GOLANG基础数据类型

讲师：KK

课程内容

- 布尔类型
- 数值类型
- Golang 字符串类型
- 枚举类型
- 指针类型

布尔类型

概念

- 布尔类型用于表示真假，类型名为bool，只有两个值true和false，占用一个字节宽度，零值为false

```
var (
    isBody bool = true
    isGirl bool = false
)
```

布尔类型

常用操作

- 逻辑运算：

  a）与(&&)

  只有左、右表达式结果都为true，运算结果为true

```
// 与
fmt.Println(isBody && isBody)
fmt.Println(isBody && isGirl)
fmt.Println(isGirl && isBody)
fmt.Println(isGirl && isGirl)
```

布尔类型

常用操作

- 逻辑运算：

  b）或(||)

  只要左、右表达式有一个为true，运算结果为true

```
// 或
fmt.Println(isBody || isBody)
fmt.Println(isBody || isGirl)
fmt.Println(isGirl || isBody)
fmt.Println(isGirl || isGirl)
```

布尔类型

常用操作

- 逻辑运算：

  c)　非(!)

  右表达式为true，运算结果为false；右表达式为false，运算结果为true

```
// 非
fmt.Println(!isBody)
fmt.Println(!isGirl)
```

布尔类型

常用操作

- 2) 关系运算
  - a) 等于(==)
  - b) 不等于(!=)

```
// 关系运算
fmt.Println(isBody == isGirl)
fmt.Println(isBody != isGirl)
```

使用fmt.Printf进行格式化参数输出，占位符:

- %t

```
// 打印
fmt.Println(isBody, isGirl)
fmt.Printf("isBody=%t, isGirl=%t\n", isBody, isGirl)
```

数值类型

整型：

Go语言提供了5种有符号、5种无符号、1种指针、1种单字节、1种单个unicode字符（unicode码点），共13种整数类型,零值均为0.

int, uint, rune, int8, int16, int32, int64, uint8, uint16, uint32, uint64, byte, uintptr

数值类型

字面量：

- 十进制表示法：以10为基数，采用0-9十个数字，逢10进位，例如：10
- 八进制表示法：以8为基数，采用0-7八个数字，逢8进位，使用0开头表示为八进制表示，例如：010
- 十六进制表示法：以16为基数，采用0-9十个数字和A-F六个字母，逢16进位，使用0X开头表示为十六进制 ，例如：0X10

```
// 分别打印十进制，八进制，十六进制标识发的字面量
fmt.Println(10, 010, 0X10)
```

数值类型

字面量-常用操作

```
var n1, n2, n3 int = 1, 8, -2
var n4 uint = 2
```

a）算术运算符：+、-、*、/、%、++、--

注意：针对/除数不能为0，且结果依然　　为整数

```
// 算术运算
fmt.Println(n1 + n2) // 9
fmt.Println(n1 - n2) // -7
fmt.Println(n1 * n2) // 8
fmt.Println(n1 / n2) // 0
fmt.Println(n1 % n2) // 1

n1++
n2--
fmt.Println(n1, n2) // 2 7
```

数值类型

字面量-常用操作

```
var n1, n2, n3 int = 1, 8, -2
var n4 uint = 2
```

b)　　关系运算符：>、>=、<、<=、==、!=

```
// 关系运算
fmt.Println(n1 > n2)
fmt.Println(n1 >= n2)
fmt.Println(n1 < n2)
fmt.Println(n1 <= n2)
fmt.Println(n1 == n2)
fmt.Println(n1 != n2)
```

数值类型

字面量-常用操作

```
var n1, n2, n3 int = 1, 8, -2
var n4 uint = 2
```

c)　　位运算符：&、|、^、<<、>>、&^

对于负整数在计算机中使用补码进行表示，对应正整数二进制表示取反+1

针对左、右移的右操作数必须为无符号整型

```
// 位运算
// 2 => 0010
// 7 => 0111
// -2 => 1110
fmt.Println(n1 & n2)  // 0010
fmt.Println(n1 | n2)  // 0111
fmt.Println(n1 ^ n2)  // 0101
fmt.Println(n1 &^ n2) // 0000
fmt.Println(n2 << n4) // 0001 1100
fmt.Println(n2 >> n4) // 0001
fmt.Println(n3 << n4) // 1111 1000
fmt.Println(n3 >> n4) // 1111 1111
```

数值类型

字面量-常用操作

```
var n1, n2, n3 int = 1, 8, -2
var n4 uint = 2
```

d)　　赋值运算符：=、+=、-=、*=、/=、%=、&=、|=、^=、<<=、>>=

```
// 赋值运算
n1 += n2
n2 -= n1
n4 <<= n4
fmt.Println(n1, n2, n4)
```

数值类型

字面量-常用操作

```
var n1, n2, n3 int = 1, 8, -2
var n4 uint = 2
```

e)　　类型转换：

Go不会对自动对数据类型转换，因此左、右操作数类型必须一致或某个字面量，可通过类型名(数据)的语法将数据转换为对应类型。需要注意值截断和值溢出问题

```
//类型转换
fmt.Printf("%T %T %T %T %T %T\n", n1, uint(n1), n4, int(n4), uint8(n4), int64(n4))
```

数值类型

格式化输出

- 使用fmt.Printf进行格式化参数输出，占位符:
- %b：二进制
- %c：字符
- %d：十进制
  - %+d表示对正整数带+符号
  - %nd表示最小占位n个宽度且右对齐
  - %-nd表示最小占位n个宽度且左对齐
  - %0nd表示最小占位n个宽度且右对齐，空字符使用0填充
- %o：八进制，%#o带0的前缀
- %x、%X：十六进制,%#x(%#X)带0x(0X)的前缀
- %U: Unicode码点，%#U带字符的Unicode码点
- %q：带单引号的字符

数值类型

格式化输出

```
//打印
var (
    n5 int  = 10
    n6 int  = -10
    c0 byte = 'a'
    u0 rune = '牛'
)

fmt.Printf("%d %b %o %x %X %#o %#x %#X\n", n5, n5, n5, n5, n5, n5, n5, n5)
fmt.Printf("%d|%+d|%10d|%-10d|%010d|%+-10d|%+010d|\n", n5, n5, n5, n5, n5, n5, n5)
fmt.Printf("%d|%+d|%10d|%-10d|%010d|%+-10d|%+010d|\n", n6, n6, n6, n6, n6, n6, n6)
fmt.Printf("%c %c %q %q\n", c0, u0, c0, u0)
fmt.Printf("%U %#U\n", u0, u0)
```

数值类型

浮点型：

◆浮点数用于表示带小数的数字，Go提供float32和float64两种浮点类型

字面量：

- 十进制表示法：3.1415926
- 科学记数法：1e-5

```
var (
    f1 float32 = 3.1415926
    f2 float32 = 3E-3
    f3 float64 = 3.1E10
)
```

数值类型

浮点型-常用操作

```
var (
    f1 float32 = 3.1415926
    f2 float32 = 3E-3
    f3 float64 = 3.1E10
)
```

a) 算术运算符：+、-、*、/、++、--

注意：针对/除数不能为0

```
// 算术运算
fmt.Println(f1 + f2)
fmt.Println(f1 - f2)
fmt.Println(f1 * f2)
fmt.Println(f1 / f2)
f1++
f2--
fmt.Println(f1, f2)
```

数值类型

浮点型-常用操作

```
var (
    f1 float32 = 3.1415926
    f2 float32 = 3E-3
    f3 float64 = 3.1E10
)
```

b) 关系运算符：>、>=、<、<=

浮点型不能进行==或!=比较，可选择使用两个浮点数的差在一定区间内则认为相等

```
// 关系运算
fmt.Println(f1 > f2)
fmt.Println(f1 >= f2)
fmt.Println(f1 < f2)
fmt.Println(f1 <= f2)
```

数值类型

浮点型-常用操作

```
var (
    f1 float32 = 3.1415926
    f2 float32 = 3E-3
    f3 float64 = 3.1E10
)
```

c) 赋值运算符：=、 +=、 -=、 *=、 /=

```
// 赋值运算
f1 += f1
fmt.Println(f1)
```

数值类型

浮点型-常用操作

```
var (
    f1 float32 = 3.1415926
    f2 float32 = 3E-3
    f3 float64 = 3.1E10
)
```

d) 类型转换：

Go不会对自动对数据类型转换，因此左、右操作数类型必须一致或某个字面量，可通过类型名(数据)的语法将数据转换为对应类型。需要注意值截断和值溢出问题

```
// 类型转换
fmt.Printf("%T %T\n", f1, float64(f1))
```

数值类型

浮点型-格式化输出

使用fmt.Printf进行格式化参数输出，占位符:

- %f、%F：十进制表示法
  - ✓ %n.mf表示最小占n个宽度并且保留m位小数
- %e、%E：科学记数法表示
- %g、%G：自动选择最紧凑的表示方法%e(%E)或%f(%F)

```
fmt.Printf("%f %f %F %F\n", f1, f3, f1, f3)
fmt.Printf("%e %e %E %E\n", f1, f3, f1, f3)
fmt.Printf("%g %g %G %G\n", f1, f3, f1, f3)
fmt.Printf("%5.2f %5.2f\n", f1, f3)
```

字符串类型

字面量：

可解析字符串：通过双引号(")来创建，不能包含多行，支持特殊字符转义序列
原生字符串：通过反引号(`)来创建，可包含多行，不支持特殊字符转义序列

```
var (
    f1 float32 = 3.1415926
    f2 float32 = 3E-3
    f3 float64 = 3.1E10
)
```

字符串类型

特殊字符：

- \\：反斜线
- \'：单引号
- \"：双引号
- \a：响铃
- \b：退格
- \f：换页
- \n：换行
- \r：回车
- \t：制表符
- \v：垂直制表符
- \ooo：3个8位数字给定的八进制码点的Unicode字符（不能超过\377）
- \uhhhh：4个16位数字给定的十六进制码点的Unicode字符
- \Uhhhhhhhh：8个32位数字给定的十六进制码点的Unicode字符
- \xhh：2个8位数字给定的十六进制码点的Unicode字符

字符串类型

常用操作：

- 字符串连接：+
- 关系运算符：>、>=、<、<=、==、!=
- 赋值运算符：+=
- 索引：s[index]，针对只包含ascii字符的字符串
- 切片：s[start:end] ，针对只包含ascii字符的字符串

```
fmt.Println(s4 + "ABCDEFGHIJKLMNOPQRSTUVWXYZ")

s5 += "--来自Silence"
fmt.Println(s5)

fmt.Println(s1 == s2)
fmt.Println(s1 != s3)
fmt.Println("abc" > "abe")
fmt.Println("abc" < "abe")
fmt.Println("abc" >= "abe")
fmt.Println("abc" <= "abe")

fmt.Println(s4[0])
fmt.Println(s4[:], s4[3:], s4[:8], s4[3:8])
```

枚举类型

常使用iota生成器用于初始化一系列相同规则的常量，批量声明常量的第一个常量使用iota进行赋值，此时iota被重置为0，其他常量省略类型和赋值，在每初始化一个常量则加1。

enum.go

```go
package main

import "fmt"

const (
    c1 int = iota
    c2
    c3 int = iota
    c4
)

func main() {
    const (
        c5 = iota
        c6
        c7 = 7
        c8 = iota
        c9
    )

    fmt.Println(c1, c2, c3, c4, c5, c6, c7, c8, c9)
}

```

指针类型

声明

指针声明需要指定存储地址中对应数据的类型，并使用*作为类型前缀。指针变量声明后会被初始化为nil，表示空指针

```
7     var pointer01 *int
8     var pointer02 *float64
9     var pointer03 *string
10
11    fmt.Printf("%T, %T, %T\n", pointer01, pointer02, pointer03)
12    fmt.Println(pointer01, pointer02, pointer03)
13    fmt.Printf("%t, %t, %t\n", pointer01 == nil, pointer02 == nil, pointer03 == nil)
```

指针类型

初始化

a）使用&运算符+变量初始化：&运算获取变量的存储位置来初始化指针变量

b）使用new函数初始化：new函数根据数据类型申请内存空间并使用零值填充，并返回申请空间地址

```
15    var (
16        age int = 30
17        heigh float64 = 1.68
18        motto string = "少年经不得顺境，中年经不得闲境，晚年经不得逆境"
19    )
20
21    //指针变量初始化
22    pointer01, pointer02, pointer03 = &age, &heigh, &motto
23    pointer04, pointer05, pointer06 := &age, &heigh, &motto
24    pointer07, pointer08, pointer09 := new(int), new(float64), new(string)
25
26    //打印变量地址
27    fmt.Println(&age, &heigh, &motto)
28
29    //打印指针变量
30    fmt.Println(pointer01, pointer02, pointer03)
31    fmt.Println(pointer04, pointer05, pointer06)
32    fmt.Println(pointer07, pointer08, pointer09)
33
34    fmt.Println(age, heigh, motto) //打印变量值
35    fmt.Println(*pointer01, *pointer02, *pointer03) //通过指针变量访问位置存储的值
36    fmt.Printf("%v, %v, %q\n", *pointer07, *pointer08, *pointer09)
```

指针类型

操作

可通过*运算符+指针变量名来访问和修改对应存储位置的值

```
32      fmt.Println(age, heigh, motto) //打印变量值
33      fmt.Println(*pointer01, *pointer02, *pointer03) //通过指针变量访问位置存储的值
34
35      //通过指针变量访问修改存储的值
36      *pointer01 += 1
37      *pointer02 = 1.70
38      *pointer03 += "--曾国藩"
39
40      fmt.Println(*pointer01, *pointer02, *pointer03)  //通过指针变量访问位置存储的值
41      fmt.Println(age, heigh, motto)  //打印变量值
42
43      fmt.Println(&age, &heigh, &motto) //打印变量地址
44      fmt.Println(pointer01, pointer02, pointer03) //打印指针变量
```

```
46      //与赋值新变量对比
47      age2, heigh2, motto2 := age, heigh, motto
48
49      //修改新变量值
50      age2 += 1
51      heigh2 = 1.72
52      motto2 += "家书"
53
54      //打印变量
55      fmt.Println(age, heigh, motto)
56      fmt.Println(age2, heigh2, motto2)
57
58      //打印变量地址
59      fmt.Println(&age, &heigh, &motto)
60      fmt.Println(&age2, &heigh2, &motto2)
```

指针类型

指针的指针

用来存储指针变量地址的变量叫做指针的指针

```
63      //定义声明指针的指针
64      var ppointer01 **int
65      var ppointer02 **float64 = &pointer02
66      ppointer03 := &pointer03
67
68      fmt.Println(ppointer01, ppointer02, ppointer03)
69
70      ppointer01 = &pointer01
71      fmt.Println(ppointer01, ppointer02, ppointer03)
72
73      //通过指针的指针访问变量地址和变量值
74      fmt.Println(*ppointer01, *ppointer02, *ppointer03)
75      fmt.Println(**ppointer01, **ppointer02, **ppointer03)
76
77      //通过指针的指针修改和变量值
78      **ppointer01 += 1
79      **ppointer02 = 1.72
80      **ppointer03 += "家属"
81
82      fmt.Println(**ppointer01, **ppointer02, **ppointer03)
83      fmt.Println(*pointer01, *pointer02, *pointer03)
84      fmt.Println(age, heigh, motto)
85
```

祝大家学业有成

谢　谢