数字类型

整型 int 和浮点型 float

Go 语言支持整型和浮点型数字，并且原生支持复数，其中位的运算采用补码。

Go 也有基于架构的类型，例如：int、uint 和 uintptr。

这些类型的长度都是根据运行程序所在的操作系统类型所决定的：

- `int` 和 `uint` 在 32 位操作系统上，它们均使用 32 位（4 个字节），在 64 位操作系统上，它们均使用 64 位（8 个字节）。
- `uintptr` 的长度被设定为足够存放一个指针即可。

Go 语言中没有 float 类型。（Go语言中只有 `float32` 和 `float64`）

与操作系统架构无关的类型都有固定的大小，并在类型的名称中就可以看出来：

整数：

- int8（-128 -> 127）
- int16（-32768 -> 32767）
- int32（-2,147,483,648 -> 2,147,483,647）
- int64（-9,223,372,036,854,775,808 -> 9,223,372,036,854,775,807）

无符号整数：

- uint8（0 -> 255）
- uint16（0 -> 65,535）
- uint32（0 -> 4,294,967,295）
- uint64（0 -> 18,446,744,073,709,551,615）

浮点型（IEEE-754 标准）：

- float32（+- 1e-45 -> +- 3.4 * 1e38）
- float64（+- 5 *1e-324* -> *107* 1e308）

int 型是计算最快的一种类型。

整型的零值（默认值）为 0，浮点型的零值（默认值）为 0.0。

float32 精确到小数点后 7 位，float64 精确到小数点后 15 位。由于精确度的缘故，你在使用 `==` 或者 `!=` 来比较浮点数时应当非常小心。你最好在正式使用前测试对于精确度要求较高的运算。

你应该尽可能地使用 float64，因为 `math` 包中所有有关数学运算的函数都会要求接收这个类型。

你可以通过增加前缀 0 来表示 8 进制数（如：077），增加前缀 0x 来表示 16 进制数（如：0xFF），以及使用 e 来表示 10 的连乘（如： 1e3 = 1000，或者 6.022e23 = 6.022 x 1e23）。

你可以使用 `a := uint64(0)` 来同时完成类型转换和赋值操作，这样 a 的类型就是 uint64。

示例：

```
package main

func main() {
```

```
    var int_8 int8
    var int_16 int16
    var int_32 int32
    var int_64 int64
    var uint_8 uint8
    var uint_16 uint16
    var uint_32 uint32
    var uint_64 uint64
    var float_32 float32
    var float_64 float64
}
```

格式化说明符

在格式化字符串里，`%d` 用于格式化整数（`%x` 和 `%X` 用于格式化 16 进制表示的数字，`%b` 表示二进制，`%o` 代表8进制。），`%g` 用于格式化浮点型（`%f` 输出浮点数，`%e` 输出科学计数表示法），`%0d` 用于规定输出定长的整数，其中开头的数字 0 是必须的。

`%n.mg` 用于表示数字 n 并精确到小数点后 m 位，除了使用 g 之外，还可以使用 e 或者 f，例如：使用格式化字符串 `%5.2e` 来输出 3.4 的结果为 `3.40e+00`。

数字值转换

当进行类似 `a32bitInt = int32(a32Float)` 的转换时，小数点后的数字将被丢弃。这种情况一般发生当从取值范围较大的类型转换为取值范围较小的类型时，或者你可以写一个专门用于处理类型转换的函数来确保没有发生精度的丢失。下面这个例子展示如何安全地从 int 型转换为 int8：

```
func Uint8FromInt(n int) (uint8, error) {
    if 0 <= n && n <= math.MaxUint8 { // conversion is safe
        return uint8(n), nil
    }
    return 0, fmt.Errorf("%d is out of the uint8 range", n)
}
```

或者安全地从 float64 转换为 int：

```
func IntFromFloat64(x float64) int {
    if math.MinInt32 <= x && x <= math.MaxInt32 { // x lies in the integer range
        whole, fraction := math.Modf(x)
        if fraction >= 0.5 {
            whole++
        }
        return int(whole)
    }
    panic(fmt.Sprintf("%g is out of the int32 range", x))
}
```

不过如果你实际存的数字超出你要转换到的类型的取值范围的话，则会引发 panic

复数

复数，是数的概念扩展。我们把形如z=a+bi（a、b均为实数）的数称为复数。其中，a称为实部，b称为虚部，i称为虚数单位。当z的虚部b = 0时，则z为实数；当z的虚部b≠0时，实部a = 0时，常称z为纯虚数。

Go 拥有以下复数类型：

```
complex64 (32 位实数和虚数)
complex128 (64 位实数和虚数)
```

复数使用 `re+imI` 来表示，其中 `re` 代表实数部分，`im` 代表虚数部分，I 代表根号负 1。

示例：

```
var c1 complex64 = 5 + 10i
fmt.Printf("The value is: %v", c1)
#结果
The value is: (5+10i)
```

如果 `re` 和 `im` 的类型均为 float32，那么类型为 complex64 的复数 c 可以通过以下方式来获得：

```
c = complex(re, im)
```

函数 `real(c)` 和 `imag(c)` 可以分别获得相应的实数和虚数部分。

在使用格式化说明符时，可以使用 `%v` 来表示复数，但当你希望只表示其中的一个部分的时候需要使用 `%f`。

复数支持和其它数字类型一样的运算。当你使用等号 `==` 或者不等号 `!=` 对复数进行比较运算时，注意对精确度的把握。`cmath` 包中包含了一些操作复数的公共方法。如果你对内存的要求不是特别高，最好使用 complex128 作为计算类型，因为相关函数都使用这个类型的参数。