Go 语言常量

常量是一个简单值的标识符，在程序运行时，不会被修改的量。

常量中的数据类型只可以是布尔型、数字型（整数型、浮点型和复数）和字符串型。

相对于变量，常量是恒定不变的值，多用于定义程序运行期间不会改变的那些值。 常量的声明和变量声明非常类似，只是把`var`换成了`const`，常量在定义的时候必须赋值。

定义常量的语法

定义一个常量使用`const`关键字，语法格式如下：

```
const identifier  [type]= value
```

`const`：定义常量关键字

`identifier`：常量名称

`type`：常量类型

`value`：常量的值

你可以省略类型说明符 [type]，因为编译器可以根据变量的值来推断其类型。

- 显式类型定义：`const b string = "abc"`
- 隐式类型定义：`const b = "abc"`

多个相同类型的声明可以简写为：

```
const c_name1, c_name2 = value1, value2
```

例如：

```
package main

import "fmt"

func main() {
   const LENGTH int = 10
   const WIDTH int = 5
   #或者多个常量声明
   const (
      LENGTH = 100
      WIDTH = 5
   )

   //多重赋值
   const LENGTH, WIDTH = 10, 5

   var area int

   area = LENGTH * WIDTH
```

```
    fmt.Printf("面积为 : %d", area)
}
```

常量的值必须是能够在编译时就能够确定的；你可以在其赋值表达式中涉及计算过程，但是所有用于计算的值必须在编译期间就能获得。

- 正确的做法：`const c1 = 2/3`
- 错误的做法：`const c2 = getNumber()` // 引发构建错误: `getNumber() used as value`

因为在编译期间自定义函数均属于未知，因此无法用于常量的赋值，但内置函数可以使用，如：len()。

常量可以用len(), cap(), unsafe.Sizeof()函数计算表达式的值。常量表达式中，函数必须是内置函数，否则编译不过。

数字型的常量是没有大小和符号的，并且可以使用任何精度而不会导致溢出：

```
const Ln2= 0.693147180559945309417232121458\
           176568075500134360255254120680009
const Log2E= 1/Ln2 // this is a precise reciprocal
const Billion = 1e9 // float constant
const hardEight = (1 << 100) >> 97
```

根据上面的例子我们可以看到，反斜杠 `\` 可以在常量表达式中作为多行的连接符使用。

与各种类型的数字型变量相比，你无需担心常量之间的类型转换问题，因为它们都是非常理想的数字。

不过需要注意的是，当常量赋值给一个精度过小的数字型变量时，可能会因为无法正确表达常量所代表的数值而导致溢出，这会在编译期间就引发错误。

另外，**常量也允许使用并行赋值的形式**：

```
const beef, two, c = "eat", 2, "veg"
const Monday, Tuesday, Wednesday, Thursday, Friday, Saturday = 1, 2, 3, 4, 5, 6
const (
    Monday, Tuesday, Wednesday = 1, 2, 3
    Thursday, Friday, Saturday = 4, 5, 6
)
```

如果定义多个常量时，如果省略了值，则和第一个值相同

```
const (
    A = 10
    B
    C
)
结果：A=10,B=10,C=10
```

常量还可以用作枚举：

```
const (
    Unknown = 0
    Female = 1
    Male = 2
)
```

现在，数字 0、1 和 2 分别代表未知性别、女性和男性。这些枚举值可以用于测试某个变量或常量的实际值。

iota

`iota`，特殊常量，可以认为是一个可以被编译器修改的常量。

`iota`是`go`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`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)。** `iota` 可以被用作枚举值， **使用`iota`能简化定义，在定义枚举时很有用。**

```
const (
    a = iota //0
    b = iota //1
    c = iota //2
)
```

第一个 `iota` 等于 0，每当 `iota` 在新的一行被使用时，它的值都会自动加 1；所以 `a=0, b=1, c=2` 可以简写为如下形式：

```
const (
    a = iota  //0
    b         //1
    c         //2
)
const (
    n1 = iota //0
    n2 = 100  //100
    n3 = iota //2
    n4        //3
)
const n5 = iota //0
```

`iota` 也可以用在表达式中，如：`iota + 50`。在每遇到一个新的常量块或单个常量声明时， `iota` 都会重置为 0（**简单地讲，每遇到一次 const 关键字，iota 就重置为 0**）。

当然，常量之所以为常量就是恒定不变的量，因此我们无法在程序运行过程中修改它的值；如果你在代码中试图修改常量的值则会引发编译错误。

iota 用法

iota声明中间可以插队，但是索引值还是递增的

```
package main

import "fmt"

func main() {
    const (
        a = iota //0
        b        //1
        c        //2
        d = "ha" //独立值，iota += 1
        e        //"ha"   iota += 1
```

```
        f = 100  //iota +=1
        g        //100  iota +=1
        h = iota //7,恢复计数
        i        //8
    )
    fmt.Println(a, b, c, d, e, f, g, h, i)
}
```

以上实例运行结果为：

```
0 1 2 ha ha 100 100 7 8
```

再看个有趣的的 iota 实例：

```
package main

import "fmt"

const (
    i = 1 << iota
    j = 3 << iota
    k
    l
)

func main() {
    fmt.Println("i=", i)
    fmt.Println("j=", j)
    fmt.Println("k=", k)
    fmt.Println("l=", l)
}
```

以上实例运行结果为：

```
i= 1
j= 6
k= 12
l= 24
```

iota 表示从 0 开始自动加 1，所以 **i=1<<0**, **j=3<<1**（<< 表示左移的意思），即：i=1, j=6，这没问题，关键在 k 和 l，从输出结果看 **k=3<<2**，**l=3<<3**。

简单表述:

- **i=1**：左移 0 位，不变仍为 1。
- **j=3**：左移 1 位，变为二进制 **110**，即 6。
- **k=3**：左移 2 位，变为二进制 **1100**，即 12。
- **l=3**：左移 3 位，变为二进制 **11000**，即 24。

注：**<<n==*(2^n)**。

多个`iota`定义在一行

```
const (
    a, b = iota + 1, iota + 2 //1,2
    c, d                      //2,3
    e, f                      //3,4
)
```

使用**下划线**_跳过某些值

```
const (
    a = iota //0
    _ = iota //1  丢弃该值，常用在错误处理中
    c = iota //2
)
```