

Distributed Communication 7th practice

Li Jianhao
lijianhao288@hotmail.com

1 Basics

1.1 Example without goroutine number limit

package: "runtime"
func NumGoroutine() int
return the number of goroutines that currently exist.

package: "sync/atomic"
func LoadUint64(addr *uint64) (val uint64)
Get the value of the uint64 atomically.
func StoreUint64(addr *uint64, val uint64)
Store the value of the uint64 atomically.

package: "math/rand"
func Intn(n int) int
 $[0, n)$

```
package main                                     1
import (                                         2
    "fmt"                                       3
    "math/rand"                               4
    "runtime"                                  5
    "sync"                                      6
    "sync/atomic"                             7
    "time"                                     8
)                                               9
                                                10
var jobQueue = make(chan string, 100)          11
var maxGo uint64                               12
var wg sync.WaitGroup                          13
                                                14
func main() {                                  15
    go goroutineCounter()                     16
                                                17
                                                18
```

```

    start := time.Now()
    wg.Add(1)
    go linkSender()

    wg.Add(1)
    go workerCreator()

    wg.Wait()

    fmt.Println("Max goroutine number:", atomic.LoadUint64(&maxGo))
    duration := time.Since(start)
    fmt.Println("Time:", duration)
}
func goroutineCounter() {
    for {
        n := runtime.NumGoroutine()
        u := uint64(n)
        if u > maxGo {
            atomic.StoreUint64(&maxGo, u)
        }
        time.Sleep(50 * time.Millisecond)
    }
}
func linkSender() {
    defer wg.Done()
    links := []string{}
    var numOfLink = 1000
    for i := 0; i < numOfLink; i++ {
        fakeLink := fmt.Sprintf("http://web%d.com", i)
        links = append(links, fakeLink)
    }
    for _, link := range links {
        jobQueue <- link
    }
    close(jobQueue)
}
func workerCreator() {
    defer wg.Done()
    for link := range jobQueue {
        wg.Add(1)
        go worker(link)
    }
}
func worker(l string) {
    defer wg.Done()
    fmt.Println(linkTest(l))
}
func linkTest(link string) string {
    time.Sleep(500 * time.Millisecond)
    if rand.Intn(2) == 1 {
        return link + ": Good"
    } else {
        return link + ": Bad"
    }
}

```

Listing 1: Without limit

...	1
http://web395.com: Good	2
http://web392.com: Good	3
http://web400.com: Bad	4
http://web397.com: Bad	5
http://web396.com: Bad	6
http://web401.com: Bad	7
http://web402.com: Bad	8
http://web394.com: Bad	9
http://web399.com: Bad	10
Max goroutine number: 1002	11
Time: 507.788444ms	12

The version with select:

package main	1
	2
import (3
"fmt"	4
"math/rand"	5
"runtime"	6
"sync"	7
"sync/atomic"	8
"time"	9
)	10
	11
var jobQueue = make(chan string, 100)	12
var maxGo uint64	13
var wg sync.WaitGroup	14
var stopper = make(chan int)	15
	16
func main() {	17
go goroutineCounter()	18
	19
start := time.Now()	20
wg.Add(1)	21
go linkSender()	22
	23
wg.Add(1)	24
go workerCreator()	25
	26
wg.Wait()	27
stopper <- 0	28
fmt.Println("Max goroutine number:", maxGo)	29
duration := time.Since(start)	30
fmt.Println("Time:", duration)	31
}	32
func goroutineCounter() {	33
for {	34
select {	35
case <- stopper:	36
fmt.Println("goroutineCounter stop")	37
return	38
default:	39
n := runtime.NumGoroutine()	40
u := uint64(n)	41
if u > maxGo {	42
atomic.StoreUint64(&maxGo, u)	43
}	44
time.Sleep(50 * time.Millisecond)	45
}	46

}	47
}	48
func linkSender() {	49
defer wg.Done()	50
links := []string{}	51
var numOfLink = 1000	52
for i := 0; i < numOfLink; i++ {	53
fakeLink := fmt.Sprintf("http://web%d.com", i)	54
links = append(links, fakeLink)	55
}	56
for _, link := range links {	57
jobQueue <- link	58
}	59
close(jobQueue)	60
}	61
}	62
func workerCreator() {	63
defer wg.Done()	64
for link := range jobQueue {	65
wg.Add(1)	66
go worker(link)	67
}	68
}	69
}	70
func worker(l string) {	71
defer wg.Done()	72
fmt.Println(linkTest(l))	73
}	74
}	75
}	76
func linkTest(link string) string {	77
time.Sleep(500 * time.Millisecond)	78
if rand.Intn(2) == 1 {	79
return link + ":_Good"	80
} else {	81
return link + ":_Bad"	82
}	83
}	84

Listing 2: Without limit (Select)

...	1
http://web919.com: Good	2
http://web923.com: Good	3
http://web931.com: Bad	4
http://web966.com: Bad	5
http://web977.com: Bad	6
http://web927.com: Bad	7
http://web946.com: Bad	8
http://web981.com: Bad	9
http://web929.com: Bad	10
goroutineCounter stop	11
Max goroutine number: 1002	12
Time: 885.9704ms	13

1.2 Limit the number of goroutines

package main	1
	2
import (3
"fmt"	4
"math/rand"	5
"runtime"	6
"sync"	7
"sync/atomic"	8
"time"	9
)	10
	11
var workerPool = make(chan int, 50)	12
var jobQueue = make(chan string, 100)	13
var maxGo uint64	14
var wg sync.WaitGroup	15
	16
func main() {	17
go goroutineCounter()	18
	19
start := time.Now()	20
wg.Add(1)	21
go linkSender()	22
	23
wg.Add(1)	24
go workerCreator()	25
	26
wg.Wait()	27
	28
fmt.Println("Max goroutine number:", atomic.LoadUint64(&maxGo))	29
duration := time.Since(start)	30
fmt.Println("Time:", duration)	31
}	32
	33
func goroutineCounter() {	34
for {	35
n := runtime.NumGoroutine()	36
u := uint64(n)	37
if u > maxGo {	38
atomic.StoreUint64(&maxGo, u)	39
}	40
time.Sleep(200 * time.Millisecond)	41
}	42
}	43
	44
func linkSender() {	45
defer wg.Done()	46
links := []string{}	47
var numOfLink = 1000	48
for i := 0; i < numOfLink; i++ {	49
fakeLink := fmt.Sprintf("http://web%d.com", i)	50
links = append(links, fakeLink)	51
}	52
for _, link := range links {	53
jobQueue <- link	54
}	55
close(jobQueue)	56
}	57
	58
func workerCreator() {	59
defer wg.Done()	60
for link := range jobQueue {	61

workerPool <- 1	62
wg.Add(1)	63
go worker(link)	64
}	65
}	66
	67
func worker(link string) {	68
defer wg.Done()	69
defer func() { <-workerPool }()	70
fmt.Println(linkTest(link))	71
}	72
	73
func linkTest(link string) string {	74
time.Sleep(500 * time.Millisecond)	75
if rand.Intn(2) == 1 {	76
return link + ":␣Good"	77
} else {	78
return link + ":␣Bad"	79
}	80
}	81

Listing 3: With limit

...	1
http://web989.com: Bad	2
http://web976.com: Bad	3
http://web991.com: Bad	4
http://web999.com: Good	5
http://web997.com: Bad	6
http://web955.com: Bad	7
http://web956.com: Bad	8
Max goroutine number: 54	9
Time: 10.043942287s	10