控制goroutine 的并发执行数量

原创 techdashen 于 2023-08-28 22:16:02 发布 阅读量1k 收藏 1 点赞数
文章标签： 后端

goroutine的数量上限是1048575吗?

正常项目， 协程数量超过十万就需要引起重视。如果有上百万goroutine，一般是有问题的。

但并不是说协程数量的上限是100多w

1048575的来自类似如下的demo代码:

```
package main

import (
 "fmt"
 "math"
 "runtime"
 "time"
)

// https://zhuanlan.zhihu.com/p/568151296
func main() {

 maxCount := math.MaxInt64
 for i := 0; i < maxCount; i++ {
  go func(i int) {
```

```
        fmt.Printf("i is: %d,goroutine num: %d\n", i, runtime.NumGoroutine())

        // 模拟各种耗时较长的业务逻辑
        time.Sleep(10 * time.Second)

    }(i)
  }
}
```

执行后，很快报错

```
i is: 69583,goroutine num: 78851
i is: 69395,goroutine num: 78734
i is: 1116929,goroutine num: 1125330
i is: 1117057,goroutine num: 1125330
i is: 1116932,goroutine num: 1125330
i is: 69474,goroutine num: 78854
panic: too many concurrent operations on a single file or socket (max 1048575)

goroutine 1118001 [running]:
internal/poll.(*fdMutex).rwlock(0x14000124060, 0x60?)
        /Users/fliter/.g/go/src/internal/poll/fd_mutex.go:147 +0x134
internal/poll.(*FD).writeLock(...)
        /Users/fliter/.g/go/src/internal/poll/fd_mutex.go:239
internal/poll.(*FD).Write(0x14000124060, {0x141428fe9c0, 0x25, 0x40})
        /Users/fliter/.g/go/src/internal/poll/fd_unix.go:370 +0x48
```

`panic: too many concurrent operations on a single file or socket (max 1048575)`

但这个是因为fmt.Printf导致的:

对单个*file/socket*的并发操作数超过了系统上限，这个是标准输出造成的，具体一点，就是文件句柄数量达到限制

如下例子，去掉fmt:

```
1
```

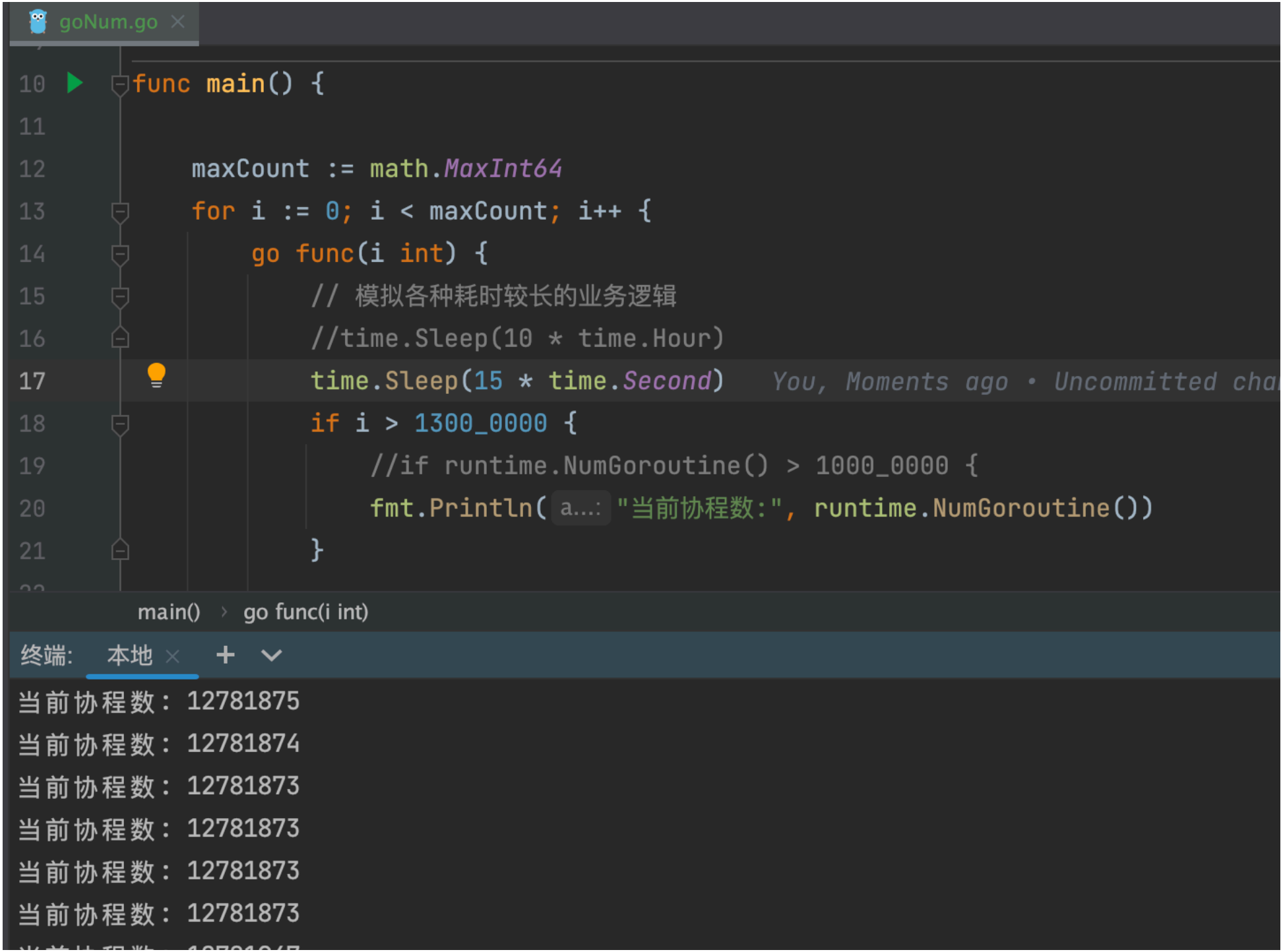
```go
package main

import (
 "fmt"
 "math"
 "runtime"
 "time"
)

func main() {

 maxCount := math.MaxInt64
 for i := 0; i < maxCount; i++ {
  go func(i int) {
   // 模拟各种耗时较长的业务逻辑
   //time.Sleep(10 * time.Hour)
   time.Sleep(15 * time.Second)
   if i > 1300_0000 {
    //if runtime.NumGoroutine() > 1000_0000 {
    fmt.Println("当前协程数:", runtime.NumGoroutine())
   }

  }(i)
 }
}
```


goNum.go
func main() {
maxCount := math.MaxInt64
for i := 0; i < maxCount; i++ {
go func(i int) {
// 模拟各种耗时较长的业务逻辑
//time.Sleep(10 * time.Hour)
time.Sleep(15 * time.Second)
You, Moments ago • Uncommitted cha
if i > 1300_0000 {
//if runtime.NumGoroutine() > 1000_0000 {
fmt.Println(a...: "当前协程数:", runtime.NumGoroutine())
}
main() › go func(i int)
终端: 本地
当前协程数： 12781875
当前协程数： 12781874
当前协程数： 12781873
当前协程数： 12781873
当前协程数： 12781873
当前协程数： 12781873

```
当前协程数：12781947
当前协程数：12781873
当前协程数：12781871
panic: too many concurrent operations on a single file or socket (max 1048575)

goroutine 14357465 [running]:
internal/poll.(*fdMutex).rwlock(0x14000124060, 0x20?)
        /Users/fliter/.g/go/src/internal/poll/fd_mutex.go:147 +0x134
internal/poll.(*FD).writeLock(...)
```

实际同一时间可以出现1000w的goroutine，可见goroutine的理论上限绝对不止100w

或者如下:

```go
package main

import (
 "fmt"
 "math"
 "runtime"
 "time"
)

func main() {

 maxCount := math.MaxInt64
 for i := 0; i < maxCount; i++ {
  go func(i int) {
   // 模拟各种耗时较长的业务逻辑
   //time.Sleep(10 * time.Hour)
   time.Sleep(15 * time.Second)
   //if i > 1300_0000 {
   if runtime.NumGoroutine() > 800_0000 {
    fmt.Println("当前协程数:", runtime.NumGoroutine())
   }

  }(i)
 }
}
```

goNum.go

```go
func main() {

    maxCount := math.MaxInt64
    for i := 0; i < maxCount; i++ {
        go func(i int) {
            // 模拟各种耗时较长的业务逻辑
            //time.Sleep(10 * time.Hour)
            time.Sleep(15 * time.Second)
            //if i > 1300_0000 {
            if runtime.NumGoroutine() > 800_0000 {
                fmt.Println( a...: "当前协程数:", runtime.NumGoroutine())
            }

        }(i)   You, Moments ago • Uncommitted changes
    }
}
```

main()

终端: 本地

```
当前协程数： 9885514
当前协程数： 9885523
当前协程数： 9885510
当前协程数： 9886579
panic: too many concurrent operations on a single file or socket (max 1048575)
```

```
goroutine 1231546 [running]:
internal/poll.(*fdMutex).rwlock(0x140000a2060, 0x20?)
        /Users/fliter/.g/go/src/internal/poll/fd_mutex.go:147 +0x134
internal/poll.(*FD).writeLock(...)
        /Users/fliter/.g/go/src/internal/poll/fd_mutex.go:239
internal/poll.(*FD).Write(0x140000a2060, {0x14635532bc0, 0x19, 0x20})
```

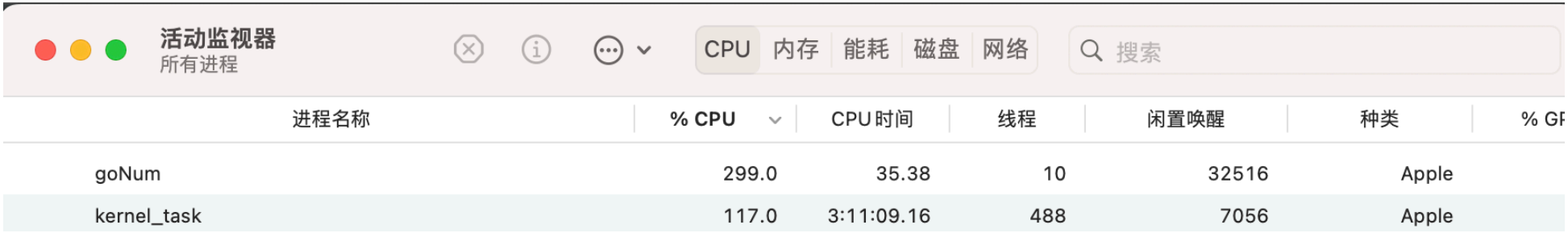
```
panic: too many concurrent operations on a single file or socket (max 1048575)

goroutine 1231546 [running]:
internal/poll.(*fdMutex).rwlock(0x140000a2060, 0x20?)
        /Users/fliter/.g/go/src/internal/poll/fd_mutex.go:147 +0x134
internal/poll.(*FD).writeLock(...)
        /Users/fliter/.g/go/src/internal/poll/fd_mutex.go:239
internal/poll.(*FD).Write(0x140000a2060, {0x14635532bc0, 0x19, 0x20})
        /Users/fliter/.g/go/src/internal/poll/fd_unix.go:370 +0x48
os.(*File).write(...)
```

```
        /Users/fliter/.g/go/src/os/file_posix.go:48
os.(*File).Write(0x140000a0008, {0x14635532bc0?, 0x19, 0x10412e25c?})
        /Users/fliter/.g/go/src/os/file.go:175 +0x60
fmt.Fprintln({0x104168cf8, 0x140000a0008}, {0x140bde92f88, 0x2, 0x2})
        /Users/fliter/.g/go/src/fmt/print.go:285 +0x74
fmt.Println(...)
        /Users/fliter/.g/go/src/fmt/print.go:294
main.main.func1(0x0?)
        /Users/fliter/go/src/shuang/0000/goNum.go:20 +0x150
created by main.main
        /Users/fliter/go/src/shuang/0000/goNum.go:14 +0x54
exit status 2
```

比较奇怪的是，如果将模拟各种耗时较长的业务逻辑的`time.Sleep(15 * time.Second)`改为`time.Sleep(10 * time.Hour)`，最终会因为内存过高而`signal: killed`。但此时goroutine数量不够多，触发不了if里面的fmt逻辑，故而不会出现`panic: too many concurrent operations on a single file or socket (max 1048575)`

（而休眠10几s的代码，内存到不了这么大，就已经因为fmt的问题panic了）



控制方式

使用有缓冲的channel，限制并发的协程数量

make(chan struct{}, 300) 创建缓冲区大小为 300 的 channel，在没有被接收的情况下，至多发送 300 个消息则被阻塞。

开启协程前，调用 ch <- struct{}{}，若缓存区满，则阻塞。 协程任务结束，调用 <-ch 释放缓冲区。

```
// 通过channel来控制并发数

package main

import (
 "fmt"
 "math"
 "runtime"
 "time"
)

func main() {

 ch := make(chan struct{}, 300)
 maxCount := math.MaxInt64
 for i := 0; i < maxCount; i++ {
  ch <- struct{}{}
```

```
 go func(i int) {
  //fmt.Printf("i is: %d,go func num: %d\n", i, runtime.NumGoroutine())

  // 模拟各种耗时较长的业务逻辑
  //time.Sleep(10 * time.Hour)
  time.Sleep(15 * time.Second)
  //if i > 1000_0000 {
  //if runtime.NumGoroutine() > 1000_0000 {
  fmt.Println("当前协程数:", runtime.NumGoroutine())
  //}

  //读取channel数据
  <-ch

 }(i)
 }
}
```

```
当前协程数: 301
当前协程数: 301
当前协程数: 301
当前协程数: 301
当前协程数: 301
当前协程数: 301
当前协程数: 301
当前协程数: 301
当前协程数: 301
...
```

同时只有301个协程(每15s，处理301个；限制太少，会大大增加程序执行完成需要的时间，具体限制多少，需要权衡，太大太小可能都有问题)

更多参考:

如何控制**golang**协程的并发数量问题[1]

golang实现并发数控制的方法[2]

golang控制并发数[3]

Golang的并发控制[4]

即所谓的

无缓冲的**channel**可以当成阻塞锁来使用 （**Go**用两个协程交替打印**100**以内的奇偶数）

有缓冲的**channel**通常可以用来控制**goroutine**的数量

来，控制一下 **goroutine** 的并发数量[5]

还有通过协程池，信号量等方式，可参考 【警惕】请勿滥用**goroutine**[6]

aceld-Go是否可以无限**go**？如何限定数量？[7]

参考资料

[1] 如何控制golang 协程的 并发 数量问题: *http://www.manongjc.com/detail/62-ixfkirkdenvuohr.html*

[2] golang实现并发数控制的方法: *http://www.qb5200.com/article/327027.html*

[3] golang控制并发数: *https://blog.csdn.net/weixin_38155824/article/details/128240704*

[4] Golang的并发控制: *https://blog.csdn.net/LINZEYU666/article/details/123020597*

[5] 来，控制一下 goroutine 的并发数量: *https://eddycjy.gitbook.io/golang/di-1-ke-za-tan/control-goroutine*

[6] 【警惕】请勿滥用goroutine: *https://juejin.cn/post/6999807716482875422#heading-5*

[7] aceld-Go是否可以无限go？如何限定数量？: *https://github.com/catandcoder/golang/blob/main/4%E3%80%81Go%E6%98%AF%E5%90%A6%E5%8F%AF%E4%BB%A5%E6%97%A0%E9%99%90go%EF%BC%9F%E5%A6%82%E4%BD%95%E9%99%90%E5%AE%9A%E6%95%B0%E9%87%8F%EF%BC%9F.md*