Go语言 标准库 Math包

关于go的int类型的解释，go的整数基本类型包括int，int8，int16，int32，int64，那么类型int有多少位(byte)？多少字节？

答：首先，1字节=8位(byte)，int8为一个字节，8位(byte)，以此类推。

int的字节数和其对应的最值，与操作系统的位数有关，若操作系统为8位，则类型int对应类型int8，最值和int8相等；若操作系统为64位，则类型int对应类型int64，最值和int64相等；

大部分PC端和服务器系统为64位，单片机位8位。

math包包含一些常量和一些有用的数学计算函数,例如:三角函数、随机数、绝对值、平方等。

常量

```
fmt.Printf("Float64的最大值: %.f\n", math.MaxFloat64)
fmt.Printf("Float64最小值: %.f\n", math.SmallestNonzeroFloat64)
fmt.Printf("Float32最大值: %.f\n", math.MaxFloat32)
fmt.Printf("Float32最小值: %.f\n", math.SmallestNonzeroFloat32)
fmt.Printf("Int8最大值: %d\n", math.MaxInt8)
fmt.Printf("Int8最小值: %d\n", math.MinInt8)
fmt.Printf("Uint8最大值: %d\n", math.MaxUint8)
fmt.Printf("Int16最大值: %d\n", math.MaxInt16)
fmt.Printf("Int16最小值: %d\n", math.MinInt16)
fmt.Printf("Uint16最大值: %d\n", math.MaxUint16)
fmt.Printf("Int32最大值: %d\n", math.MaxInt32)
fmt.Printf("Int32最小值: %d\n", math.MinInt32)
fmt.Printf("Uint32最大值: %d\n", math.MaxUint32)
fmt.Printf("Int64最大值: %d\n", math.MaxInt64)
fmt.Printf("Int64最小值: %d\n", math.MinInt64)
fmt.Printf("圆周率默认值: %v\n", math.Pi)
```

常量如下：

```
Float64的最大值:
179769313486231570814527423731704356798070567525844996598917476803157260780028538760589558632766878171540458953514382464234321326889464182768467546703537516986049910576551282076245490090389328944075868508455133942304583236903222948165808559332123348274797826204144723168738177180919299881250404026184124858368
Float64最小值: 0
Float32最大值: 340282346638528859811704183484516925440
Float32最小值: 0
Int8最大值: 127
Int8最小值: -128
Uint8最大值: 255
Int16最大值: 32767
Int16最小值: -32768
Uint16最大值: 65535
Int32最大值: 2147483647
Int32最小值: -2147483648
Uint32最大值: 4294967295
```

```
Int64最大值: 9223372036854775807
Int64最小值: -9223372036854775808
圆周率默认值: 3.141592653589793
```

常用函数

IsNaN函数 判断是否是NAN

```
func IsNaN(f float64) (is bool)
```

报告f是否表示一个NaN（Not A Number）值，是数值返回一个false，不是数值则返回一个true。

```
func testIsNaN() {
    fmt.Println(math.IsNaN(12321.321321))    //false
}
```

Ceil函数 向上取整

```
func Ceil(x float64) float64
```

返回一个不小于x的最小整数，简单来说就是向上取整。

```
func testCeil() {
    fmt.Println(math.Ceil(1.13456))    //2
}
```

Floor函数 向下取整

```
func Floor(x float64) float64
```

返回一个不大于x的最小整数，简单来说就是向下取整。

```
func testFloor() {
    fmt.Println(math.Floor(2.9999))    //2
}
```

Trunc函数 取整

```
func Trunc(x float64) float64
```

返回x整数部分。

```
func testTrunc() {
    fmt.Println(math.Trunc(2.9999))    //2
}
```

Abs函数 绝对值

```
func Abs(x float64) float64
```

返回x的绝对值。

```
func testAbs() {
    fmt.Println(math.Abs(1.1))   //1.1
    fmt.Println(math.Abs(-4.58)) //4.58
}
```

Max函数 比大

```
func Max(x, y float64) float64
```

返回x和y中最大值。

```
func testMax() {
    fmt.Println(math.Max(1000,200))    //1000
}
```

Min函数 比小

```
func Min(x, y float64) float64
```

返回x和y中最小值

```
func testMin() {
    fmt.Println(math.Min(1000,200))    //200
}
```

Dim函数 差值和比较取最大值

```
func Dim(x, y float64) float64
```

函数返回x-y和0中的最大值。

```
func testDim() {
    fmt.Println(math.Dim(1000,2000))    //0
    fmt.Println(math.Dim(1000,200))    //800
}
```

Mod函数 取余

```
func Mod(x, y float64) float64
```

取余运算，可以理解为 x-Trunc(x/y)*y，结果的正负号和x相同。

```
func testMod() {
    fmt.Println(math.Mod(123,0))    //NaN
    fmt.Println(math.Mod(123,10))    //3
}
```

Sqrt函数 平方根运算

```
func Sqrt(x float64) float64
```

返回x的二次方根,平方根。

```
func testSqrt() {
    fmt.Println(math.Sqrt(144))    //12
}
```

Cbrt函数 二次方根

```
func Cbrt(x float64) float64
```

返回x的二次方根,平方根

```
func testCbrt() {
    fmt.Println(math.Cbrt(1728))    //12
}
```

Hypot函数

```
func Hypot(p, q float64) float64
```

返回Sqrt(p * p + q * q)，注意要避免不必要的溢出或下溢。

```
func testHypot() {
    fmt.Println(math.Hypot(12,12))    //16.970562748477143
}
```

Pow函数 求幂

```
func Pow(x, y float64) float64
```

求幂，x的y次方

```
func testPow() {
    fmt.Println(math.Pow(2,3))    //8
}
```

Sin函数 正弦

```
func Sin(x float64) float64
```

求正弦

```
func testSin() {
    fmt.Println(math.Sin(12))    //-0.5365729180004349
}
```

Cos函数 余弦

```
func Cos(x float64) float64
```

求余弦

```
func testCos() {
    fmt.Println(math.Cos(12))    //0.8438539587324921
}
```

Tan函数 正切

```
func Tan(x float64) float64
```

求正切

```
func testTan() {
    fmt.Println(math.Tan(12))    //-0.6358599286615807
}
```

Log函数

```
func Log(x float64) float64
```

求自然对数

```
func testLog() {
    fmt.Println(math.Log(2))    //0.6931471805599453
}
```

Log2函数

```
func Log2(x float64) float64
```

求2为底的对数

```
func testLog2() {
    fmt.Println(math.Log2(128))    //7
}
```

Log10函数

```
func Log10(x float64) float64
```

求10为底的对数

```
func testLog10() {
    fmt.Println(math.Log10(10000))    //4
}
```

Signbit函数 判断值正负

```
func Signbit(x float64) bool
```

如果x是一个负数或者负零，返回true

```
func testSignbit() {
    fmt.Println(math.Signbit(10000))    //false
    fmt.Println(math.Signbit(-200))    //true
}
```

随机数math/rand

math/rand包是go提供用来产生各种各样随机数的包。

注意：rand生成的数值虽然说是随机数，但它其实是**伪随机数**，关于为什么是伪随机数，而不是真正的随机数，本文不做详细讲解，因为我也不是太清楚，只是提出这一点；简单说一下我的理解：真正的随机数是无规则可循的，就像抛硬币，正反面是真正随机的，这是一个真正的随机案例。计算机底层生成一个数值，究其根源它也是程序员们根据某种算法得到的数值，而是要是人为操控的计算就一定有规律可循，只是这个规律不是肉眼可见的。所以说是伪随机数。

rand实现的几个方法：

函数	说明
`func (r *Rand) Int() int`	返回一个非负的伪随机int值。
`func (r *Rand) Int31() int32`	返回一个int32类型的非负的31位伪随机数。
`func (r *Rand) Intn(n int) int`	返回一个取值范围在[0,n)的伪随机int值，如果n<=0会panic。
`func Int63() int64`	返回一个int64类型的非负的63位伪随机数。
`func Uint32() uint32`	返回一个uint32类型的非负的32位伪随机数。
`func Uint64() uint64`	返回一个uint64类型的非负的32位伪随机数。
`func Int31n(n int32) int32`	返回一个取值范围在[0,n)的伪随机int32值，如果n<=0会panic。
`func Int63n(n int64) int64`	返回一个取值范围在[0, n)的伪随机int64值，如果n<=0会panic。
`func Float32() float32`	返回一个取值范围在[0.0, 1.0)的伪随机float32值。
`func Float64() float64`	返回一个取值范围在[0.0, 1.0)的伪随机float64值。
`func Perm(n int) []int`	返回一个有n个元素的，[0,n)范围内整数的伪随机的切片。
`func Read(p []byte) (n int, err error)`	生成len§个伪随机数，伪随机数的范围为0-255；并将伪随机数存入p，返回len§和可能发生的错误。
`func NewSource(seed int64) Source`	使用给定的种子创建一个伪随机资源。
`func New(src Source) *Rand`	返回一个使用src随机源生成一个Rand。

简单示例：

```
package main

import (
  "fmt"
  "math/rand"
)

func main() {
  // 直接调用rand的方法生成伪随机int值
  fmt.Println(rand.Int()) // 5577006791947779410
  fmt.Println(rand.Int31()) // 2019727887
  fmt.Println(rand.Intn(5)) // 2
}
```

但是当把代码运行多次发现，结果都是一样的。不管怎么运行代码，产生的结果都是这三个数，结果不会变。**这是因为我们还没有设置随机数种子的原因。**

```
func (r *Rand) Seed(seed int64)
```

使用给定的seed来初始化生成器到一个确定的状态，这就是设置随机种子。

```
package main

import (
  "fmt"
  "math/rand"
)

func main() {
  // 直接调用rand的方法生成伪随机int值
  rand.Seed(time.Now().Unix()) // 设置种子，我们以当前时间的秒；当然也可以用毫秒，微秒等
  fmt.Println(rand.Int())
  fmt.Println(rand.Int31())
  fmt.Println(rand.Intn(5))
}
```

上面的代码，多次运行，就会发现，结果是不一样的了。

实例演示：

```
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func init() {
    // 以时间作为初始化种子
    rand.Seed(time.Now().UnixNano())
}
```

```
func myRand() {
    for i := 0; i < 10; i++ {
        a := rand.Int()
        fmt.Printf("a: %v\n", a)
    }
    fmt.Println("----------")

    for i := 0; i < 10; i++ {
        a := rand.Intn(100)
        fmt.Printf("a: %v\n", a)
    }
    fmt.Println("----------")

    for i := 0; i < 10; i++ {
        a := rand.Float32()
        fmt.Printf("a: %v\n", a)
    }
    fmt.Println("----------")

    source := rand.NewSource(time.Now().UnixNano()) // 使用当前的纳秒生成一个随机源，也就是随机种子。NewSource()方法就等价于前面的rand.Seed()方法,都是用来设置随机种子。，这两种方法本质上没有区别。
    ran := rand.New(source) // 生成一个rand
    fmt.Println(ran.Int()) // 获取随机数
}

func main() {
    myRand()
}
```