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 Golang 语言社区-综合面试题 2

第一节 iota 的使用 2

一、公众账号： 2

二、核心内容讲解： 2

第一季　Golang 语言社区-综合面试题

第一节 iota 的使用

一、公众账号：

回复关键字：客服　　获取课程助教的微信

二、核心内容讲解：

iota 是 golang 语言的常量计数器,只能在常量的表达式中使用。

iota 在 const 关键字出现时将被重置为 0(const 内部的第一行之前)，const 中每新增一行常量声明将使 iota 计数一次(iota 可理解为 const 语句块中的行索引)。

使用 iota 能简化定义，在定义枚举时很有用。

举例如下：

1、iota 只能在常量的表达式中使用。

```
fmt.Println(iota)
```

编译错误： undefined: iota

2、每次 const 出现时，都会让 iota 初始化为 0.【自增长】

```
const a = iota // a=0
const (
  b = iota        //b=0
  c                //c=1
)
```

3、自定义类型

自增长常量经常包含一个自定义枚举类型，允许你依靠编译器完成自增设置。

```
type Stereotype int

const (
   TypicalNoob Stereotype = iota // 0
   TypicalHipster                // 1
   TypicalUnixWizard             // 2
   TypicalStartupFounder         // 3
)
```

下面是来自 time 包的例子，它首先定义了一个 Weekday 命名类型，然后为一周的每天定义了一个常量，从周日 0 开始。在其它编程语言中，这种类型一般被称为枚举类型。

```
type Weekday int

const (
    Sunday Weekday = iota
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
    Saturday
)
```

周一将对应 0，周一为 1，如此等等。

4、可跳过的值

设想你在处理消费者的音频输出。音频可能无论什么都没有任何输出，或者它可能是单声道，立体声，或是环绕立体声的。

这可能有些潜在的逻辑定义没有任何输出为 0，单声道为 1，立体声为 2，值是由通道的数量提供。

所以你给 Dolby 5.1 环绕立体声什么值。

一方面，它有 6 个通道输出，但是另一方面，仅仅 5 个通道是全带宽通道（因此 5.1 称号 - 其中 .1 表示的是低频效果通道）。
不管怎样，我们不想简单的增加到 3。
我们可以使用下划线跳过不想要的值。

```
type AudioOutput int

const (
    OutMute AudioOutput = iota // 0
    OutMono                    // 1
    OutStereo                  // 2
    _
    _
    OutSurround                // 5
)
```

5、位掩码表达式

iota 可以做更多事情，而不仅仅是 increment。更精确地说，iota 总是用于 increment，但是它可以用于表达式，在常量中的存储结果值。

```
type Allergen int

const (
    IgEggs Allergen = 1 << iota // 1 << 0 which is 00000001
    IgChocolate                 // 1 << 1 which is 00000010
    IgNuts                      // 1 << 2 which is 00000100
    IgStrawberries              // 1 << 3 which is 00001000
    IgShellfish                 // 1 << 4 which is 00010000
)
```

这个工作是因为当你在一个 const 组中仅仅有一个标示符在一行的时候，它将使用增长的 iota 取得前面的表达式并且再运用它，。在 Go 语言的 spec 中， 这就是所谓的隐性重复最后一个非空的表达式列表。

如果你对鸡蛋，巧克力和海鲜过敏，把这些 bits 翻转到 “on” 的位置（从左到右映射 bits）。然后你将得到一个 bit 值 00010011，它对应十进制的 19。

```
fmt.Println(IgEggs | IgChocolate | IgShellfish)

// output:
// 19
```

我们也可以在复杂的常量表达式中使用 iota，下面是来自 net 包的例子，用于给一个无符号整数的最低 5bit 的每个 bit 指定一个名字：

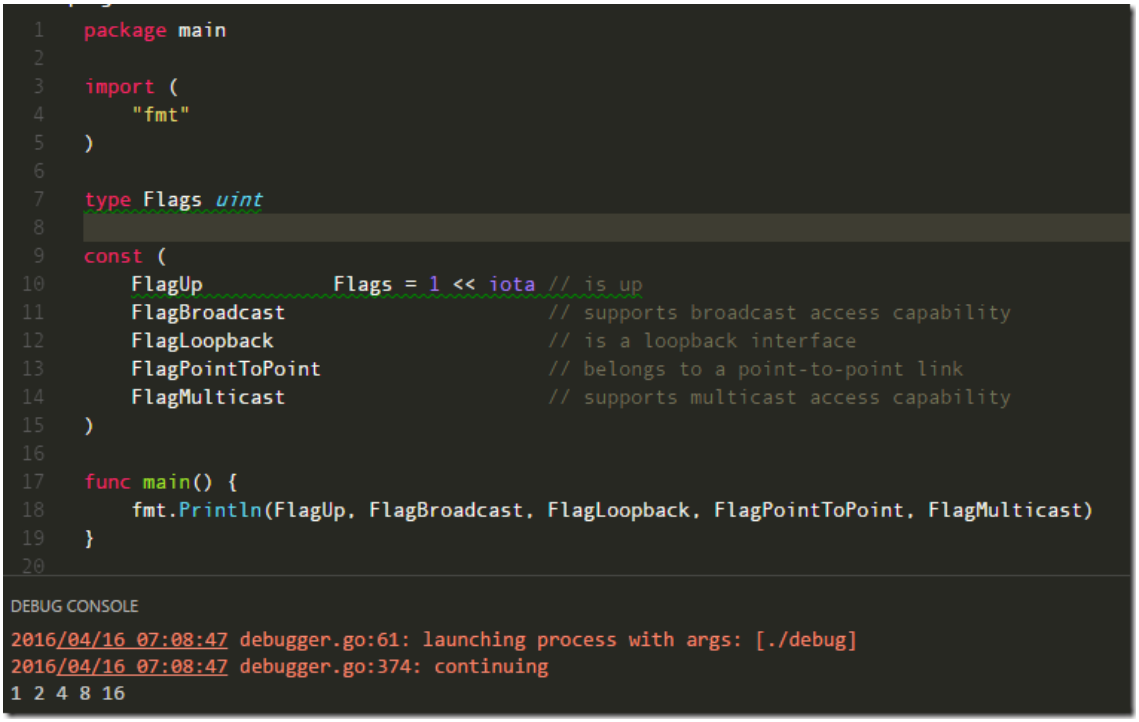
```
type Flags uint

const (
    FlagUp Flags = 1 << iota // is up
    FlagBroadcast            // supports broadcast access capability
    FlagLoopback             // is a loopback interface
    FlagPointToPoint         // belongs to a point-to-point link
```

```
    FlagMulticast             // supports multicast access capability
)
```

随着 iota 的递增，每个常量对应表达式 1 << iota，是连续的 2 的幂，分别对应一个 bit 位置。使用这些常量可以用于测试、设置或清除对应的 bit 位的值：

测试结果：



随着 iota 的递增，每个常量对应表达式 1 << iota，是连续的 2 的幂，分别对应一个 bit 位置。使用这些常量可以用于测试、设置或清除对应的 bit 位的值：

```
package main

import (
    "fmt"
)

type Flags uint

const (
    FlagUp           Flags = 1 << iota // is up
    FlagBroadcast                      // supports broadcast access
capability
    FlagLoopback                       // is a loopback interface
    FlagPointToPoint                   // belongs to a point-to-point
link
```

```
    FlagMulticast                       // supports multicast access
capability
)

func IsUp(v Flags) bool     { return v&FlagUp == FlagUp }
func TurnDown(v *Flags)     { *v &^= FlagUp }
func SetBroadcast(v *Flags) { *v |= FlagBroadcast }
func IsCast(v Flags) bool   { return
v&(FlagBroadcast|FlagMulticast) != 0 }

func main() {
    var v Flags = FlagMulticast | FlagUp
    fmt.Printf("%b %t\n", v, IsUp(v)) // "10001 true"
    TurnDown(&v)
    fmt.Printf("%b %t\n", v, IsUp(v)) // "10000 false"
    SetBroadcast(&v)
    fmt.Printf("%b %t\n", v, IsUp(v))   // "10010 false"
    fmt.Printf("%b %t\n", v, IsCast(v)) // "10010 true"
}
```

运行结果：

10001 true

10000 false

10010 false

10010 true

6、定义数量级

```
type ByteSize float64

const (
    _           = iota                  // ignore first value by
assigning to blank identifier
    KB ByteSize = 1 << (10 * iota)      // 1 << (10*1)
    MB                                  // 1 << (10*2)
    GB                                  // 1 << (10*3)
    TB                                  // 1 << (10*4)
    PB                                  // 1 << (10*5)
    EB                                  // 1 << (10*6)
    ZB                                  // 1 << (10*7)
    YB                                  // 1 << (10*8)
)
```

下面是一个更复杂的例子，每个常量都是 1024 的幂：

```
const (
    _ = 1 << (10 * iota)
    KiB // 1024
    MiB // 1048576
    GiB // 1073741824
    TiB // 1099511627776             (exceeds 1 << 32)
    PiB // 1125899906842624
    EiB // 1152921504606846976
    ZiB // 1180591620717411303424    (exceeds 1 << 64)
    YiB // 1208925819614629174706176
)
```

不过 iota 常量生成规则也有其局限性。例如，它并不能用于产生 1000 的幂（KB、MB 等），因为 Go 语言并没有计算幂的运算符。

7、定义在一行的情况

```
const (
    Apple, Banana = iota + 1, iota + 2
    Cherimoya, Durian
    Elderberry, Fig
)
```

iota 在下一行增长，而不是立即取得它的引用。

```
// Apple: 1
// Banana: 2
// Cherimoya: 2
// Durian: 3
// Elderberry: 3
// Fig: 4
```

8、中间插队

```
const (
   i = iota
   j = 3.14
   k = iota
   l
)
```

那么打印出来的结果是 i=0,j=3.14,k=2,l=3