜
2. 可訂製
3. 輕量化
4. 穩穩定



Linux 發行版是什麼？

- <https://zh.wikipedia.org/zh-tw/Linux%E5%8F%91%E8%A1%8C%E7%89%88>
- Ubuntu (易用, GUI 好)
- Debian (輕量化, 穩定)
- CentOS (穩定, 難用)
- Red Hat Enterprise (附帶更多的 OS 安全機制)



為何 golang 後端喜歡使用 Linux ? (1)

- 開放原始碼: Linux 是開放原始碼的作業系統,這意味著開發人員可以自由地查看、修改和分發其原始碼。這種開放性使開發人員能夠深入了解系統的內部運作,並根據自己的需求進行定制。
- 穩定性和可靠性: Linux 以其穩定性和可靠性而聞名。與其他作業系統相比,Linux 系統崩潰和故障的頻率較低。這對於需要持續運行的後端服務非常重要。
- 高效的命令列介面: Linux 提供了功能強大的命令列介面(CLI),允許開發人員快速執行複雜的任務,如檔案操作、程序管理和遠端管理。許多後端開發工具和框架都有很好的 CLI 支援。



為何 golang 後端喜歡使用 Linux ? (2)

- 軟體套件管理: Linux 發行版通常帶有套件管理器,如 apt 和 yum,使開發人員可以輕鬆安裝、更新和管理各種軟體套件。這簡化了開發環境的設置和維護。
- 伺服器軟體支援: 許多常用的後端服務器軟體,如 Apache、Nginx、MySQL 和 PostgreSQL,都原生支援 Linux。這使得在 Linux 上部署和管理後端服務變得更容易。
- 雲端和容器支援: Linux 是雲端計算和容器化技術的核心。許多雲端平台,如 AWS 和 Google Cloud,都使用 Linux 作為其基礎設施。Docker 和 Kubernetes 等流行的容器化工具也依賴於 Linux 的特性。
- 社群支援: Linux 擁有龐大而活躍的開發者社群。這意味著開發人員可以輕鬆找到問題的答案、分享經驗並參與開源專案。



記憶體管理

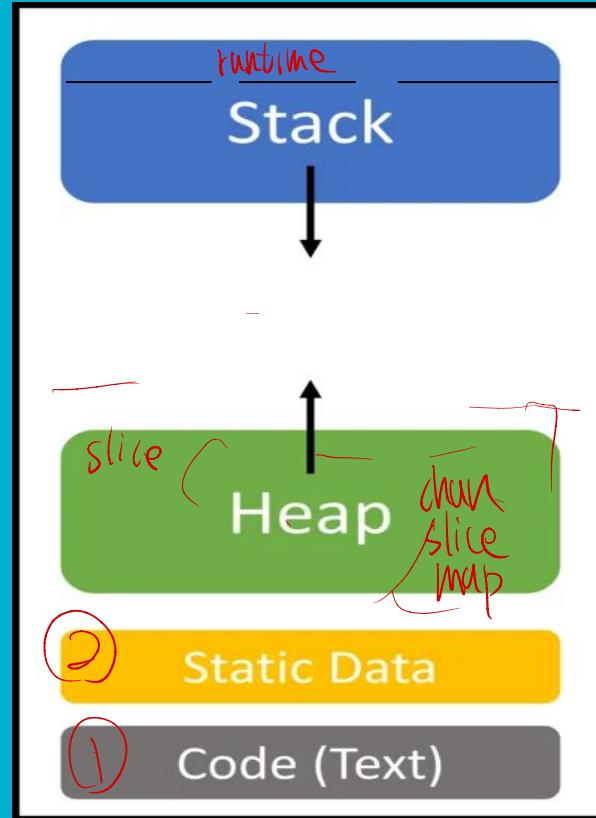
- ❖ Stack
- ❖ Heap
- ❖ Static(global) Data
- ❖ Code
- ❖ Address Space

program → Address Space → process → main()

↑
OS

↓ ↓ ↓
t1 t2 t3

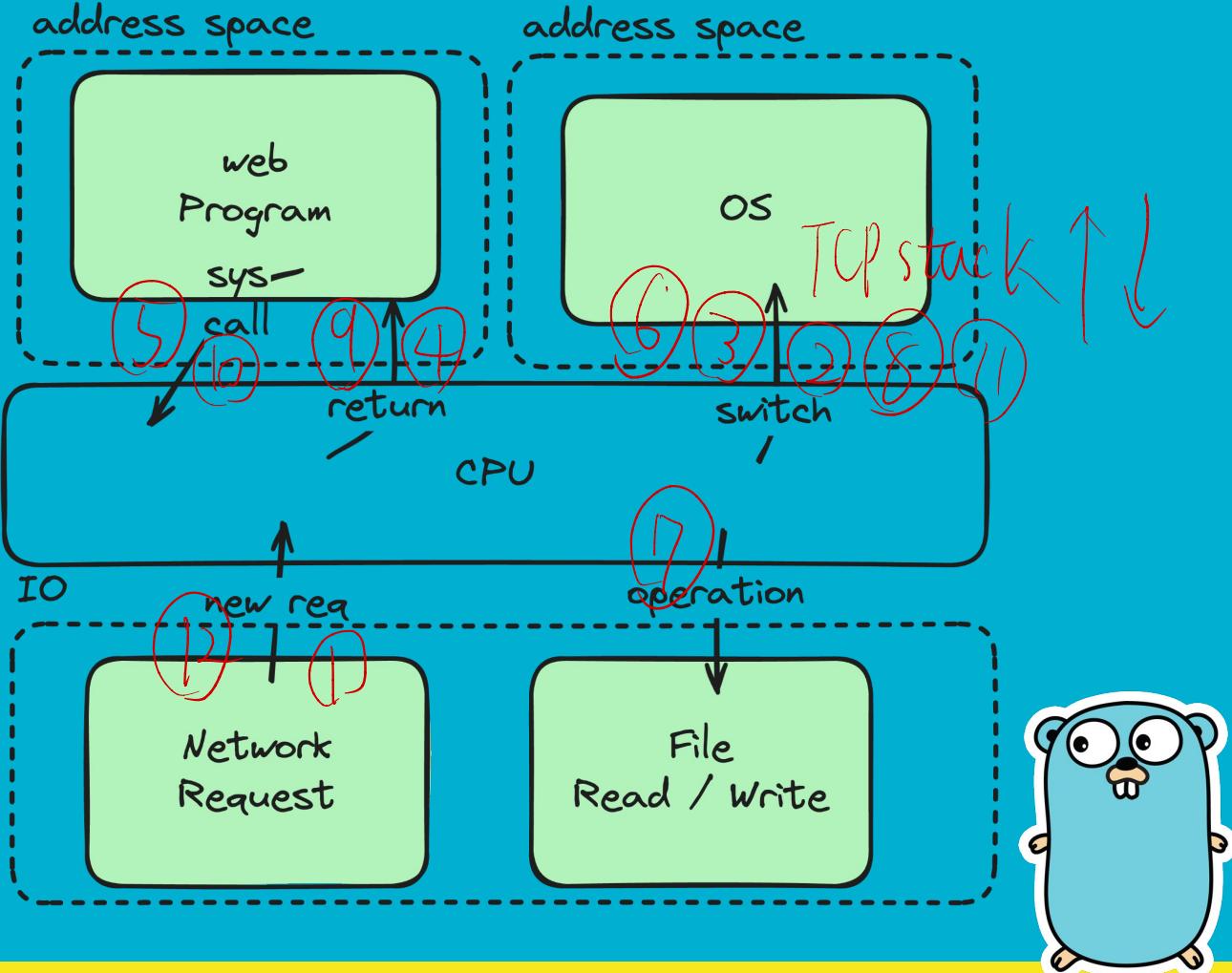
① address space for process



IO 管理

- ❖ network IO
- ❖ disk IO
- ❖ context switch
- ❖ system call
- ❖ IO 是最慢的！

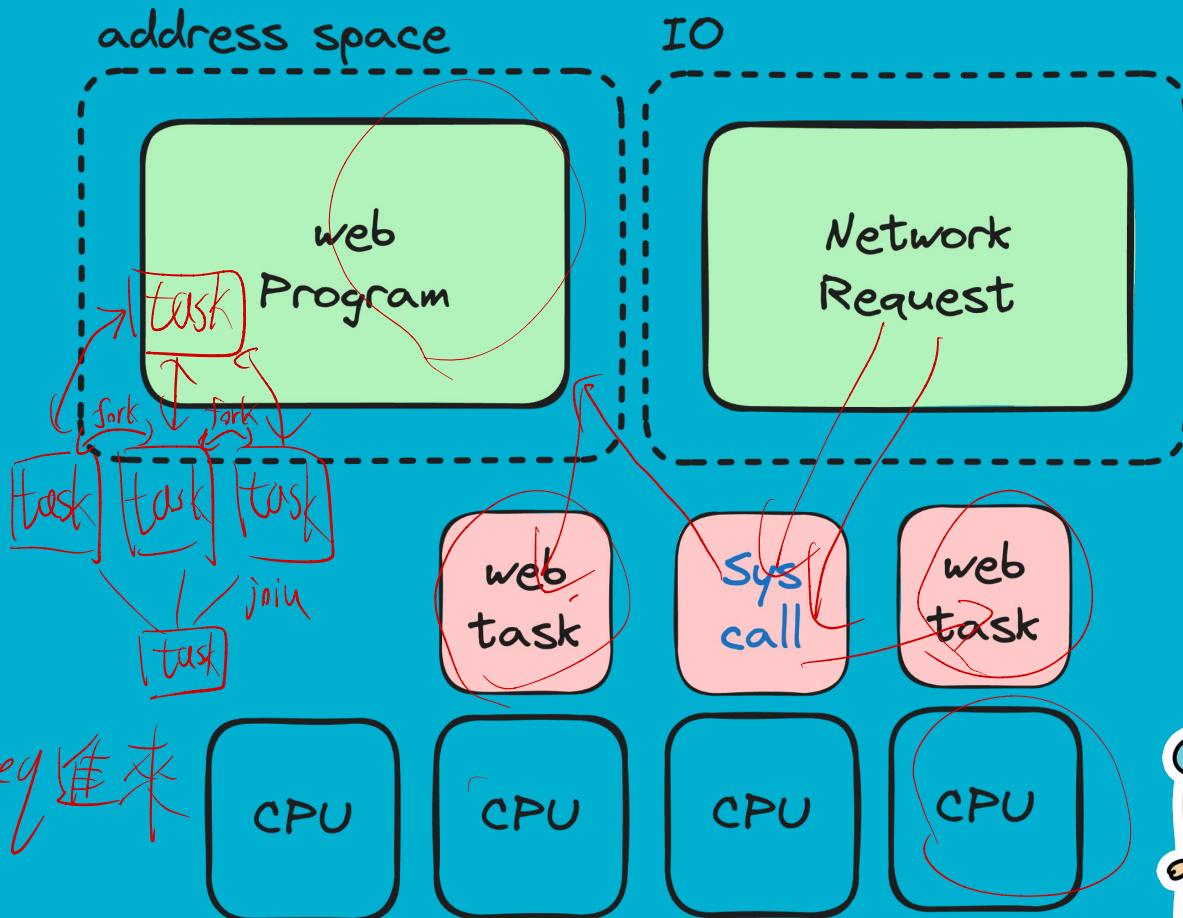
1. 收到外部 API req
2. DB 調用



CPU 管理

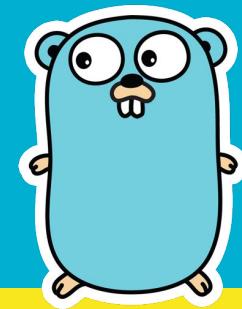
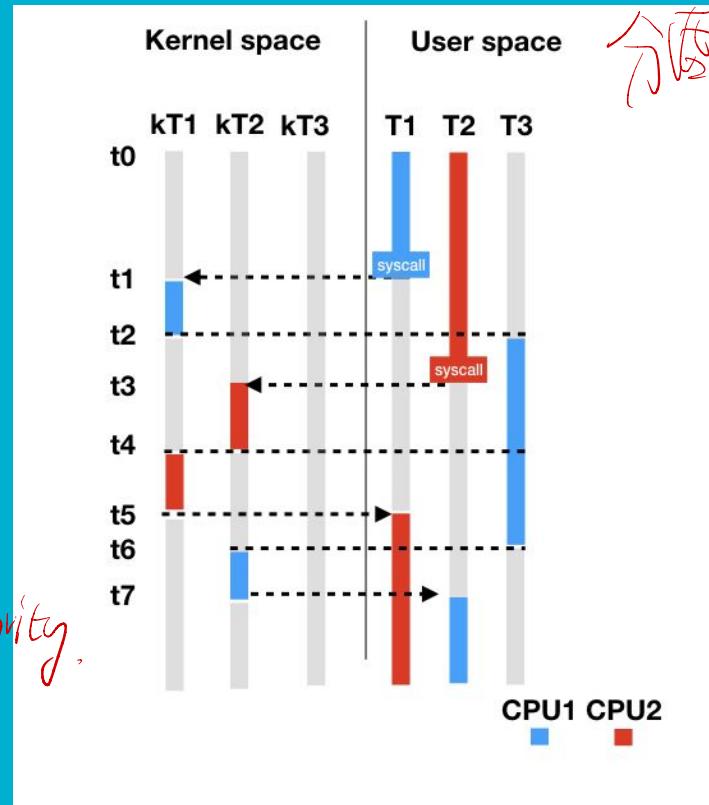
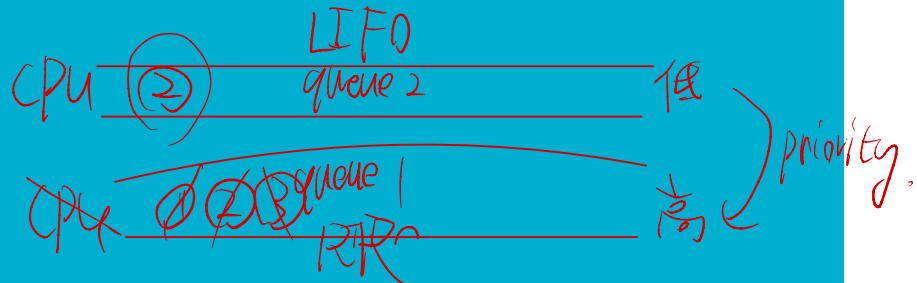
- ❖ CPU
- ❖ Task
 - ~~thread~~
 - ~~process~~
- ❖ Program
- ❖ System Call

同時2个http req進來



CPU Scheduler

- ❖ RR \rightarrow 1 : $\frac{1}{n}$
- ❖ FIFO
- ❖ LIFO
- ❖ Multi-Layer-Queue



Golang 三大組件

三大組件

- ❖ Compiler
- ❖ Linker
- ❖ Runtime



compiler

Go 的 compiler 負責將 Go 原始碼編譯為高效的機器碼。

- 語法分析: 有沒有無法理解的語法
- 型別檢查: 確認沒有型別錯誤
- 程式碼最佳化: 優化冗余程式碼
- 機器碼生成: 轉換成特定 CPU 指令集的機器碼

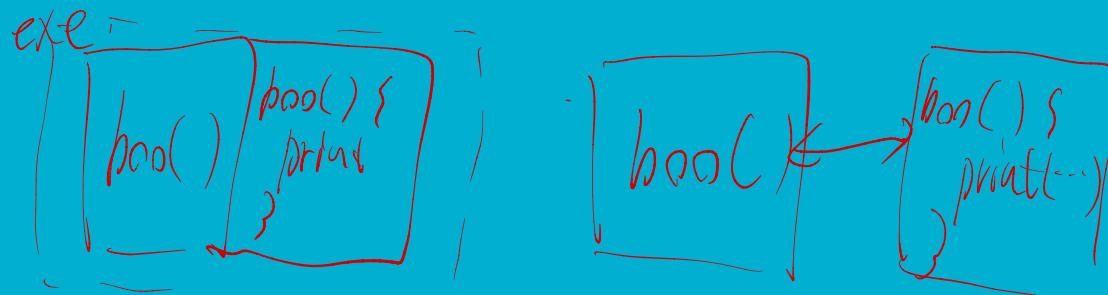
Go 的編譯器以編譯速度快而聞名,這得益於它的簡單語法和模組化設計。此外, Go 的編譯器還支援跨平台編譯,允許在一個平台上編譯出在另一個平台上運行的二進位檔案。

```
GOOS=linux GOARCH=ppc64 go build -o binaryname
```

```
GOOS=<OS> GOARCH=<CPU_ISA> go build -o binaryname
```



Linker (1)



linker 負責將編譯器生成的目的檔連結為最終的可執行檔案或套件。

- 符號解析: 理解外部模組 (函數, 變數)
- 重定位: 調用不同檔案, 設定相對地址
- 套件初始化: 設置 `func init()` 的執行順序



Go 的連結器設計得很精簡,因為 Go 的程式碼組織方式(如套件機制)和執行模型(如靜態連結)使得連結過程相對簡單。

Go 建議盡量使用靜態連結。Why? What is cgo?



Linker (2)

Go 語言建議盡量使用靜態連結原因如下:

- **部署簡單:** 靜態連結將所有依賴項打包到單個可執行檔案中包括 Go 的 runtime 和標準函式庫。這使得部署 Go 程式變得非常簡單,因為只需要複製一個可執行檔案到目標機器上即可運行,無需擔心目標機器上是否安裝了相應的函式庫。
- **避免版本衝突:** 靜態連結可以避免動態連結庫的版本衝突問題。在動態連結中,如果系統上安裝了不同版本的共享函式庫可能會導致程式運行時出現錯誤。靜態連結消除了這種風險,因為所有的依賴項都被包含在可執行檔案中。
- **效能優勢:** 靜態連結的程式在啟動時不需要載入共享函式庫因此啟動速度略快於動態連結的程式。此外,靜態連結的程式在執行時也不需要進行函式庫的動態解析,因此可以獲得一定的效能優勢。

cgo 是 Go 語言的一個特殊功能,它允許 Go 程式呼叫 C 語言的程式碼。cgo 使得 Go 程式可以與現有的 C 語言函式庫進行互動操作,這對於使用一些底層系統函式庫或者整合遺留的 C 語言程式碼非常有用。



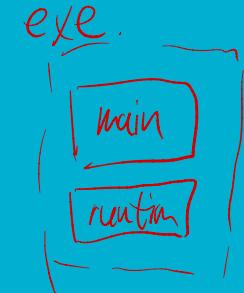
Runtime

Go 的 runtime 是一個強大的執行時系統, 負責管理 Go 程式的執行。

- 資源分配
- 垃圾回收器 *GC*
- goroutine 排程器
- channel 通信 ✓
-

Runtime 使 Go 能夠高效地執行並發程式, 並提供了豐富的執行時支援, 如
反射、型別斷言和 panic/recover 機制。

Runtime 還負責與作業系統互動, 管理 OS Thread 和系統呼叫。

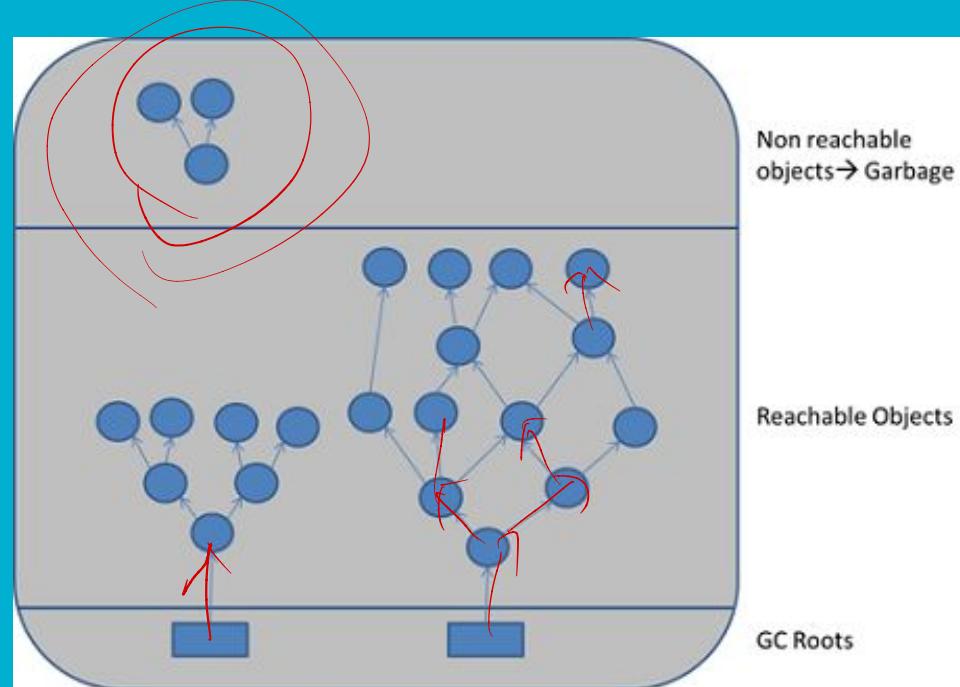


Garbage Collection

Java / JS

清理 heap 中的動態記憶體

「slice
map
chan」

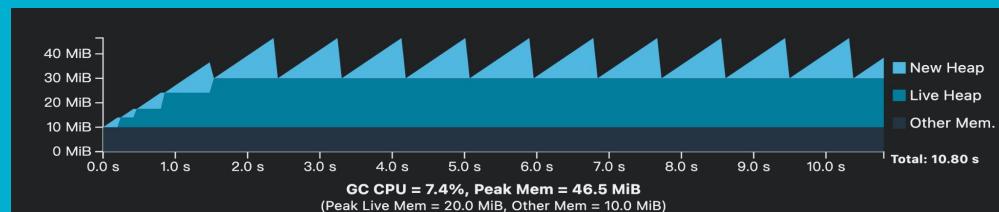
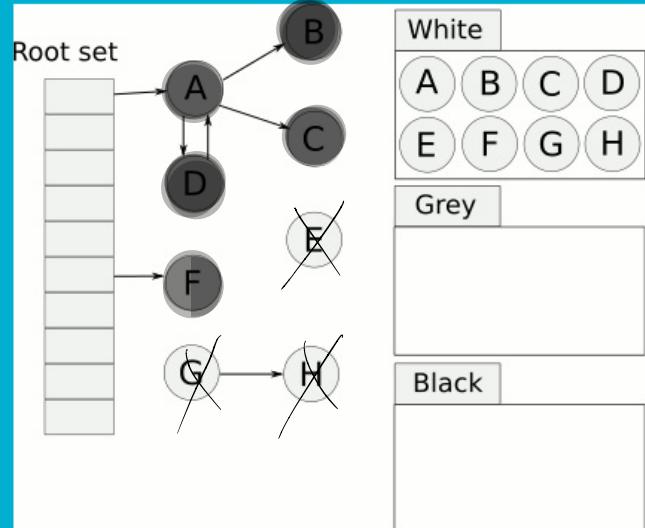


ref: <https://www.dynatrace.com/resources/ebooks/javabook/how-garbage-collection-works/>



Golang GC

- ❖ Stack 由編譯器管理, Heap 由 GC 管理
- ❖ 記憶體物件 = data + tag
- ❖ 由 runtime 記下所有記憶體物件
- ❖ GC cycle = 標記 \Leftrightarrow 刪除
- ❖ 使用 3 色標記法完成 GC cycle, 每次進入 `go sched()` 時檢查
- ❖ 應該盡可能地降低 GC cycle 頻率: 參數 GOGC & Memory Limit (PD控制)

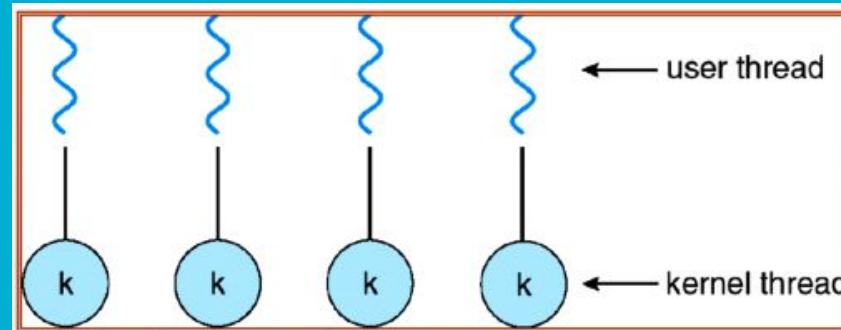


ref: <https://tip.golang.org/doc/gc-guide>



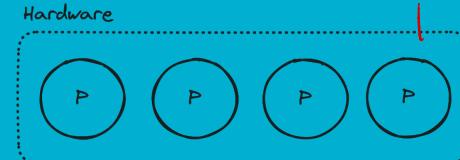
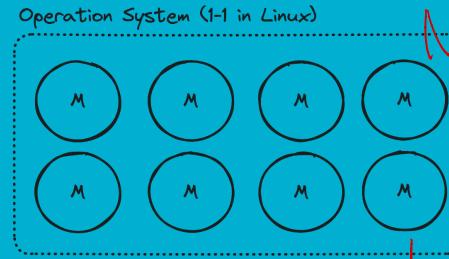
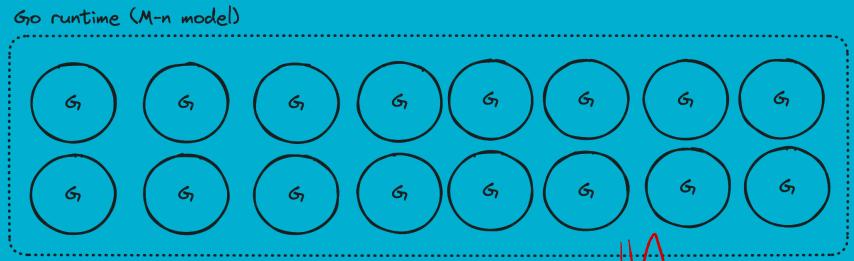
Linux 1-1 scheduler

- user thread 和 kernel thread 一一對應。
- 當 user thread 被中斷時,控制權轉移到 kernel thread。
- kernel thread 運行 scheduler 邏輯,決定是否讓出 CPU。
- 若讓出 CPU,則另一個 thread 接管 CPU 並執行。
- thread pool 充分利用所有 CPU 資源。
- 一對一的 thread 模型避免了 kernel thread 成為性能瓶頸。



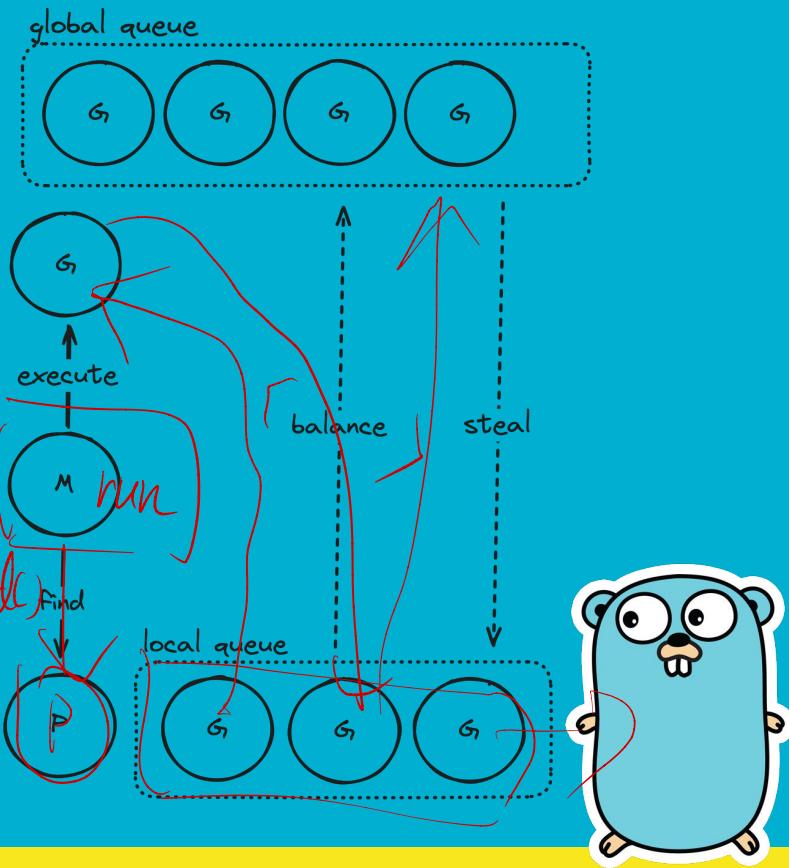
Golang M-n Scheduler (1)

- G, Goroutine 輕量化執行實體
 - M, OS Thread 作業系統提供的執行實體
 - P, Processor 實體 CPU 的映射
-
- G 的數量在單一裝置可以破百萬
 - 幾個 P 決定了最多幾個 G 同時執行
 - M 是由 OS 管理的外部執行平台



Golang M-n Scheduler (2)

- ❖ OS 啟動 M
- ❖ M 找到 P 後就有資格運行 G
- ❖ M 運行 G 優先從 P 的 queue 找
- ❖ 當今天 G 創建新 G 也放到 P 的 queue
- ❖ 當 M 發現 P 太多 G, 上傳到 global
- ❖ 當 M 發現 P 太少 G, 偷竊到 local
- ❖ 每個 G 的中斷都會觸發 M 重選 G
- ❖ 中斷包含: IO / chan / time event /



Lab

設計模式介紹與實戰

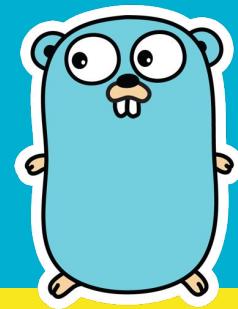


DDOS 攻擊概述

DOS

DOS (Denial of Service) 攻擊是一種網路攻擊手法旨在使目標系統或服務無法正常運作。以下是DOS 攻擊的主要特點和常見類型：

1. 目的：
 - 使目標系統或服務癱瘓無法提供正常服務
 - 導致目標系統資源耗盡如 CPU、記憶體、網路頻寬等
2. 攻擊方式：
 - 發送大量的請求或數據包超過目標系統的處理能力
 - 利用目標系統或協議的弱點導致其無法正常處理請求
3. 常見類型：
 - Flood 攻擊：
 - UDP Flood: 發送大量 UDP 數據包,耗盡目標系統資源
 - ICMP Flood: 發送大量 ICMP 回應請求數據包,消耗目標系統資源
 - SYN Flood: 發送大量 SYN 請求,但不完成 TCP 三次握手,耗盡目標系統資源
 - Ping of Death: 發送超過最大尺寸的ICMP 回應請求數據包,導致目標系統崩潰
 - Slowloris: 發送不完整的HTTP 請求,佔用目標系統的連接資源



DDOS

分散式 DOS 攻擊, 以下提供實作流程簡述

1. 實作 DOS client 端
2. 實作 DOS server 端
3. 用計策讓 victim 下載並且執行 DOS client
4. DOS client 用計策讓 victim 用 email 擴散 DOS client 執行檔
5. DOS client 定期 polling DOS server, 接收最新指令
6. DOS client 依照指令調用多核盡可能地對目標發動 DOS 攻擊



實作 ptt 高性能爬蟲

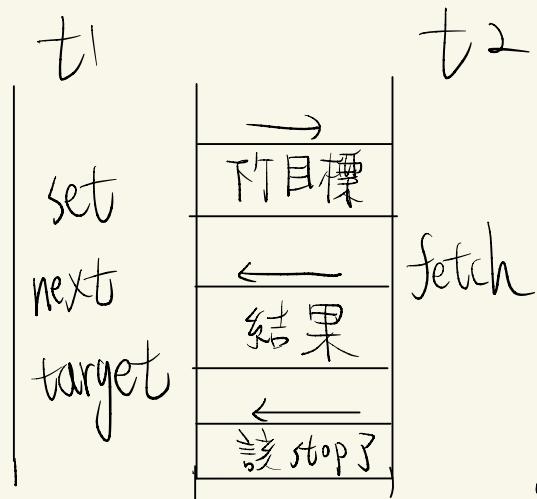
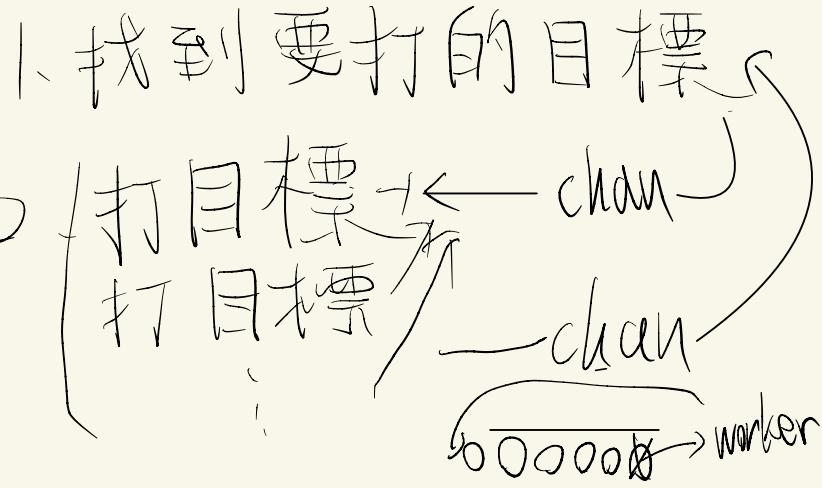
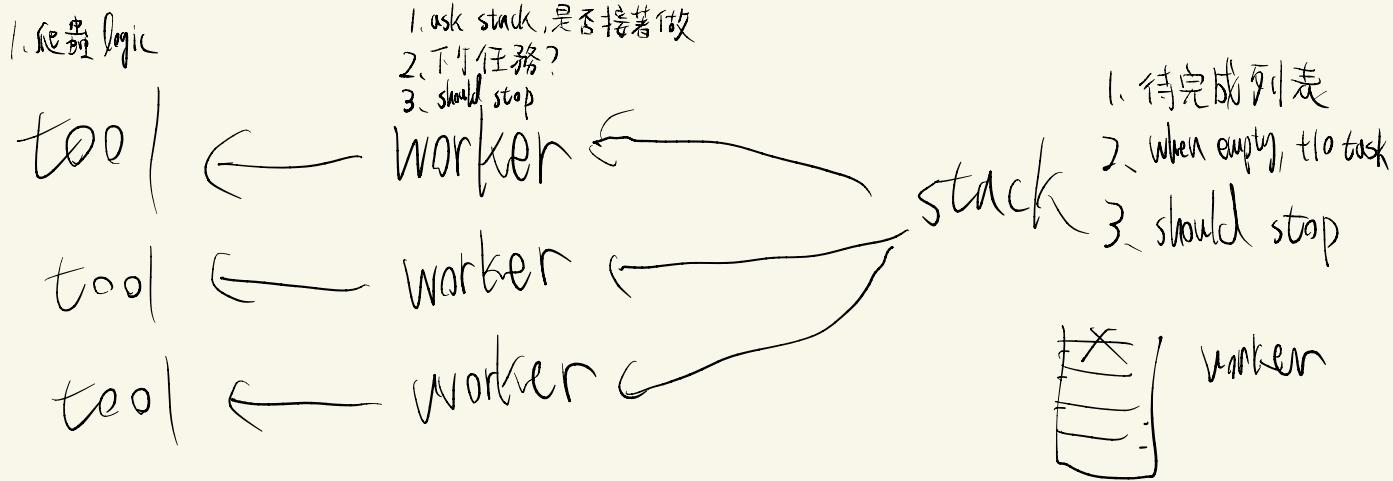
SPEC

- 輸入特定關鍵字
- 可以有 N 個 task 同時獲取所有八卦版內含關鍵字之標題 ($N > 1$)
- ex:
 - input: 貓貓
 - output: 八卦版內所有含貓貓的標題

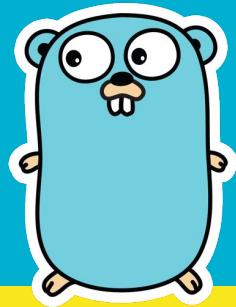
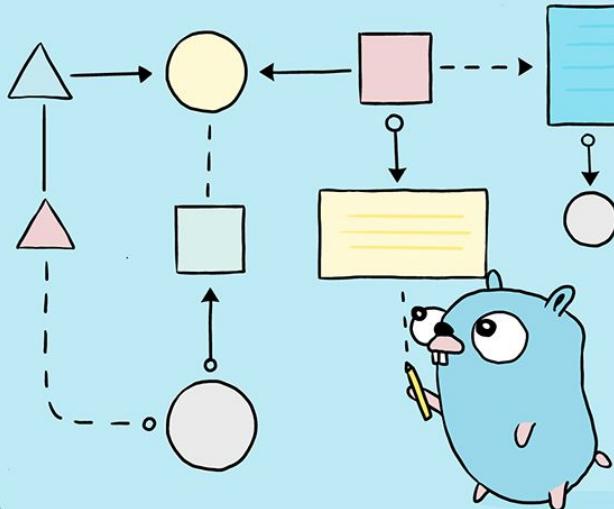
相關 URL : <https://www.ptt.cc/bbs/Gossiping/search?page=1&q=test>



OOP



Q&A 時間



作業: ptt 高性能爬蟲

1. 物件導向
2. 函數導向

Link: <https://github.com/leon123858/go-tutorial/tree/main/web-crawler>

