

新年彩蛋 | 完整版思考题答案

2019-02-02 郝林

Go语言核心36讲

[进入课程 >](#)



你好，我是郝林。

在 2019 年的春节来临之际，我恰好也更新完了专栏所有的配图和思考题答案。希望这些可以帮助到你，在新的一年中，祝你新年快乐，Go 语言学习之路更加顺利。

基础概念篇

1. Go 语言在多个工作区中查找依赖包的时候是以怎样的顺序进行的？

答：你设置的环境变量 `GOPATH` 的值决定了这个顺序。如果你在 `GOPATH` 中设置了多个工作区，那么这种查找会以从左到右的顺序在这些工作区中进行。

你可以通过试验来确定这个问题的答案。例如：先在一个源码文件中导入一个在你的机器上并不存在的代码包，然后编译这个代码文件。最后，将输出的编译错误信息与 `GOPATH` 的值进行对比。



3. 默认情况下，我们可以让命令源码文件接受哪些类型的参数值？

答：这个问题通过查看`flag`代码包的文档就可以回答了。概括来讲，有布尔类型、整数类型、浮点数类型、字符串类型，以及`time.Duration`类型。

4. 我们可以把自定义的数据类型作为参数值的类型吗？如果可以，怎样做？

答：狭义上讲是不可以的，但是广义上讲是可以的。这需要一些定制化的工作，并且被给定的参数值只能是序列化的。具体可参见`flag`代码包文档中的例子。

5. 如果你需要导入两个代码包，而这两个代码包的导入路径的最后一级是相同的，比如：`dep/lib/flag`和`flag`，那么会产生冲突吗？

答：这会产生冲突。因为代表两个代码包的标识符重复了，都是`flag`。

6. 如果会产生冲突，那么怎样解决这种冲突？有几种方式？

答：接上一个问题。很简单，导入代码包的时候给它起一个别名就可以了，比如：`import libflag "dep/lib/flag"`。或者，以本地化的方式导入代码包，如：`import . "dep/lib/flag"`。

7. 如果与当前的变量重名的是外层代码块中的变量，那么意味着什么？

答：这意味着这两个变量成为了“可重名变量”。在内层的变量所处的那个代码块以及更深层次的代码块中，这个变量会“屏蔽”掉外层代码块中的那个变量。

8. 如果通过`import . XXX`这种方式导入的代码包中的变量与当前代码包中的变量重名了，那么 Go 语言是会把它们当做“可重名变量”看待还是会报错呢？

答：这两个变量会成为“可重名变量”。虽然这两个变量在这种情况下的作用域都是当前代码包的当前文件，但是它们所处的代码块是不同的。

当前文件中的变量处在该文件所代表的代码块中，而被导入的代码包中的变量却处在声明它的那个文件所代表的代码块中。当然，我们也可以认为被导入的代码包所代表的代码块包含了



在当前文件中，本地的变量会“屏蔽”掉被导入的变量。

9. 除了《程序实体的那些事儿 3》一文中提及的那些，你还认为类型转换规则中有哪些值得注意的地方？

答：简单来说，我们在进行类型转换的时候需要注意各种符号的优先级。具体可参见 Go 语言规范中的转换部分。

10. 你能具体说说别名类型在代码重构过程中可以起到的哪些作用吗？

答：简单来说，我们可以通过别名类型实现外界无感知的代码重构。具体可参见 Go 语言官方的文档 Proposal: Type Aliases。

数据类型和语句篇

11. 如果有多个切片指向了同一个底层数组，那么你认为应该注意些什么？

答：我们需要特别注意的是，当操作其中一个切片的时候是否会影响到其他指向同一个底层数组的切片。

如果是，那么问一下自己，这是你想要的结果吗？无论如何，通过这种方式来组织或共享数据是不正确的。你需要做的是，要么彻底切断这些切片的底层联系，要么立即为所有的相关操作加锁。

12. 怎样沿用“扩容”的思想对切片进行“缩容”？

答：关于切片的“缩容”，可参看官方的相关 wiki。不过，如果你需要频繁的“缩容”，那么就可能需要考虑其他的数据结构了，比如：container/list 代码包中的 List。

13. container/ring 包中的循环链表的适用场景都有哪些？

答：比如：可重用的资源（缓存等）的存储，或者需要灵活组织的资源池，等等。

14. container/heap 包中的堆的适用场景又有哪些呢？



15. 字典类型的值是并发安全的吗？如果不是，那么在我们只在字典上添加或删除键 - 元素对的情况下，依然不安全吗？

答：字典类型的值不是并发安全的，即使我们只是增减其中的键值对也是如此。其根本原因是，字典值内部有时候会根据需要进行存储方面的调整。

16. 通道的长度代表着什么？它在什么时候会通道的容量相同？

通道的长度代表它当前包含的元素值的个数。当通道已满时，其长度会与容量相同。

17. 元素值在经过通道传递时会被复制，那么这个复制是浅表复制还是深层复制呢？

答：浅表复制。实际上，在 Go 语言中并不存在深层次的复制，除非我们自己来做。

18. 如果在 `select` 语句中发现某个通道已关闭，那么应该怎样屏蔽掉它所在的分支？

答：很简单，把 `nil` 赋给代表了这个通道的变量就可以了。如此一来，对于这个通道（那个变量）的发送操作和接收操作就会永远被阻塞。

19. 在 `select` 语句与 `for` 语句联用时，怎样直接退出外层的 `for` 语句？

答：这一般会用到 `goto` 语句和标签（label），具体请参看 Go 语言规范的这部分。

20. `complexArray1` 被传入函数的话，这个函数中对该参数值的修改会影响到它的原值吗？

答：文中 `complexArray1` 变量的声明如下：

复制代码

```
1 complexArray1 := [3][]string{  
2     []string{"d", "e", "f"},  
3     []string{"g", "h", "i"},  
4     []string{"j", "k", "l"},  
5 }
```



达妄有心忤修改了。虽然complexArray1本身是一个数组，但是其中的元素却都是切片。如果对complexArray1中的元素进行增减，那么原值就不会受到影响。但若要修改它已有的元素值，那么原值也会跟着改变。

21. 函数真正拿到的参数值其实只是它们的副本，那么函数返回给调用方的结果值也会被复制吗？

答：函数返回给调用方的结果值也会被复制。不过，在一般情况下，我们不用太在意。但如果函数在返回结果值之后依然保持执行并会对结果值进行修改，那么我们就需要注意了。

22. 我们可以在结构体类型中嵌入某个类型的指针类型吗？如果可以，有哪些注意事项？

答：当然可以。在这时，我们依然需要注意各种“屏蔽”现象。由于某个类型的指针类型会包含与前者有关联的所有方法，所以我们更要注意。

另外，我们在嵌入和引用这样的字段的时候还需要注意一些冲突方面的问题，具体请参看Go语言规范的这一部分。

23. 字面量struct{}代表了什么？又有什么用处？

答：字面量struct{}代表了空的结构体类型。这样的类型既不包含任何字段也没有任何方法。该类型的值所需的存储空间几乎可以忽略不计。

因此，我们可以把这样的值作为占位值来使用。比如：在同一个应用场景下，map[int][int]bool类型的值占用更少的存储空间。

24. 如果我们把一个值为nil的某个实现类型的变量赋给了接口变量，那么在这个接口变量上仍然可以调用该接口的方法吗？如果可以，有哪些注意事项？如果不可以，原因是什么？

答：可以调用。但是请注意，这个被调用的方法在此时所持有的接收者的值是nil。因此，如果该方法引用了其接收者的某个字段，那么就会引发panic！

25. 引用类型的值的指针值是有意义的吗？如果没有意义，为什么？如果有意义，意义在哪里？



26. 用什么手段可以对 goroutine 的启用数量加以限制？

答：一个很简单且很常用的方法是，使用一个通道保存一些令牌。只有先拿到一个令牌，才能启用一个 goroutine。另外在 `go` 函数即将执行结束的时候还需要把令牌及时归还给那个通道。

更高级的手段就需要比较完整的设计了。比如，任务分发器 + 任务管道（单层的通道）+ 固定个数的 goroutine。又比如，动态任务池（多层的通道）+ 动态 goroutine 池（可由前述的那个令牌方案演化而来）。等等。

27. `runtime` 包中提供了哪些与模型三要素 G、P 和 M 相关的函数？

答：关于这个问题，我相信你一查文档便知。在线文档在这里。不过光知道还不够，还要会用。

28. 在类型 `switch` 语句中，我们怎样对被判断类型的那个值做相应的类型转换？

答：其实这个事情可以让 Go 语言自己来做，例如：

复制代码

```
1 switch t := x.(type) {  
2 // cases  
3 }
```

当流程进入到某个 `case` 子句的时候，变量 `t` 的值就已经被自动地转换为相应类型的值了。

29. 在 `if` 语句中，初始化子句声明的变量的作用域是什么？

答：如果这个变量是新的变量，那么它的作用域就是当前 `if` 语句所代表的代码块。注意，后续的 `else if` 子句和 `else` 子句也包含在当前的 `if` 语句代表的代码块之内。

30. 请列举出你经常用到或者看到的 3 个错误类型，它们所在的错误类型体系都是怎样的？你能画出一棵树来描述它们吗？

31. 请列举出你经常用到或者看到的 3 个错误值，它们分别在哪个错误值列表里？这些错误值列表分别包含的是哪个种类的错误？

答：略。这需要你自己去做，我代替不了你。

32. 一个函数怎样才能把 panic 转化为 error 类型值，并将其作为函数的结果值返回给调用方？

答：可以这样编写：

复制代码

```
1 func doSomething() (err error) {
2     defer func() {
3         p := recover()
4         err = fmt.Errorf("FATAL ERROR: %s", p)
5     }()
6     panic("Oops!!")
7 }
```

注意结果声明的写法。这是一个带有名称的结果声明。

33. 我们可以在 defer 函数中恢复 panic，那么可以在其中引发 panic 吗？

答：当然可以。这样做可以把原先的 panic 包装一下再抛出去。

Go 程序的测试

34. 除了本文中提到的，你还知道或用过 testing.T 类型和 testing.B 类型的哪些方法？它们都是做什么用的？

答：略。这需要你自己去做，我代替不了你。

35. 在编写示例测试函数的时候，我们怎样指定预期的打印内容？

答：这个问题的答案就在 testing 代码包的文档中。



答：-timeout标记的作用是设定测试函数的执行时间上限。

具体请看这里的文档。

37. 怎样在测试的时候开启测试覆盖度分析？如果开启，会有什么副作用吗？

答：go test命令可以接受-coverage标记。该标记的作用就是开启测试覆盖度分析。不过，由于覆盖度分析开启之后go test命令可能会在程序被编译之前注释掉一部分源代码，所以，若程序编译或测试失败，那么错误报告可能会记录下与原始的源代码不对应的行号。

标准库的用法

38. 你知道互斥锁和读写锁的指针类型都实现了哪一个接口吗？

答：它们都实现了sync.Locker接口。

39. 怎样获取读写锁中的读锁？

答：sync.RWMutex类型有一个名为RLocker的指针方法可以获取其读锁。

40. *sync.Cond类型的值可以被传递吗？那sync.Cond类型的值呢？

答：sync.Cond类型的值一旦被使用就不应该再被传递了，传递往往意味着拷贝。拷贝一个已经被使用的sync.Cond值会引发panic。但是它的指针值是可以被拷贝的。

41. sync.Cond类型中的公开字段L是做什么用的？我们可以在使用条件变量的过程中改变这个字段的值吗？

答：这个字段代表的是当前的sync.Cond值所持有的那个锁。我们可以在使用条件变量的过程中改变该字段的值，但是在改变之前一定要搞清楚这样做的影响。

42. 如果要对原子值和互斥锁进行二选一，你认为最重要的三个决策条件应该是什么？

答：我觉得首先需要考虑下面几个问题。

操作被保护数据的代码是集中的还是分散的？如果是分散的，是否可以变为集中的？

在搞清楚上述问题（以及你关注的其他问题）之后，优先使用原子值。

43. 在使用WaitGroup值实现一对多的 goroutine 协作流程时，怎样才能让分发子任务的 goroutine 获得各个子任务的具体执行结果？

答：可以考虑使用锁 + 容器（数组、切片或字典等），也可以考虑使用通道。另外，你或许也可以用上`golang.org/x/sync/errgroup`代码包中的程序实体，相应的文档在这里。

44. Context值在传达撤销信号的时候是广度优先的还是深度优先的？其优势和劣势都是什么？

答：它是深度优先的。其优势和劣势都是：直接分支的产生时间越早，其中的所有子节点就会越先接收到信号。至于什么时候是优势、什么时候是劣势还要看具体的应用场景。

例如，如果子节点的存续时间与资源的消耗是正相关的，那么这可能就是一个优势。但是，如果每个分支中的子节点都很多，而且各个分支中的子节点的产生顺序并不依从于分支的产生顺序，那么这种优势就很可能会变成劣势。最终的定论还是要看测试的结果。

45. 怎样保证一个临时对象池中总有比较充足的临时对象？

答：首先，我们应该事先向临时对象池中放入足够多的临时对象。其次，在用完临时对象之后，我们需要及时地把它归还给临时对象池。

最后，我们应该保证它的`New`字段所代表的值是可用的。虽然`New`函数返回的临时对象并不会被放入池中，但是起码能够保证池的`Get`方法总能返回一个临时对象。

46. 关于保证并发安全字典中的键和值的类型正确性，你还能想到其他的方案吗？

答：这是一道开放的问题，需要你自己去思考。其实怎样做完全取决于你的应用场景。不过，我们应该尽量避免使用反射，因为它对程序性能还是有一定的影响的。



EncodeRune函数等。我们需要根据输入的不同来选择和使用它们。具体可以查看该代码包的文档。

48. strings.Builder和strings.Reader都分别实现了哪些接口？这样做有什么好处吗？

答：strings.Builder类型实现了3个接口，分别是：fmt.Stringer、io.Writer和io.ByteWriter。

而strings.Reader类型则实现了8个接口，即：io.Reader、io.ReaderAt、io.ByteReader、io.RuneReader、io.Seeker、io.ByteScanner、io.RuneScanner和io.WriterTo。

好处是显而易见的。实现的接口越多，它们的用途就越广。它们会适用于那些要求参数的类型为这些接口类型的地方。

49. 对比strings.Builder和bytes.Buffer的String方法，并判断哪一个更高效？原因是什么？

答：strings.Builder的String方法更高效。因为该方法只对其所属值的内容容器（那个字节切片）做了简单的类型转换，并且直接使用了底层的值（或者说内存空间）。它的源码如下：

复制代码

```
1 // String returns the accumulated string.
2 func (b *Builder) String() string {
3     return *(*string)(unsafe.Pointer(&b.buf))
4 }
```

数组值和字符串值在底层的存储方式其实是一样的。所以从切片值到字符串值的指针值的转换可以是直截了当的。又由于字符串值是不可变的，所以这样做也是安全的。

50. `io`包中的同步内存管道的运作机制是什么？

答：我们实际上已经在正文中做了基本的说明。

`io.Pipe`函数会返回一个`io.PipeReader`类型的值和一个`io.PipeWriter`类型的值，并将它们分别作为管道的两端。而这两个值在底层其实只是代理了同一个`*io.pipe`类型值的功能而已。

`io.pipe`类型通过无缓冲的通道实现了读操作与写操作之间的同步，并且通过互斥锁实现了写操作之间的串行化。另外，它还使用原子值来处理错误。这些共同保证了这个同步内存管道的并发安全性。

51. `bufio.Scanner`类型的主要功用是什么？它有哪些特点？

答：`bufio.Scanner`类型俗称带缓存的扫描器。它的功能还是比较强大的。

比如，我们可以自定义每次扫描的边界，或者说内容的分段方法。我们在调用它的`Scan`方法对目标进行扫描之前，可以先调用其`Split`方法并传入一个函数来自定义分段方法。

在默认情况下，扫描器会以行为单位对目标内容进行扫描。`bufio`代码包提供了一些现成的分段方法。实际上，扫描器在默认情况下会使用`bufio.ScanLines`函数作为分段方法。

又比如，我们还可以在扫描之前自定义缓存的载体和缓存的最大容量，这需要调用它的`Buffer`方法。在默认情况下，扫描器内部设定的最大缓存容量是64K个字节。

换句话说，目标内容中的每一段都不能超过64K个字节。否则，扫描器就会使它的`Scan`方法返回`false`，并通过其`Err`方法给予我们一个表示“`token too long`”的错误值。这里的“`token`”代表的就是一段内容。

关于`bufio.Scanner`类型的更多特点和使用注意事项，你可以通过它的文档获得。

52. 怎样通过`os`包中的 API 创建和操纵一个系统进程？



这两者都会返回一个`*os.Process`类型的值。该类型提供了一些方法，比如，用于杀掉当前进程的`Kill`方法，又比如，可以给当前进程发送系统信号的`Signal`方法，以及会等待当前进程结束的`Wait`方法。

与此相关的还有`os.ProcAttr`类型、`os.ProcessState`类型、`os.Signal`类型，等等。你可以通过积极的实践去探索更多的玩法。

53. 怎样在`net.Conn`类型的值上正确地设定针对读操作和写操作的超时时间？

答：`net.Conn`类型有 3 个可用于设置超时时间的方法，分别是：`SetDeadline`、`SetReadDeadline`和`SetWriteDeadline`。

这三个方法的签名是一模一样的，只是名称不同罢了。它们都接受一个`time.Time`类型的参数，并都会返回一个`error`类型的结果。其中的`SetDeadline`方法是用来同时设置读操作超时和写操作超时的。

有一点需要特别注意，这三个方法都会针对任何正在进行以及未来将要进行的相应操作进行超时设定。

因此，如果你要在一个循环中进行读操作或写操作的话，最好在每次迭代中都进行一次超时设定。

否则，靠后的操作就有可能因触达超时时间而直接失败。另外，如果有必要，你应该再次调用它们并传入`time.Time`类型的零值来表达不再限定超时时间。

54. 怎样优雅地停止基于 HTTP 协议的网络服务程序？

答：`net/http.Server`类型有一个名为`Shutdown`的指针方法可以实现“优雅的停止”。也就是说，它可以在不中断任何正处在活动状态的连接的情况下平滑地关闭当前的服务器。

它会先关闭所有的空闲连接，并一直等待。只有活动的连接变为空闲之后，它才会关闭它们。当所有的连接都被平滑地关闭之后，它会关闭当前的服务器并返回。当有错误发生时，



另外，你还可以通过调用 Server 值的 RegisterOnShutdown 方法来注册可以在服务器即将关闭时被自动调用的函数。

更确切地说，当前服务器的 Shutdown 方法会以异步的方式调用如此注册的所有函数。我们可以利用这样的函数来通知长连接的客户端“连接即将关闭”。

55. runtime/trace 代码包的功用是什么？

答：简单来说，这个代码包是用来帮助 Go 程序实现内部跟踪操作的。其中的程序实体可以帮助我们记录程序中各个 goroutine 的状态、各种系统调用的状态，与 GC 有关的各种事件，以及内存相关和 CPU 相关的变化，等等。

通过它们生成的跟踪记录可以通过 go tool trace 命令来查看。更具体的说明可以参看 runtime/trace 代码包的文档。

有了 runtime/trace 代码包，我们就可以为 Go 程序加装上可以满足个性化需求的跟踪器了。Go 语言标准库中有的代码包正是通过使用该包实现了自身的功能，例如 net/http/pprof 包。

好了，全部的思考题答案已经更新完了，你如果还有疑问，可以给我留言。祝你新春快乐，学习愉快。再见。

[戳此查看 Go 语言专栏文章配套详细代码。](#)



GO语言核心36讲

3个月带你通关 GO语言

郝林

《Go 并发编程实战》作者
GoHackers 技术社群发起人
前轻松筹大数据负责人



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金奖励**。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 尾声 | 愿你披荆斩棘，所向无敌

精选留言 (13)

写留言

3



...

2019-02-18

大神 发现一篇文章go的陷阱，

<http://ju.outofmemory.cn/entry/351623>

描述了很多对go不满的地方和陷阱，我想知道对于开发者在大型项目中如何避免或者进入陷阱以及如何排查。或者有什么规范要求

展开 ▼

作者回复: 大部分所谓的陷阱或者坑，都是由于不了解语言机理而犯的错误。使用编程语言B的理念和哲学去理解编程语言A必然会出现问题。



冯小贤



阅读 111



Kyle Liu

2019-03-03

1

非常感谢老师，3个月前对go一知半解的情况下看老师的文章只是走个流程很模糊。现在对go有了一个整体的认知再看老师的文章，非常受用也解决了工作中许多疑惑。



阿海

2019-02-03

1

谢谢郝老师的新年彩蛋，祝郝老师和大家新年快乐，心想事成



傻乐

2019-04-30

1

今天才真正看完，从开课到现在，有点滞后太多，因为我是数据方向的，学完收益真高，现在所有的数据深层次的bug都可以结合编程思想定位解决，还可以自己写想要的工具，谢谢

作者回复: 赞！



Geek_1b0d6...

2019-04-12

1

终于坚持这看完了整个一个专栏，谢谢郝林老师的这个专栏，让我可以对go语言的了解不在浮于表面



小强

2019-04-08

1

感谢老师，通过专栏学到很多，最近工作中需要用go去重构一部分应用，在写代码中发展，以前是基于自己的编程基础去看的本专栏，导致go的很多基础知识没记牢固，在此再定个目标，反复多过几遍，希望能真正的掌握本专栏的内容，而不光是浅层面的理解。

展开 ^

 **写点啥呢**

2019-02-14



老师有心啊，非常感谢！

展开▼

 **失了智的沫...**

2019-02-11



新年快乐，过完年才看到彩蛋，😁

展开▼

 **言十年**

2019-02-03



新年快乐

展开▼

 **方向**

2019-02-02



刚刚打开学习的课程，发现多了一讲，原来是彩蛋，太棒了

 **我来也**

2019-02-02



哈哈 还有新年彩蛋！

老师用心了。

 **beiliu**

2019-02-02



郝老师，新年快乐，一路走来，受益良多。

展开▼



下载APP 