Golang slice map channel 小技巧

作者：Allenxuxu
原文链接：https://ld246.com/article/1618658465384
来源网站：链滴

Slice vs Array

Slice 和 Array 是不同的类型

```
package main

func main() {
   s := make([]int, 100)
   printSlice(s)

   var a [100]int
   printArray(a)
}

func printSlice(s []int) {
   println(len(s)) // 100
   println(cap(s)) // 100
}

func printArray(a [100]int) {
   println(len(a)) // 100
   println(cap(a)) // 100
}
```

Slice 结构体

```
type slice struct {
    array unsafe.Pointer
    len   int
    cap   int
}
```

下面的汇编表明，当类型是 slice 的时候，打印 len 或者 cap 的时候，会去栈上取数据:

```
MOVQ 0x28(SP), AX
  MOVQ AX, 0x8(SP)
  CALL 0xbfc              [1:5]R_CALL:runtime.printlock<1>
  MOVQ 0x8(SP), AX
  MOVQ AX, 0(SP)
```

**而当类型是 array 时候，直接用的 0x64 (10进制 100)： **MOVQ $0x64, 0(SP)

查看具体汇编代码

```
go tool compile -N -l main.go
go tool objdump main.o
```

Slice 的自动扩容

Slice 可以使用 append 函数新增数据，当容量不足的时候，会自动新申请一块空间，将原有数据复过去，再新增数据。

https://github.com/golang/go/blob/2ebe77a2fda1ee9ff6fd9a3e08933ad1ebaea039/src/runtime/slice.go#L125

- **当 cap < 1024 的时候，每次 *2 **
- **当 cap >= 1024 的时候，每次 * 1.25**
- **其中还会涉及内存对齐的调整**

```
package main

import "fmt"

func main() {
  var s []int

  for i:=0;i<3;i++ {
    s = append(s, i)
  }
  fmt.Println(s) // [0 1 2]

  modifySlice(s)
  fmt.Println(s) // [1024 1 2]
}

func modifySlice(s []int) {
  s = append(s, 2048)
  s[0] = 1024
  fmt.Println(s) // [1024 1 2 2048]
}
```

Golang 中都是值传递，所以 modifySlice 函数的入参，只是复制了 slice struct 中 array，len，cap 三个字段的值来初始化函数内的局部变量 s 。

所以，当函数内部进行 append 发送扩容了的话，会新申请一块空间，然后让 array 指针指向他。数外部的 slice 变量是不会变化的，array 指针仍然不变。

所以，个人觉得，对于需要在函数内部 append slice 的情况一律传递 *[]int 。

var s []int == nil

```
package main

import (
  "encoding/json"
  "fmt"
)

func main() {
  var s []int
  d, _ := json.Marshal(s)
  fmt.Println(string(d)) // null
  fmt.Println(s == nil) // true

  s2 := []int{}
```

```
    d, _ = json.Marshal(s2)
    fmt.Println(string(d)) // []
    fmt.Println(s2 == nil) // false
}
```

高效的 append

```

func BenchmarkAppendSlice(b *testing.B) {
    for i := 0; i < b.N; i++ {
        s := make([]int, 0, 10000)
        for j := 0; j < 10000; j++ {
            s = append(s, j)
        }
    }
}

func BenchmarkAppendSliceIndexed(b *testing.B) {
    for i := 0; i < b.N; i++ {
        s := make([]int, 10000)
        for j := 0; j < 10000; j++ {
            s[j] = j
        }
    }
}

BenchmarkAppendSlice-12                110247          10832 ns/op
BenchmarkAppendSliceIndexed-12         137204           8585 ns/op
```

因为 append 操作内部会每次去检查容量是不是够，即每次调用 runtime.growslice ，下面为截取部分汇编。

```
MOVQ AX, 0x50(SP)
 LEAQ 0(IP), SI          [3:7]R_PCREL:type.int
 MOVQ SI, 0(SP)
 MOVQ BX, 0x8(SP)
 MOVQ AX, 0x10(SP)
 MOVQ DX, 0x18(SP)
 MOVQ CX, 0x20(SP)
 NOPL 0(AX)
 CALL 0x5d6              [1:5]R_CALL:runtime.growslice<1>
 MOVQ 0x28(SP), BX
 MOVQ 0x30(SP), AX
 MOVQ 0x38(SP), DX
 LEAQ 0x1(AX), CX
 MOVQ 0x50(SP), AX
 JMP 0x57b
```

边界检查消除

https://gfw.go101.org/article/bounds-check-elimination.html

```
func normal(s []int) int {
```

```
    i := 0

    i += s[0]
    i += s[1]
    i += s[2]
    i += s[3]
    i += s[4]

    return i
}

func bce(s []int) int {
    _ = s[4]

    i := 0
    i += s[0]
    i += s[1]
    i += s[2]
    i += s[3]
    i += s[4]

    return i
}
```

第一种情况下，golang 会在每一次按下标取值时调用 runtime.panicIndex 检查是否越界。

下面是截取的部分汇编：

下面的需要开启优化选项来编译 go tool compile main.go

normal

```
CALL 0x8ce              [1:5]R_CALL:runtime.panicIndex
MOVL $0x3, AX
CALL 0x8d8              [1:5]R_CALL:runtime.panicIndex
MOVL $0x2, AX
NOPL
CALL 0x8e3              [1:5]R_CALL:runtime.panicIndex
MOVL $0x1, AX
CALL 0x8ed              [1:5]R_CALL:runtime.panicIndex
XORL AX, AX
CALL 0x8f4              [1:5]R_CALL:runtime.panicIndex
```

Bce

```
CALL 0xab8              [1:5]R_CALL:runtime.printint<1>
CALL 0xabd              [1:5]R_CALL:runtime.printnl<1>
CALL 0xac2              [1:5]R_CALL:runtime.printunlock<1>
MOVQ 0x18(SP), BP
ADDQ $0x20, SP
RET
MOVL $0x3, AX
CALL 0xad6              [1:5]R_CALL:runtime.panicIndex
```

总结

- **Golang 中 数组 和 slice 是两种完全不同的类型，也有着不同的行为**
  - **数组的不可改变，是其类型声明的一部分**
- **Slice 本质是一个 struct，包含一个执行数据地址的指针，len 字段记录长度，cap 字段记录容量**
- **Golang 中函数传参**
  - **如果是 slice ，则会复制内部的三个字段值来初始化一个新的 slice 变量**
- **如果是 array，则会重新申请一块内存，复制整个数组的内容到新的 array 变量**
- * 如果在函数内部 append 这个slice，一定要传递 []int
- **Slice 的 append 函数内部会每次都调用 runtime.growslice ，检查是否需要扩容，在容量已经定的情况下，用 index 更高效。**
- **边界检查优化**

Map

Map 主要是要了解一些源码实现，详情可以看下面的文章分析

https://qcrao.com/2019/05/22/dive-into-go-map/

总结

- **Map 删除 key 不会缩容，也不会释放空间**
- **Map 的 key value 都不可取值**
- **Map 内部和 slice 类似，都是用指针指向具体存储，所以用 map 作为函数参数，在函数内部可以改map**
  - **不同的是，如果在函数内部 map 发生扩容，是会作用于外部的 map 的，因为 map 内部采用链法，不同于 slice 的申请一个新空间然后复制过去**
- **Map 非并发安全，并发访问使用 sync.Map**

Channel

https://qcrao.com/2019/07/22/dive-into-go-channel/

总结

- **Channel 内部使用锁实现**
- **Channel 会触发调度**
- **Channel 发送的数据是值拷贝的，有必要的话需要传指针减少复制开销**
- **For 循环里面的 select 内部的 break 只会跳出 select**