Golang 개념 정리

2020. 06. 06 (토)

신 재 환

Go – 클로저(Closure)

- 리턴 값으로 익명 함수(Anonymous function)를 반환하는 형태를 지닌다.
- 익명 함수와 유사하지만, 반환되는 익명 함수 외부에 최소 한 개의 지역 변수를 포함한다.
- 클로저가 호출될 때, 이전의 지역 변수 값을 유지한다는 특성을 제공한다.

Go – 클로저(Closure)

- 클로저 예제 코드

```go
package main

import "fmt"

//              ↓ 리턴 값이 익명 함수
func calc() func(x int) int {
        a, b := 3, 5            // 지역 변수는 함수가 끝나면 소멸되지만
        return func(x int) int {
                return a*x + b // 클로저이므로 함수를 호출 할 때마다 변수 a와 b의 값을 사용할 수 있음
        }
        // ↑ 익명 함수를 리턴
}

func main() {
        f := calc() // calc 함수를 실행하여 리턴값으로 나온 클로저를 변수에 저장

        fmt.Println(f(1)) // 8
        fmt.Println(f(2)) // 11
        fmt.Println(f(3)) // 14
        fmt.Println(f(4)) // 17
        fmt.Println(f(5)) // 20
}
```

```
8
11
14
17
20
```

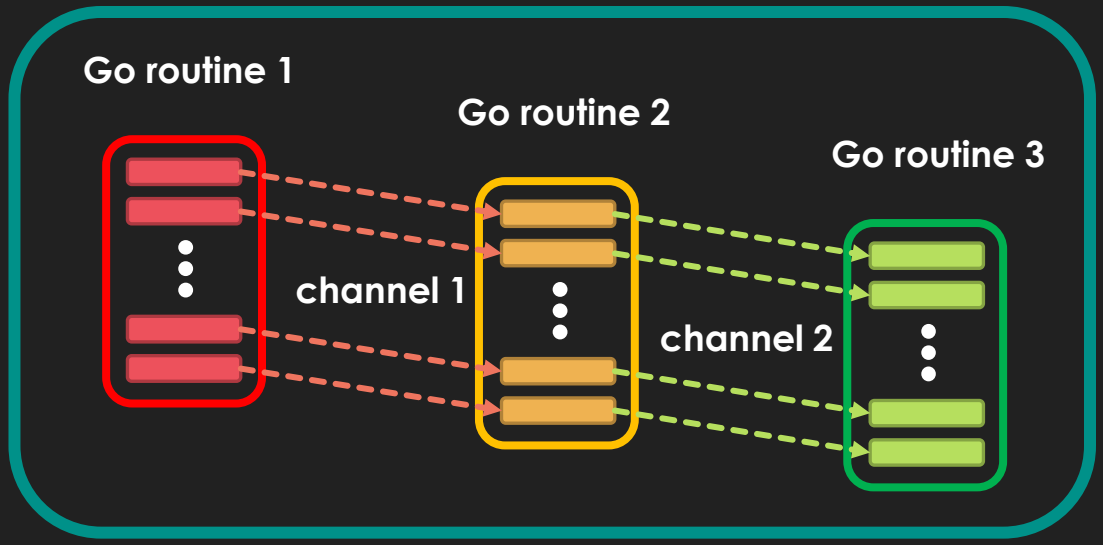
실행 결과

Go – 병렬성 및 동시성

- 병렬성(Parallelism): 동시 처리의 물리적인 개념
  - 작업을 여러 CPU 코어에서 나눠서 동시에 처리하는 상태
- 동시성(Concurrency): 동시 처리의 논리적인 개념
  - 단일 코어에서 여러 스레드를 생성하여 각 스레드에 코어를 할당하는 시간을 쪼개어 동시에 처리되는 것 처럼 보이는 상태 (시분할 형태로 처리)
- Go 루틴 : Go 언어 자체적으로 병렬성 및 동시성을 지원하는 경량 스레드
- 채널(Channel) : Go 루틴끼리 데이터를 주고 받기 위해 사용되는 통로

Go – 동시성 철학

- "메모리 공유를 통해 통신하지 말고 통신을 통해 메모리를 공유하라"
  - 스레드와 메모리 공유가 아닌, Go 루틴과 채널 중심의 프로그래밍
  - 수행할 전체 작업을 여러 개의 Go루틴으로 나누어 파이프 라인 모델을 설계
  - 하나의 Go루틴에서 처리한 결과값을 채널을 통해 다음 Go 루틴으로 전달



Go – Select 구문

- Select 구문
  - Go 루틴, 채널과 함께 Go 언어에서 동시성을 위해 가장 중요한 개념 중 하나
  - Go 루틴에서 사용되는 여러 채널에 대한 동기화를 수행
  - 정확한 동시성 프로그램을 보다 간편하게 구현할 수 있도록 함

Go – Select 구문

- Select 구문 특징 #1
  - case 문의 채널 값이 들어올 때 까지 select문에서 차단(block) 되며, 로직의 취소, 시간초과(time-out), 대기(waiting) 등을 수행
    - time.After() 함수를 사용하여 시간 초과 핸들링

```
var c <-chan int
select {
case <-c: ❶
case <-time.After(1 * time.Second):
    fmt.Println("Timed out.")
}
```

❶ This case statement will never become unblocked because we're reading from a `nil` channel.

This produces:

```
Timed out.
```

Go – Select 구문

- Select 구문 특징 #2
  - Switch – case 구문과 유사한 형태를 지니지만, 각 case의 조건이 순차적으로 검사 되지 않음 (각각의 case 구문이 거의 동일한 확률로 선택됨)
  - 여러 case의 채널에 값이 존재하는 경우 랜덤으로 수행

```
c1 := make(chan interface{}); close(c1)
c2 := make(chan interface{}); close(c2)

var c1Count, c2Count int
for i := 1000; i >= 0; i-- {
    select {
    case <-c1:
        c1Count++
    case <-c2:
        c2Count++
    }
}

fmt.Printf("c1Count: %d\nc2Count: %d\n", c1Count, c2Count)
```

This produces:

```
c1Count: 505
c2Count: 496
```

Go – Select 구문

- Select 구문 특징 #3
  - 모든 case의 채널에 값이 없어 차단(block)된 경우 실행되는 default 구문을 지원
  - default 구문은 주로 for – select(무한루프) 형태로 사용됨

```
done := make(chan interface{})
go func() {
    time.Sleep(5*time.Second)
    close(done)
}()

workCounter := 0
loop:
for {
    select {
    case <-done:
        break loop
    default:
    }

    // Simulate work
    workCounter++
    time.Sleep(1*time.Second)
}

fmt.Printf("Achieved %v cycles of work before signalled to stop.\n", workCounter)
```