Programmeertalen: Go

Donderdaglecture

Koen van Elsen, Sander van Oostveen

Universiteit van Amsterdam

12 maart 2020

Inhoud

Paradigma

- Omschrijf concurrency

Paradigma

- Omschrijf concurrency
- Hoe maak je code concurrent?

Paradigma

- Omschrijf concurrency
- Hoe maak je code concurrent?
- Wat is een goroutine?

Paradigma

- Omschrijf concurrency
- Hoe maak je code concurrent?
- Wat is een goroutine?

`https://en.wikipedia.org/wiki/Concurrency_(computer_science)`

Oplossingen en concurrent

- Goroutine voor een filter, dan doorgeven

Oplossingen en concurrent

- Goroutine voor een filter, dan doorgeven
- Goroutine die de filter start, nieuwe filter aanmaakt en on-the-fly doorgeeft

Oplossingen en concurrent

- Goroutine voor een filter, dan doorgeven
- Goroutine die de filter start, nieuwe filter aanmaakt en on-the-fly doorgeeft
- Main die voor elk element een goroutine start

Oplossingen en concurrent

- Goroutine voor een filter, dan doorgeven
- Goroutine die de filter start, nieuwe filter aanmaakt en on-the-fly doorgeeft
- Main die voor elk element een goroutine start ,maar dan wel wacht op de vorige

Oplossingen en concurrent

- Goroutine voor een filter, dan doorgeven
- Goroutine die de filter start, nieuwe filter aanmaakt en on-the-fly doorgeeft
- Main die voor elk element een goroutine start ,maar dan wel wacht op de vorige
- Main die voor elk element een goroutine start en deze in een channel verzamelt

Oplossingen en concurrent

- Goroutine voor een filter, dan doorgeven
- Goroutine die de filter start, nieuwe filter aanmaakt en on-the-fly doorgeeft
- Main die voor elk element een goroutine start ,maar dan wel wacht op de vorige
- Main die voor elk element een goroutine start en deze in een channel verzamelt

sync WaitGroup

- Een manier om te wachten tot een bepaald proces klaar is.

sync WaitGroup

- Een manier om te wachten tot een bepaald proces klaar is.
- Toch zien we deze liever niet, waarom niet?

sync WaitGroup

- Een manier om te wachten tot een bepaald proces klaar is.
- Toch zien we deze liever niet, waarom niet?
- Denk aan het motto van Go: "Share memory by communicating, don't communicate by sharing memory."

sync WaitGroup

- Een manier om te wachten tot een bepaald proces klaar is.
- Toch zien we deze liever niet, waarom niet?
- Denk aan het motto van Go: "Share memory by communicating, don't communicate by sharing memory."
- Daarbij komt dat het erg omslachtig kan worden als je dit voor meerdere processen doet die, in een bepaalde volgorde, op elkaar moeten wachten. Hiervoor kun je dan beter eerst goed nadenken over het programma en wat er nodig is om dit in de juiste volgorde te doen.

Inhoud

Leaky Goroutines: Slecht!

```
routine(channel chan int) {
    for{
        i <- channel
        if condition {
            newChannel := make(chan int)
            go routine(newChannel)
        }
    }
}
```

- Loop eindigt niet!

Leaky Goroutines: Slecht!

```
routine(channel chan int) {
    for{
        i <- channel
        if condition {
            newChannel := make(chan int)
            go routine(newChannel)
        }
    }
}
```

- Loop eindigt niet!
- Channel word niet gesloten!

Leaky Goroutines: Slecht!

```
routine(channel chan int) {
    for{
        i <- channel
        if condition {
            newChannel := make(chan int)
            go routine(newChannel)
        }
    }
}
```

- Loop eindigt niet!
- Channel word niet gesloten!
- Goroutine blijft bestaan na programma terminatie(Leak)

Leaky Goroutines: Goed!

```
routine(channel chan int) {
    for i := range channel {
        if condition {
            newChannel := make(chan int)
            go routine(newChannel)
        }
    }
    close(newChannel)
}
```

- Geen oneindige loops.

Leaky Goroutines: Goed!

```
routine(channel chan int) {
    for i := range channel {
        if condition {
            newChannel := make(chan int)
            go routine(newChannel)
        }
    }
    close(newChannel)
}
```

- Geen oneindige loops.
- Channels worden gesloten!

Leaky Goroutines: Goed!

```
routine(channel chan int) {
    for i := range channel {
        if condition {
            newChannel := make(chan int)
            go routine(newChannel)
        }
    }
    close(newChannel)
}
```

- Geen oneindige loops.
- Channels worden gesloten!
- Lezen uit een gesloten channel returned 0.

Leaky Goroutines: Goed!

```
routine(channel chan int) {
    for i := range channel {
        if condition {
            newChannel := make(chan int)
            go routine(newChannel)
        }
    }
    close(newChannel)
}
```

- Geen oneindige loops.
- Channels worden gesloten!
- Lezen uit een gesloten channel returned 0.
- Hierdoor eindigt de loop!

Leaky Goroutines: Goed!

```
routine(channel chan int) {
    for i := range channel {
        if condition {
            newChannel := make(chan int)
            go routine(newChannel)
        }
    }
    close(newChannel)
}
```

- Geen oneindige loops.
- Channels worden gesloten!
- Lezen uit een gesloten channel returned 0.
- Hierdoor eindigt de loop!
- Termineer het programma pas als de laatste channel 0 returned.

Concurrency

- Waarom gebruik je Goroutines?

Concurrency

- Waarom gebruik je Goroutines? Concurrency!

Concurrency

- Waarom gebruik je Goroutines? Concurrency!
- Ideaal om zoveel mogelijk Goroutines tegelijkertijd te laten lopen!

Concurrency

- Waarom gebruik je Goroutines? Concurrency!
- Ideaal om zoveel mogelijk Goroutines tegelijkertijd te laten lopen!
- Wachten totdat huidige Goroutine klaar is is(in dit geval) niet concurrent!

Concurrency: Slecht!

```
routine(lst []int) {
    newlst := make([]int)
    for i := 0; i < len(lst); i++ {
        if condition {
            newlst := append(newlst, lst[i])
        }
    }
    go routine(newlst)
}
```

- Stuur niet steeds een slice door in sequentie.

Concurrency: Slecht!

```
routine(lst []int) {
    newlst := make([]int)
    for i := 0; i < len(lst); i++ {
        if condition {
            newlst := append(newlst, lst[i])
        }
    }
    go routine(newlst)
}
```

- Stuur niet steeds een slice door in sequentie.
- Stuur 1 voor 1 waardes door, zodat de verschillende Goroutines concurrent kunnen werken.

Shared Memory

"Share memory by communicating, don't communicate by sharing memory."

Shared Memory

"Share memory by communicating, don't communicate by sharing memory."

- WaitGroups delen een counter tussen verschillende Goroutines. Shared memory!

Shared Memory

"Share memory by communicating, don't communicate by sharing memory."

- WaitGroups delen een counter tussen verschillende Goroutines. Shared memory!
- Channels zijn de manier om te communiceren, maar ook om te synchroniseren.

Shared Memory

"Share memory by communicating, don't communicate by sharing memory."

- WaitGroups delen een counter tussen verschillende Goroutines. Shared memory!
- Channels zijn de manier om te communiceren, maar ook om te synchroniseren.
- Channels kosten minder overhead.

Shared Memory

"Share memory by communicating, don't communicate by sharing memory."

- WaitGroups delen een counter tussen verschillende Goroutines. Shared memory!
- Channels zijn de manier om te communiceren, maar ook om te synchroniseren.
- Channels kosten minder overhead.
- Niet alle synchronisatie is slecht! In de team opdracht: sync.Once!

Inhoud

sync.Once

Het eenmalig uitvoeren van een bepaald stuk code, bijvoorbeeld het klaarzetten van iets waar alle andere goroutines (van dezelfde functie) gebruik van kunnen maken.

sync.Once

Het eenmalig uitvoeren van een bepaald stuk code, bijvoorbeeld het klaarzetten van iets waar alle andere goroutines (van dezelfde functie) gebruik van kunnen maken.

```
var serverOpstart sync.Once

func main() {
        serverAanroep()
        serverAanroep()
}

func serverAanroep() {
        serverOpstart.Do(func() {
                fmt.Println("Doe dit tijdens de opstart van een server.")
        })
        fmt.Println("Doe de rest van de code voor elke aanroep!")
}
```

sync.Once

Het eenmalig uitvoeren van een bepaald stuk code, bijvoorbeeld het klaarzetten van iets waar alle andere goroutines (van dezelfde functie) gebruik van kunnen maken.

```
var serverOpstart sync.Once

func main() {
        serverAanroep()
        serverAanroep()
}

func serverAanroep() {
        serverOpstart.Do(func() {
                fmt.Println("Doe dit tijdens de opstart van een server.")
        })
        fmt.Println("Doe de rest van de code voor elke aanroep!")
}
```

```
Doe dit tijdens de opstart van een server.
Doe de rest van de code voor elke aanroep!
Doe de rest van de code voor elke aanroep!
```

Inhoud

Go Recursion

- In de Sieve riep je Goroutines recursief aan.
- In de team opdracht is het beter om dit niet recursief te doen.
- Maak daarom alleen nieuwe Goroutines aan vanuit je hoofd functie.
- Voorkom hiermee dat meerdere Goroutines tegelijkertijd sync.Once gaan gebruiken!

Inhoud

Sander

- Byte en wat kun je er allemaal mee doen
- De breedte klopt niet in de output voor de noordzijde, dit is geen probleem en mag je fixen.
- De oplossing voor het kortste pad(Expert) mag gegeven worden in comments; Je mag er ook code voor schrijven, maar zorg er wel voor dat je nogsteeds alle tests haalt op codegrade.

Doolhof als zoekboom

sync.Once: eenmalige initialisatie

```
import "sync" // https://golang.org/pkg/sync/
import "fmt"
var s []int
var initialise sync.Once

func fill() {
        fmt.Println("Initialising!")
        s = []int{91, 42, 54}
}

func getIndex(x int, result chan int) {
        initialise.Do(fill) // fill is only executed at the first getIndex invocation
        for i, v := range s {
                if x == v {
                        result <- i
                        return
                }
        }
        result <- -1 // Not found
}
```

Succes met de Teamopdracht

Zoek je **eigen** weg de maze uit!