

并发编程

作者：少林之巅

目录

1. 并发和并行

2. Goroutine初探

3. Goroutine实战

4. Channel介绍

并发和并行

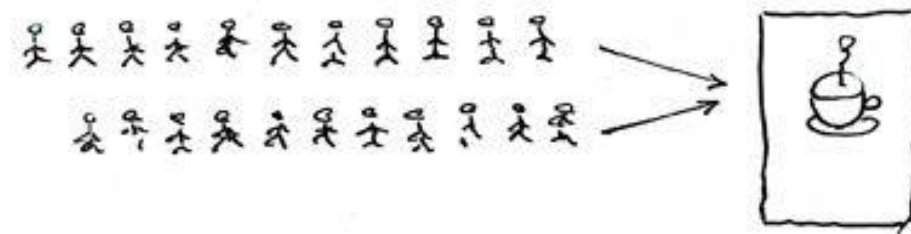
1. 概念

A. 并发：同一时间段内执行多个操作。

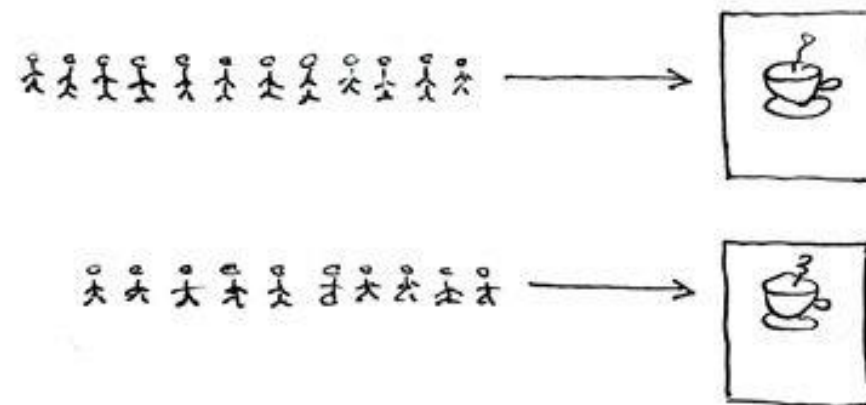
B. 并行：同一时刻执行多个操作。

并发和并行

Concurrent = Two Queues One Coffee Machine



Parallel = Two Queues Two Coffee Machines



© Joe Armstrong 2013

Goroutine初探

3. 多线程

- A. 线程是由操作系统进行管理，也就是处于内核态。
- B. 线程之间进行切换，需要发生用户态到内核态的切换。
- C. 当系统中运行大量线程，系统会变的非常慢。
- D. 用户态的线程，支持大量线程创建。也叫协程或goroutine。

Goroutine使用介绍

4. 创建goroutine

```
package main

import (
    "fmt"
)

func hello() {
    fmt.Println("Hello world goroutine")
}

func main() {
    go hello()
    fmt.Println("main function")
}
```

Goroutine使用介绍

5. 修复代码

```
package main

import (
    "fmt"
    "time"
)

func hello() {
    fmt.Println("Hello world goroutine")
}

func main() {
    go hello()
    time.Sleep(1*time.Second)
    fmt.Println("main function")
}
```

Goroutine使用介绍

6. 启动多个goroutine

```
package main

import (
    "fmt"
    "time"
)

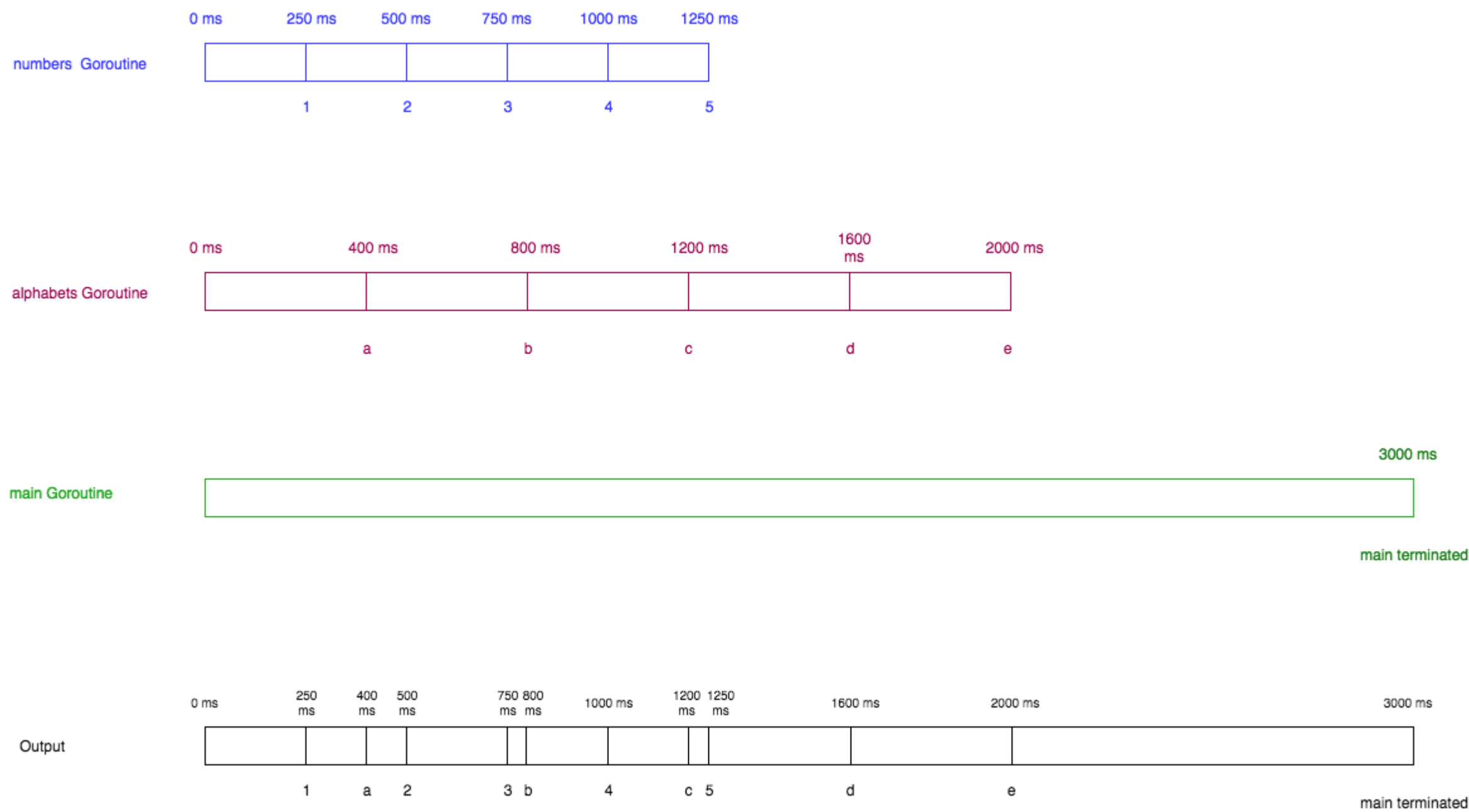
func numbers() {
    for i := 1; i <= 5; i++ {
        time.Sleep(250 * time.Millisecond)
        fmt.Printf("%d ", i)
    }
}

func alphabets() {
    for i := 'a'; i <= 'e'; i++ {
        time.Sleep(400 * time.Millisecond)
        fmt.Printf("%c ", i)
    }
}

func main() {
    go numbers()
    go alphabets()
    time.Sleep(3000 * time.Millisecond)
    fmt.Println("main terminated")
}
```


Goroutine使用介绍

7. 程序分析



Goroutine使用介绍

8. 多核控制

- A. 通过runtime包进行多核设置
- B. GOMAXPROCS设置当前程序运行时占用的cpu核数
- C. NumCPU获取当前系统的cpu核数

Goroutine原理浅析

9. Goroutine原理浅析

- A. 一个操作系统线程对应用户态多个goroutine
- B. 同时使用多个操作系统线程
- C. 操作系统线程对goroutine是多对多关系, 即M:N

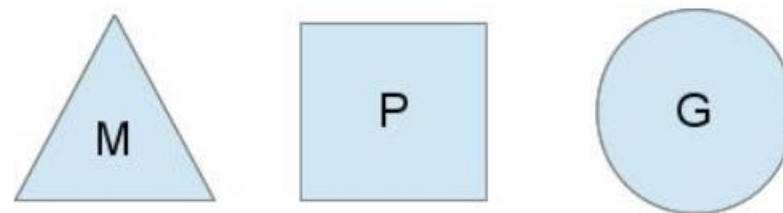
Goroutine原理浅析

10. 模型抽象

A. 操作系统线程： M

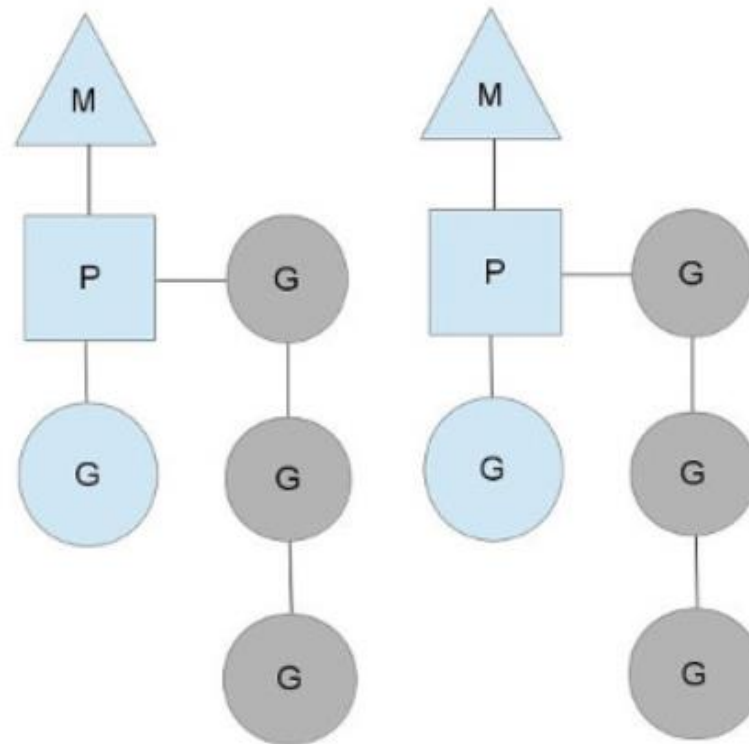
B. 用户态线程（goroutine）： G

C. 上下文对象： P



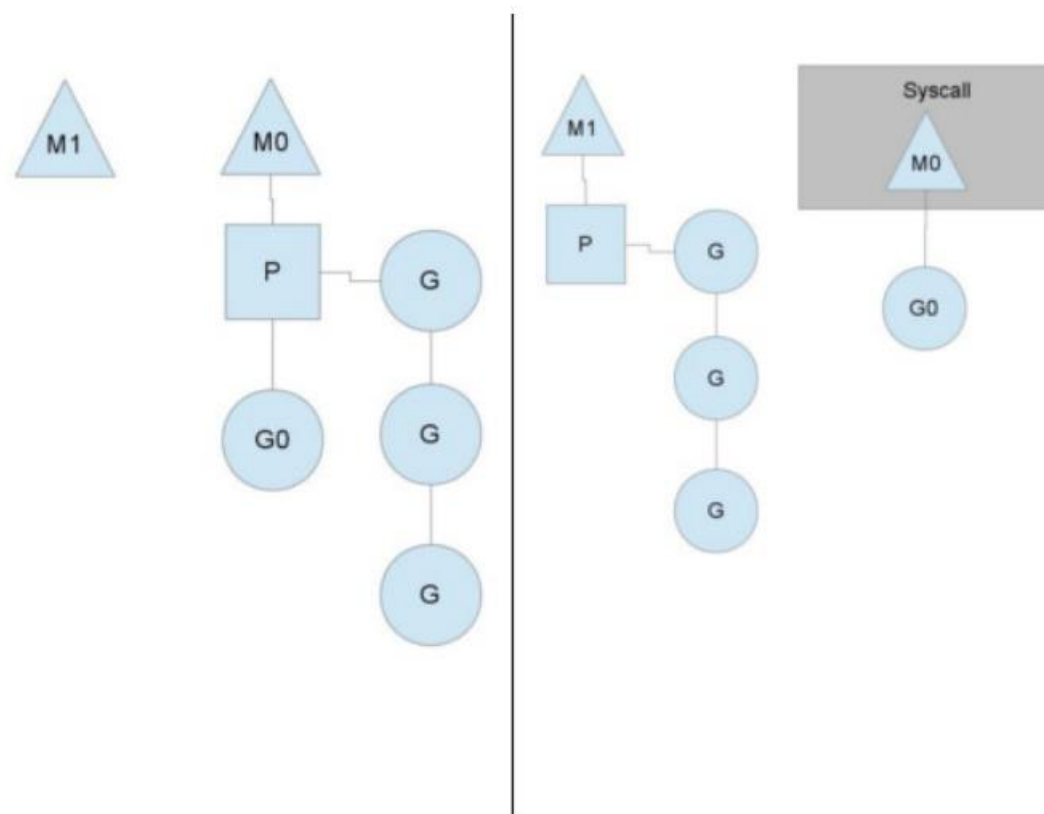
Goroutine原理浅析

11. goroutine调度



Goroutine原理浅析

12. 系统调用怎么处理



Channel介绍

13. channel介绍

- A. 本质上就是一个队列，是一个容器
- B. 因此定义的时候，需要指定容器中元素的类型
- C. `var 变量名 chan 数据类型`

Channel介绍

```
package main

import "fmt"

func main() {
    var a chan int
    if a == nil {
        fmt.Println("channel a is nil, going to define it")
        a = make(chan int)
        fmt.Printf("Type of a is %T", a)
    }
}
```


14. 元素入队和出队

```
var a chan int
```

A. 入队操作, `a <- 100`

B. 出队操作: `data := <- a`

Channel介绍

15. 阻塞chan

```
package main

import "fmt"

func main() {
    var a chan int
    if a == nil {
        fmt.Println("channel a is nil, going to define it")
        a = make(chan int)
        a <- 10
        fmt.Printf("Type of a is %T", a)
    }
}
```

Channel介绍

16. 使用chan来进行goroutine同步

```
package main

import (
    "fmt"
)

func hello(done chan bool) {
    fmt.Println("Hello world goroutine")
    done <- true
}

func main() {
    done := make(chan bool)
    go hello(done)
    <-done
    fmt.Println("main function")
}
```

Channel介绍

17. 使用chan来进行goroutine同步

```
package main

import (
    "fmt"
    "time"
)

func hello(done chan bool) {
    fmt.Println("hello go routine is going to sleep")
    time.Sleep(4 * time.Second)
    fmt.Println("hello go routine awake and going to write to done")
    done <- true
}

func main() {
    done := make(chan bool)
    fmt.Println("Main going to call hello go goroutine")
    go hello(done)
    <-done
    fmt.Println("Main received data")
}
```

Channel介绍

18. 单向chan

```
package main

import "fmt"

func sendData(sendch chan<- int) {
    sendch <- 10
}

func readData(sendch <-chan int) {
    sendch <- 10
}

func main() {
    chn1 := make(chan int)
    go sendData(chn1)
    readData(chn1)
}
```

Channel介绍

19. chan关闭

```
package main

import (
    "fmt"
)

func producer(chnl chan int) {
    for i := 0; i < 10; i++ {
        chnl <- i
    }
    close(chnl)
}

func main() {
    ch := make(chan int)
    go producer(ch)
    for {
        v, ok := <-ch
        if ok == false {
            break
        }
        fmt.Println("Received ", v, ok)
    }
}
```

Channel介绍

20. for range操作

```
package main

import (
    "fmt"
)

func producer(chnl chan int) {
    for i := 0; i < 10; i++ {
        chnl <- i
    }
    close(chnl)
}

func main() {
    ch := make(chan int)
    go producer(ch)
    for v := range ch {
        fmt.Println("Received ",v)
    }
}
```