go语言基本语法和数据类型

[toc]

🤹go语言官方编程指南：https://golang.org/#

> go语言的官方文档学习笔记很全，推荐去官网学习

🤹我的学习笔记：github: https://github.com/3293172751/golang-rearn

区块链技术（也称之为分布式账本技术），是一种互联网数据库技术，其特点是去中心化，公开透明，让每一个人均可参与的数据库记录

> ❤️💕💕关于区块链技术，可以关注我，共同学习更多的区块链技术。博客 http://nsddd.top

---

1. 🐧应用场景

1. 区块链

区块链技术（也称之为分布式账本技术），是一种互联网数据库技术，其特点是去中心化，公开透明，让每一个人均可参与的数据库记录

> ❤️💕💕关于区块链技术，可以关注我或者博客http://nsddd.top

2. 后端服务器开发

如美团后台流量支撑程序

主要支撑后台流量（排序，推荐，搜索等），提供负载均衡。

3. 云计算和云服务

内容分发网络

2. 🗃️特点

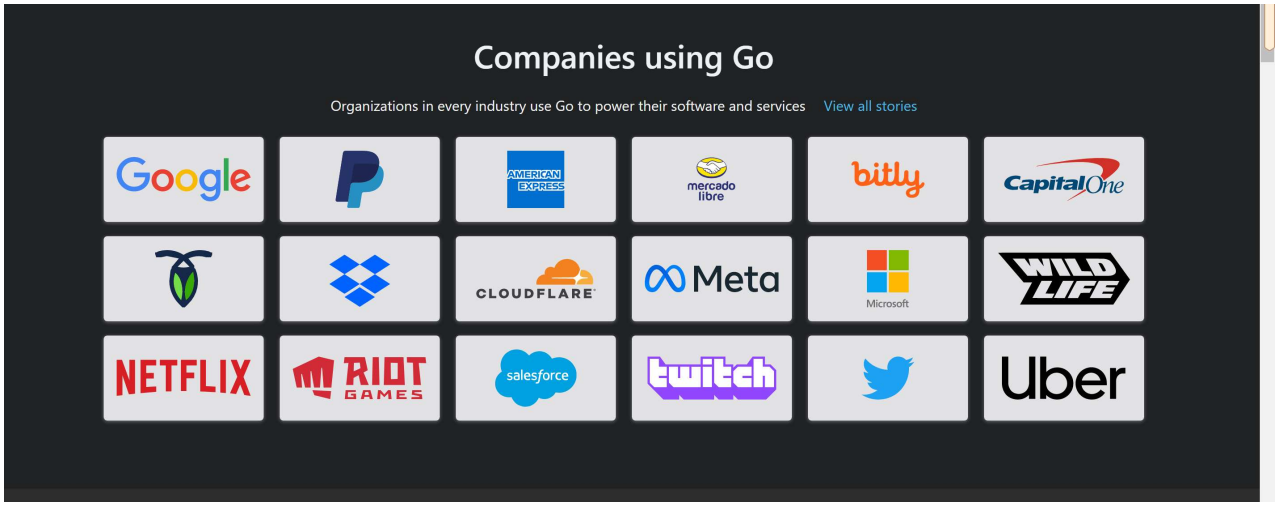
go语言保证了既能达到静态编译语言的安全与性能，又达到了动态语言开发维护的高效率，使用一个表达式来形容go语言： `go = C +python`

由此说明go语言既有C静态语言程序运行速度，有能达到python动态语言的快速开发。

- go语言包概念：**go 语言的每一个文件都是属于一个包，不可以独立存在。**
- go语言的垃圾回收机制：**内存自动回收，不需要开发人员管理（避免出现内存泄漏）**

> **go语言的天然并发**
>
> （重要❤️）
>
> 1. 从语言层面支持并发，实现简单。
> 2. Goroutine,轻量级线程，**可以实现大并发处理，高效利用多核。**
> 3. 基于CPS并发模型实现。

- Go语言管道通信机制
- 函数可以返回多个值

> Go语言的应用场景也是非常广的：
>
> 
>

3. ⚒️使用工具

我使用的工具有两种,分别对应的是window系统和Linux系统

其他工具:

- sublime text
- Emacs(特别强大,可称之为集成开发环境)

> 还有很多不列举出来了,根据自己喜欢的选择

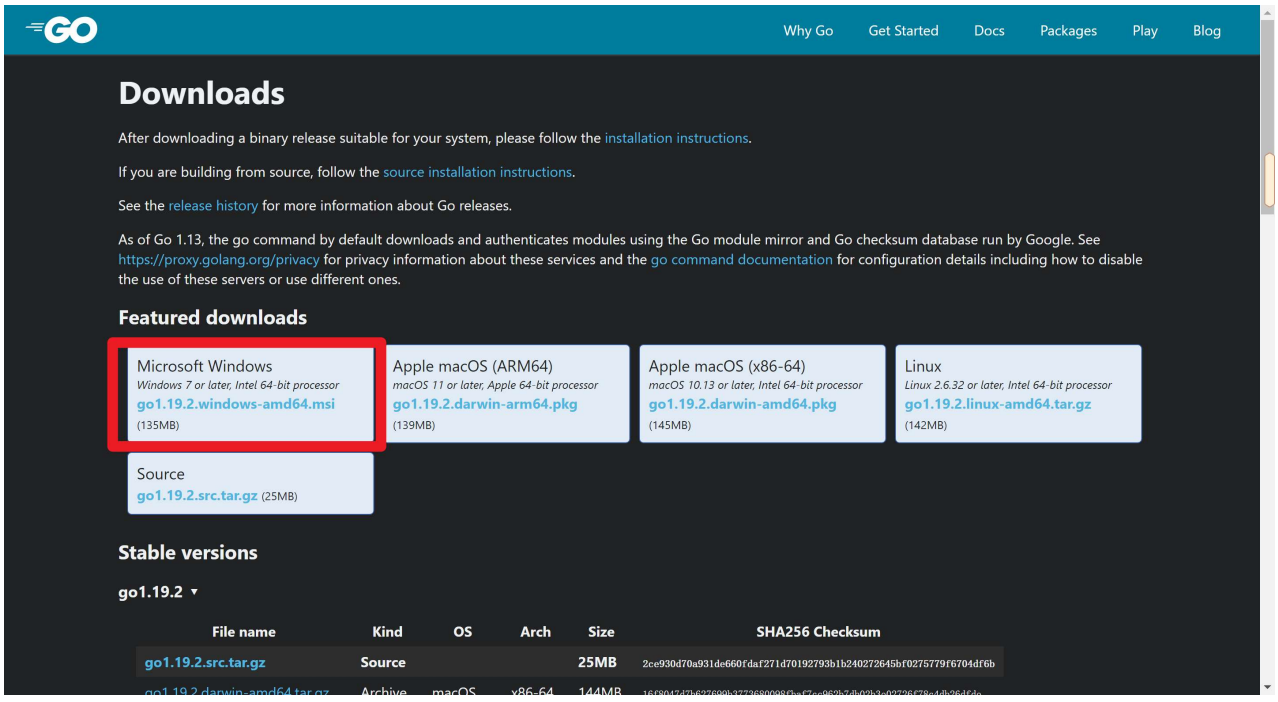
4. 🔨 环境搭建

ubuntu centos

```sh
sudo apt install golang
```

4.1 Windows下搭建go 开发环境 - 暗转和配置SDK

> 在win环境下安装Go语言（现阶段用 `1.18~1.19` 版本都可以）：
>
> - [x] 官方地址



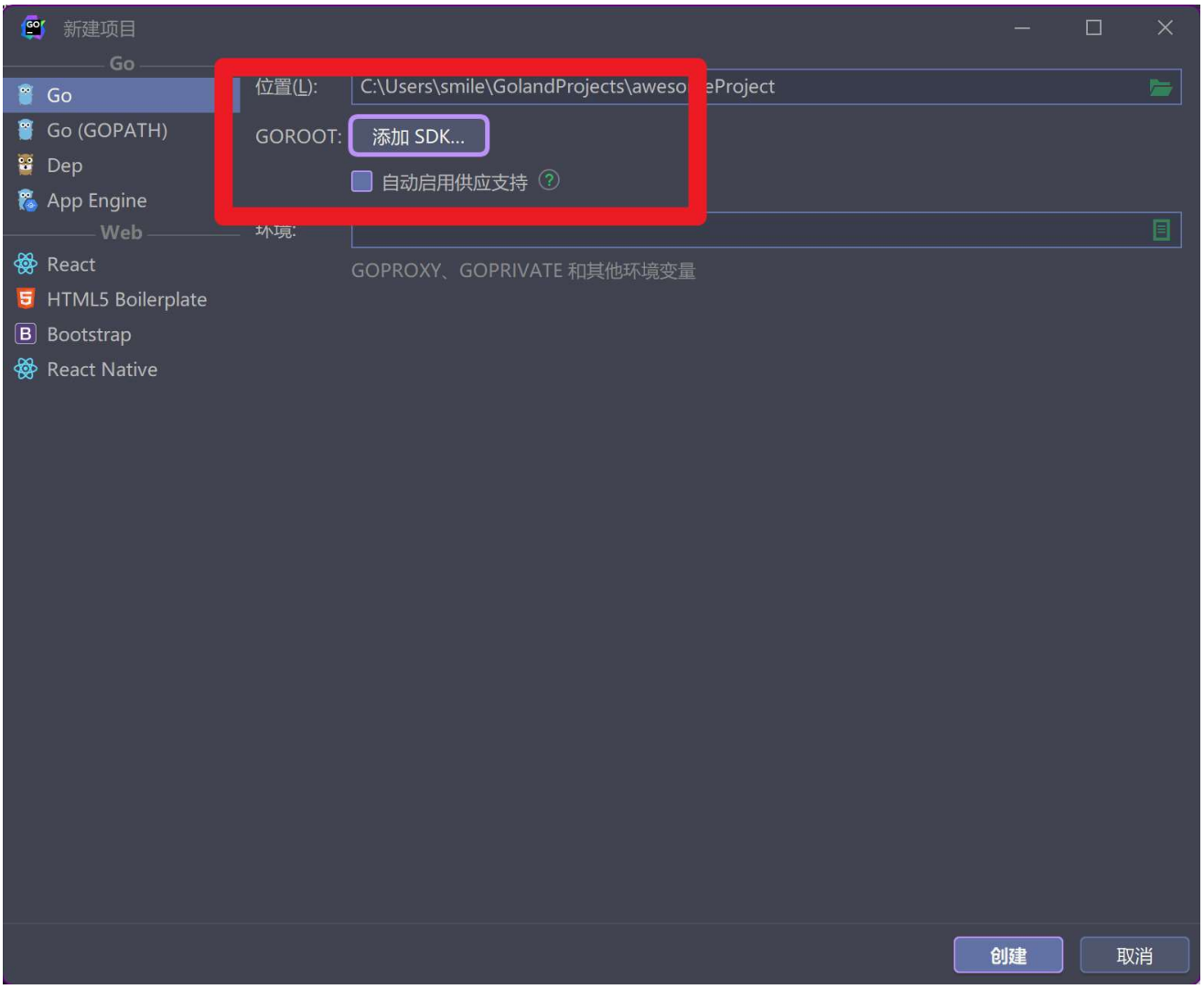
> ⚠️ 可能出现没办法访问（网络防火墙），我将安装包上传到迅雷。

```
链接：https://pan.xunlei.com/s/VNDaGl3svnHgctoYVReZQ9T_A1?pwd=intg#
提取码：intg
复制这段内容后打开手机迅雷App，查看更方便
```

SDK:software development kit 软件开发工具包

提供给开发者的工具,编译成可执行文件,包括___工具和API___

- 官网(需要外网): `golang.org.`
- msi:是exe文件安装,推荐使用zip类型安装,解压既可以使用



4.2 Linux下搭建go环境

> Ubuntu的安装地址（centos把 `apt` 换成 `yum` 即可）

```sh
apt install golang
```

- 使用 `uanme -a` 查看当前Linux的版本号和基本信息.
- 使用 `ftp` 将文件传输到Linux中,或者使用apt(ubuntu) wget下载
- 使用 `tar -zxvf` 解压

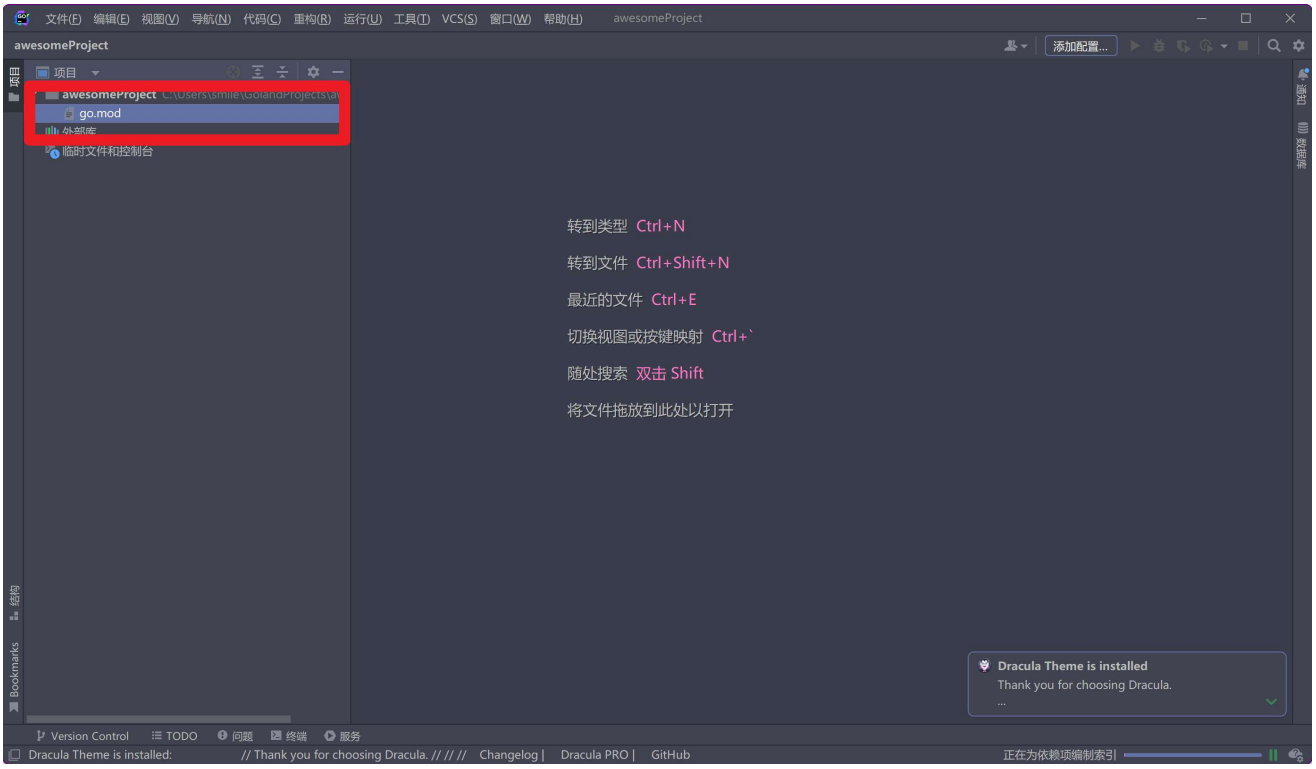
Linux下go 的配置:

1. 在 `/etc/profile` 文件中添加三种语句(`root权限`)

```sh
export GOROOT =/opt/go

export PATH = $GOROOT/bin:$PATH

export GOPATH = $HOME/goprojects/
```

2. 注销用户再使用，这时才能生效

ide：golang



> 接下来可以愉快的编程了

推荐一个很好用的主题，个人也比较喜欢

5. 📘 Go 语言的快速入门

Go 语言结构

在我们开始学习 Go 编程语言的基础构建模块前，让我们先来了解 Go 语言最简单程序的结构。

5.1Go Hello World 实例

Go 语言的基础组成有以下几个部分：

- 包声明
- 引入包
- 函数
- 变量
- 语句 & 表达式
- 注释

```go
package main

import "fmt"  //表示引入了一个包，类似于python 引入改包后就可以使用包内函数

func main() {
            /* {不能单独放在一行，func是一个关键词，后面是一个函数，main主函数，即
            /* 这是我的第一个简单的程序,这是注释 */
  fmt.Println("Hello, World!")
}           //fmt 表示调用Printlt输出hello word.
```

5.2 程序的各个部分：

> 1. 第一行代码 *package main* 定义了包名。你必须在中非注释的第一行指明这个文件属于哪个包，如：package main。package main表示一个可独立执行的程序，每个 Go 应用程序都包含一个名为 main 的包。

3. 下一行 *func main()* 是程序开始执行的函数。main 函数是每一个可执行程序所必须包含的，一般来说都是在启动后第一个执行的函数（如果有 init() 函数则会先执行该函数）。
4. 下一行 /.../ 是注释，在程序执行时将被忽略。单行注释是最常见的注释形式，你可以在任何地方使用以 // 开头的单行注释。多行注释也叫块注释，均已以 /* 开头，并以 */ 结尾，且不可以嵌套使用，多行注释一般用于包的文档描述或注释成块的代码片段。
5. 下一行 fmt.Println(...) 可以将字符串输出到控制台，并在最后自动增加换行字符 \n。 使用 fmt.Print("hello, world\n") 可以得到相同的结果。 Print 和 Println 这两个函数也支持使用变量，如：fmt.Println(arr)。如果没有特别指定，它们会以默认的打印格式将变量 arr 输出到控制台。
6. 当标识符（包括常量、变量、类型、函数名、结构字段等等）以一个大写字母开头，如：Group1，那么使用这种形式的标识符的对象就可以被外部包的代码所使用（客户端程序需要先导入这个包），这被称为导出（像面向对象语言中的 public）；标识符如果以小写字母开头，则对包外是不可见的，但是他们在整个包的内部是可见并且可用的（像面向对象语言中的 protected ）。

5.3 执行代码

1. 我们还可以使用 **go build** 命令来生成二进制文件：

   windows下面生成``exe 文件 ，Linux下生成 sh` 无后缀的可执行文件

```sh
$ go build hello.go
$ ls
hello    hello.go
$ ./hello
Hello, World!
```

   💔**注意：** 使用**go run** 命令也可以直接运行hello.go 程序。类似于python编译脚本语言的形式，但是在生产中，我们实际上先编译，后执行。但是这个时候编译的速度会变慢，而**直接执行二进制文件速度会快很多。**

   编译指定二进制文件名称： 类似于C/C++

   windows: `go build -o myhello.exe hello.go`

   linux: `go build -o myhello hello.go`

5.4 ❤️注意

- 需要注意的是 { 不能单独放在一行，所以以下代码在运行时会产生错误

```go
package main
import "fmt"

func main() {       // 错误，{ 不能在单独的行上
  fmt.Println("Hello, World!")
}
```

- window**编译可执行文件的时候不是使用./编译的，而是直接编译exe文件**
- go语言**严格区分大小写**
- go方法由一条条语句构成，每一条语句结尾 **不需要分号**
- **go编译器是一行行的编译的，因此我们一行只写一条语句**，不要把多条语句写在同一行，否则报错
- **go语言定义的变量或者import包如果没有使用到，代码不能编译通过，避免冗余**

go语言转义字符（escape char）

字符	含义
\t	一个制表符
\n	换行符
\\	\
\r	回车
"	一个"

go语言常见错误和解决方法

1. the system cannot find the file specified
2. .....不是内部或者外部命令，也不是可运行程序或者处理文件

一般这种错误都是因为源文件名不存在或者当前路径错误

Go语言目录结构

> 很重要的，在根目录下面创建 `goproject` ，在这里面新建一个 `src` 目录，在 `src` 里面写入 `go_code` 目录，用来写代码，在这个目录里面写入项目，项目名称可以叫 `project01` ， `project02` ，每个项目里面都有一个 `main` 目录这是主目录，里面包含 `main.go` 文件

我们在开发程序的时候或者是开发项目的时候，一定要按照Go语言的规定目录进行

Go语言基本语法

Go 标记

Go 程序可以由多个标记组成，可以是关键字，标识符，常量，字符串，符号。如以下 GO 语句由 6 个标记组成：

```
fmt.Println("Hello, World!")
```

6 个标记是(每行一个)：

```go
1. fmt
2. .
3. Println
4. (
5. "Hello, World!"
6. )
```

行分隔符

在 Go 程序中，一行代表一个语句结束。每个语句不需要像 C 家族中的其它语言一样以分号 ; 结尾，因为这些工作都将由 Go 编译器自动完成。

如果你打算将多个语句写在同一行，它们则必须使用 ; 人为区分，但在实际开发中我们并不鼓励这种做法。

```
3    fmt.Println("hello golang!")
```

注释(comment)

注释不会被编译，每一个包应该有相关注释。（和C语言C++类似）

单行注释是最常见的注释形式，你可以在任何地方使用以 // 开头的单行注释。多行注释也叫块注释，均已以 /* 开头，并以 */ 结尾。如：

```go
1  // 单行注释
2  /*
3   Author by
4   我是多行注释
5   */
```

标识符

标识符用来命名变量、类型等程序实体。一个标识符实际上就是一个或是多个字母(A~Z和a~z)数字(0~9)、下划线_组成的序列，但是第一个字符必须是字母或下划线而不能是数字。

以下是有效的标识符：

```go
1  mahesh   kumar   abc   move_name   a_123
2  myname50   _temp   j   a23b9   retVal
```

以下是无效的标识符：

- 1ab（以数字开头）
- case（Go 语言的关键字）
- a+b（运算符是不允许的）

字符串连接

Go 语言的字符串可以通过 + 实现

```
3  func main() {
4    fmt.Println("Google" + "Runoob")
5  }
6  //**如果要一行输出：**
7  fmt.Println("hello,word",
8          "hello golang")
```

以上实例输出结果为：

```
1  GoogleRunoob
```

关键字

下面列举了 Go 代码中会使用到的 25 个关键字或保留字：

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

除了以上介绍的这些关键字，Go 语言还有 36 个预定义标识符：

append	bool	byte	cap	close	complex	complex64	c
copy	false	float32	float64	imag	int	int8	ir
int32	int64	iota	len	make	new	nil	p
print	println	real	recover	string	true	uint	u

程序一般由关键字、常量、变量、运算符、类型和函数组成。

程序中可能会使用到这些分隔符：括号 ()，中括号 [] 和大括号 {}。

go语言的空格

Go 语言中变量的声明必须使用空格隔开，如：

```
var age int;
```

语句中适当使用空格能让程序更易阅读。

无空格：

```
fruit=apples+oranges;
```

在变量与运算符间加入空格，程序看起来更加美观，如：

```
fruit = apples + oranges;
```

格式化字符串

Go 语言中使用 fmt.Sprintf 格式化字符串并赋值给新串：

go

```
package main

import (
  "fmt"
)

func main() {
  // %d 表示整型数字，%s 表示字符串*
  var stockcode = 123
  var enddate = "2020-12-31"
  var url = "Code=%d&endDate=%s"
  var target_url = fmt.Sprintf(url,stockcode,enddate)  //格式化字符串
  fmt.Println(target_url)
}
```

输出结果为：

golang 标准库api文档

1. **官网地址（需要外网）**： https://golang.org.pkg

   可以查看golang中所有的接口

2. **golang中文网**： https://studygolang.com/pkgdoc

为了方便在Windows下使用命令，熟悉一下dos:磁盘操作系统

DOS命令

`copy` 移动文件 --参考linux中cp

`move` 剪切文件 --参考mv命令

`del` 删除文件 --参考Linux中rm 和rmdir

`cls` 清屏 --程序中会使用到，在Linux中用clear

语言的数据类型

在 Go 编程语言中，**数据类型用于声明函数和变量。**

数据类型的出现是为了把数据分成所需内存大小不同的数据，编程的时候需要用大数据的时候才需要申请大内存，就可以充分利用内存。

Go 语言按类别有以下几种数据类型：

序号	类型和描述
1	**布尔型**：布尔型的值只可以是常量 true 或者 false。一个简单的例子：var b bool = true。
2	**数字类型**： 整型 int 和浮点型 float32、float64，Go 语言支持整型和浮点型数字，并且支持复数，其中位的运算采用补码。

3	**字符串类型**: 字符串就是一串固定长度的字符连接起来的字符序列。Go 的字符串是由单个字节连接起来的。Go 语言的字符串的字节使用 UTF-8 编码标识 Unicode 文本。
4	**派生类型**: 包括：(a) 指针类型（Pointer）(b) 数组类型(c) 结构化类型(struct)(d) Channel 类型(e) 函数类型(f) 切片类型(g) 接口类型（interface）(h) Map 类型

数据类型

Go 也有基于架构的类型，例如：int、uint 和 uintptr。

序号	类型和描述
1	**uint8** 无符号 8 位整型 (0 到 255)
2	**uint16** 无符号 16 位整型 (0 到 65535)
3	**uint32** 无符号 32 位整型 (0 到 4294967295)
4	**uint64** 无符号 64 位整型 (0 到 18446744073709551615)
5	**int8** 有符号 8 位整型 (-128 到 127)
6	**int16** 有符号 16 位整型 (-32768 到 32767)
7	**int32** 有符号 32 位整型 (-2147483648 到 2147483647)
8	**int64** 有符号 64 位整型 (-9223372036854775808 到 9223372036854775807)

浮点型

序号	类型和描述
1	**float32** IEEE-754 32位浮点型数
2	**float64** IEEE-754 64位浮点型数
3	**complex64** 32 位实数和虚数
4	**complex128** 64 位实数和虚数

其他数字类型

以下列出了其他更多的数字类型：

序号	类型和描述
1	**byte** 类似 uint8
2	**rune** 类似 int32
3	**uint** 32 或 64 位
4	**int** 与 uint 一样大小
5	**uintptr** 无符号整型，用于存放一个指针

变量

- Go 语言变量名由字母、数字、下划线组成，其中**首个字符不能为数字。**
- 变量表示内存中的存储区域，有自己的名称和类型
- 声明变量的一般形式是使用 `var` 关键字：

```
var identifier type
```

可以一次声明多个变量：

```
var identifier1, identifier2 type
```

```go
package main
import "fmt"
func main() {
  var a string = "Runoob"  //定义一个类型为string的变量a
  fmt.Println(a)

  var b, c int = 1, 2

```

以上实例输出结果为：

```
Runoob
1 2
```

变量声明

第一种，指定变量类型，如果没有初始化， 则变量默认为零值 。

```go
var v_name v_type
v_name = value
```

零值就是变量没有做初始化时系统默认设置的值。

```go
package main
import "fmt"
func main() {

  // 声明一个变量并初始化
  var a = "RUNOOB"
  fmt.Println(a)

  // 没有初始化就为零值
  var b int
  fmt.Println(b)

  // bool 零值为 false
  var c bool
  fmt.Println(c)
}
```

以上实例执行结果为：

```
RUNOOB
0
false
```

- 布尔类型为 false
- **字符串为 ""（空字符串）**
- 以下几种类型为 nil：

```go
var a *int
var a []int
var a map[string] int
var a chan int
var a func(string) int
var a error // error 是接口
```

案例

```go
package main

import "fmt"

func main() {
  var i int
  var f float64
  var b bool
  var s string
  fmt.Printf("%v %v %v %q\n", i, f, b, s) //打印默认值
}
```

输出结果是：

```
0 0 false ""
```

第二种，根据值自行判定变量类型。

```
var v_name = value
```

> 由于设置了 `d` 值，所以打印出来的d是 `true`

```
3  func main() {
4    var d = true
5    fmt.Println(d)
6  }
```

输出结果是：

```
1  true
```

第三种，如果变量已经使用 var 声明过了，再使用 := 声明变量，就产生编译错误，格式：

```
1  v_name := value
```

声明的同时可以直接赋值使用 :=

例如：

```go
1  var intVal int
2  intVal :=1   // 这时候会产生编译错误，因为 intVal 已经声明，不需要重新声明
```

直接使用下面的语句即可：

```go
1  intVal := 1   // 此时不会产生编译错误，因为有声明新的变量，因为 := 是一个声明语句
```

intVal := 1 相等于：

```go
1  var intVal int
2  intVal =1
```

可以将 var f string = "Runoob" 简写为 f := "Runoob"：

实例

```
3  func main() {
4    f := "Runoob" // var f string = "Runoob"
5    fmt.Println(f)
6  }
```

输出结果是:

```
1  Runoob
```

多变量声明

go

```
1  //类型相同多个变量, 非全局变量
2  var vname1, vname2, vname3 type
3
4  vname1, vname2, vname3 = v1, v2, v3
5
6  var vname1, vname2, vname3 = v1, v2, v3     // 和 python 很像,不需要显示声明类型
7
8  vname1, vname2, vname3 := v1, v2, v3
9  // 出现在 := 左侧的变量不应该是已经被声明过的，否则会导致编译错误
10
11 // 这种因式分解关键字的写法一般用于声明全局变量
12 var (
13     vname1 v_type1
14     vname2 v_type2
15 )
```

实例

go

```
1  package main
2
3  var x, y int
4  var (  // 这种因式分解关键字的写法一般用于声明全局变量*
5    a int
6    b bool
7  )
```

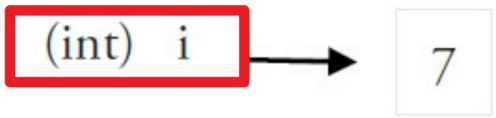
```
/*也可以使用最简洁方法:
 c,d,e,f := 1,2,123,"hello"
*/
//这种不带声明格式的只能在函数体中出现
//g, h := 123, "hello"

func main(){
  g, h := 123, "hello"       //定义的是局部变量
  println(x, y, a, b, c, d, e, f, g, h)
}
```

以上实例执行结果为:

```
0 0 0 false 1 2 123 hello 123 hello
```

值类型和引用类型

所有像 int、float、bool 和 string 这些基本类型都属于值类型，使用这些类型的变量直接指向存在内存中的值:



32 bit word

Fig 4.1: Value type

当使用等号 `=` 将一个变量的值赋值给另一个变量时，如: `j = i` ，实际上是**在内存中将 i 的值进行了拷贝**:

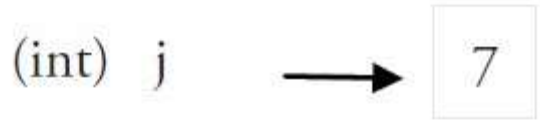


Fig 4.2: Assignment of value types

你可以通过 `&i` 来获取变量 i 的内存地址，例如： `0xf840000040` （每次的地址都可能不一样）。

❗**值类型变量的值存储在堆中。**

内存地址会根据机器的不同而有所不同，甚至相同的程序在不同的机器上执行后也会有不同的内存地址。因为每台机器可能有不同的存储器布局，并且位置分配也可能不同。

更复杂的数据通常会需要使用多个字，这些数据一般使用引用类型保存。

一个引用类型的变量 r1 存储的是 r1 的值所在的内存地址（数字），或内存地址中第一个字所在的位置。

!

Fig 4.3: Reference types and assignment

这个内存地址称之为指针，这个指针实际上也被存在另外的某一个值中。

同一个引用类型的指针指向的多个字可以是在连续的内存地址中（内存布局是连续的），这也是计算效率最高的一种存储形式；也可以将这些字分散存放在内存中，每个字都指示了下一个字所在的内存地址。

当使用赋值语句 `r2 = r1` 时，只有引用（地址）被复制。

简短形式，使用 := 赋值操作符

我们知道可以在变量的初始化时省略变量的类型而由系统自动推断，声明语句写上 var 关键字其实是显得有些多余了，因此我们可以将它们简写为 `a := 50` 或 `b := false` 。

a 和 b 的类型（int 和 bool）将由编译器自动推断。

这是使用变量的首选形式，但是它只能被用在函数体内，而不可以用于全局变量的声明与赋值。使用操作符 := 可以高效地创建一个新的变量，称之为初始化声明。

注意事项

如果在相同的代码块中，我们不可以再次对于相同名称的变量使用初始化声明，例如：a := 20 就是不被允许的，编译器会提示错误 `no new variables on left side of :=` ，但是 a = 20 是可以的，因为这是给相同的变量赋予一个新的值。

如果你在定义变量 a 之前使用它，则会得到编译错误 undefined: a。

如果你声明了一个局部变量却没有在相同的代码块中使用它，同样会得到编译错误，例如下面这个例子当中的变量 a：

```go
package main

import "fmt"

func main() {
  var a string = "abc"
  fmt.Println("hello, world")
}
```

尝试编译这段代码将得到错误 **a declared and not used**。

此外，单纯地给 a 赋值也是不够的，这个值必须被使用，所以使用

```
fmt.Println("hello, world", a)
```

会移除错误。

```
var a, b, c int
```

多变量可以在同一行进行赋值，如：

```
var a, b int
var c string
a, b, c = 5, 7, "abc"
```

go

上面这行假设了变量 a，b 和 c 都已经被声明，否则的话应该这样使用：

```
a, b, c := 5, 7, "abc"
```

右边的这些值以相同的顺序赋值给左边的变量，所以 a 的值是 5， b 的值是 7，c 的值是 "abc"。

这被称为 并行 或 同时 赋值。

如果你想要交换两个变量的值，则可以简单地使用 `a, b = b, a` ，两个变量的类型必须是相同。

空白标识符 _ 也被用于抛弃值，如值 5 在：_, b = 5, 7 中被抛弃。

_ 实际上是一个只写变量，你不能得到它的值。这样做是因为 Go 语言中你必须使用所有被声明的变量，但有时你并不需要使用从一个函数得到的所有返回值。

并行赋值也被用于当一个函数返回多个返回值时，比如这里的 val 和错误 err 是通过调用 Func1 函数同时得到：

```
val, err = Func1(var1)
```

END 链接

- ⬆️上一节🔗 ⬇️下一节🔗
  - Ⓜ️回到目录🏠
- 👊参与贡献💞 ❤️ 💖)

在GitHub上贡献此页面

上次更新: 2022/10/6 20:49:39

贡献者: xiongxinwei

go语言常量、基本数据类型、字符串转化 →