

Go 语言基本数据类型

主讲教师：（大地）

合作网站：www.itying.com （IT 营）

我的专栏：<https://www.itying.com/category-79-b0.html>

1、Golang 数据类型介绍.....	1
2、整型.....	1
3、浮点型.....	5
4、布尔值.....	6
5、字符串.....	7
6、byte 和 rune 类型.....	10
7、修改字符串.....	11

1、Golang 数据类型介绍

Go 语言中数据类型分为：基本数据类型和复合数据类型

基本数据类型有：

整型、浮点型、布尔型、字符串

复合数据类型有：

数组、切片、结构体、函数、map、通道（channel）、接口等。

2、整型

整型分为以下两个大类：

有符号整形按长度分为：int8、int16、int32、int64

对应的无符号整型：uint8、uint16、uint32、uint64

类型	范围	占用空间	有无符号
int8	(-128 到 127) -2^7 到 2^7-1	1 个字节	有
int16	(-32768 到 32767) -2^{15} 到 $2^{15}-1$	2 个字节	有
int32	(-2147483648 到 2147483647) -2^{31} 到 $2^{31}-1$	4 个字节	有
int64	(-9223372036854775808 到 9223372036854775807) -2^{63} 到 $2^{63}-1$	8 个字节	有
uint8	(0 到 255) 0 到 2^8-1	1 个字节	无
uint16	(0 到 65535) 0 到 $2^{16}-1$	2 个字节	无
uint32	(0 到 4294967295) 0 到 $2^{32}-1$	4 个字节	无
uint64	(0 到 18446744073709551615) 0 到 $2^{64}-1$	8 个字节	无

关于字节：

字节也叫 Byte，是计算机数据的基本存储单位。8bit(位)=1Byte(字节) 1024Byte(字节)=1KB 1024KB=1MB 1024MB=1GB 1024GB=1TB 。在电脑里一个中文字是占两个字节的。

特殊整型

类型	描述
uint	32 位操作系统上就是 uint32，64 位操作系统上就是 uint64
int	32 位操作系统上就是 int32，64 位操作系统上就是 int64
uintptr	无符号整型，用于存放一个指针

注意： 在使用 int 和 uint 类型时，不能假定它是 32 位或 64 位的整型，而是考虑 int 和 uint 可能在不同平台上的差异。

注意事项： 实际项目中整数类型、切片、map 的元素数量等都可以用 int 来表示。在涉及到二进制传输、为了保持文件的结构不会受到不同编译目标平台字节长度的影响，不要使用 int 和 uint。

```
package main
import (
    "fmt"
)
func main() {
    var num int64
    num = 123
    fmt.Printf("值:%v 类型%T", num, num)
}
```

unsafe.Sizeof

unsafe.Sizeof(n1) 是 unsafe 包的一个函数，可以返回 n1 变量占用的字节数

```
package main

import (
    "fmt"
    "unsafe"
)

func main() {

    var a int8 = 120
    fmt.Printf("%T\n", a)
    fmt.Println(unsafe.Sizeof(a))
}
```

int 不同长度直接的转换

```
package main
import (
    "fmt"
)
func main() {
    var num1 int8
    num1 = 127
    num2 := int32(num1)
    fmt.Printf("值:%v 类型%T", num2, num2) //值:127 类型 int32
}
```

数字字面量语法（Number literals syntax）（了解）

Go1.13 版本之后引入了数字字面量语法，这样便于开发者以二进制、八进制或十六进制浮点数的格式定义数字，例如：

v := 0b00101101， 代表二进制的 101101，相当于十进制的 45。 v := 0o377，代表八进制的

377，相当于十进制的 255。 $v := 0x1p-2$ ，代表十六进制的 1 除以 2^2 ，也就是 0.25。而且还允许我们用 _ 来分隔数字，比如说：

$v := 123_456$ 等于 123456。

我们可以借助 fmt 函数来将一个整数以不同进制形式展示。

```
package main

import "fmt"

func main(){

    // 十进制

    var a int

    a = 10

    fmt.Printf("%d \n", a) // 10

    fmt.Printf("%b \n", a) // 1010 占位符%b 表示二进制


    // 八进制 以 0 开头

    var b int

    b = 077

    fmt.Printf("%o \n", b) // 77


    // 十六进制 以 0x 开头

    var c int

    c = 0xff

    fmt.Printf("%x \n", c) // ff

    fmt.Printf("%X \n", c) // FF

    fmt.Printf("%d \n", c) // 255

}
```

参考: <http://docscn.studygolang.com/pkg/fmt/>

3、浮点型

Go 语言支持两种浮点型数: float32 和 float64。这两种浮点型数据格式遵循 IEEE 754 标准: float32 的浮点数的最大范围约为 $3.4e38$, 可以使用常量定义: math.MaxFloat32。float64 的浮点数的最大范围约为 $1.8e308$, 可以使用一个常量定义: math.MaxFloat64。

打印浮点数时, 可以使用 fmt 包配合动词%f, 代码如下:

```
package main
import (
    "fmt"
    "math"
)

func main() {
    fmt.Printf("%f\n", math.Pi) //默认保留 6 位小数
    fmt.Printf("%.2f\n", math.Pi) //保留 2 位小数
}
```

Go 语言中浮点数默认是 float64

```
num := 1.1
fmt.Printf("值: %v--类型:%T", num, num) //值: 1.1--类型:float64
```

Golang 中 float 精度丢失问题

几乎所有的编程语言都有精度丢失这个问题, 这是典型的二进制浮点数精度损失问题, 在定长条件下, 二进制小数和十进制小数互转可能有精度丢失。

```
d := 1129.6
fmt.Println((d * 100)) //输出: 112959.99999999999
```

```
var d float64 = 1129.6
fmt.Println((d * 100)) //输出: 112959.99999999999
```

```
m1 := 8.2  
m2 := 3.8  
fmt.Println(m1 - m2) // 期望是 4.4, 结果打印出了 4.399999999999999
```

使用第三方包来解决精度损失问题:

<https://github.com/shopspring/decimal>

Golang 科学计数法表示浮点类型

```
num8 := 5.1234e2 // ? 5.1234 * 10 的 2 次方  
num9 := 5.1234E2 // ? 5.1234 * 10 的 2 次方 shift+alt+向下的箭头  
num10 := 5.1234E-2 // ? 5.1234 / 10 的 2 次方 0.051234  
  
fmt.Println("num8=", num8, "num9=", num9, "num10=", num10)
```

4、布尔值

Go 语言中以 bool 类型进行声明布尔型数据, 布尔型数据只有 true (真) 和 false (假) 两个值。

注意:

1. 布尔类型变量的默认值为 false。
2. Go 语言中不允许将整型强制转换为布尔型。
3. 布尔型无法参与数值运算, 也无法与其他类型进行转换。

```
package main  
import (  
    "fmt"  
    "unsafe"  
)  
func main() {  
    var b = true  
    fmt.Println(b, "占用字节: ", unsafe.Sizeof(b))  
}
```

5、字符串

Go 语言中的字符串以原生数据类型出现，使用字符串就像使用其他原生数据类型（int、bool、float32、float64 等）一样。Go 语言里的字符串的内部实现使用 UTF-8 编码。字符串的值为双引号(")中的内容，可以在 Go 语言的源码中直接添加非 ASCII 码字符，例如：

```
s1 := "hello"  
s2 := "你好"
```

字符串转义符

Go 语言的字符串常见转义符包含回车、换行、单双引号、制表符等，如下表所示。

转义符	含义
\r	回车符（返回行首）
\n	换行符（直接跳到下一行的同列位置）
\t	制表符
\'	单引号
\"	双引号
\\	反斜杠

举个例子，我们要打印一个 Windows 平台下的一个文件路径：

```
package main  
import (  
    "fmt"  
)  
func main() {  
    fmt.Println("str := \"c:\\Code\\demo\\go.exe\"")  
}
```

多行字符串

Go 语言中要定义一个多行字符串时，就必须使用反引号字符：

```
s1 := `第一行  
第二行`
```

第三行
、

`fmt.Println(s1)`

反引号间换行将被作为字符串中的换行，但是所有的转义字符均无效，文本将会原样输出。

字符串的常用操作

方法	介绍
<code>len(str)</code>	求长度
<code>+</code> 或 <code>fmt.Sprintf</code>	拼接字符串
<code>strings.Split</code>	分割
<code>strings.Contains</code>	判断是否包含
<code>strings.HasPrefix</code> , <code>strings.HasSuffix</code>	前缀/后缀判断
<code>strings.Index()</code> , <code>strings.LastIndex()</code>	子串出现的位置
<code>strings.Join(a[]string, sep string)</code>	join 操作

`len(str)`求字符串的长度

```
var str = "this is str"
fmt.Println(len(str))
```

拼接字符串

```
var str1 = "你好"
var str2 = "golang"

fmt.Println(str1 + str2)

var str3 = fmt.Sprintf("%v %v", str1, str2)
fmt.Println(str3)
```

`strings.Split` 分割字符串

```
var str = "123-456-789"
```



```
var arr = strings.Split(str, "-")  
fmt.Println(arr)
```

拼接字符串

```
var str = "this is golang"  
var flag = strings.Contains(str, "golang")  
fmt.Println(flag)
```

判断首字符尾字母是否包含指定字符

```
var str = "this is golang"  
var flag = strings.HasPrefix(str, "this")  
fmt.Println(flag)  
  
var str = "this is golang"  
var flag = strings.HasSuffix(str, "go")  
fmt.Println(flag)
```

判断字符串出现的位置

```
var str = "this is golang"  
var index = strings.Index(str, "is") //从前往后  
fmt.Println(index)  
  
var str = "this is golang"  
var index = strings.LastIndex(str, "is") //从后往前  
fmt.Println(index)
```

Join 拼接字符串

```
var str = "123-456-789"  
var arr = strings.Split(str, "-")  
var str2 = strings.Join(arr, "*")  
fmt.Println(str2)
```

6、byte 和 rune 类型

组成每个字符串的元素叫做“字符”，可以通过遍历字符串元素获得字符。字符用单引号（'）包裹起来，如：

```
package main
import "fmt"
func main() {
    a := 'a'
    b := '0'
    //当我们直接输出 byte（字符）的时候输出的是这个字符对应的码值
    fmt.Println(a)
    fmt.Println(b)

    //如果我们要输出这个字符，需要格式化输出
    fmt.Printf("%c--%c", a, b) // %c 相应 Unicode 码点所表示的字符
}
```

字节（byte）：是计算机中 数据处理 的基本单位，习惯上用大写 B 来表示,1B（byte,字节）= 8bit（位）

字符：是指计算机中使用的字母、数字、字和符号

一个汉子占用 3 个字节 一个字母占用一个字节

```
a := "m"
fmt.Println(len(a)) //1

b := "张"
fmt.Println(len(b)) //3
```

Go 语言的字符有以下两种：

1. uint8 类型，或者叫 byte 型，代表了 ASCII 码的一个字符。
2. rune 类型，代表一个 UTF-8 字符。

当需要处理中文、日文或者其他复合字符时，则需要用到 rune 类型。rune 类型实际是一个 int32。

Go 使用了特殊的 rune 类型来处理 Unicode，让基于 Unicode 的文本处理更为方便，也可以使用 byte 型进行默认字符串处理，性能和扩展性都有照顾。

```
// 遍历字符串
package main

import "fmt"

func main() {
    s := "hello 张三"
    for i := 0; i < len(s); i++ { //byte
        fmt.Printf("%v(%c) ", s[i], s[i])
    }
    fmt.Println()

    for _, r := range s { //rune
        fmt.Printf("%v(%c) ", r, r)
    }
    fmt.Println()
}
```

输出:

```
104(h) 101(e) 108(l) 108(l) 111(o) 32( ) 229(à) 188(¼) 160( ) 228(ä) 184(,) 137()
104(h) 101(e) 108(l) 108(l) 111(o) 32( ) 24352(张) 19977(三)
```

因为 UTF8 编码下一个中文汉字由 3 个字节组成，所以我们不能简单的按照字节去遍历一个包含中文的字符串，否则就会出现上面输出中第一行的结果。

字符串底层是一个 byte 数组，所以可以和 []byte 类型相互转换。字符串是不能修改的，字符串是由 byte 字节组成，所以字符串的长度是 byte 字节的长度。 rune 类型用来表示 utf8 字符，一个 rune 字符由一个或多个 byte 组成。

rune 类型实际是一个 int32

```
c3 := "营"
c4 := '营'
fmt.Printf("C3 的类型%T--C4 的类型%T", c3, c4) //C3 的类型 string--C4 的类型 int32
```

7、修改字符串

要修改字符串，需要先将其转换成 []rune 或 []byte，完成后再转换为 string。无论哪种转换，都会重新分配内存，并复制字节数组。



```
func changeString() {  
    s1 := "big"  
    // 强制类型转换  
    byteS1 := []byte(s1)  
    byteS1[0] = 'p'  
    fmt.Println(string(byteS1))  
  
    s2 := "白萝卜"  
    runeS2 := []rune(s2)  
    runeS2[0] = '红'  
    fmt.Println(string(runeS2))  
}
```