一周汇总 2019-04-28

每日一学

问题01：

关于空 slice 的声明：

当我们需要声明一个空 slice 时，一般来说 `var t []string` 优于 `t := []string{}`。前者是 slice 的 nil 值；后者是 non-nil 值但长度是 0。但两者在功能性上是一致的：它们的 len 和 cap 都是 0。但 nil slice 是更好的方式。 特别的，对于编码 JSON，因为 nil slice 会编码为 null，而 []string{} 编码为 JSON 的数组 []。因此这种情形下，根据场景，non-nil 但长度是 0 的 slice 可能是比 nil slice 更推荐的形式。

注意：在设计接口时，不应该对 nil slice、non-nil 但 0 长度的 slice 进行区分，这可能会导致一些细微的错误。

讨论结果：

1. make 也可以产生 non-nil 的 slice，长度为 0；
2. `a := []int{}` 会产生 non-nil 的 slice，`a := []int(nil)` 会产生 nil 的 slice；
3. make(T) 会返回一个类型T 的初始值。

问题02：

关于随机数： 在 Go 语言中，产生随机数（伪随机数）有两种方式：

- math/rand 包
- crypto/rand 包

一般的场景，我们使用 math/rand 包即可，但要注意随机数种子的问题，一般使用 time.Now().UnixNano() 作为种子。同时需要注意，两个程序，如果设置同样的种子，那么它们同样的调用次数，得到的随机数是一样的，这也就是为什么叫伪随机数。不过 math/rand 简单好用，很多场景下人们都会优先使用它。（新手使用 math/rand 可能会问：为什么我的程序随机数不随机，每次运行得到的结果都是同一个，根本不是随机嘛，你知道原因了吗？）

所以，对于需要产生密钥的场景，我们应该使用 crypto/rand 包来产生随机数。

留两个问题：

1. 请为 math/rand 和 crypto/rand 写性能测试，给出你的测试结果；
2. 你有兴趣了解这两种随机数的实现方式吗？有没有可能将将它们结合起来？

讨论结果：

1. math/rand 测试结果：200000 10350 ns/op； crypto/rand 测试结果：2000000 599 ns/op 测试代码：

```
1 package main
2
```

```
import (
    cRand "crypto/rand"
    "math/big"
    mRand "math/rand"
    "time"
    "testing"
)

func CryptoRandInt(max int64) (*big.Int, error) {
    return cRand.Int(cRand.Reader, big.NewInt(max))
}
func MathRandInto() int {
    seed := time.Now().UnixNano()
    mRand.Seed(seed)
    return mRand.Int()
}
func BenchmarkCryptoRandInt(b *testing.B) {
    for i := 0; i < b.N; i++ {
        CryptoRandInt(999999999999999999)
    }
}
func BenchmarkMathRandInto(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MathRandInto()
    }
}
```

- **星主回复**：这里 MathRandInto 不公平，种子应该在外面设置，而不是在该函数里，这样每次都设置，性能肯定有影响。

2. 代码：

```
const letterBytes = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

func MRand() {
    b := make([]byte, len(letterBytes))
    for i := range b {
        mrand.Seed(time.Now().UnixNano())
        b[i] = letterBytes[mrand.Intn(len(letterBytes))]
    }
    //fmt.Println(len(b))

}

func CRand() {

    b := []byte(letterBytes)
    crand.Read(b)
    //fmt.Println(len(b))
    //fmt.Println(b)
}


```

```
//BenchmarkCRand-8       2000000           988 ns/op
func BenchmarkCRand(b *testing.B) {
    for i := 0; i < b.N; i++ {
        CRand()
    }
}
//BenchmarkMRand-8          2000        591545 ns/op
func BenchmarkMRand(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MRand()
    }
}
```

- **星主回复**：一样的问题，MRand 不公平，设置太多次种子，影响实际性能。

3. 之前写的也是将种子设置在了随机函数里面，所以修改了下代码：

```
func init(){
    rand.Seed(time.Now().UnixNano())
}
func MathRandIntNumber()int{
    return rand.Int()
}

func CryptoRandIntNumber(m int64)(*big.Int, error){
       // crypto/rand 定义别名为crypto，避免和math/rand冲突
    return crypto.Int(crypto.Reader, big.NewInt(m))
}
//测试代码
func BenchmarkMathRandIntNumber(b *testing.B){
    for i := 0; i < b.N; i++{
        MathRandIntNumber()
    }
}

func BenchmarkCryptoRandIntNumber(b *testing.B){
    for i := 0; i < b.N; i++{
        CryptoRandIntNumber(math.MaxInt64)
    }
}
//性能测试结果如下
BenchmarkMathRandIntNumber-8        100000000           17.7 ns/op
BenchmarkCryptoRandIntNumber-8        2000000          915 ns/op
```

4. 性能测试在哪里有资料不～没写过
    - go语言圣经里的基准测试就是吧

问题03：

关于代码缩进： 内容有较多代码，为了方便阅读，请移步Go中文网。 付费用户「每日一学」2019-04-24：关于代码缩进 - Go语言中文网 - Golang中文社...

问题04：

关于 Go 命名风格

1. 整体上，Go 采用驼峰风格；
2. 包名全小写，且简短。这种包名真心觉得很差：abc_defg，一定改改吧，标准库包名就没有含 _ 的；
3. 变量名中，有一点特别提醒一下：对于常见的缩写，应该采用全小写或全大写，比如：ID，URL，不应该定义为：Id、Url，如果不按此要求，golint 会有提示；(https://github.com/golang/lint)" />，这里推荐另外一个更强大的工具...
4. 常量单词采用首字母大写的方式，不同于其他语言，全大写，如：FirstName

ORM 中数据库 id 经常遇见，你是怎么处理的呢？还有哪些命名风格需要注意，欢迎一起讨论~

讨论结果：

1. 之前python，shell全爱用下划线，看过go规范之后就全部改过来了；
2. 感觉这篇文章不错：https://gist.github.com/Akagi201/f720076788b90687ec36；

问题05：

从此再无**类型接收者选择的困惑**

Go 新手经常很困惑，不知道该使用值接收者还是指针接收者。这个问题在之前每日分享中有过一次，见 付费用户「每日一学」2019-04-11：Go 类型的本质 - Go语言中文网 - Golang中...

今天我看到了官方有一些更好地判定方法或建议，所以分享给大家。

粗暴的结论：如果你不知道怎么选择，那就使用指针。但有时候，使用值接收者会更合理，尤其是效率考虑，比如：不需要修改的小 struct、基础数据类型。以下是一些有用的指导方针：

- 如果接收者是 map、func 或 chan，不用使用指针。如果是 slice，并且方法不会 reslice 或 从分配 slice，不要使用指针；
- 如果方法需要修改接收者，必须使用指针；
- 如果接收者是包含了 sync.Mutex 或类似的同步字段的结构体（struct），接收者必须使用指针，避免拷贝；
- 如果接收者是一个大的结构体或数组，使用指针会更高效。但多大是大？如果所有元素（struct 的字段或数组元素）作为方法的参数传递认为太大，那么作为接收者也是太大。（粗暴一些，所有元素内存超过指针大小，可以考虑使用指针）；
- 如果接收者是结构体、数组或 slice，同时，它们的元素是指针，且指向的数据要改变，那么使用指针接收者会更合理；（有点绕，那就总原则：用指针没错）；
- 如果接收者是小的数组，或小的没有可变字段或指针的结构体，或者结构体字段只是简单的基础类型，值接收者会更合理；值接收者能减少垃圾回收的压力，一般会优先分配在栈上（记住是一般，因为有可能逃逸）；但除非有性能测试验证，否则别因为可以介绍垃圾回收压力，就选择值接收者；

最后再强调一下，如果你拿不定主意，那就用指针接收者。

问题06：

关于简式声明语法（:=）付费用户「每日一学」2019-04-28：关于简式声明语法（:=） - Go语言中文网 - Gol...

讨论结果：

1. 那个 x 是因为在 {} 中属于子域才可以 :=。

面试题

问题01：

`import .`这种包导入形式是什么意思？一般什么情况下会使用？

讨论结果：

1. 测试的时候可以使用，其他地方不要用。参考：CodeReviewComments · golang/go Wiki · GitHub
2. 可以省略导入包的包名，直接用函数

问题02：

关于 map，以下代码有什么问题，应该如何改进？

```go
package main

type Person struct {
    Age int
}

func (p *Person) GrowUp() {
    p.Age++
}

func main() {
    m := map[string]Person{
        "zhangsan": Person{Age: 20},
    }
    m["zhangsan"].Age = 23
    m["zhangsan"].GrowUp()
}
```

讨论结果：

1. map 回传的是一个新的值，修改它不会影响到原 map 中存储的值；
2. 两种修改方式：

```go
// 1.修改完后覆盖
p := m["zhangsan"]
p.Age = 23
p.GrowUp()
m["zhangsan"] = p
// 2.使用指针
m := map[string]*Person{
  "zhangsan": &Person{Age: 20},
}
```

3. go语言中的map的元素不可以寻址，因为map中的元素的地址是变化的，这意味着寻址的结果是无意义的。

资源分享

1. 这里面的，我计划挑一些我认为不错的，慢慢跟大家一起学习探讨Golang 学习路线图 2019 - Go语言中文网 - Golang中文社区
2. 关于两个随机数的问题，各位球友陆陆续续给出了自己的解答，我这里推荐一篇文章，大家可以看看：两个随机数函数的故事 - Go语言中文网 - Golang中文社区
3. GopherChina 2019 讲师PPT 链接：https://pan.baidu.com/s/1VqbYrfm4X1jwaT18gWQlhA 提取码：659f

爬虫系列

1. 爬虫系列1：总体规划 - Go语言中文网 - Golang中文社区
2. 爬虫系列2：前端相关知识 - Go语言中文网 - Golang中文社区
3. 爬虫系列3：系统设计（一） - Go语言中文网 - Golang中文社区

同学问的问题

问题01：

写的后台服务在运维那里跑起来后碰到了内存泄露的问题，我用go tool pprof性能分析工具拉下来了heap.profile文件做分析，但没看出是哪里泄露了。请问还有其他解决方法吗？谢谢。

对话：

conversation1：

- 程序因为内存泄露挂掉了？有没有coredump 文件？
  - 是的，内存占用太多被系统杀死了。没有coredump文件。运维那里显示已占用3个G，并且还在不断涨，涨势缓慢；我用pprof工具查看常驻内存使用情况是占1个G。所以很困惑。
- 如果 pprof 看不出什么问题，看看代码，哪些地方可能占用较多内存，是否存在循环不断增加内存的情况。

conversation2：

- 也可以看看服务器进程，其实分很多情况，有可能是操作db，也有可能是读大文件到内存，没有分批读等，对弈第一种，可以看当前服务器状态，第二种，就得看代码了
    - 谢谢，目前已经定位是协程泄露造成的

问题02：

博主，编写项目的时候，应该如何设计目录，也就是开发准备？想用go来重构旧项目，但不了解go这块如何合理的设计目录结构。 另外除了这个，还需要做什么开发前准备。

星主回答：

GitHub - golang-standards/project-layout: Standard Go Project Layout 这个说明供参考。

我个人建议可以加上其他语言成熟的 MVC 项目结构加上这个的部分融合，对于一个项目组来说，保持一致、比较合理就是最好的。我在后面的实战项目会专门讲解目录结构的问题。预计5.1后产出。

每周链接

- GCTT | Go 中的 import 声明
- 在此将Go语言中一些和项目结构设计相关的经验记录下来分享给大家，一起学习。 Go项目结构设计过程点滴记录 - Go语言中文网 - Golang中文社区
- B站源码分析来了 一探B站后台架构, 他山之石, 何以攻玉? -- 仅从一个一线Golang开发者的角度谈B站4.22... 看看大公司项目目录结构。
- GCTT 出品 | 在 Go 运行时中的 Strings
- GopherChina第一天小结
- 之前问星球里一个关于slice扩容的问题，网上看了一篇文章，写的不错，分享一下：Go slice扩容分析之 不是double或1.25那么简单 - 简书