go+区块链培训 讲师:张长志

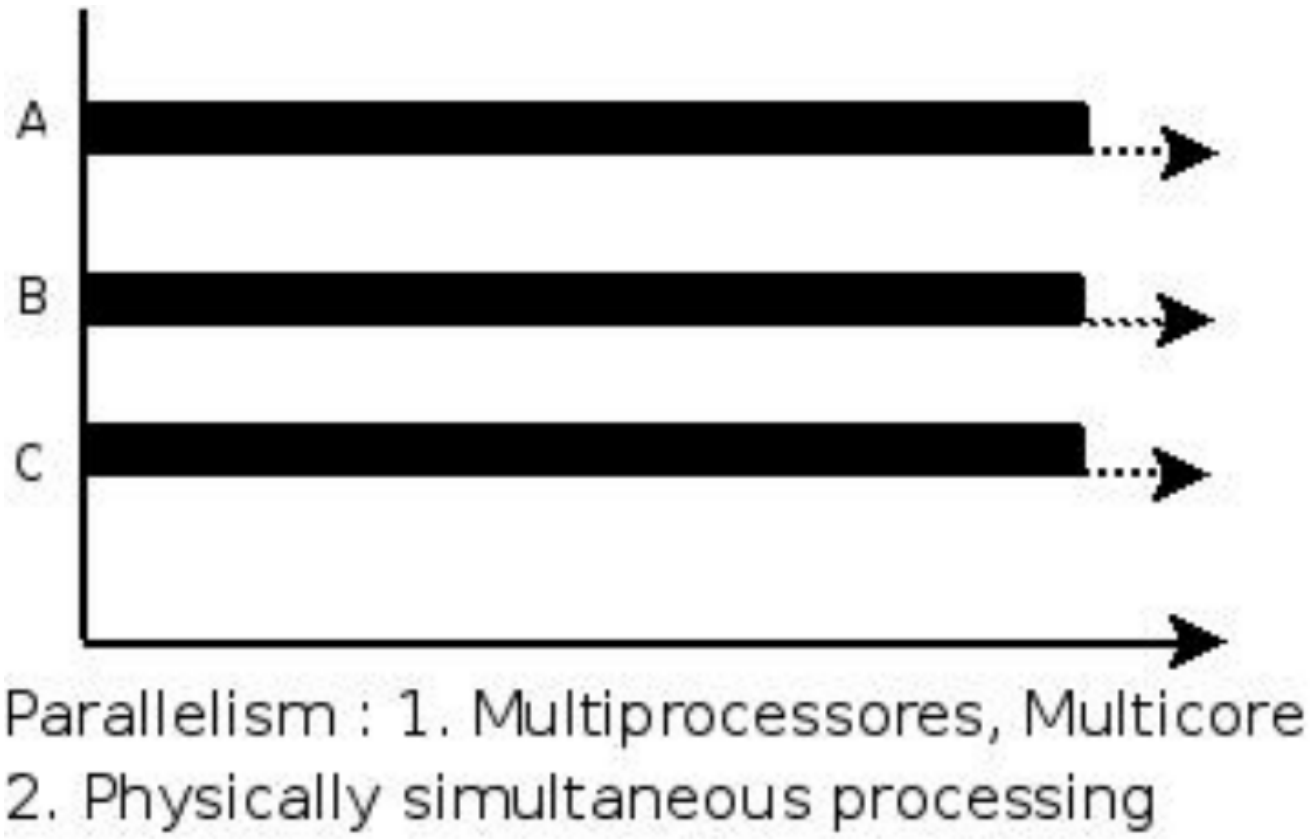
并发

并发和并行

1、并行

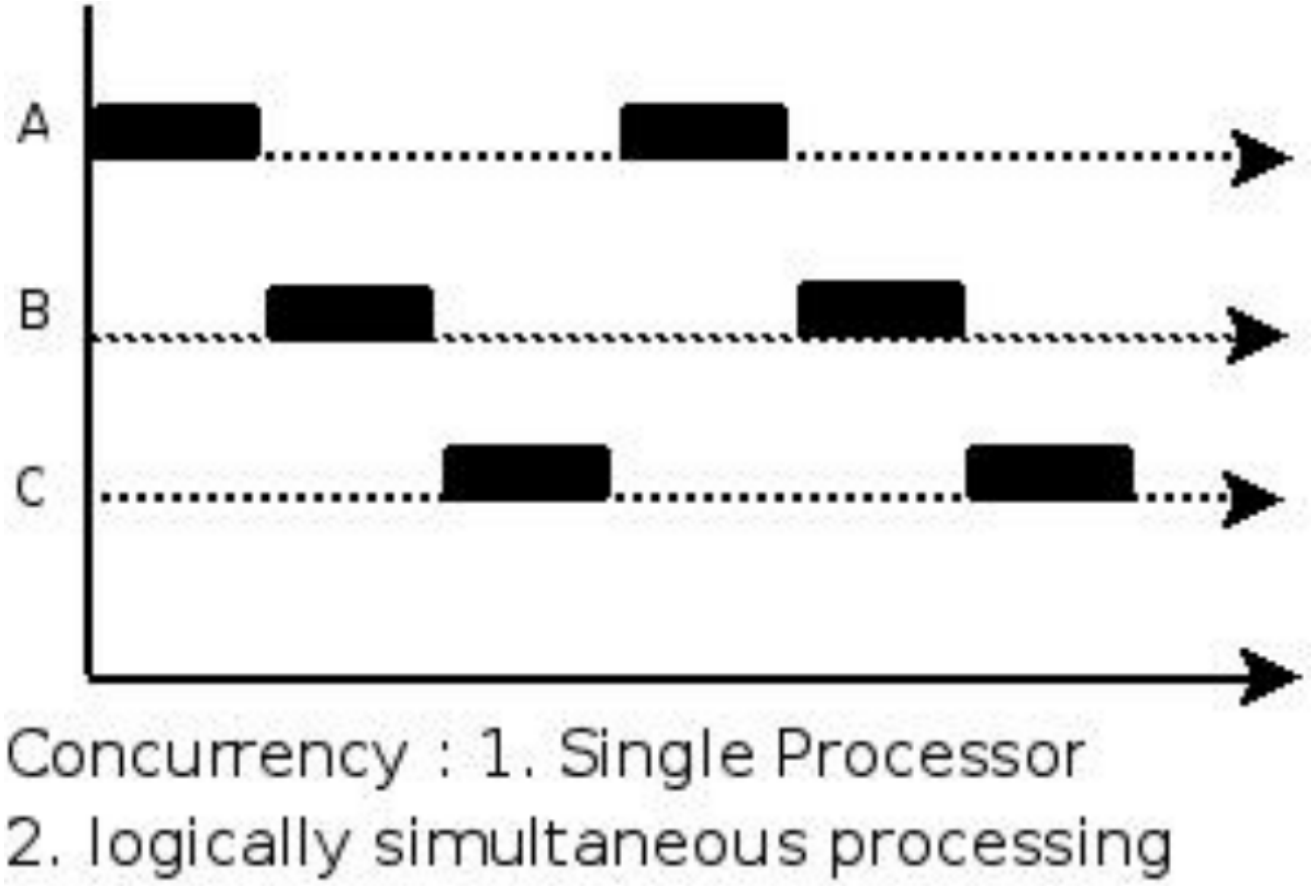
A 12：00------ A处理机上运行

B 12：00------B处理机上运行



2、并发

指在同一时刻只能有一条指令执行，但多个进程指令被快速的轮换执行，使得在宏观上具有多个进程同时执行的效果，但在微观上并不是同时执行的，只是把时间分成若干段，使多个进程快速交替的执行。



go在并发上面的优势

有人把go语言比作21世纪的C语言，第一因为go语言设计简单，第二，21世纪最重要的就是并发程序设计，而go语言底层就是支持了并发和并行。同时 并发程序内存管理有时候非常复杂，而go语言提供了自动垃圾回收机制。

go语言并发编程内置了上层API基于顺序通信模型。避免锁的麻烦。因为go语言通过相当安全的通道发送和接受数据以实现同步，这就大大简化了并发程序的编写。

一般情况下，一个普通的桌面计算机跑十几个二十几个线程有点负载过大，但是同样这台机器可以轻松的跑上**上千甚至过万个goroutine**进行资源竞争。

goroutine是什么

goroutine是go并发设计的核心。goroutine说到底就是协程。但是它比线程更小，十几个goroutine可能体现在底层就是5~6个线程。go语言内部帮我们实现这些goroutine的共享内存。执行goroutine只需要极少的栈内存（大概4~5KB），当然会根据相应的数据进行伸缩，也正是如此，可以运行上万个并发任务。goroutine比thread更易用，更高效 更轻便。

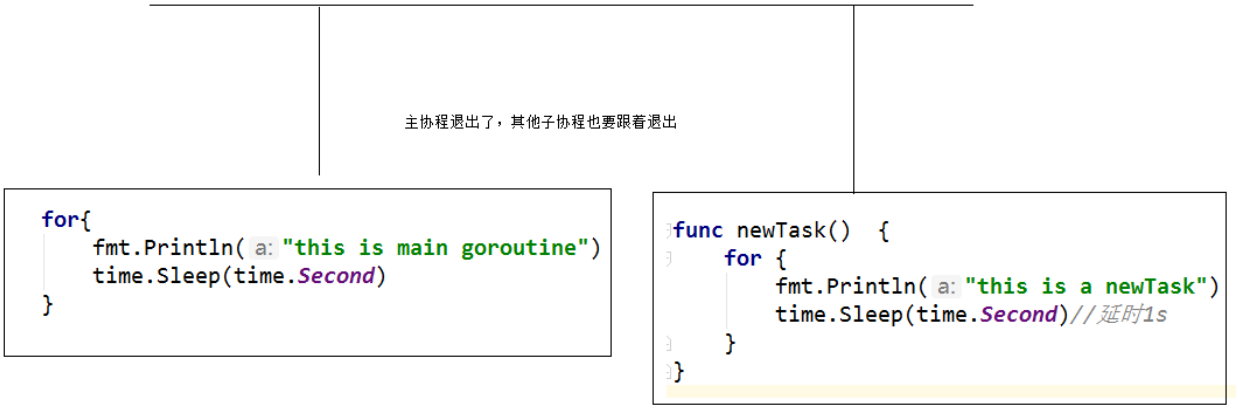
创建goroutine

只需要在函数调用语句前添加**go**关键字，就可创建并发执行单元。开发人员不用java了解任何机制，调度器就会将安排到合适的系统上执行。

在并发编程里面。我们通常想将一个过程切分成几块，然后让每个goroutine各自负责一块工作。当一个程序启动时，其主函数在一个单独的goroutine里面运行，这个主函数我们叫main goroutine。 新的goroutine会用go语句来实现。

```
package main

import (
   "fmt"
   "time"
)

func newTask()  {
   for {
      fmt.Println("this is a newTask")
      time.Sleep(time.Second)//延时1s
   }
}


func main()  {
   go newTask() //go 关键字就新建一个协程，新建一个任务

   for{
      fmt.Println("this is main goroutine")
      time.Sleep(time.Second)
   }

}
```

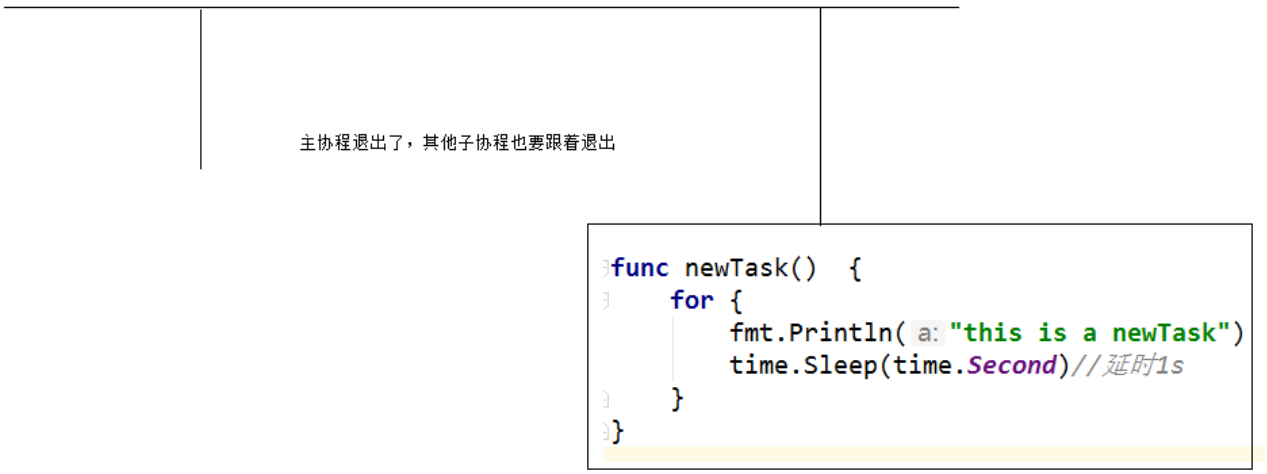
主goroutine退出 子的也就退出




```go
package main

import (
   "fmt"
   "time"
)

func newTask()  {
   for {
      fmt.Println("this is a newTask")
      time.Sleep(time.Second)//延时1s
   }
}


func main()  {
   go newTask() //go 关键字就新建一个协程，新建一个任务

   /*for{
      fmt.Println("this is main goroutine")
      time.Sleep(time.Second)
   }*/
   i := 0
   for{
        i++
      fmt.Println("this is main goroutine")
      time.Sleep(time.Second)
      if i==2{
         break
      }

   }

}
```

通过匿名创建案例

```
package main

import (
   "fmt"
   "time"
)
//主程序退出 子程序跟着退出，子程序退出需要时间
func main(){
   go func() {
      i := 0
      for {
         i++
         fmt.Println("子协程 i=",i)
         time.Sleep(time.Second)
      }
   }()

   i := 0
   for{
      i++
      fmt.Println("main i=",i)
      time.Sleep(time.Second)
      if i == 2{
         break
      }
   }
}
```



主协程先退出导致子协程没有来的及调用

```
package main

import (
   "fmt"
   "time"
)
//主程序退出 子程序跟着退出，导致子程序没有调用
func main(){
   go func() {
      i := 0
      for {
         i++
         fmt.Println("子协程 i=",i)
         time.Sleep(time.Second)
      }
   }()


}
```

goshed的使用

让出时间片，先让别的协程执行，它执行完，最回来执行此协程

```
package main

import (
   "fmt"
   "runtime"
)

func main(){
   go func(){
      for i:=0;i<5;i++{
         fmt.Println("go")
      }
   }()

   for i:=0;i<2;i++{
      runtime.Gosched()
      fmt.Println("hello")
      //让出时间片，先让别的协程执行，它执行完，最回来执行此协程

      //time.Sleep(time.Second)
```

```
    }


}
```

GOexit的使用

终止协程

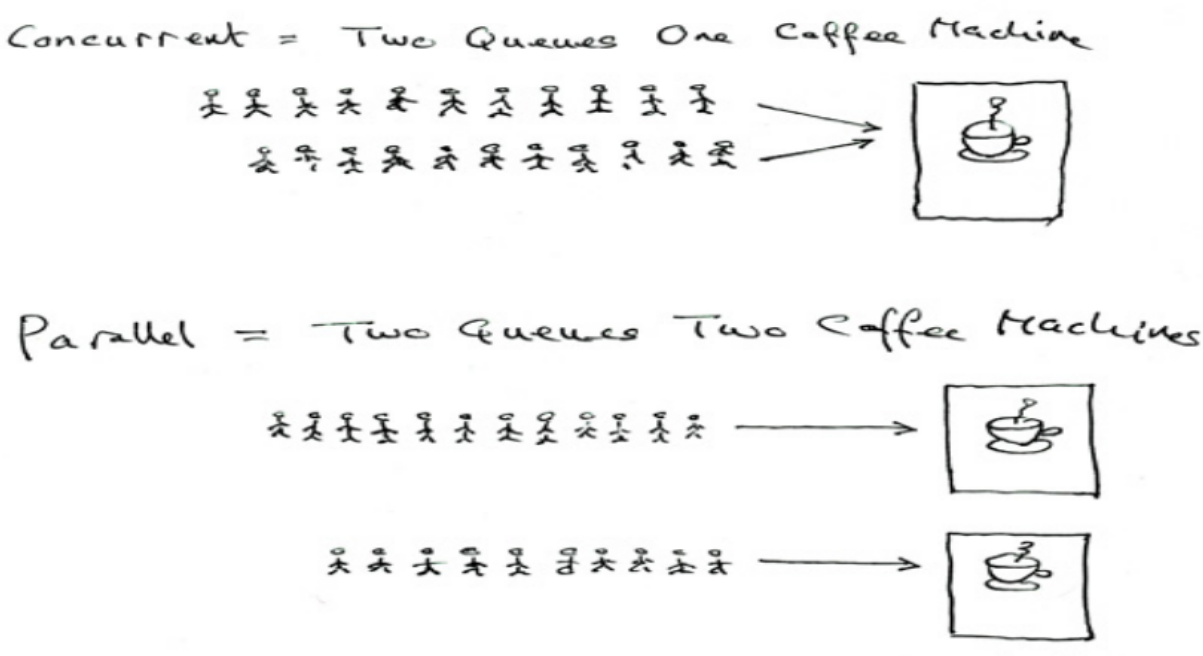
```
package main

import (
    "fmt"
    "runtime"
)

func test()  {
    defer fmt.Println("ccccccc")
    //return  //终止函数
    runtime.Goexit() //终止所有的协程
    fmt.Println("ddddddddd")
}

func main(){
    //创建新的协程
    go func() {
        fmt.Println("aaaaaaa")
          test()
        fmt.Println("bbbbbbbb")
    }()

     for{

     }


}
```

资源抢占问题

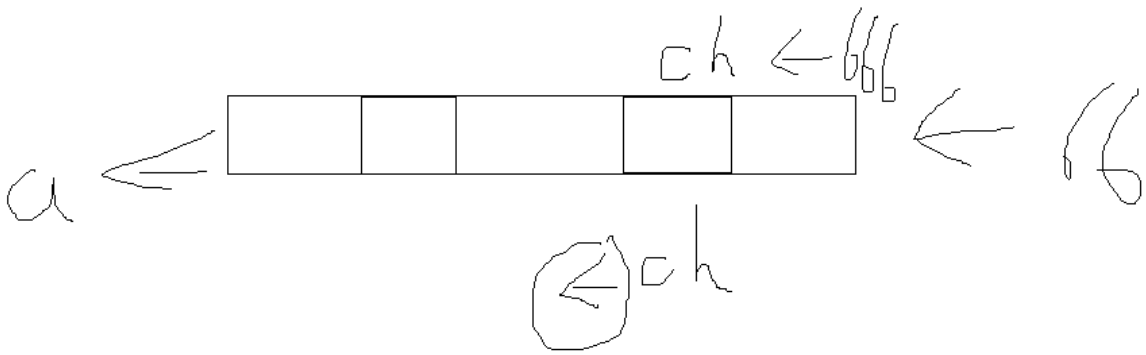
```
package main

import (
    "fmt"
    "time"
```

```
)

//定义一个打印机，参数字符串，按照每个字符打印
func Printer(str string)  {
   for _,data :=range str{
      fmt.Printf("%c",data)
      time.Sleep(time.Second)
   }
   fmt.Printf("\n")
}

func Person1()  {
  Printer("hello")
}

func Person2()  {
   Printer("world")
}
func main(){
   //2个协程共有一个资源
   go Person1()
   go Person2()

   for{

   }

}
```

并发和并行图表示



同过channel实现同步



在协程直接通信我们一般使用chan，实现同步问题

```
package main

import (
   "time"
   "fmt"
)
//
var ch = make(chan int)

//定义一个打印机，参数字符串，按照每个字符打印
func Printer1(str string)  {
   for _,data :=range str{
      fmt.Printf("%c",data)
      time.Sleep(time.Second)
   }
   fmt.Printf("\n")
}

//person3 执行完之后 就会执行person4 ，在协程直接通信我们一般使用chan
func person3(){
  Printer1("hello")
  ch <- 666 //给管道写数据，发送
}

func person4()  {
    <-ch //从管道里面取数据，接收 如果通道里面没有数据就会阻塞
   Printer1("world")

}
func main()  {
```

```
    go person3()
    go person4()

    for{

    }
}
```