

# GO 性能优化

By @miraclesu



舜飞科技

网络运营全流程解决方案供应商

# 概要

- string & profiling
- slice & array
- slice & map 初始化
- 并发
- 缓存

**string & profiling**

# string 连接1

## fmt VS "+"

```
12     str      = "hello gohpers!"
13     sep      = ","
14 )
15
16 func BenchmarkFmt(b *testing.B) {
17     for i := 0; i < b.N; i++ {
18         _ = fmt.Sprintf("%s%s%s%s", str, sep, str, sep, str)
19     }
20 }
21
22 func BenchmarkPlus(b *testing.B) {
23     for i := 0; i < b.N; i++ {
24         _ = str + sep + str + sep + str
25     }
26 }
```

BenchmarkFmt	1000000	1617 ns/op
BenchmarkPlus	5000000	393 ns/op

# string 连接2

fmt VS "+"

```
11     intA int    = 12345
12     intB int64 = 67890
13     str      = "hello gohpers!"
14     sep      = ", "
15 )
16
17 func BenchmarkFmt(b *testing.B) {
18     for i := 0; i < b.N; i++ {
19         _ = fmt.Sprintf("%d%s%s%d", intA, sep, str, sep, intB)
20     }
21 }
22
23 func BenchmarkPlus(b *testing.B) {
24     for i := 0; i < b.N; i++ {
25         _ = strconv.Itoa(intA) + sep + str + sep + strconv.FormatInt(intB, 10)
26     }
27 }
28
```

BenchmarkFmt	10000000	1324 ns/op
BenchmarkPlus	50000000	751 ns/op

# string 连接3

strings.join VS "+"

```
8 func plus(a []string, sep string) string {
9     if len(a) == 0 {
10         return ""
11     }
12     if len(a) == 1 {
13         return a[0]
14     }
15
16     str := a[0]
17     for _, s := range a[1:] {
18         str += sep + s
19     }
20     return str
21 }
22
23 func join(a []string, sep string) string {
24     return strings.Join(a, sep)
25 }
```

# string 连接3

## strings.join VS "+"

```
12     str      = "hello gohpers!"
13     strs     = []string{str, str, str, str, str, str, str, str, str, str}
14     sep      = ","
15 )
16
17 func BenchmarkPlus(b *testing.B) {
18     for i := 0; i < b.N; i++ {
19         _ = plus(strs, sep)
20     }
21 }
22
23 func BenchmarkJoin(b *testing.B) {
24     for i := 0; i < b.N; i++ {
25         _ = join(strs, sep)
26     }
27 }
```

BenchmarkPlus	500000	4659 ns/op
BenchmarkJoin	1000000	1491 ns/op

# string 连接4

## strings.Join VS bytes.Buffer

```
23 func join(a []string, sep string) string {
24     return strings.Join(a, sep)
25 }
26
27 func buffer(a []string, sep string) string {
28     if len(a) == 0 {
29         return ""
30     }
31     if len(a) == 1 {
32         return a[0]
33     }
34
35     var buf bytes.Buffer
36     buf.WriteString(a[0])
37     for _, s := range a[1:] {
38         buf.WriteString(sep)
39         buf.WriteString(s)
40     }
41     return buf.String()
42 }
```



# string 连接4

## strings.Join VS bytes.Buffer

```
10     str      = "hello gohpers!"
11     strs     = []string{str, str, str, str, str, str, str, str, str, str}
12     sep      = ","
13 )
14
15 func BenchmarkJoin(b *testing.B) {
16     for i := 0; i < b.N; i++ {
17         _ = join(strs, sep)
18     }
19 }
20
21 func BenchmarkBuffer(b *testing.B) {
22     for i := 0; i < b.N; i++ {
23         _ = buffer(strs, sep)
24     }
25 }
```

BenchmarkJoin	1000000	1505 ns/op
BenchmarkBuffer	500000	2886 ns/op

# string 连接4-1

## strings.Join VS bytes.Buffer

```
23 func join(a []string, sep string) []byte {
24     return []byte(strings.Join(a, sep))
25 }
26
27 func buffer(a []string, sep string) []byte {
28     if len(a) == 0 {
29         return []byte{}
30     }
31     if len(a) == 1 {
32         return []byte(a[0])
33     }
34
35     var buf bytes.Buffer
36     buf.WriteString(a[0])
37     for _, s := range a[1:] {
38         buf.WriteString(sep)
39         buf.WriteString(s)
40     }
41     return buf.Bytes()
42 }
```

# string 连接4-1

## strings.Join VS bytes.Buffer

```
10     str      = "hello gohpers!"
11     strs     = []string{str, str, str, str, str, str, str, str, str, str}
12     sep      = ","
13 )
14
15 func BenchmarkJoin(b *testing.B) {
16     for i := 0; i < b.N; i++ {
17         _ = join(strs, sep)
18     }
19 }
20
21 func BenchmarkBuffer(b *testing.B) {
22     for i := 0; i < b.N; i++ {
23         _ = buffer(strs, sep)
24     }
25 }
```

BenchmarkJoin	1000000	1824 ns/op
BenchmarkBuffer	1000000	2588 ns/op

内个...内个，我对bytes.Buffer  
情有独钟，能不能让Ta快点？

# profiling

- `go test -c`
- `go test -test.bench=. -test.cpuprofile=cpu.prof`
- `go tool pprof bench.test cpu.prof`

# string 连接4-2

## strings.Join VS bytes.Buffer

```
23 func join(a []string, sep string) string {
24     return strings.Join(a, sep)
25 }
26
27 func buffer(buf *bytes.Buffer, a []string, sep string) string {
28     if len(a) == 0 {
29         return ""
30     }
31     if len(a) == 1 {
32         return a[0]
33     }
34
35     buf.WriteString(a[0])
36     for _, s := range a[1:] {
37         buf.WriteString(sep)
38         buf.WriteString(s)
39     }
40     return buf.String()
41 }
```

# string 连接4-2

## strings.Join VS bytes.Buffer

```
11     str      = "hello gohpers!"
12     strs     = []string{str, str, str, str, str, str, str, str, str, str}
13     sep      = ","
14 )
15
16 func BenchmarkJoin(b *testing.B) {
17     for i := 0; i < b.N; i++ {
18         _ = join(strs, sep)
19     }
20 }
21
22 func BenchmarkBuffer(b *testing.B) {
23     buf := &bytes.Buffer{}
24     for i := 0; i < b.N; i++ {
25         buf.Reset()
26         _ = buffer(buf, strs, sep)
27     }
28 }
```

BenchmarkJoin	1000000	1500 ns/op
BenchmarkBuffer	1000000	1482 ns/op

# string 连接4-3

## strings.Join VS bytes.Buffer

```
23 func join(a []string, sep string) []byte {
24     return []byte(strings.Join(a, sep))
25 }
26
27 func buffer(buf *bytes.Buffer, a []string, sep string) []byte {
28     if len(a) == 0 {
29         return []byte{}
30     }
31     if len(a) == 1 {
32         return []byte(a[0])
33     }
34
35     buf.WriteString(a[0])
36     for _, s := range a[1:] {
37         buf.WriteString(sep)
38         buf.WriteString(s)
39     }
40     return buf.Bytes()
41 }
```



# string 连接4-3

## strings.Join VS bytes.Buffer

```
11     str      = "hello gohpers!"
12     strs      = []string{str, str, str, str, str, str, str, str, str, str}
13     sep       = ","
14 )
15
16 func BenchmarkJoin(b *testing.B) {
17     for i := 0; i < b.N; i++ {
18         _ = join(strs, sep)
19     }
20 }
21
22 func BenchmarkBuffer(b *testing.B) {
23     buf := &bytes.Buffer{}
24     for i := 0; i < b.N; i++ {
25         buf.Reset()
26         _ = buffer(buf, strs, sep)
27     }
28 }
```

BenchmarkJoin	1000000	1791 ns/op
BenchmarkBuffer	1000000	1162 ns/op

# string 和 []byte

- 如果可以的话，尽量用多[]byte，少用string
- 尽可能少地在两者之间做转换
- `append([]byte, string...)`
- `copy([]byte, string)`

# strconv

- `func AppendBool(dst []byte, b bool) []byte`
- `func AppendFloat(dst []byte, f float64, fmt byte, prec int, bitSize int) []byte`
- `func AppendInt(dst []byte, i int64, base int) []byte`
- `func AppendUint(dst []byte, i uint64, base int) []byte`
- `func FormatBool(b bool) string`
- `func FormatFloat(f float64, fmt byte, prec, bitSize int) string`
- `func FormatInt(i int64, base int) string`
- `func FormatUint(i uint64, base int) string`

**slice & array**

# slice & array

```
3  const SIZE = 1000
4
5  var (
6      Arr = [SIZE]string{}
7      Sli = make([]string, 0, SIZE)
8      str = "hello gohpers!"
9  )
10
11 func init() {
12     for i := 0; i < SIZE; i++ {
13         Arr[i] = str
14         Sli = append(Sli, str)
15     }
16 }
```

```
18 func arrayFunc(a [SIZE]string) {
19     for _, s := range a {
20         _ = s
21     }
22 }
23
24 func sliceFunc(a []string) {
25     for _, s := range a {
26         _ = s
27     }
28 }
29
```

# slice & array

```
7 func BenchmarkArray(b *testing.B) {  
8     for i := 0; i < b.N; i++ {  
9         arrayFunc(Arr)  
10    }  
11 }  
12  
13 func BenchmarkSlice(b *testing.B) {  
14     for i := 0; i < b.N; i++ {  
15         sliceFunc(Sli)  
16     }  
17 }
```

BenchmarkArray	200000	11101 ns/op
BenchmarkSlice	2000000	822 ns/op

# slice & array

- 数组是值传递
- slice是引用传递

# slice 坑

```
func operatSlice(s []int, num int) []int {  
    for i := 0; i < num; i++ {  
        s = append(s, 3)  
    }  
    s[0] = num  
    return s  
}
```

```
func main() {  
    s := make([]int, 0, 3)  
    s = append(s, 1)  
    fmt.Printf("s=%+v\n", s)  
    fmt.Println("=====")  
    s1 := operatSlice(s, 2)  
    fmt.Printf("s=%+v\n", s)  
    fmt.Printf("s1=%+v\n", s1)  
    fmt.Println("=====")  
    s2 := operatSlice(s, 3)  
    fmt.Printf("s=%+v\n", s)  
    fmt.Printf("s1=%+v\n", s1)  
    fmt.Printf("s2=%+v\n", s2)  
}
```

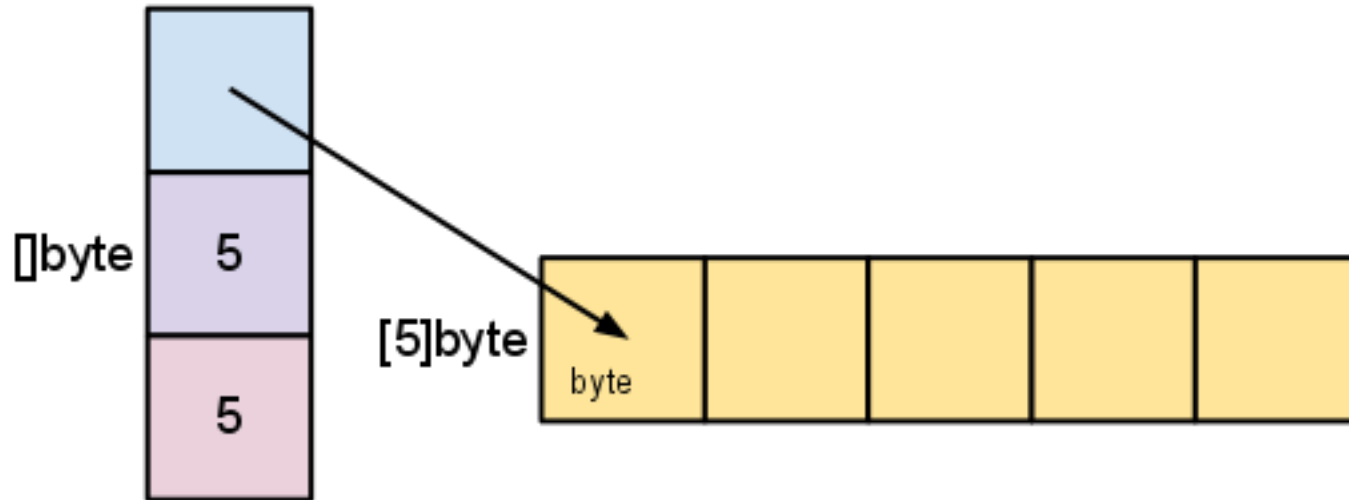
```
s=[ 1]  
=====  
s=[ 2]  
s1=[2 3 3]  
=====  
s=[ 2]  
s1=[2 3 3]  
s2=[3 3 3 3]
```



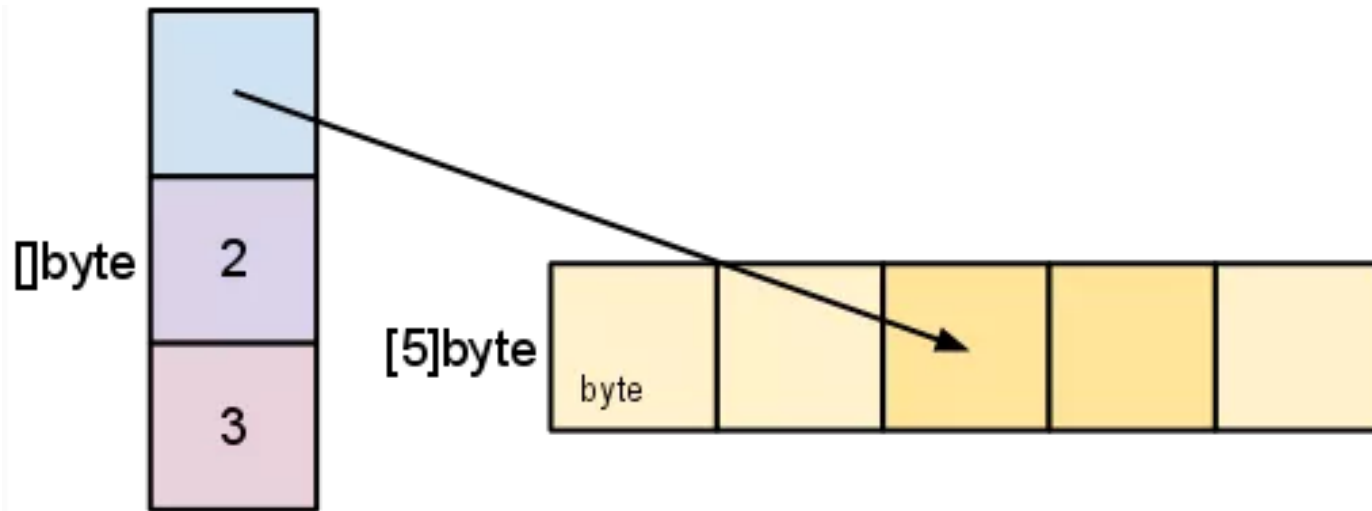
# slice 结构



# `make([]byte, 5)`



**$s = s[2:4]$**



**slice & map 初始化**

# slice 初始化

```
3  const (  
4      SIZE = 1000  
5      STR  = "hello gohpers!"  
6  )
```

```
8  func sliceFunc() []string {  
9      s := make([]string, 0)  
10     for i := 0; i < SIZE; i++ {  
11         s = append(s, STR)  
12     }  
13     return s  
14 }
```

```
16 func sliceCapFunc() []string {  
17     s := make([]string, 0, SIZE)  
18     for i := 0; i < SIZE; i++ {  
19         s = append(s, STR)  
20     }  
21     return s  
22 }
```

# slice 初始化测试结果

```
19 func BenchmarkSlice(b *testing.B) {  
20     for i := 0; i < b.N; i++ {  
21         sliceFunc()  
22     }  
23 }  
24  
25 func BenchmarkSliceCap(b *testing.B) {  
26     for i := 0; i < b.N; i++ {  
27         sliceCapFunc()  
28     }  
29 }
```

BenchmarkSlice	50000	33351 ns/op
BenchmarkSliceCap	100000	16432 ns/op

# map 初始化

```
3  const (  
4      SIZE = 1000  
5      STR  = "hello gohpers!"  
6  )
```

```
24 func mapFunc() map[int]string {  
25     m := make(map[int]string)  
26     for i := 0; i < SIZE; i++ {  
27         m[i] = STR  
28     }  
29     return m  
30 }
```

```
32 func mapCapFunc() map[int]string {  
33     m := make(map[int]string, SIZE)  
34     for i := 0; i < SIZE; i++ {  
35         m[i] = STR  
36     }  
37     return m  
38 }
```

# map 初始化测试结果

```
7 func BenchmarkMap(b *testing.B) {
8     for i := 0; i < b.N; i++ {
9         mapFunc()
10    }
11 }
12
13 func BenchmarkMapCap(b *testing.B) {
14     for i := 0; i < b.N; i++ {
15         mapCapFunc()
16     }
17 }
```

BenchmarkMap	5000	277715 ns/
op		
BenchmarkMapCap	10000	136396 ns/
op		



# slice or map?

BenchmarkSlice	50000	33351 ns/op
BenchmarkMap	5000	277715 ns/op
BenchmarkSliceCap	100000	16432 ns/op
BenchmarkMapCap	10000	136396 ns/op

# slice & map Read

```
7  const (  
8      SIZE = 1000  
9      STR  = "hello gohpers!"  
10 )  
11  
12 var (  
13     S    = make([]string, 0, SIZE)  
14     M    = make(map[int]string, SIZE)  
15 )  
16  
17 func init() {  
18     for i := 0; i < SIZE; i++ {  
19         S = append(S, STR)  
20         M[i] = STR  
21     }  
22 }
```

```
24 func sliceRead() string {  
25     i := rand.Intn(SIZE)  
26     return S[i]  
27 }  
28  
29 func mapRead() string {  
30     i := rand.Intn(SIZE)  
31     return M[i]  
32 }
```

# slice & map Read 测试结果

```
31 func BenchmarkMapRead(b *testing.B) {  
32     for i := 0; i < b.N; i++ {  
33         mapRead()  
34     }  
35 }  
36  
37 func BenchmarkSliceRead(b *testing.B) {  
38     for i := 0; i < b.N; i++ {  
39         sliceRead()  
40     }  
41 }
```

BenchmarkMapRead	10000000	155
ns/op		
BenchmarkSliceRead	20000000	86.8
ns/op		

并发

# 串行泡茶



# 总用时 26分钟

- 洗水壶 (1分)
- 烧开水 (15分)
- 洗茶壶 (2分)
- 拿茶叶 (1分)
- 泡茶 (5分)
- 洗茶杯 (2分)

如果我要泡4杯茶？  
并行





问题：  
每26分钟生产一杯茶

# 并发I



21分钟

# 烧开水最费时间！那么并发他！



# 并发II



并发5, 每3分钟烧出一壶开水



$$1+3+5 = 9\text{分}$$

# 泡茶（5分钟）已经成为瓶颈



# 并发2



并发5, 每3分钟烧出一壶开水



并发4, 每1.25分钟 泡一壶茶

$$1+3+2 = 6\text{分}$$





# 并发3



每3分钟一壶

并发大于并行，包含并行



缓存

# 提前优化是万恶之源

# Q & A

苏创绩 @miraclesu-创绩

## 欢迎加入我们



舜飞科技

网络运营全流程解决方案供应商