Go의 동시성 패턴 및 context

2020. 06. 13 (토)

신 재 환

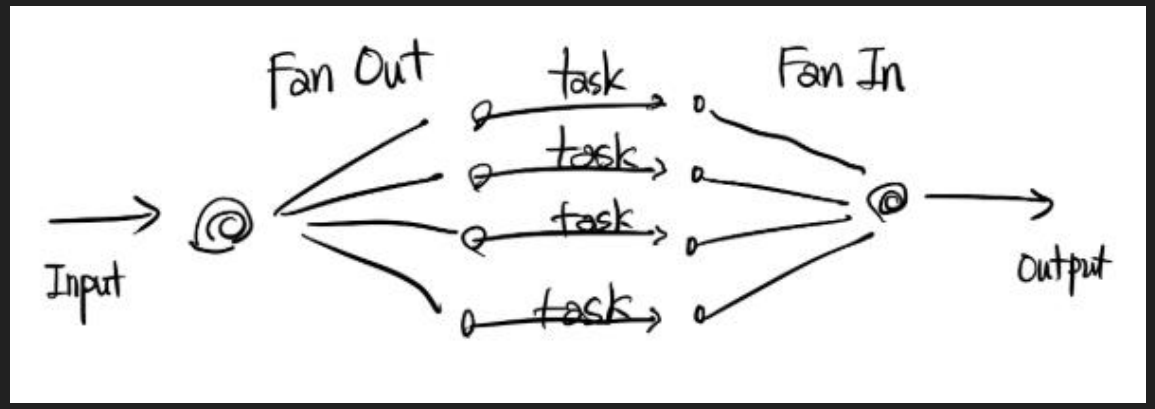
Pipeline

- 채널(channel)로 연결된 여러 개의 Go 루틴 stage들로 구성

- 각 stage에는 임의의 수의 inbound, outbound 채널이 존재하며, 첫 번째와 마지막 stage는 예외
  - 첫 번째 stage : 데이터를 다음 단계로 내보내는 outbound 채널만 존재
    - Source 또는 producer 라고 표현
  - 마지막 stage : 이전 stage에서 데이터를 받아오는 inbound 채널만 존재
    - Sink 또는 consumer 라고 표현

Pipeline

- Fan-out 과 Fan-in
  - Fan-out
    - 하나의 채널을 입력으로 다수의 Go 루틴이 데이터를 처리
    - 입력 데이터의 처리를 여러 worker에서 수행하여 부하를 분산시키는 목적으로 사용
  - Fan-in
    - 다수의 Go 루틴으로부터 처리된 결과를 merge 과정을 거쳐 하나의 채널로 취합



Pipeline

- 파이프라인 Go 루틴(stage)의 종료
  - 입력 채널(inbound)의 데이터를 읽고 처리하며, 입력 채널이 닫히는 경우 stage 종료
  - 메시지를 보내는 작업이 모두 끝나면 출력 채널(outbound) 채널을 닫고 stage 종료
- 그러나, 실제 상황에서 다음 stage로 값을 넘기지 못하는 경우 발생
  - 입력 채널의 일부만을 필요로 하는 경우
  - 입력 채널의 값에 오류가 있어 일찍 종료되는 경우
- 다음 stage는 도착하지 않는 값을 무한히 대기하며, 종료되지 못함

- Go 루틴의 경우 **가비지 컬렉터가 작동하지 않기 때문에** 메모리 누수 발생
  - Go 루틴이 언제 종료될지 모른 채로 Go 루틴을 실행시켜서는 안된다
    "Never start a goroutine without knowing how it will stop"

Pipeline

- Stopping short
  - 채널 생성시 전송될 값의 수를 알고있는 경우, 채널의 buffer를 설정
    - 채널이 가득 차는 상황을 방지하여 block 되지 않도록 함
    - 임시 방편에 불과하며, 채널에 추가적인 입력이 발생하는 경우, 다시 block됨
  - 모든 채널에 대해 **수신할 값의 수**와 **소비할 값의 수**를 알고 있어야만 사용 가능
  - 수신 stage가 송신 stage에게 중지 신호를 보내는 방법이 필요

Pipeline

- Explicit Cancellation (example03.go)
  - 메모리 누수를 막기 위해 상위 stage의 Go 루틴을 명시적으로 취소
  - 중지 신호를 전달하는 done 채널을 입력 파라미터로 전송
  - for-select 구문을 사용하여, 프로그램 종료 후 done 채널이 닫히는 경우 Go 루틴이 return(종료) 하도록 작성

```go
func main() {
    done := make(chan struct{})
    defer close(done)

    in := gen(done, nums...: 2, 3, 5, 7, 11, 13)

    for n := range in {
        fmt.Println(n)
    }
}
```

```go
func gen(done chan struct{}, nums ...int) <-chan int {
    out := make(chan int, len(nums))
    go func() {
        defer close(out)
        for _, n := range nums {
            select {
            case out <- n:
            case <-done:
                return
            }
        }
    }()
    return out
}
```

Context

- context 패키지는 마감시간(deadline), 취소(cancellation) 신호, 및 기타 요청범위 값을 API 경계와 프로세스간에 전달하는 Context 타입을 정의
- 서버에 들어오는 요청은 context를 생성해야 하며, 서버로 나가는 호출은 context를 승인해야 함

- context.Context 타입 사용 시 주의사항
  - 관례적으로 함수의 첫 번째 파라미터로 사용한다
  - Context를 구조체에 넣지 않고 함수의 파라미터로 전달해서 사용한다
  - Custom Context type을 만들어서 사용하지 않는다
  - 사용할 Context가 확실하지 않으면 nil Context 대신 context.TODO를 전달한다

Context - Variables

- **Canceled**
  - context가 취소될 때 Context.Err()에 의해 호출되는 error

```
var Canceled = errors.New( text: "context canceled")
```

- **DeadlineExceeded**
  - Context의 마감시간(deadline)이 지났을 때 Context.Err()에 의해 호출되는 error

```
var DeadlineExceeded error = deadlineExceededError{}

type deadlineExceededError struct{}
```

Context – Functions

```
func WithCancel(parent Context) (ctx Context, cancel CancelFunc)

func WithDeadline(parent Context, d time.Time) (Context, CancelFunc)

func WithTimeout(parent Context, timeout time.Duration) (Context, CancelFunc)

func WithValue(parent Context, key, val interface{}) Context

func Background() Context

func TODO() Context
```

Context – Functions #1

- **WithCancel()**

```
func WithCancel(parent Context) (ctx Context, cancel CancelFunc)
```

  - 새로운 Done 채널을 가진 부모의 복사본과 취소 함수를 반환
  - 반환된 취소 함수(cancel())가 호출되거나, 부모 컨텍스트의 Done 채널이 닫히는 경우 해당 컨텍스트의 Done 채널이 닫힘
  - 컨텍스트가 취소되면 관련된 자원의 할당이 해제되므로, 작업이 완료되는 즉시 취소 함수(cancel())를 호출해야 함

Context – Functions #2

- **WithDeadline()**

```
func WithDeadline(parent Context, d time.Time) (Context, CancelFunc)
```

  - 마감시간이 d보다 늦지 않도록 설정된 부모 컨텍스트의 복사본과 취소 함수를 반환
  - 부모 컨텍스트의 최종 기한이 이미 d보다 빠른 경우 **WithDeadline(parent, d)**는 parent와 의미적으로 동일함
  - 마감시간이 만료되거나, 취소 함수의 호출 및 부모 컨텍스트의 Done 채널이 닫히는 경우 해당 컨텍스트의 Done 채널이 닫힘

- **WithTimeout()**

```
func WithTimeout(parent Context, timeout time.Duration) (Context, CancelFunc)
```

  - WithDeadline(parent, time.Now().Add(timeout))을 호출
  - 주요 내용은 위와 동일

Context – Functions #3

- **WithValue()**

```
func WithValue(parent Context, key, val interface{}) Context
```

  - key에 연관된 값이 val인 parent의 복사본을 반환
  - 프로세스 및 API를 전송하는 요청 범위 데이터에만 사용한다
  - key의 type은 패키지 간에 충돌을 피하기 위해 새로운 유형으로 정의하여 사용한다 (string 등의 built-in type이 아니여야 함)

Context – Functions #4

- **Background()**

```
func Background() Context
```

  - nil이 아닌, 빈 컨텍스트를 반환
  - 반환되는 컨텍스트는 취소되지 않으며 value를 갖지 않고 마감시간이 없음
  - main 함수, 초기화, 테스트 등에서 수신 요청에 대한 최상위 컨텍스트로 사용

- **TODO()**

```
func TODO() Context
```

  - nil이 아닌, 빈 컨텍스트를 반환
  - 사용할 컨텍스트가 확실하지 않거나, 아직 사용할 수 없는 경우 context.TODO 로 생성한 컨텍스트를 사용

Context – Methods

- 4 가지 메소드를 제공하며, 여러 Go 루틴에서 동시에 호출 가능

```
type Context interface {
    Deadline() (deadline time.Time, ok bool)

    Done() <-chan struct{}

    Err() error

    Value(key interface{}) interface{}
}
```

Context – Methods #1

- **Deadline()**

```
Deadline() (deadline time.Time, ok bool)
```

  - 수행된 작업이 취소되어야 하는 시간(deadline) 및 bool 값을 반환
  - 마감시간(deadline)이 설정되지 않은 경우 ok == false, 설정된 경우 ok==true 반환
  - 연속 호출하는 경우 동일한 결과를 반환

Context – Methods #2

- **Done()**

```
Done() <-chan struct{}
```

  - WithCancel()의 경우 반환된 cancel 함수가 호출될 때 닫힌 채널을 반환
  - WithDeadline(), WithTimeout()의 경우 마감 시간이 만료되거나, 반환된 cancel 함수가 호출될 때 닫힌 채널을 반환

- **Err()**

```
Err() error
```

  - 컨텍스트의 Done 채널이 닫히지 않은 경우 nil을 반환
  - Done 채널이 닫힌 경우, 닫힌 이유를 설명하는 nil이 아닌 error를 리턴
  - 연속 호출하는 경우 동일한 오류를 반환

Context – Methods #3

- **Value()**

```
Value(key interface{}) interface{}
```

  - 컨텍스트의 key에 연관된 value를 반환하고, 연관된 value가 없는 경우 nil을 반환
  - 동일한 key를 사용한 연속 호출은 동일한 value를 반환
  - 함수에 선택적 매개변수를 전달하는 것이 아니라, 프로세스 및 API를 전송하는 요청 범위 데이터에만 컨텍스트 value를 사용한다
  - key의 type은 패키지 간에 충돌을 피하기 위해 새로운 유형으로 정의하여 사용한다 (string 등의 built-in type이 아니여야 함)