基础数据类型

在 vars.go / func 点兵() 有过简要介绍~

自定义类型 type

```
type Weekday int // Weekday wd -- int wd
```

也即C/C++ typedef

```
typedef int Weekday;
```

甚至还有 #define 和 using

```
#define Weekday int
```

```
using Weekday = int;
```

我的天哪C学长😘

整型

基础类型

类型	解释
int8, int16, int32, int64, int(32 or 64，看情况)	int8, 8bit 有符号整型
uint8, uint16, uint32, uint64, uint	无符号整型
type(= uint8), rune(= int32)	字节(8 bit)，Unicode码点(32 bit)
uintptr	

不管到底是不是都是32bit， int 和 int32 都是**不同类型**（的兄弟类型），需要将 int 当作 int32 类型的地方需要一个显式的类型转换操作

while in C/C++

Go	C/C++
int8	int8_t(C99+)
int16	int16_t
int32	int32_t
int64	int64_t
int	int
uint8	uint8_t
uint16	uint16_t
uint32	uint32_t
uint64	uint64_t
uint	unsigned int
uintptr	uintptr_t(C99+)

C/C++还有__int128_t！win!😘
C学长还是你C学长~
学弟的模仿罢了 ——C学长

溢出

此事于《计组春秋》亦有记载

int -- 2的补码形式，最高位符号位
$-2^{n-1} - 2^{n-1} - 1$
e.g int8：-128~127

uint -- 无符号，非负数
$0 - 2^n - 1$
e.g. uint8：0~255

计算结果是溢出，超出的高位的bit位部分将被丢弃

```
var u uint8 = 255
fmt.Println(u, u+1, u*u) // "255 0 1"

var i int8 = 127
fmt.Println(i, i+1, i*i) // "127 -128 1"
```

原来C学长也会溢出吗😋

运算符

```
*      /      %      <<       >>     &       &^
+      -      |      ^
==     !=     <      <=       >      >=
&&
||
```

恍然大悟😍

麻麻？ &^ 是什么动物？
孩子，那是fvv...
咳咳，**按位清除(AND NOT)**

```
z = x &^ y  // 等价于：z = x & (~y)
```

用 y 对 x 按位清除，如果y的某位是1，则x对应位被置为0；如果y的某位是0，则x对应位不变

```
1011 &^ 1000 = 0011
```

Go有 &^ ，但C没有，此为一胜！
Go一胜，C零胜，此为二胜！
...
Go学长完胜！！ 🥳 😋

类姓制度

严格的类姓制度，跨类别操作会——报错！ 👺

```
var apples int32 = 1
var oranges int16 = 2
var compote int = apples + oranges // compile error
// invalid operation: apples + oranges (mismatched types int32 and int16)
```

但是

```
int a = 1;
long long b = 2;
int c = a + b; // √
```

太封建了Go！ 🥰

apple真的很喜欢orange酱，于是
很简单，转成int就是了🤪👆

```
var compote = int(apples) + int(oranges)
```

不同类型需要**显示转换**！
ps: float -> int 小数截断

```
f := 1.99
fmt.Println(int(f)) // "1"
```

浮点数

基础类型

float32 float64

算术规范：IEEE754浮点数国际标准

范围？可查看math包里的常量：
`math.MaxFloat32`，`math.MaxFloat64` 等

科学计数法

$aeb = a * 10^b$

```
const Avogadro = 6.02214129e23  // 阿伏伽德罗常数
const Planck   = 6.62606957e-34 // 普朗克常数
```

fmt.Printf小技巧

C printf厨狂喜

```go
func printFloat() {
	// var f float64 = 114.514
	f := 114.514
	fmt.Println("====基础print====")
	fmt.Println(f)
	fmt.Printf("%f\n", f)
	fmt.Printf("%v\n", f)
	fmt.Printf("%#v\n", f)

	// 控制小数位数
	fmt.Println("====控制小数位数====")
	fmt.Printf("%.2f\n", f) // 两位小数，自动四舍五入
	// 但是四舍五入稍微有点问题（估计是二进制的问题）
	// 可以尝试分别打印114.514, 114.515, 114.516

	// 控制总宽度
	fmt.Println("====控制宽度&对齐====")
	fmt.Printf("%10.3f\n", f) // 右对齐
	fmt.Printf("%-10.3f\n", f) // 左对齐
	fmt.Printf("%010.3f\n", f) // 填充前导0

	// 正负号？
	fmt.Println("====控制正负号====")
	fmt.Printf("%+f\n", f)
	fmt.Printf("%-f\n", -f) // 这个纯在玩了
	fmt.Printf("%+f\n", -f) // 想蒙混过关？该罚！

	// 科学计数法
	fmt.Println("====科学计数法====")
	fmt.Printf("%e\n", f)
	fmt.Printf("%.3e\n", f) // 保留3位小数

	// 自动选择格式
	fmt.Print("====","%","g","自动选择是否使用科学计数法====\n")
	fmt.Printf("%g\n", 123456.789) // 自动选择 %f 或 %e: 123456.789
	fmt.Printf("%g\n", 1.23456789e5)
	fmt.Printf("%g\n", 1.23456789e8) // 输出: 1.23456789e+08
	// 短的就不用，长的就用
}
```

详见bt.go

0 INF NaN?

```
var z float64 // 缺省为0
fmt.Println(z, -z, 1/z, -1/z, z/z) // "0 -0 +Inf -Inf NaN"
```

nan：not a number喵?

```
nan := math.NaN()
fmt.Println(nan == nan, nan < nan, nan > nan) // "false false false"
```

非数到底是个什么数? 😋
C: 用 `isnan()` 来爱抚一下😘

复数

基础类型

complex64 = float32实部 + float32虚部

complex128 = float64实部 + float64虚部

使用手册

写好的Class Complex? 🥳
C学长还得练~

```
var x complex128 = complex(1, 2) // 构造函数？（喜）1+2i
var y complex128 = complex(3, 4) // 3+4i
fmt.Println(x*y) // 真的是复数乘哎 重构*运算符？😘 "(-5+10i)"
fmt.Println(real(x*y)) // real求实部"-5"
fmt.Println(imag(x*y)) // imag求虚部 "10"
```

i的含金量?

```
x := 1 + 2i
y := 3 + 4i
// 构造函数？不熟😊
fmt.Println(1i * 1i) // "(-1+0i)", i^2 = -1
```

more funcs about complex can be found in `math\cmplx` 库
e.g. $\sqrt{-1}$

```
fmt.Println(cmplx.Sqrt(-1)) // "(0+1i)"
```

布尔型

true or false💏

短路行为（省事行为）

如果运算符左边值已经可以确定整个布尔表达式的值，那么运算符右边的值将不再被求值

```
s != "" && s[0] == 'x'
```

如果s已经判断为空了，就不会再判断 `s[0]` 是否为 `'x'` 了（这样就不会越界访问了）

1 ≠ true 0 ≠ false?

不同类型需要**显示类型转换**!

C学长，你拿 `int flag` 当 `bool` 的好日子一去不复返了!

```
var a int = 1
        if a {
                print(1)
        } // × non-boolean condition in if statement
```

显式 `int to bool` ，封装成函数

```
func itob(i int) bool { return i != 0 }
```

同理 bool to int ， 封装成函数

```
// btoi returns 1 if b is true and 0 if false.
func btoi(b bool) int {
    if b {
        return 1
    }
    return 0
}
```

字符串

string在哪里向来都是博大精深的
鉴于不(lan)太(ai)专(wan)业(qi)，就先贴一个blog
Go 字符串

"" vs ``

" "

```
str := "hello, world!\n"
fmt.Println(str)
```

和C++的差不多

``

```
str := `hello,
world
!\n
`
fmt.Println(str)
```

支持换行，用于定义多行字符串，但转义字符不会转义
\n 会直接输出 \n ， 不会变成换行

那一天的转义不再转义（悲）

遍历

```
s := "hello, wolrd!"
for i, r := range s {
        fmt.Printf("%d\t%q\t%d\n", i, r, r) // 下标，UTF-8字符，UTF-8字符对应的值
}

/*
0       'h'     104
1       'e'     101
2       'l'     108
3       'l'     108
4       'o'     111
5       ','     44
6       ' '     32
7       'w'     119
8       'o'     111
9       'l'     108
10      'r'     114
11      'd'     100
12      '!'     33
*/
```

转换

详见 strconv 函数

strconv 在哪？在那遥远的地方~~😋

常量

const大法

```
const N = 500005
// ()批量声明
const (
    e  = 2.71828182845904523536028747135266249775724709369995957496696763
    pi = 3.14159265358979323846264338327950288419716939937510582097494459
)
```

批量声明省事法

如果是批量声明的常量，除了第一个外其它的常量右边的初始化表达式都可以省略

世袭制！🤓👆

```
const (
    a = 1 // 第一个a = 1
    b // 第二个省略，照抄a，b = 1
    c = 2
    d // 同理，照抄c，d = 2
)

fmt.Println(a, b, c, d) // "1 1 2 2"
```

iota常量生成器（012345自动机）

```
type Weekday int

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

在一个const声明语句中，在第一个声明的常量所在的行，iota将会被置为0，然后在每一个有常量声明的行加一

又根据《批量声明省事法》，iota直接世袭，1234567
结果：Sunday - Saturday : 0 1 2 3 4 5 6

番外篇《C学长，看看你的iota》

```
vector<int>number(10);

iota(number.begin(), number.end(), 66);
for (int i = 0; i < 10; i++) {
        cout << number[i] << " ";       //输出：66 67 68 69 70 71 72 73 74 75
}

// iota(起始，终止，基数);
```