# Go语言基础

## 常量

常量使用关键字const声明，下面有几个例子：

const limit = 512

const top uint16 = 1421

const Pi float64 = 3.1415926

const x,y int = 1,3 //多重赋值

当需要设置多个常量的时候，不必重复使用const关键字，可以使用以下语法:

const (

Cyan = 0

Black = 1

White = 2

)

Go 语言还预定义了这些常量：true、false、iota。 iota是一个可以被编译器修改的常量，在const关键字出现时被重置为 0，在下一个 const 出现之前，每出现一次 iota,其所代表的数字自动加 1。下面通过一个例子讲解 iota 的用法：

const (

a = iota //a == 0

b = iota //b ==1

c = iota //c == 2

)

const d = iota //d==0,因为const的出现，iota被重置为0

## 变量

Go 语言引入了关键字var对变量进行声明，也可以使用:=来对变量直接进行初始化，Go 编译器会自动推导出该变量的类型，这大大的方便了开发者的工作。但是需要注意的是:=左侧的变量不能是已经被声明过的，否则会导致编译器错误。以下是 Go 声明和初始化变量的各种方法：

var a int

var b string

var c float64

var d [5] int //数组

var e [] int //数组切片

var f \* int //正确

var v1 int = 5 //正确

var v2 = 5 //正确，编译器自动推导出V2类型

v3 := 5 //正确，编译器自动推导出V3的类型

Go 语言提供了大多数语言不支持的多重赋值，这使得变量的交换变得十分简单。下面通过一个例子来了解 Go 语言的多重赋值：

i := 2

j := 3

i, j = j, i //交换i和j的值，此时i == 3，j == 2

## 数据类型

### 3.1整型

Go 语言提供了 11 种整型，如下列表所示

| **类型** | **说明** |
| --- | --- |
| byte | 等同于 uint8 |
| int | 依赖于不同平台下的实现，可以是 int32 或者 int64 |
| int8 | [-128, 127] |
| int16 | [-32768, 32767] |
| int32 | [-2147483648, 2147483647] |
| int64 | [-9223372036854775808, 9223372036854775807] |
| rune | 等同于 int32 |
| uint | 依赖于不同平台下的实现，可以是 uint32 或者 uint64 |
| uint8 | [0, 255] |
| uint16 | [0, 65535] |
| uint32 | [0, 4294967295] |
| uint64 | [0, 18446744073709551615] |
| uintptr | 一个可以恰好容纳指针值的无符号整型（对 32 位平台是 uint32, 对 64 位平台是 uint |

在C语言中我们可以通过sizeof操作符查看类型的字节长度，在 Go 语言中可以通过unsafe.Sizeof函数进行，使用vim创建源文件type\_length.go，输入以下代码：

package main

import (

"fmt"

"unsafe"

)

func main() {

a := 12

fmt.Println("length of a: ", unsafe.Sizeof(a))

var b int = 12

fmt.Println("length of b(int): ", unsafe.Sizeof(b))

var c int8 = 12

fmt.Println("length of c(int8): ", unsafe.Sizeof(c))

var d int16 = 12

fmt.Println("length of d(int16): ", unsafe.Sizeof(d))

var e int32 = 12

fmt.Println("length of e(int32): ", unsafe.Sizeof(e))

var f int64 = 12

fmt.Println("length of f(int64): ", unsafe.Sizeof(f))

}以上代码中，首先声明了目前的源文件属于main包，然后导入了fmt和unsafe包，fmt包用于格式化字符串，unsafe包含了用于获取 Go 语言类型信息的方法。然后在 main 函数中，我们分别声明了几种类型的整型变量，并通过unsafe.Sizeof方法获取该类型的字节长度。最后我们通过以下方法运行type\_length.go，同时打印出了输出：

$ go run type\_length.go

length of a: 8

length of b(int): 8

length of c(int8): 1

length of d(int16): 2

length of e(int32): 4

length of f(int64): 8

×

实验剩余时间 44 分钟， 剩余时间小于10分钟才可以延时

取消

**Vim使用**

vim 编辑器可以简单分为“命令控制模式” 和 “文本编辑模式”两种，默认启动编辑器时，工作在“命令控制模式”下。

（1）命令控制模式：该模式下主要控制vim实现，如启动“编辑”、“查找”、“定位”、“快速删除”、“跳转到指定行”等等一系列功能。

**（2）文本编辑模式**：该模式下可以对文本进行数据写入/修改/追加等操作

默认打开vim编辑器时，是处于“命令控制模式”下，此时的文本是无法编辑的，要实现文本编辑，就需要切换到文本编辑模式下。

此时输入“i”（小写）即可进入“文本编辑模式”，切换到此模式后，我们就可以对该文档进行数据写入等编辑了。在终端输入：vim test.c （此时目录下没有这个文件）回车打开。在“文本编辑”模式下完成数据的写入或修改后，接下来就是如何“保存”的问题了，此时需要**切换回到“命令控制”模式**下执行文件保存。

按下键盘的“Esc”键退出“文本编辑”模式，回到“命令控制”模式，接着输入“：wq”命令，回车执行就可以保存文件并退出编辑器了。这样一个简单的文件创建/编辑/保存的流程就结束了，此时在当前目录下会自动生成一个“test.c”的文件

## 2.浮点型

Go 语言提供了两种浮点类型和两种复数类型, 具体如下：

| **类型** | **说明** |
| --- | --- |
| float32 | ±3.402 823 466 385 288 598 117 041 834 845 169 254 40x1038 计算精度大概是小数点后 7 个十进制数 |
| float64 | ±1.797 693 134 862 315 708 145 274 237 317 043 567 981x1038 计算精度大概是小数点后 15 个十进制数 |
| complex32 | 复数，实部和虚部都是 float32 |
| complex64 | 复数，实部和虚部都是 float64 |

## 3.布尔类型

Go 语言提供了内置的布尔值true和false。Go 语言支持标准的逻辑和比较操作，这些操作的结果都是布尔值。值得注意的地方是可以通过!b的方式反转变量b的真假。需要注意的是布尔类型不能接受其他类型的赋值，不支持自动或强制的类型转换。实例代码如下：

var a bool

a = true

b := (2 == 3) //b也会被推导为bool类型

//错误示范

var b bool

b = 1 //编译错误

b = bool(1) //编译错误

## 4.字符串

Go 语言中的字符串是 [UTF-8](http://zh.wikipedia.org/wiki/UTF-8) 字符的一个序列（当字符为 ASCII 码时则占用 1 个字节，其它字符根据需要占用 2-4 个字节）。Go 语言中字符串的可以使用双引号( " )或者反引号( ` )来创建。双引号用来创建可解析的字符串字面量，所谓可解析的是指字符串中的一些符号可以被格式化为其他内容，如\n在在输出时候会被格式化成换行符， 如果需要按照原始字符输出必须进行转义。而反引号创建的字符串原始是什么样，那输出还是什么，不需要进行任何转义。以下是几个例子:

t1 := "\"hello\"" //内容： "hello"

t2 := `"hello"` //内容：和t1一致

t3 := "\u6B22\u8FCE" //内容：欢迎

Go 语言中的部分转义字符如下表所示：

| **转义字符** | **含义** |
| --- | --- |
| \\ | 表示反斜线 |
| ' | 单引号 |
| " | 双引号 |
| \n | 换行符 |
| \uhhhh | 4 个 16 进制数字给定的 Unicode 字符 |
| 语法 | 描述 |
| s += t | 将字符串 t 追加到 s 末尾 |
| s + t | 将字符串 s 和 t 级联 |
| s[n] | 从字符串 s 中索引位置为 n 处的原始字节 |
| s[n:m] | 从位置 n 到位置 m-1 处取得的字符（字节）串 |
| s[n:] | 从位置 n 到位置 len(s)-1 处取得的字符（字节）串 |
| s[:m] | 从位置 0 到位置 m-1 处取得的字符（字节）串 |
| len(s) | 字符串 s 中的字节数 |
| len([]rune(s)) | 字符串 s 中字符的个数，可以使用更快的方法 utf8.RuneCountInString() |
| [ ]rune(s) | 将字符串 s 转换为一个 unicode 值组成的串 |
| string(chars) | chars 类型是[]rune 或者[]int32, 将之转换为字符串 |
| [ ]byte(s) | 无副本的将字符串 s 转换为一个原始的字节的切片数组，不保证转换的字节是合法的 UTF-8 编码字节 |

## 5.格式化字符串

| **格式化指令** | **含义** |
| --- | --- |
| %% | %字面量 |
| %b | 一个二进制整数，将一个整数格式化为二进制的表达方式 |
| %c | 一个 Unicode 的字符 |
| %d | 十进制数值 |
| %o | 八进制数值 |
| %x | 小写的十六进制数值 |
| %X | 大写的十六进制数值 |
| %U | 一个 Unicode 表示法表示的整形码值，默认是 4 个数字字符 |
| %s | 输出以原生的 UTF-8 字节表示的字符，如果 console 不支持 UTF-8 编码，则会输出乱码 |
| %t | 以 true 或者 false 的方式输出布尔值 |
| %v | 使用默认格式输出值，或者使用类型的 String()方法输出的自定义值，如果该方法存在的话 |
| %T | 输出值的类型 |

常用的格式化指令修饰符如下：

* 空白 如果输出的数字为负，则在其前面加上一个减号"-"。如果输出的是整数，则在前面加一个空格。使用 %x 或者 %X 格式化指令输出时，会在结果之间添加一个空格。例如 fmt.Printf("% X", "实")输出 E5 AE 9E
* #
  + %#o 输出以 0 开始的八进制数据
  + %#x 输出以 0x 开始的十六进制数据
* + 让格式化指令在数值前面输出+号或者-号，为字符串输出 ASCII 字符（非 ASCII 字符会被转义），为结构体输出其字段名
* - 让格式化指令将值向左对齐（默认值为像右对齐）
* 0 让格式指令以数字 0 而非空白进行填充

## 6.数组

多维数组可以简单地使用自身为数组的元素来创建。数组的元素使用操作符号[ ]来索引，索引从 0 开始，到 len(array)-1 结束。数组使用以下语法创建：

* \*[length]Type\*
* \*[N]Type{value1, value2, ..., valueN}\*
* \*[...]Type{value1, value2, ..., valueN}\*

如果使用了...（省略符）操作符，Go 语言会为我们自动计算数组的长度。在任何情况下，一个数组的长度都是固定的并且不可修改。数组的长度可以使用len()函数获得。由于数组的长度是固定的，因此数组的长度和容量都是一样的，因此对于数组而言cap()和len()函数返回值都是一样的。

## 7.切片

Go 语言的切片比数组更加灵活，强大而且方便。数组是按值传递的（即是传递的副本），而切片是引用类型，传递切片的成本非常小，而且是不定长的。而且数组是定长的，而切片可以调整长度。创建切片的语法如下：

* *make([ ]Type, length, capacity)*
* *make([ ]Type, length)*
* \*[ ]Type{}\*
* \*[ ]Type{value1, value2, ..., valueN}\*

内置函数make()用于创建切片、映射和通道。一个切片的容量即为隐藏数组的长度，而其长度则为不超过该容量的任意值。另外可以通过内置的函数append()来增加切片的容量。**（ vim slice\_arry.go）**

## 8.包

Go 语言组织代码的方式是包，包是各种类型和函数的集合。在包中，如果标示符(类型名称，函数名称，方法名称)的首字母是大写，那这些标示符是可以被导出的，也就是说可以在包以外直接使用。前面我们也提到了$GOPATH环境变量（指向一个或多个目录），以及其子目录src目录的，当我们使用import关键字导入包的时候，Go 语言会在$GOPATH，GOROOT目录中搜索包。