课程安排，请关注**微信公众平台**或者**官方微博**

编程语言： Golang 与 html5

编程工具： Goland 和 HBuilder

预计平均一周左右更新一或二节课程

授人以鱼，不如授人以渔业。

大家好，

欢迎来到 **字节教育** 课程的学习

字节教育官网：www.ByteEdu.Com

腾讯课堂地址：Gopher.ke.qq.Com

技术交流群 ： 221 273 219

微信公众号 ：Golang 语言社区

微信服务号 ：Golang 技术社区

目录：

第一季　从零开始搭建游戏服务器

第十四节　高效协程池推荐（并不加入架构）

一、公众账号：

关键字回复：客服　获取课程助教的微信（助教 MM）

二、服务器具体编码：

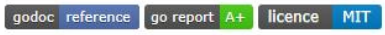
<1>.github 地址：

https://github.com/Golangltd/ants

<2>.介绍

ants

A goroutine pool for Go



ants 是一个高性能的协程池，实现了对大规模 goroutine 的调度管理、goroutine 复用，允许使用者在开发并发程序的时候限制协程数量，复用资源，达到更高效执行任务的效果。

功能:

- 实现了自动调度并发的 goroutine，复用 goroutine
- 提供了友好的接口：任务提交、获取运行中的协程数量、动态调整协程池大小
- 资源复用，极大节省内存使用量；在大规模批量并发任务场景下比原生 goroutine 并发具有更高的性能

安装

```
go get -u github.com/panjf2000/ants
```

使用包管理工具 glide 安装:

```
glide get github.com/panjf2000/ants
```

使用

写 go 并发程序的时候如果程序会启动大量的 goroutine ， 势必会消耗大量的系统资源（内存，CPU），通过使用 ants，可以实例化一个协程池，复用 goroutine ， 节省资源，提升性能：

```
package main

import (
            "fmt"
            "sync"
            "sync/atomic"

            "github.com/panjf2000/ants"
            "time"
)

var sum int32
```

```
func myFunc(i interface{}) error {
            n := i.(int)
            atomic.AddInt32(&sum, int32(n))
            fmt.Printf("run with %d\n", n)
            return nil
}

func demoFunc() error {
            time.Sleep(10 * time.Millisecond)
            fmt.Println("Hello World!")
            return nil
}

func main() {
            runTimes := 1000

            // use the common pool
            var wg sync.WaitGroup
            for i := 0; i < runTimes; i++ {
                        wg.Add(1)
                        ants.Submit(func() error {
                                    demoFunc()
                                    wg.Done()
                                    return nil
                        })
            }
            wg.Wait()
            fmt.Printf("running goroutines: %d\n", ants.Running())
            fmt.Printf("finish all tasks.\n")

            // use the pool with a function
            // set 10 the size of goroutine pool
            p, _ := ants.NewPoolWithFunc(10, func(i interface{}) error {
                        myFunc(i)
                        wg.Done()
                        return nil
            })
            // submit tasks
            for i := 0; i < runTimes; i++ {
                        wg.Add(1)
                        p.Serve(i)
            }
            wg.Wait()
            fmt.Printf("running goroutines: %d\n", p.Running())
            fmt.Printf("finish all tasks, result is %d\n", sum)
}
```

任务提交

提交任务通过调用 ants.Submit(func())方法：

```
ants.Submit(func() {})
```

自定义池

ants 支持实例化使用者自己的一个 Pool ， 指定具体的池容量；通过调用 NewPool 方法可以实例化一个新的带有指定容量的 Pool ， 如下：

```
// set 10000 the size of goroutine pool
p, _ := ants.NewPool(10000)
// submit a task
p.Submit(func() {})
```

动态调整协程池容量

需要动态调整协程池容量可以通过调用 ReSize(int)：

```
pool.ReSize(1000) // Readjust its capacity to 1000
pool.ReSize(100000) // Readjust its capacity to 100000
```

该方法是线程安全的。

Benchmarks

系统参数：

```
OS : macOS High Sierra
Processor : 2.7 GHz Intel Core i5
Memory : 8 GB 1867 MHz DDR3
```

```
andypan ··· github.com > panjf2000 > ants go test -bench=. -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkGoroutine-4                   1        1412072220 ns/op        128648424 B/op   1137895 allocs/op
BenchmarkPoolGroutine-4                1        1183985784 ns/op        37851280 B/op    1200179 allocs/op
PASS
ok      github.com/panjf2000/ants       2.693s
andypan ··· github.com > panjf2000 > ants go test -bench=. -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkGoroutine-4                   1        1233761461 ns/op        123666280 B/op   1128050 allocs/op
BenchmarkPoolGroutine-4                1        1090242715 ns/op        34051952 B/op    1186013 allocs/op
PASS
ok      github.com/panjf2000/ants       2.411s
andypan ··· github.com > panjf2000 > ants go test -bench=. -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkGoroutine-4                   1        27052612690 ns/op       1418164488 B/op 11658073 allocs/op
BenchmarkPoolGroutine-4                1        12517234956 ns/op       188974512 B/op  10200139 allocs/op
PASS
ok      github.com/panjf2000/ants       43.730s
andypan ··· github.com > panjf2000 > ants go test -bench=. -benchmem=true -run=none -memprofile mem.out
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkGoroutine-4                   1        25942291180 ns/op       1363038792 B/op 11517290 allocs/op
BenchmarkPoolGroutine-4                1        13976975822 ns/op       186483376 B/op  10200141 allocs/op
PASS
ok      github.com/panjf2000/ants       42.520s
andypan ··· github.com > panjf2000 > ants vi mem.out
andypan ··· github.com > panjf2000 > ants go test -bench=. -benchmem=true -run=none -test.memprofile mem.out
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkGoroutine-4                   1        26997432811 ns/op       1394907336 B/op 11597722 allocs/op
BenchmarkPoolGroutine-4                1        17253949543 ns/op       188975280 B/op  10200147 allocs/op
PASS
ok      github.com/panjf2000/ants       51.180s
```

上图中的前两个 benchmark 测试结果是基于 100w 任务量的条件，剩下的几个是基于 1000w 任务量的测试结果，`ants` 的默认池容量是 5w。

- BenchmarkGoroutine-4 代表原生 goroutine
- BenchmarkPoolGroutine-4 代表使用协程池 `ants`

Benchmarks with Pool

```
andypan ··· github.com panjf2000 ants go test -run="TestNoPool" -v
=== RUN   TestNoPool
--- PASS: TestNoPool (1.64s)
        ants_test.go:68: memory usage:130 MB
PASS
ok      github.com/panjf2000/ants       1.700s
andypan ··· github.com panjf2000 ants go test -run="TestDefaultPool" -v
=== RUN   TestDefaultPool
--- PASS: TestDefaultPool (1.46s)
        ants_test.go:49: running workers number:48396
        ants_test.go:52: memory usage:53 MB
PASS
ok      github.com/panjf2000/ants       1.477s
andypan ··· github.com panjf2000 ants go test -run="TestNoPool" -v
=== RUN   TestNoPool
--- PASS: TestNoPool (47.42s)
        ants_test.go:68: memory usage:1442 MB
PASS
ok      github.com/panjf2000/ants       48.012s
andypan ··· github.com panjf2000 ants go test -run="TestDefaultPool" -v
=== RUN   TestDefaultPool
--- PASS: TestDefaultPool (24.11s)
        ants_test.go:49: running workers number:716797
        ants_test.go:52: memory usage:684 MB
PASS
ok      github.com/panjf2000/ants       24.325s
andypan ··· github.com panjf2000 ants go test -run="TestNoPool" -v
```

Benchmarks with PoolWithFunc

```
andypan ··· github.com panjf2000 ants go test -bench="GoroutineWithFunc" -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkGoroutineWithFunc-4           1        45359154353 ns/op       1443930600 B/op 11725197 allocs/op
PASS
ok      github.com/panjf2000/ants       46.006s
andypan ··· github.com panjf2000 ants go test -bench="AntsPoolWithFunc" -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkAntsPoolWithFunc-4            1        15677278117 ns/op       40789864 B/op     261302 allocs/op
--- BENCH: BenchmarkAntsPoolWithFunc-4
        ants_benchmark_test.go:91: running goroutines: 50000
PASS
ok      github.com/panjf2000/ants       15.696s
andypan ··· github.com panjf2000 ants _
```

吞吐量测试

10w 任务量

```
andypan ··· github.com > panjf2000 > ants go test -bench="Goroutine$" -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkGoroutine-4           5         310334784 ns/op        31119840 B/op     147390 allocs/op
PASS
ok      github.com/panjf2000/ants       3.058s
andypan ··· github.com > panjf2000 > ants go test -bench="Goroutine$" -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkGoroutine-4           5         236169267 ns/op        38258771 B/op     168555 allocs/op
PASS
ok      github.com/panjf2000/ants       2.382s
andypan ··· github.com > panjf2000 > ants go test -bench="AntsPool$" -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkAntsPool-4           10         152347574 ns/op         3680629 B/op      23901 allocs/op
PASS
ok      github.com/panjf2000/ants       1.733s
```

100w 任务量

```
andypan ··· github.com > panjf2000 > ants go test -bench="Goroutine$" -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkGoroutine-4           1        6596307910 ns/op        537984248 B/op   2004769 allocs/op
PASS
ok      github.com/panjf2000/ants       8.601s
andypan ··· github.com > panjf2000 > ants go test -bench="AntsPool$" -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkAntsPool-4            1        1598450502 ns/op        37947504 B/op     246024 allocs/op
PASS
ok      github.com/panjf2000/ants       1.620s
```

1000w 任务量

```
andypan ··· github.com > panjf2000 > ants go test -bench="AntsPool$" -benchmem=true -run=none
goos: darwin
goarch: amd64
pkg: github.com/panjf2000/ants
BenchmarkAntsPool-4            1        15934332408 ns/op       41056368 B/op     264075 allocs/op
PASS
ok      github.com/panjf2000/ants       15.961s
andypan ··· github.com > panjf2000 > ants
```

1000w 任务量的场景下，我的电脑已经无法支撑 golang 的原生 goroutine 并发，所以只测出了使用 ants 池的测试结果。