

17 | go语句及其执行规则（下）

2018-09-19 郝林

Go语言核心36讲

[进入课程 >](#)



讲述：黄洲君

时长 09:31 大小 4.36M



你好，我是郝林，今天我们继续分享 go 语句执行规则的内容。


在上一篇文章中，我们讲到了 goroutine 在操作系统的并发编程体系，以及在 Go 语言并发编程模型中的地位和作用等一系列内容，今天我们继续来聊一聊这个话题。

知识扩展

问题 1：怎样才能让主 goroutine 等待其他 goroutine？

我刚才说过，一旦主 goroutine 中的代码执行完毕，当前的 Go 程序就会结束运行，无论其他的 goroutine 是否已经在运行了。那么，怎样才能做到等其他的 goroutine 运行完毕之后，再让主 goroutine 结束运行呢？

其实有很多办法可以做到这一点。其中，最简单粗暴的办法就是让主 goroutine “小睡” 一会儿。

 复制代码

```
1 for i := 0; i < 10; i++ {  
2     go func() {  
3         fmt.Println(i)  
4     }()  
5 }  
6 time.Sleep(time.Millisecond * 500)
```

在for语句的后边，我调用了time包的Sleep函数，并把time.Millisecond * 500的结果作为参数值传给了它。time.Sleep函数的功能就是让当前的 goroutine（在这里就是主 goroutine）暂停运行一段时间，直到到达指定的恢复运行时间。

我们可以把一个相对的时间传给该函数，就像我在这里传入的“500 毫秒”那样。

time.Sleep函数会在被调用时用当前的绝对时间，再加上相对时间计算出在未来的恢复运行时间。显然，一旦到达恢复运行时间，当前的 goroutine 就会从“睡眠”中醒来，并开始继续执行后边的代码。

这个办法是可行的，只要“睡眠”的时间不要太短就好。不过，问题恰恰就在这里，我们让主 goroutine “睡眠”多长时间才是合适的呢？如果“睡眠”太短，则很可能不足以让其他的 goroutine 运行完毕，而若“睡眠”太长则纯属浪费时间，这个时间就太难把握了。

你可能会想到，既然不容易预估时间，那我们就让其他的 goroutine 在运行完毕的时候告诉我们好了。这个思路很好，但怎么做呢？

你是否想到了通道呢？我们先创建一个通道，它的长度应该与我们手动启用的 goroutine 的数量一致。在每个手动启用的 goroutine 即将运行完毕的时候，我们都要向该通道发送一个值。

注意，这些发送表达式应该被放在它们的go函数体的最后面。对应的，我们还需要在main函数的最后从通道接收元素值，接收的次数也应该与手动启用的 goroutine 的数量保持一致。关于这些你可以到 demo39.go 文件中，去查看具体的写法。

其中有一个细节你需要注意。我在声明通道`sign`的时候是以`chan struct{}`作为其类型的。其中的类型字面量`struct{}`有些类似于空接口类型`interface{}`，它代表了既不包含任何字段也不拥有任何方法的空结构体类型。

注意，`struct{}`类型值的表示法只有一个，即：`struct{}{}`。并且，它占用的内存空间是0字节。确切地说，这个值在整个 Go 程序中永远都只会存在一份。虽然我们可以无数次地使用这个值字面量，但是用到的却都是同一个值。

当我们仅仅把通道当作传递某种简单信号的介质的时候，用`struct{}`作为其元素类型是再好不过的了。顺便说一句，我在讲“结构体及其方法的使用法门”的时候留过一道与此相关的思考题，你可以返回去看一看。

再说回当下的问题，有没有比使用通道更好的方法？如果你知道标准库中的代码包`sync`的话，那么可能会想到`sync.WaitGroup`类型。没错，这是一个更好的答案。不过具体的使用方式我在后边讲`sync`包的时候再说。

问题 2：怎样让我们启用的多个 goroutine 按照既定的顺序运行？

在很多时候，当我沿着上面的主问题以及第一个扩展问题一路问下来的时候，应聘者往往会被这第二个扩展问题难住。

所以基于上一篇主问题中的代码，怎样做到让从0到9这几个整数按照自然数的顺序打印出来？你可能会说，我不用 goroutine 不就可以了嘛。没错，这样是可以，但是如果我不考虑这样做呢。你应该怎么解决这个问题？

当然了，众多应聘者回答的其他答案也是五花八门的，有的可行，有的不可行，还有的把原来的代码改得面目全非。我下面就说说我的思路，以及心目中的答案吧。这个答案并不一定是最佳的，也许你在看完之后还可以想到更优的答案。


首先，我们需要稍微改造一下`for`语句中的那个`go`函数，要让它接受一个`int`类型的参数，并在调用它的时候把变量`i`的值传进去。为了不改动这个`go`函数中的其他代码，我们可以把这个参数也命名为`i`。

```
2     go func(i int) {
3         fmt.Println(i)
4     }(i)
5 }
```

只有这样，Go 语言才能保证每个 goroutine 都可以拿到一个唯一的整数。其原因与go函数的执行时机有关。

我在前面已经讲过了。在go语句被执行时，我们传给go函数的参数*i*会先被求值，如此就得到了当次迭代的序号。之后，无论go函数会在什么时候执行，这个参数值都不会变。也就是说，go函数中调用的fmt.Println函数打印的一定会是那个当次迭代的序号。

然后，我们在着手改造for语句中的go函数。

 复制代码

```
1 for i := uint32(0); i < 10; i++ {
2     go func(i uint32) {
3         fn := func() {
4             fmt.Println(i)
5         }
6         trigger(i, fn)
7     }(i)
8 }
```

我在go函数中先声明了一个匿名的函数，并把它赋给了变量fn。这个匿名函数做的事情很简单，只是调用fmt.Println函数以打印go函数的参数*i*的值。

在这之后，我调用了个名叫trigger的函数，并把go函数的参数*i*和刚刚声明的变量fn作为参数传给了它。注意，for语句声明的局部变量*i*和go函数的参数*i*的类型都变了，都由int变为了uint32。至于为什么，我一会儿再说。

再来说trigger函数。该函数接受两个参数，一个是uint32类型的参数*i*，另一个是func()类型的参数fn。你应该记得，func()代表的是既无参数声明也无结果声明的函数类型。

```
1 trigger := func(i uint32, fn func()) {  
2     for {  
3         if n := atomic.LoadUint32(&count); n == i {  
4             fn()  
5             atomic.AddUint32(&count, 1)  
6             break  
7         }  
8         time.Sleep(time.Nanosecond)  
9     }  
10 }
```

`trigger`函数会不断地获取一个名叫`count`的变量的值，并判断该值是否与参数`i`的值相同。如果相同，那么就立即调用`fn`代表的函数，然后把`count`变量的值加1，最后显式地退出当前的循环。否则，我们就先让当前的 `goroutine` “睡眠” 一个纳秒再进入下一个迭代。

注意，我操作变量`count`的时候使用的都是原子操作。这是由于`trigger`函数会被多个 `goroutine` 并发地调用，所以它用到的非本地变量`count`，就被多个用户级线程共用了。因此，对它的操作就产生了竞态条件（`race condition`），破坏了程序的并发安全性。

所以，我们总是应该对这样的操作加以保护，在`sync/atomic`包中声明了很多用于原子操作的函数。

另外，由于我选用的原子操作函数对被操作的数值的类型有约束，所以我对`count`以及相关的变量和参数的类型进行了统一的变更（由`int`变为了`uint32`）。

纵观`count`变量、`trigger`函数以及改造后的`for`语句和`go`函数，我要做的是，让`count`变量成为一个信号，它的值总是下一个可以调用打印函数的`go`函数的序号。

这个序号其实就是启用 `goroutine` 时，那个当次迭代的序号。也正因为如此，`go`函数实际的执行顺序才会与`go`语句的执行顺序完全一致。此外，这里的`trigger`函数实现了一种自旋（`spinning`）。除非发现条件已满足，否则它会不断地进行检查。

最后要说的是，因为我依然想让主 `goroutine` 最后一个运行完毕，所以还需要加一行代码。不过既然有了`trigger`函数，我就没有再使用通道。


```
1 trigger(10, func(){}))
```

调用`trigger`函数完全可以达到相同的效果。由于当所有我手动启用的 `goroutine` 都运行完毕之后，`count`的值一定会是10，所以我就把10作为了第一个参数值。又由于我并不想打印这个10，所以我把一个什么都不做的函数作为了第二个参数值。

总之，通过上述的改造，我使得异步发起的`go`函数得到了同步地（或者说按照既定顺序地）执行，你也可以动手自己试一试，感受一下。

总结

在本篇文章中，我们接着上一篇文章的主问题，讨论了当我们想让运行结果更加可控的时候，应该怎样去做。

主 `goroutine` 的运行若过早结束，那么我们的并发程序的功能就很可能无法全部完成。所以我们往往需要通过一些手段去进行干涉，比如调用`time.Sleep`函数或者使用通道。我们在后面的文章中还会讨论更高级的手段。

另外，`go`函数的实际执行顺序往往与其所属的`go`语句的执行顺序（或者说 `goroutine` 的启用顺序）不同，而且默认情况下的执行顺序是不可预知的。那怎样才能让这两个顺序一致呢？其实复杂的实现方式有不少，但是可能会把原来的代码改得面目全非。我在这里提供了一种比较简单、清晰的改造方案，供你参考。

总之，我希望通过上述基础知识以及三个连贯的问题帮你串起一条主线。这应该会让你更快地深入理解 `goroutine` 及其背后的并发编程模型，从而更加游刃有余地使用`go`语句。

思考题

1. `runtime`包中提供了哪些与模型三要素 G、P 和 M 相关的函数？（模型三要素内容在上一篇）

[戳此查看 Go 语言专栏文章配套详细代码。](#)

GO语言核心36讲

3个月带你通关GO语言

郝林

《Go 并发编程实战》作者
GoHackers 技术社群发起人
前轻松筹大数据负责人



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 16 | go语句及其执行规则（上）

下一篇 18 | if语句、for语句和switch语句

精选留言 (32)

写留言



来碗绿豆汤

2018-09-19

16

我有一个更简单的实现方式，如下

```
func main(){
    ch := make(chan struct{})
    for i:=0; i < 100; i++{
        go func(i int){...
```

展开 ▾

作者回复: 这些go函数的真正执行谁先谁后是不可控的，所以这样做不行的。



xiao豪

2018-09-19

👍 12

回楼上，atomic的加操作和读操作只有32位和64位整数型，所以必须要把int转为intxx。之所以这么做是因为int位数是根据系统决定的，而原子级操作要求速度尽可能的快，所以明确了整数的位数才能最大地提高性能。



AskerIve

2018-09-19

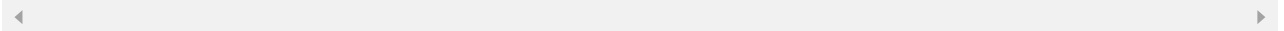
👍 8

package main

```
import (  
    "fmt"  
    "sync/atomic"...
```

展开 ▾

作者回复: 可以加个sleep



AskerIve

2018-09-19

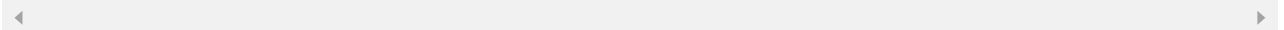
👍 4

package main

```
import (  
    "fmt"  
    "sync/atomic"...
```

展开 ▾

作者回复: Win下可能会有问题，你在bif语句后边加一句time.sleep(time.Nanosecond)。github上的代码我已经更新了。



新垣结裤

2018-09-21

👍 3

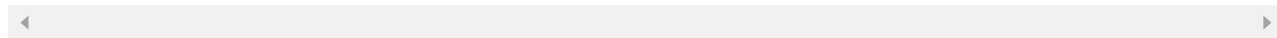
```
func main() {  
    num := 10  
    chs := [num+1]chan struct{}}
```



```
for i := 0; i < num+1; i++ {  
    chs[i] = make(chan struct{})...
```

展开 ▾

作者回复: 搞这么多通道有些浪费啊。另外切片不是并发安全的数据类型，最好不要这样用。



嗽大猫的鱼

2018-09-20

👍 3

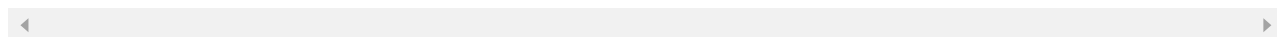
老师，最近从头学习，前面一直没跟着动手，也没自己总结。这几天在整理每章的重点！

<https://github.com/wenxuwan/go36>

刚写完第二章，突然发现自己动手总结和只看差好多。我会继续保持喜欢总结！

展开 ▾

作者回复: 很好，加油！



老茂

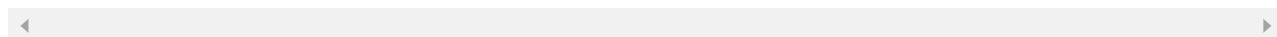
2018-10-15

👍 2

不加sleep程序不能正常结束的情况貌似跟cpu核数有关，我是4核cpu，打印0到2每次都可以正常执行；0到3以上就会有卡主的情况，卡主时cpu达到100%，load会超过4。猜测是不是此时所有cpu都在处理count==0的for循环，没有空闲的cpu执行atomic.AddUint32(&count, 1)？

展开 ▾

作者回复: Go语言调度goroutine是准抢占式的，虽然会防止某个goroutine运行太久，并做换下处理。但是像简单的死循环这种有可能会换下失败，尤其是windows下，这跟操作系统的底层支持有关。不过一般情况下不用担心。



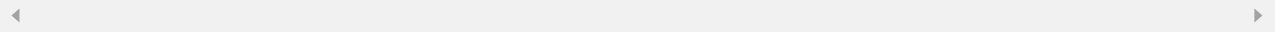
cygnus

2018-09-19

👍 2

demo40的执行结果不是幂等的，程序经常无法正常结束退出，只有极少数几次有正确输出。

作者回复: 你在win下执行的嘛？



sky

2018-09-19

👍 2

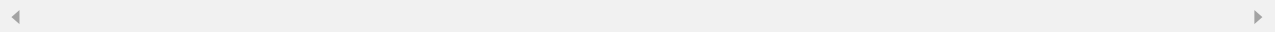
win64版本：go1.10.2

linux64版本：go1.11

linux下实际运行和预期一样，但为何win下会一直运行不会停止呢，且CPU也已经是100%表示不解呀

展开 ▾

作者回复: 可以加个sleep。



冰激凌的眼...

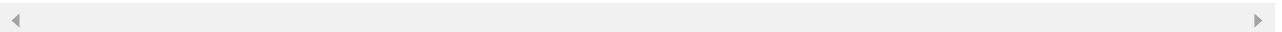
2018-09-19

👍 2

“ 否则，我们就先让当前的 goroutine “睡眠” 一个纳秒再进入下一个迭代。 ”

示例代码里没有这个睡眠代码

作者回复: 代码已经更新了。



志鑫

2019-05-11

👍 1

//个人笔记：使用一个通道来控制
package main

import "fmt"

...

展开 ▾



枫林火山



2019-04-01



老师，关于顺序打印的demo40.go的优化版本，同来碗绿豆汤同学的实现

```
package main
```

```
import "fmt"
```

```
...
```

展开 ▾

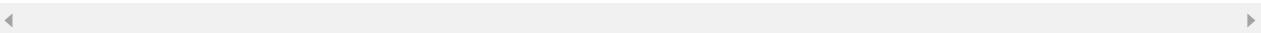
作者回复: 我又看了一下“来碗绿豆汤”同学写的代码。我可能当时没看清楚，或者没说清楚。

他写的这段代码单从“顺序打印数字”的要求上看是可以的。但是这样做就变成纯同步的流程了，go函数就完全没必要写了。把go函数中的代码拿出来、删掉go函数，再把通道的相关代码也删掉，岂不是更直截了当？像这样：

```
for i := 0; i < num; i++ {  
    fmt.Println(i)  
}
```

这个题目的要求是“使得在for循环中启用的多个goroutine按照既定的顺序运行”。你也可以把它理解为“在异步的情况下顺序的打印数字”。所以，“来碗绿豆汤”同学写的代码只满足了其中一个要求，而没有让go函数们自由的异步执行。

我的那个版本demo40.go是让各个go函数（确切地说，是它们调用的trigger函数）自行地检查所需条件，然后再在条件允许的情况下打印数字。这也叫“自旋”。这与纯同步的流程是有本质上的区别的。



Wind

2019-03-05



```
func main(){  
    ch := make(chan struct{})  
    for i:=0; i < 100; i++){  
        go func(i int){  
            fmt.Println(i)...
```

展开 ▾



SuperP ♥...

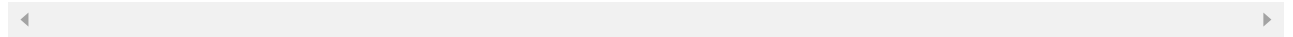
2018-10-11



runtime.GOMAXPROCS 这个应该能控制P的数量

展开 ∨

作者回复: 对，可以。



timmy21

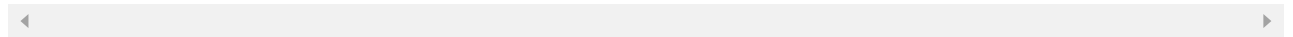
2018-09-19

👍 1

有一个问题不太清楚，当i和count不相等时，您提到了睡眠1纳秒，可是我没看到有相关的sleep被调用。这是如何做到的？

展开 ∨

作者回复: 代码已经更新了，漏掉了。



Zzz

2019-05-15

👍

很多同学都提到在循环内部增加<-ch的方式保证上面的go语句执行完再继续下一次循环。这种方式虽然达到了顺序打印的效果，但是实际上整个程序只需要一个goroutine就可以完成：主函数阻塞时执行go语句，然后再唤醒主函数。

展开 ∨



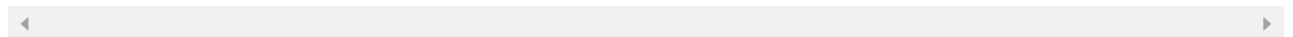
肖恩

2019-05-08

👍

第一遍看好多都看不懂，看到后边回过头来看，发现用自旋goroutine实现，真实奇妙；现在想想，除了文章中实现方式，可以用channel同步实现；还可以用sync.WaitGroup实现

作者回复: 祝贺你升级了；)



Nixus

2019-04-13

👍

```
func main() {  
    ch := make(chan int, 10)
```

```
for i := 0; i < 10; i++ {  
    ch <- i  
    go func() {...
```

展开 ▾

作者回复: 这样做倒是可以顺序打印出数字, 但是跟原题不太对应。另外, 在正式的程序中, 用 `len(ch)` 判断通道是否已经使用完毕不是一个好办法。



左氧佛沙星...

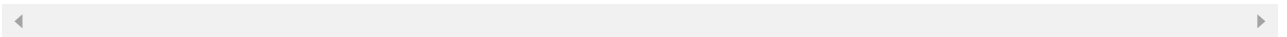
2019-04-06



```
func main() {  
    ch := make(chan int, 100)  
    for i := int(0); i < 100; i++ {  
        go func() {  
            fmt.Println(i)...
```

展开 ▾

作者回复: 你这段代码里的go函数的真正执行顺序实际上是不确定的啊。



枫林火山

2019-04-01



谢谢老师的讲解

展开 ▾