Go语言 Goroutine 协程（轻量级线程）

Goroutine 介绍

goroutine 是一种非常轻量级的实现，由 Go 运行时（runtime）管理。Go 程序会智能地将 goroutine 中的任务合理地分配给每个 CPU。可在单个进程里执行成千上万的并发任务，它是Go语言并发设计的核心。

说到底 goroutine 其实就是线程，但是它比线程更小，十几个 goroutine 可能体现在底层就是五六个线程，而且Go语言内部也实现了 goroutine 之间的内存共享。

使用 go 关键字就可以创建 goroutine，将 go 声明放到一个需调用的函数之前，在相同地址空间调用运行这个函数，这样该函数执行时便会作为一个独立的并发线程，这种线程在Go语言中则被称为 goroutine。

Go 程序从 main 包的 main() 函数开始，在程序启动时，Go 程序就会为 main() 函数创建一个默认的 goroutine。

创建 goroutine

Golang 中的并发是**函数**相互独立运行的能力。**Goroutines** 是并发运行的函数。Golang 提供了 Goroutines 作为并发处理操作的一种方式。

Go 程序中使用 **go** 关键字为一个函数创建一个 goroutine。一个函数可以被创建多个 goroutine，一个 goroutine 必定对应一个函数。

格式

为一个普通函数创建 goroutine 的写法如下：

```
go 函数名( 参数列表 )
```

- 函数名：要调用的函数名。
- 参数列表：调用函数需要传入的参数。

使用 go 关键字创建 goroutine 时，被调用函数的**返回值会被忽略。**

如果需要在 goroutine 中返回数据，请使用后面介绍的通道（channel）特性，通过通道把数据从 goroutine 中作为返回值传出。

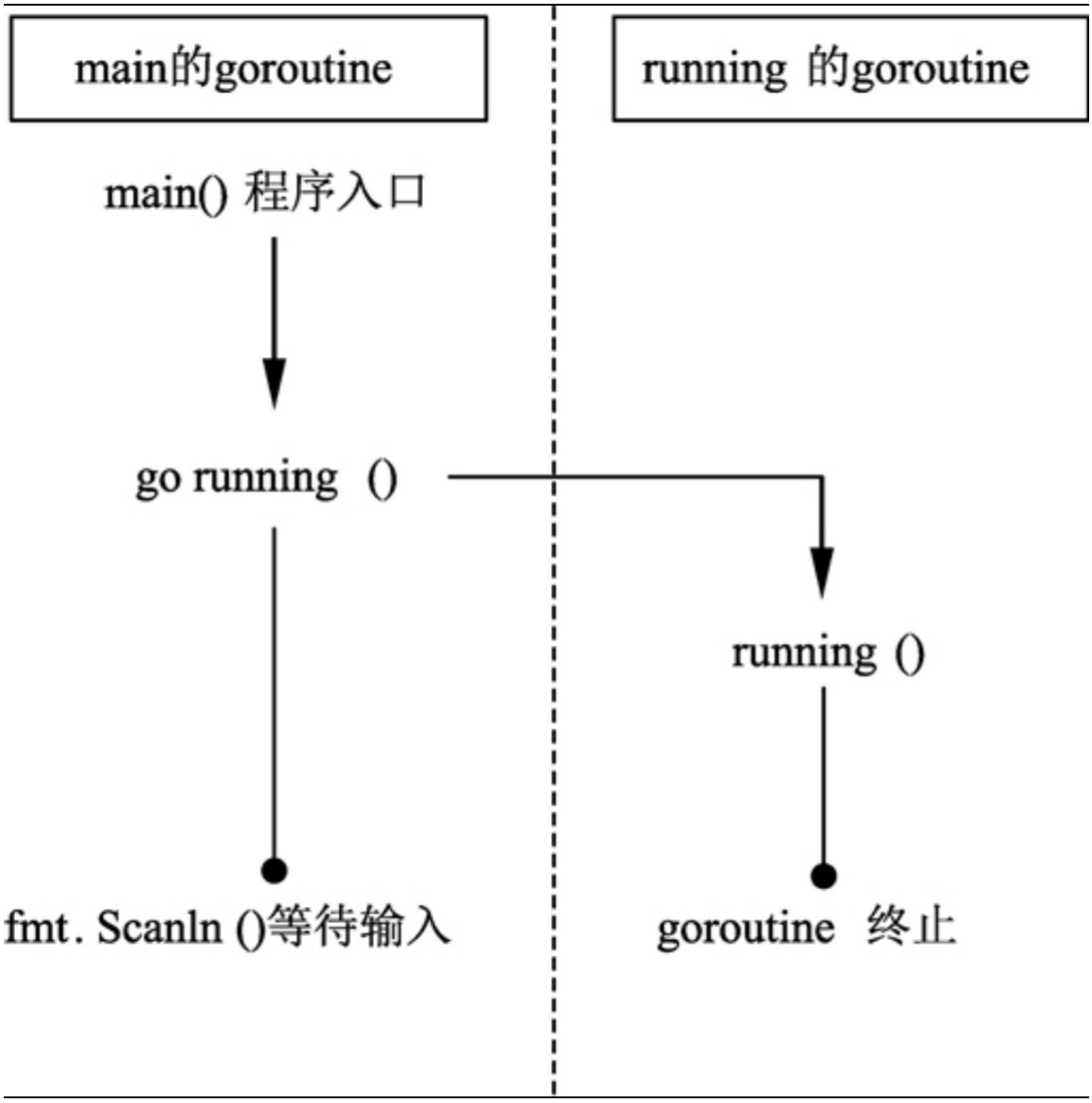
示例：使用 go 关键字，将 running() 函数并发执行，每隔一秒打印一次计数器，而 main 的 goroutine 则等待用户输入，两个行为可以同时进行。

```
package main

import (
    "fmt"
    "time"
)

func running() {
    var times int
```

```
    // 构建一个无限循环
    for {
        times++
        fmt.Println("tick", times)

        // 延时1秒
        time.Sleep(time.Second)
    }
}

func main() {
    // 并发执行程序
    go running()

    // 接受命令行输入，不做任何事情
    var input string
    fmt.Scanln(&input)
}
```

代码执行后，命令行会不断地输出 tick，同时可以使用 fmt.Scanln() 接受用户输入。两个环节可以同时进行。

这段代码的执行顺序如下图所示。



这个例子中，Go 程序在启动时，运行时（runtime）会默认为 main() 函数创建一个 goroutine。在 main() 函数的 goroutine 中执行到 go running 语句时，归属于 running() 函数的 goroutine 被创建，running() 函数开始在自己的 goroutine 中执行。此时，main() 继续执行，两个 goroutine 通过 Go 程序的调度机制同时运作。

使用匿名函数创建goroutine

go 关键字后也可以为匿名函数或闭包启动 goroutine

格式

使用匿名函数或闭包创建 goroutine 时，除了将函数定义部分写在 go 的后面之外，还需要加上匿名函数的调用参数，格式如下：

```
go func( 参数列表 ){
    函数体
}( 调用参数列表 )
```

- 参数列表：函数体内的参数变量列表。
- 函数体：匿名函数的代码。
- 调用参数列表：启动 goroutine 时，需要向匿名函数传递的调用参数。

示例：在 main() 函数中创建一个匿名函数并为匿名函数启动 goroutine。匿名函数没有参数。代码将并行执行定时打印计数的效果。

```
package main

import (
    "fmt"
    "time"
)

func main() {
    // 并发执行程序
    go func() {
        var times int
        // 构建一个无限循环
        for {
            times++
            fmt.Println("tick", times)
            // 延时1秒
            time.Sleep(time.Second)
        }
    }() // 匿名函数调用

    // 接受命令行输入，不做任何事情
    var input string
    fmt.Scanln(&input)
}
```

代码执行后，命令行会不断地输出 tick，同时可以使用 fmt.Scanln() 接受用户输入。两个环节可以同时进行。

提示

- 所有 goroutine 在 main() 函数结束时会一同结束。
- goroutine 虽然类似于线程概念，但是从调度性能上没有线程细致，而细致程度取决于 Go 程序的 goroutine 调度器的实现和运行环境。
- 终止 goroutine 的最好方法就是自然返回 goroutine 对应的函数。虽然可以用 golang.org/x/net/context 包进行 goroutine 生命期深度控制，但这种方法仍然处于内部试验阶段，并不是官方推荐的特性。
- 截止 Go 1.9 版本，暂时没有标准接口获取 goroutine 的 ID。