Ngày 7: Lập trình đồng thời và song song với GO

Bài 1 – Khái niệm Concurrency và Parallelism

Bài này sẽ giúp bạn học các nội dung sau:

- Phân biệt được processes, threads, gorountine
- Phân biệt concurrency (đồng thời) và parallelism (song song)
- Làm quen với gorountine, channels, pipelines

Tạo goroutine

Khảo sát chương trình `GoRoutineFibo.go` sau:

```
package main

import (
    "fmt"
    "os"
    "strconv"
)

func PrintFibo(count int) {
    fmt.Printf("Hiển thị %d số Fibonaci đầu tiên lớn 2", count)
    a := 1
    b := 2
    fibo := a + b
    for i := 0; i < count; i++ {
        fibo = a + b
        fmt.Print(fibo, " ")
        a = b
        b = fibo
    }
}

func main() {
    n, _ := strconv.Atoi(os.Args[1])
    go PrintFibo(n)
    fmt.Printf("Đã gọi hàm tính %d số Fibonaci", n)
}
```

Chạy chương trình với tham số là 10:

```
go run .\GoRoutineFibo.go 10
```

Kết quả hiển thị ra như sau:

```
Đã gọi hàm tính 10 số Fibonaci
```

Hàm `PrintFibo(n)` được gọi với từ khóa **go** phía trước ý nói là lời gọi hàm này xong thì thực hiện lệnh tiếp theo ngay; hàm PrintFibo(n) sẽ chạy nền (background).

Vì lệnh tiếp theo thực hiện xong mà hàm `PrintFibo(n)` có thể vừa mới bắt đầu chạy, chưa kịp chạy và hiển thị số liệu gì hết thì hàm `main()` đã kết thúc. Tức là chương trình đã kết thúc trong khi nội dung các lệnh bên trong hàm `PrintFibo(n)` chưa kịp thực hiện.

Để thấy được kết quả của hàm `PrintFibo(n)` thì bạn phải thêm 1 dòng để đợi chương trình chạy xong. Cụ thể đợi 1 giây để hàm PrintFibo chạy xong.

```
package main

import (
    "fmt"
    "os"
    "strconv"
    "time"
)

func PrintFibo(count int) {
    fmt.Printf("Hiển thị %d số Fibonaci đầu tiên lớn 2.\n", count)
    a := 1
    b := 2
    fibo := a + b
    for i := 0; i < count; i++ {
        fibo = a + b
        fmt.Print(fibo, " ")
        a = b
        b = fibo
    }
}

func main() {
    n, _ := strconv.Atoi(os.Args[1])
    go PrintFibo(n)
    fmt.Printf("Đã gọi hàm tính %d số Fibonaci", n)
    time.Sleep(time.Second)
}
```

Câu hỏi đặt ra là không thể biết lúc nào hàm `PrintFibo(n)` chạy xong, đợi 1 giây như ở trên thì chỉ là hú họa.

Đợi hàm Goroutine chạy xong

Xem chương trình được nâng cấp bằng cách sử dụng biến `waitGroup` như sau:

```
package main

import (
    "fmt"
    "os"
```

```
    "strconv"
    "sync"
)

var waitGroup sync.WaitGroup

func PrintFibo(count int) {
    fmt.Printf("Hiển thị %d số Fibonaci đầu tiên lớn 2.\n", count)
    a := 1
    b := 2
    fibo := a + b
    for i := 0; i < count; i++ {
        fibo = a + b
        fmt.Print(fibo, " ")
        a = b
        b = fibo
    }
    defer waitGroup.Done()
}

func main() {
    n, _ := strconv.Atoi(os.Args[1])

    waitGroup.Add(1)
    go PrintFibo(n)

    fmt.Printf("Đã gọi hàm tính %d số Fibonaci.\n", n)
    waitGroup.Wait()
}
```

Sử dụng channel cho goroutine

Khảo sát chương trình sau:

```
package main

import (
    "fmt"
    "time"
)

func writeToChannel(c chan int, x int) {
    fmt.Println(x)
    c <- x
    close(c)
}

func main() {
    c := make(chan int)
    go writeToChannel(c, 10)
    time.Sleep(time.Second)
```

```
    fmt.Println("Read:", <-c)
}
```

Chương trình gọi routine writeToChannel với 2 tham số:

- Tham số thứ nhất có kiểu là channel (viết tắt là chan) hỗ trợ chuyển số nguyên.
- Tham số thứ hai là số nguyên cần truyền.

Sử dụng cú pháp **c <- x** để truyền một số nguyên x vào kênh c.

Dùng cú pháp **value, status <-c** để lấy giá trị từ kênh c. Kết quả trả là cặp giá trị:

- value: giá trị lấy ra từ kênh c.
- status: là trạng thái của kênh. **true** có nghĩa là kênh đang mở (open); **false** có nghĩa là kênh đã đóng (closed).

Sau khi lấy giá trị thì kênh sẽ trống, nếu lấy một lần nữa thì sẽ trả lại giá trị 0 và status là false.

```
package main

import (
    "fmt"
    "time"
)

func writeToChannel(c chan int, x int) {
    fmt.Println(x)
    c <- x
    close(c)
}

func main() {
    c := make(chan int)
    go writeToChannel(c, 10)
    time.Sleep(time.Second)

    value, status := (<-c)

    fmt.Printf("Read value: %d; status=%t\n", value, status)

    value, status = (<-c)
    fmt.Printf("Read value again: %d; status=%t\n", value, status)
}
```

Kết quả:

```
10
```

```
Read value: 10; status=true
Read value again: 0; status=false
```

Khảo sát thêm ví dụ GoRoutineMessage.go sau:

- Chương trình minh họa 2 Go Routine gởi (sender) và nhận (receiver) xử lý dữ liệu chuỗi.
- Routine receiver nhận vào một chuỗi và xử lý bằng cách chuyển toàn bộ nội dung thành chữ thường.

```go
package main

import (
    "fmt"
    "strings"
)

func sender(c chan string, msg string) {
    c <- msg
}

func receiver(c chan string) {
    for {
        msg := <-c

        // Convert the message to lowercase
        msg = strings.ToLower(msg)
        fmt.Printf("Convert received msg to lowercase: %s\n", msg)
    }
}

func main() {
    var c chan string = make(chan string)

    go sender(c, "Demo Sender & Receiver by Go Routine")
    go receiver(c)

    for i := 1; i < 10; i++ {
        go sender(c, fmt.Sprintf("Thông báo mới từ VIỆT NAM #%d", i))
    }

    var input string
    fmt.Scanln(&input)
}
```

Output:

```
PS D:\MyCodes\MyGo\GoRoutine> go run .\GoRoutineMessage.go
```

```
Convert received msg to lowercase: demo sender & receiver by go
routine
Convert received msg to lowercase: thông báo mới từ việt nam #1
Convert received msg to lowercase: thông báo mới từ việt nam #2
Convert received msg to lowercase: thông báo mới từ việt nam #4
Convert received msg to lowercase: thông báo mới từ việt nam #5
Convert received msg to lowercase: thông báo mới từ việt nam #6
Convert received msg to lowercase: thông báo mới từ việt nam #7
Convert received msg to lowercase: thông báo mới từ việt nam #8
Convert received msg to lowercase: thông báo mới từ việt nam #3
Convert received msg to lowercase: thông báo mới từ việt nam #9
```

✍Tóm tắt

Kết thúc bài này bạn đã trải nghiệm và làm quen với khái niệm Routine trong GOLANG. Khi viết một lệnh cần gọi hàm mà không cần phải đợi hàm đấy chạy xong mà muốn chuyển sang lệnh tiếp theo thì thêm từ khóa `go` phía trước lời gọi hàm.

Bạn cũng đã làm quen với cách truyền dữ liệu thông qua channel (kênh) giữa các Routine.