Go快速入门-上篇

> 讲师：杜Sir
>
> 训练营地址：https://youdianzhishi.com/web/course/1035

一、Go语言特性

1.1 Go语言介绍

- Go 即 Golang，是 Google 公司 2009 年 11 月正式对外公开的一门编程语言。
- 根据 Go 语言开发者自述，近 10 多年，从单机时代的 C 语言到现在互联网时代的 Java，都没有令人满意的开发语言，而 C++往往给人的感觉是，花了 100%的经历，却只有 60%的开发效率，产出比太低，Java 和 C#的哲学又来源于 C++。
- 并且，随着硬件的不断升级，这些 语言不能充分的利用硬件及 CPU。
- 因此，一门高效、简洁、开源的语言诞生了。
- Go 语言不仅拥有静态编译语言的安全和高性能，而且又达到了动态语言开发速度和易 维护性。
- 有人形容 Go 语言：Go = C + Python ，说明 Go 语言既有 C 语言程序的运行速度，又能达到 Python 语言的快速开发。
- Go **语言是非常有潜力的语言**，是因为它的应用场景是目前互联网非常热门的几个领域
- 比如 WEB 开发、区块链开发、大型游戏服务端开发、分布式/云计算开发。
- 国内比较知名的B 站就是用 Go 语言开发的，像 Goggle、阿里、京东、百度、腾讯、小米、360 的很多应用也是使用 Go 语言开发的。

1.2 Go语言优势

- 天生支持高并发
- 自动垃圾回收机制
- 不需要环境依赖
- 云原生无缝接入
- 社区活跃度
- 易上手

二、Go 运行环境

- golang下载地址 https://go.dev/dl/
- 确定go版本信息

```
go version # 查看go版本
go env # 查看 go 环境
```

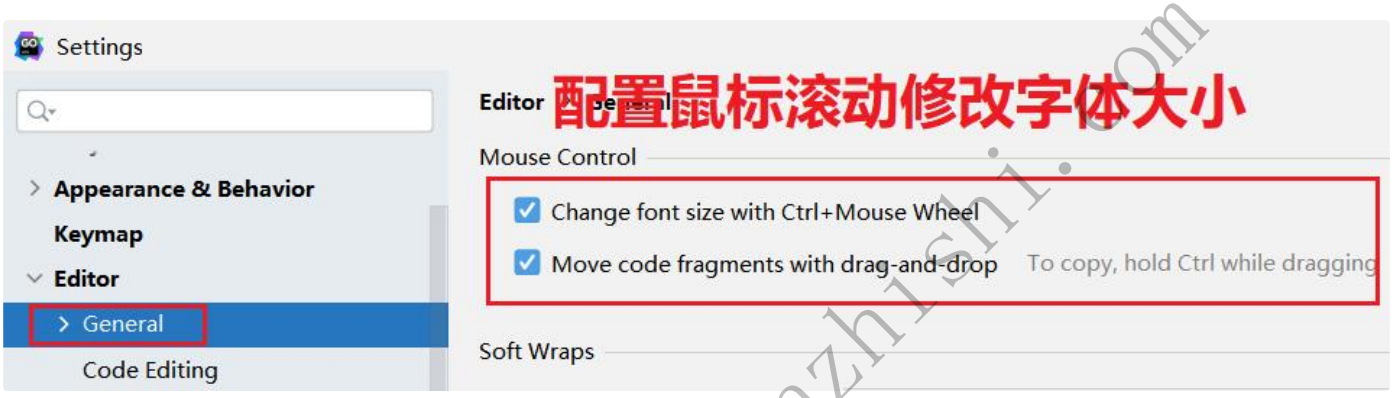
2.1 下载Golang IDE

https://www.jetbrains.com/go/download/#section=windows

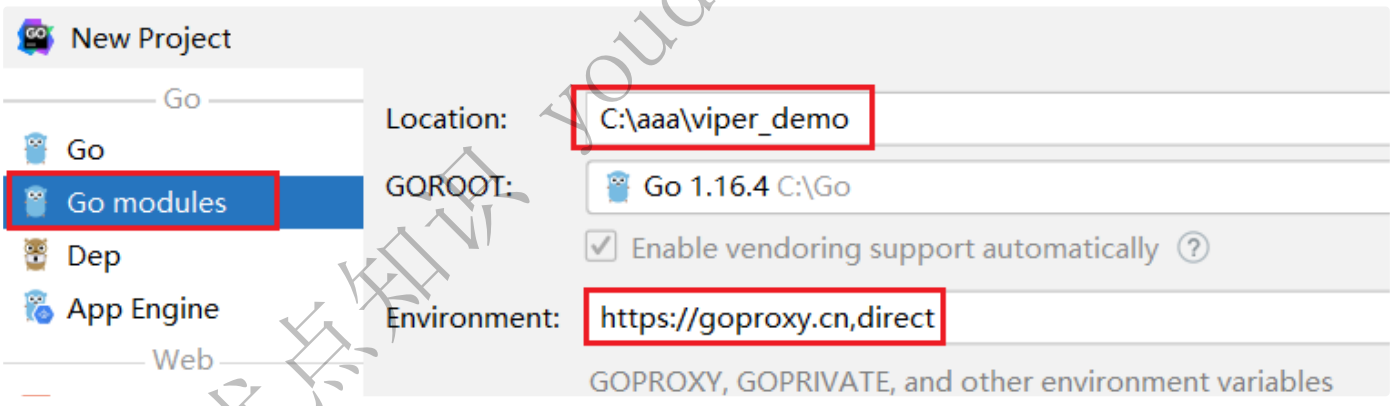
- liteIDE 运行速度快，代码提示特别好用，但是调试功能不太好用
- VSCode 调试功能好用，但是代码提示非常一般，写起来特别费劲
- GoLand 各项功能非常完善，但是是收费的，并且占用资源较多

2.2 配置

- 配置鼠标滚动修改字体大小
- 字体的更改方法：File → Settings → Editor → Font → Size，推荐选18或者20
- 主题的更改方法：File → Settings → Editor → Color Scheme → Scheme，推荐选Colorful Darcula



2.3 创建项目



2.4 go mod

- go1.11版本开始支持go modules
- 包的存放路径为{GOPATH}，windows默认在%USERPROFILE%\go(C:\Users\xxx\go)
- go.mod记录依赖包的名字以及版本号等信息
- go.sum记录依赖包的校验信息
- 使用go mod初始化项目：

```
mkdir go-demo
cd go-demo
go mod init go-demo #生成go.mod和go.sum
go env -w GOPROXY="https://goproxy.cn,direct"
```

- 下载未安装但使用到的包

```
go mod tidy
```

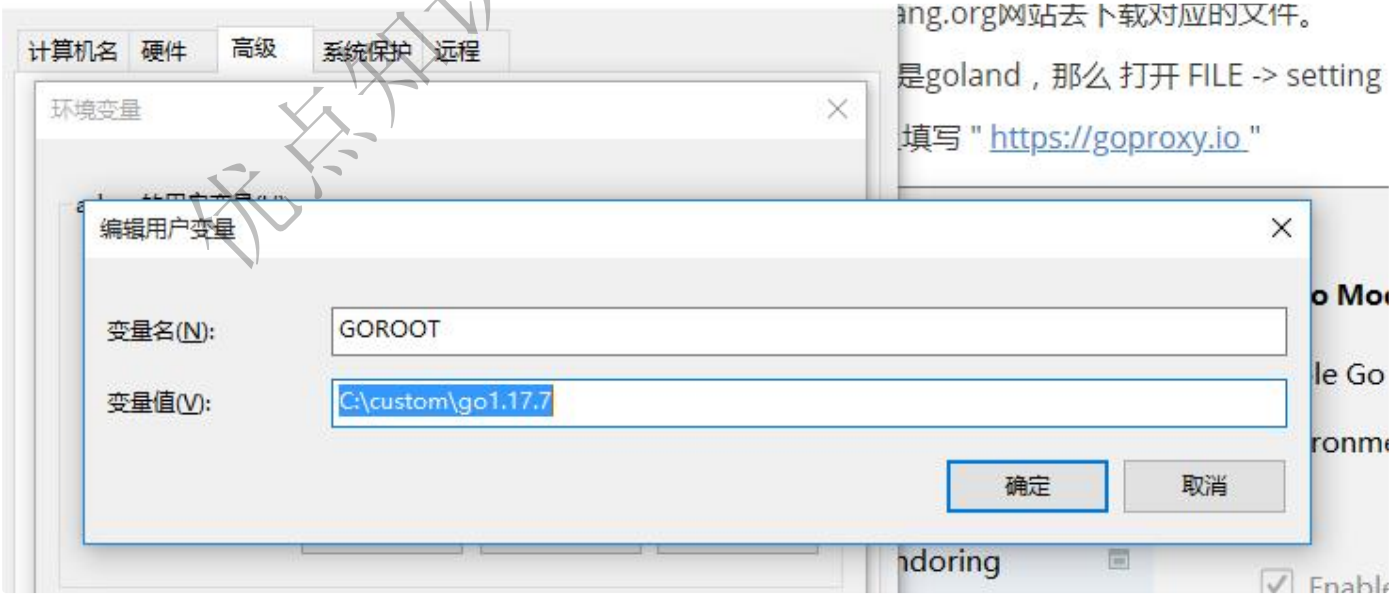
- 修改依赖包版本：修改`go.mod`中依赖包的版本号后，执行`go mod tidy`即可

2.5 解决部分包无法下载问题

- 使用golang 开发有时会遇到 golang unrecognized import path "golang.org/x" 之类的错误。
- 原因是无法访问golang.org网站去下载对应的文件。
- 如果使用的开发IDE是goland，那么 打开 FILE → setting → Go Modules 选项
  - 在proxy 选项上填写 " https://goproxy.io "

2.6 Goland无法识别GO SDK的问题

- 设置环境变量{GOROOT}到golang的安装路径（比如我的安装路径是C:\custom\go1.17.7）



- 编辑{GOROOT}/src/runtime/internal/sys/zversion.go文件，添加如下内容：

zversion.go

```
// Code generated by go tool dist; DO NOT EDIT.

package sys

const StackGuardMultiplierDefault = 1
const TheVersion = `go1.17.7`
```

2.7 运行时windows系统不兼容

```
go env -w GOEXE=.exe
go env -w GOOS=windows
```

三、Go 基本语法

3.1 变量定义方法

1. var定义变量

- var 变量名 类型 = 表达式

```
var name string = "zhangsan"
var age int = 21
var isOk bool
```

2. 类型推导方式定义变量

- a 在函数内部，可以使用更简略的 := 方式声明并初始化变量。
- **注意**：短变量只能用于声明局部变量，不能用于全局变量的声明

```
// 变量名 := 表达式
n := 10
var age = 18
```

3. 一次定义多个变量

```
package main
import "fmt"
func main() {
    var username, sex string
    username = "张三"
    sex = "男"
    fmt.Println(username, sex)
}
```

4. 批量声明变量

```
package main
import "fmt"
func main() {
    var (
        a string
        b int
        c bool
    )
    a = "张三"
    b = 10
    c = true
    fmt.Println(a,b,c)
}
```

3.2 常量定义

- 声明了 pi 和 e 这两个常量之后，在整个程序运行期间它们的值都不能再发生变化了。

```
const pi = 3.1415
const e = 2.7182
// 多个常量也可以一起声明
const (
    pi = 3.1415
    e = 2.7182
)
```

- const 同时声明多个常量时，如果省略了值则表示和上面一行的值相同。

```
const (
    n1 = 100
    n2
    n3
)
// 上面示例中，常量 n1、n2、n3 的值都是 100
```

3.3 fmt包

- Println:
  - 一次输入多个值的时候 Println 中间有空格
  - Println 会自动换行，Print 不会
- Print:
  - 一次输入多个值的时候 Print 没有 中间有空格
  - Print 不会自动换行
- Printf
  - Printf 是格式化输出，在很多场景下比 Println 更方便
- Sprintf

- Sprintf 是格式化输出，返回字符串，不打印，常用于变量的拼接以及赋值

```
package main

import "fmt"

func main() {
    fmt.Print("zhangsan","lisi","wangwu") // zhangsanlisiwangwu
    fmt.Println("zhangsan","lisi","wangwu") // zhangsan lisi wangwu
    name := "zhangsan"
    age := 20
    fmt.Printf("%s 今年 %d 岁", name, age) // zhangsan 今年 20 岁
    info := fmt.Sprintf("姓名: %s, 性别: %d", name, 20)
  fmt.Println(info)
}
```

3.4 Init函数和main函数

1. main函数

```
// Go语言程序的默认入口函数(主函数): func main()
// 函数体用 {} 一对括号包裹

func main(){
    //函数体
}
```

2. init函数

- go语言中 init函数用于包（package)的初始化，该函数是go语言的一个重要特性。
- 有下面的特征:
  - init函数是用于程序执行前做包的初始化的函数，比如初始化包里的变量等
  - 每个包可以拥有多个init函数
  - 同一个包中多个init函数的执行顺序go语言没有明确的定义(说明)
  - 不同包的init函数按照包导入的依赖关系决定该初始化函数的执行顺序
  - init函数不能被其他函数调用，而是在main函数执行之前，自动被调用

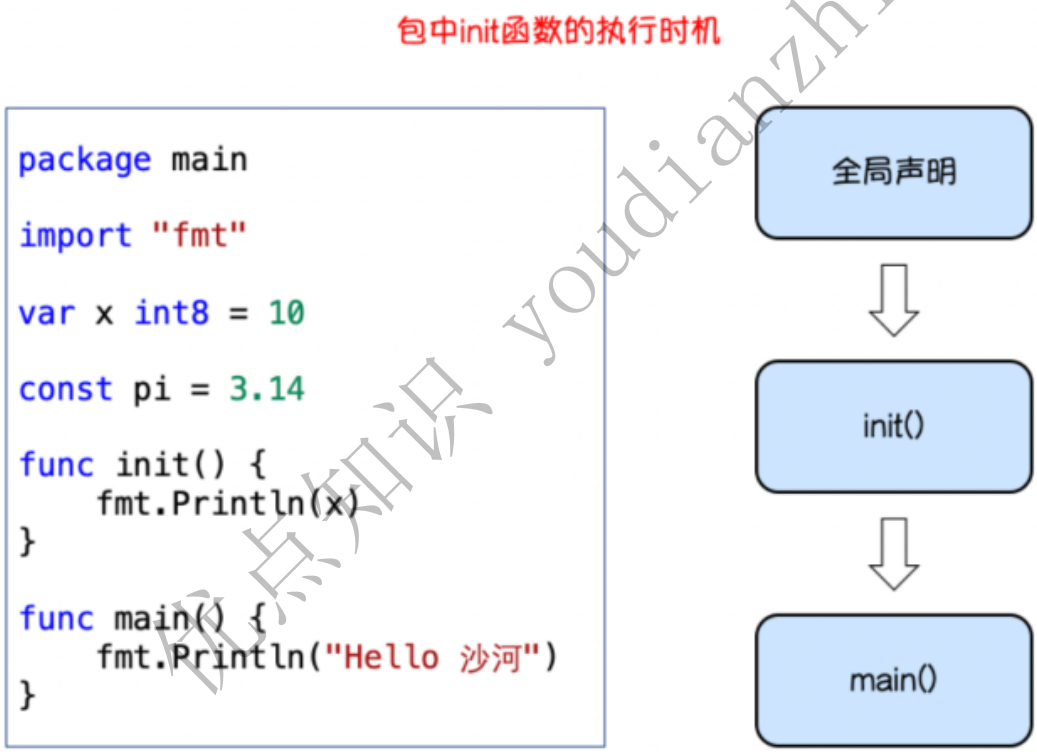
```
package main

import "fmt"

// Go语言程序的默认入口函数(主函数): func main()
// 函数体用 {} 一对括号包裹

func init() {
  fmt.Println("我是init函数")
}

func main(){
```

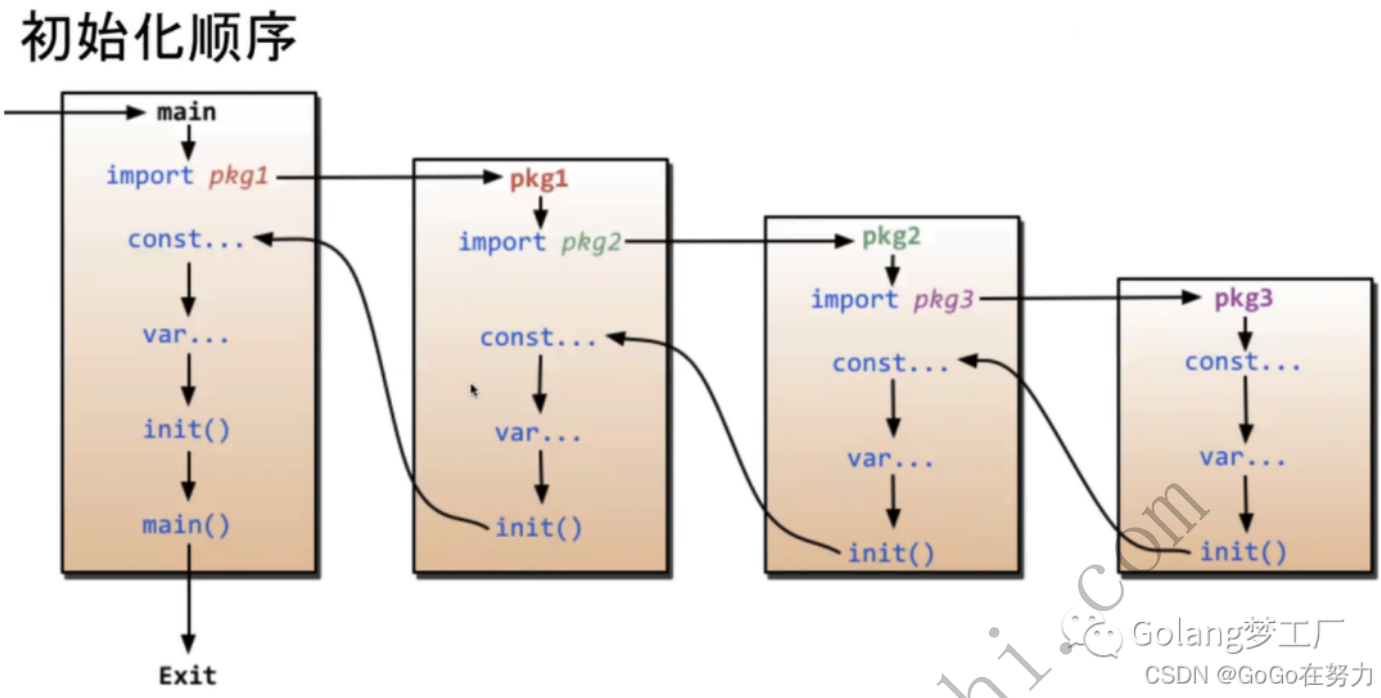
```
    //函数体
    fmt.Println("我是mian函数")
}
```

3. init函数和main函数的异同

- 相同点:
  - 两个函数在定义时不能有任何的参数和返回值，且Go程序自动调用。
- 不同点:
  - init可以应用于任意包中，且可以重复定义多个。
  - main函数只能用于main包中，且只能定义一个。
- 两个函数的执行顺序:
  - 对同一个go文件的 init() 调用顺序是从上到下的。
  - 对同一个package中不同文件是按文件名字符串比较“从小到大”顺序调用各文件中的 init() 函数。
  - 对于不同的 package ，如果不相互依赖的话，按照main包中"先 import 的后调用"的顺序调用其包中的 init()
  - 如果 package 存在依赖，则先调用最早被依赖的 package 中的 init() ，最后调用 main 函数。

4. init()函数介绍





3.5 golang中关键字

- var和const ：变量和常量的声明
- package and import ： 包和导入
- func ： 用于定义函数和方法
- return ：用于从函数和方法返回
- defer someCode ：在函数退出之前执行
- go ：用于并行
- select 用于选择不同类型的通讯
- interface 用于定义接口
- struct 用于定义抽象数据类型
- break、case、continue、for、fallthrough、else、if、switch、goto、default 流程控制
- chan 用于channel通讯
- type 用于声明自定义类型
- map 用于声明map类型数据
- range 用于读取slice、map、channel数据

3.6 命名规范

- Go是一门区分大小写的语言。命名规则涉及变量、常量、全局函数、结构体、接口、方法等的命名。
- Go语言从语法层面进行了以下限定：任何需要对外暴露的名字必须以大写字母开头，不需要对外暴露的则应该以小写字母开头。
- 当命名（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：Analysize，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）；
- **命名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private ）
- 包名称
  - 保持package的名字和目录保持一致，尽量采取有意义的包名，简短，有意义，尽量和标准库不要冲突。包名应该为**小写**单词，不要使用下划线或者混合大小写。

```
package domain
```

- 文件命名
  - 尽量采取有意义的文件名，简短，有意义，应该为**小写**单词，使用**下划线**分隔各个单词。

```
approve_service.go
```

- 结构体命名
  - 采用驼峰命名法，首字母根据访问控制大写或者小写struct 申明和初始化格式采用多行，例如 下面：

```
type MainConfig struct {
    Port string `json:"port"`
    Address string `json:"address"`
}
```

- **接口命名**命名规则基本和上面的结构体类型单个函数的结构名以 “er” 作为后缀，例如 Reader ,Writer 。
  - 命名以“er”结尾，如：Writer，xxxHandler，Helper，Manager等
  - 接口方法声明 = 方法名+方法签名如：methodA (param1, param2) outputTypeList

```
type Reader interface {
  Read(p []byte) (n int, err error)
}
```

- 变量命名
  - 和结构体类似，变量名称一般遵循驼峰法，首字母根据访问控制原则大写或者小写 但遇到特有名词时，需要遵循以下规则：如果变量为私有，且特有名词为首个单词，则使用小写
  - 如 appService若变量类型为 bool 类型，则名称应以 Has，Is，Can 或 Allow 开头

```
var isExist bool
var hasConflict bool
var canManage bool
var allowGitHook bool
```

- **常量命名**常量均需使用全部大写字母组成，并使用下划线分词
  - const APP_URL = "https://www.baidu.com" 如果是枚举类型的常量，需要先创建相应类型

```
type Scheme string
const (
    HTTP Scheme = "http"
    HTTPS Scheme = "https"
)
```

四、基本数据类型

4.1 内置类型

1. 值类型

```
bool
int(32 or 64), int8, int16, int32, int64
uint(32 or 64), uint8(byte), uint16, uint32, uint64
float32, float64
string
complex64, complex128
array // 固定长度的数组
```

2. 引用类型:（指针类型）

```
slice // 序列数组(最常用)
map // 映射
chan // 管道
```

4.2 内置函数

- Go 语言拥有一些不需要进行导入操作就可以使用的内置函数。
- 它们有时可以针对不同的类型进行操作，例如：len、cap 和 append，
- 或必须用于系统级的操作，例如：panic。 因此，它们需要直接获得编译器的支持。

```
append // 用来追加元素到数组、slice中,返回修改后的数组、slice
close // 主要用来关闭channel
delete // 从map中删除key对应的value
panic // 停止常规的goroutine （panic和recover：用来做错误处理）
recover // 允许程序定义goroutine的panic动作
real // 返回complex的实部 （complex、real imag：用于创建和操作复数）
imag // 返回complex的虚部
make // 用来分配内存，返回Type本身(只能应用于slice, map, channel)
new // 用来分配内存，主要用来分配值类型，比如int、struct。返回指向Type的指针
cap // capacity是容量的意思，用于返回某个类型的最大容量（只能用于切片和 map）
copy // 用于复制和连接slice，返回复制的数目
len // 来求长度，比如string、array、slice、map、channel ，返回长度
print、println // 底层打印函数，在部署环境中建议使用 fmt 包
```

4.3 基本类型介绍

类型	长度(字节)	默认值	说明
bool	1	false	
byte	1	0	uint8
rune	4	0	Unicode Code Point, int32
int, uint	4 或 8	0	32 或 64 位
int8, uint8	1	0	-128 ~ 127, 0 ~ 255，byte 是 uint8 的别名
int16, uint16	2	0	-32768 ~ 32767, 0 ~ 65535
int32, uint32	4	0	-21 亿~ 21 亿, 0 ~ 42 亿，rune 是 int32 的别名
int64, uint64	8	0	
float32	4	0.0	
float64	8	0.0	
complex64	8		
complex128	16		
uintptr	4 或 8		以存储指针的 uint32 或 uint64 整数
array			值类型
struct			值类型
string		""	UTF-8 字符串
slice		nil	引用类型
map		nil	引用类型
channel		nil	引用类型
interface		nil	接口
function		nil	函数

五、数字

5.1 数字类型

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

2. 整型分为两大类

- 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

5.2 数字定义

1. 定义数字类型

```
package main
import "fmt"
func main() {
    var a int8 = 4
    var b int32 = 4
    var c int64 = 4
    d := 4
    fmt.Printf("a: %T %v \n", a, a)
    fmt.Printf("b: %T %v \n", b, b)
```

```
    fmt.Printf("c: %T %v \n", c, c)
    fmt.Printf("d: %T %v \n", d, d)
}
/*
a: int8 4
b: int32 4
c: int64 4
d: int 4
*/
```

2. reflect.TypeOf查看数据类型

```
package main
import (
    "fmt"
    "reflect"
)
func main() {
    c := 10
    fmt.Println( reflect.TypeOf(c) ) // int
}
```

5.3 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)) // true 占用字节:  1
}
```

六、字符串

6.1 字符串

1. 字符串

- 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

```
s1 := "hello"
s2 := "你好"
```

2. 多行字符串

- 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。

```
package main
import (
    "fmt"
)
func main() {
    s1 := `
    第一行
    第二行
    第三行`
    fmt.Println(s1)
}
```

3. byte和rune

- Go 语言的字符有以下两种

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

- 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。

```
package main
import "fmt"
func main() {
    // “美国第一”
    s := "美国第一"
    s_rune := []rune(s)
    fmt.Println( "中国" + string(s_rune[2:])) // 中国第一
}
```

- rune类型用来表示utf8字符，一个rune字符由一个或多个byte组成。

```go
package main

import (
  "fmt"
)

func main(){
  strs := "我是谁"
  a := []rune(strs)
  b := []byte(strs)
  fmt.Printf("值: %d, 类型: %T \n", a, a)
  fmt.Printf("值: %d, 类型: %T", b, b)
    //值: [25105 26159 35841], 类型: []int32
  //值: [230 136 145 230 152 175 232 176 129], 类型: []uint8
}
```

6.2 字符串的常用操作

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

1. len(str)

```go
package main
import (
    "fmt"
)
func main() {
    var str = "this is str"
    fmt.Println(len(str))  // 11
}
```

2. +(拼接)

```
package main
import (
    "fmt"
)
func main() {
    var str1 = "你好"
    var str2 = "golang"
    fmt.Println(str1 + ", " + str2)
  fmt.Println(fmt.Sprintf("%s, %s", str1, str2))
}
```

3. strings.Split()

```
package main
import (
    "fmt"
    "strings"
)
func main() {
    var s = "123-456-789"
    var arr = strings.Split(s, "-")
    fmt.Println(arr)  // [123 456 789]
}
```

4. strings.Join()

```
package main
import (
    "fmt"
    "strings"
)
func main() {
    var str = "123-456-789"
    var arr = strings.Split(str, "-")    // [123 456 789]
    var str2 = strings.Join(arr, "*")    // 123*456*789
    fmt.Println(arr)
    fmt.Println(str2)
}
```

5. 单引号

- 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（’）

```go
package main
import "fmt"
func main() {
    a := 'a'
    name := "zhangsan"
    //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)    // 97 这里输出的是 a 字符串的 ASCII值
    fmt.Println(name)   // zhangsan
    //如果我们要输出这个字符，需要格式化输出
    fmt.Printf("的值是%c", a)  // a的值是a
}
```

6.3 字符串遍历

1. 遍历字符串

```go
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])
        // 104(h) 101(e) 108(l) 108(l) 111(o) 32( ) 229(å) 188(¼) 160() 228(ä) 184(¸)
137()
    }
    fmt.Println() // 打印一个换行
    for _, r := range s { //rune
        fmt.Printf("%v⇒%c ", r, r)
        // 104⇒h 101⇒e 108⇒l 108⇒l 111⇒o 32⇒  24352⇒张 19977⇒三
    }
    fmt.Println()
}
```

6.4 转String

1. strconv

```go
package main
import (
    "fmt"
    "strconv"
)
func main() {
    //1、int 转换成 string
  var num1 int = 20
  s1 := strconv.Itoa(num1)
  fmt.Printf("类型:  %T ,值=%v \n", s1, s1) // 类型:  string ,值=20
```

```
  // 2、float 转 string
  var num2 float64 = 20.113123
  /* 参数 1: 要转换的值
     参数 2: 格式化类型
     参数 3: 保留的小数点 -1（不对小数点格式化）
     参数 4: 格式化的类型
  */
  s2 := strconv.FormatFloat(num2, 'f', 2, 64)
  fmt.Printf("类型:  %T ,值=%v \n", s2, s2) // 类型:  string ,值=20.11
  // 3、bool 转 string
  s3 := strconv.FormatBool(true)
  fmt.Printf("类型:  %T ,值=%v \n", s3, s3) // 类型:  string ,值=true
  //4、int64 转 string
  var num3 int64 = 20
  s4 := strconv.FormatInt(num3, 10) /* 第二个参数10为 进制 */
  fmt.Printf("类型 %T ,值=%v \n", s4, s4) // 类型 string ,值=20
}
```

6.5 string与int转换

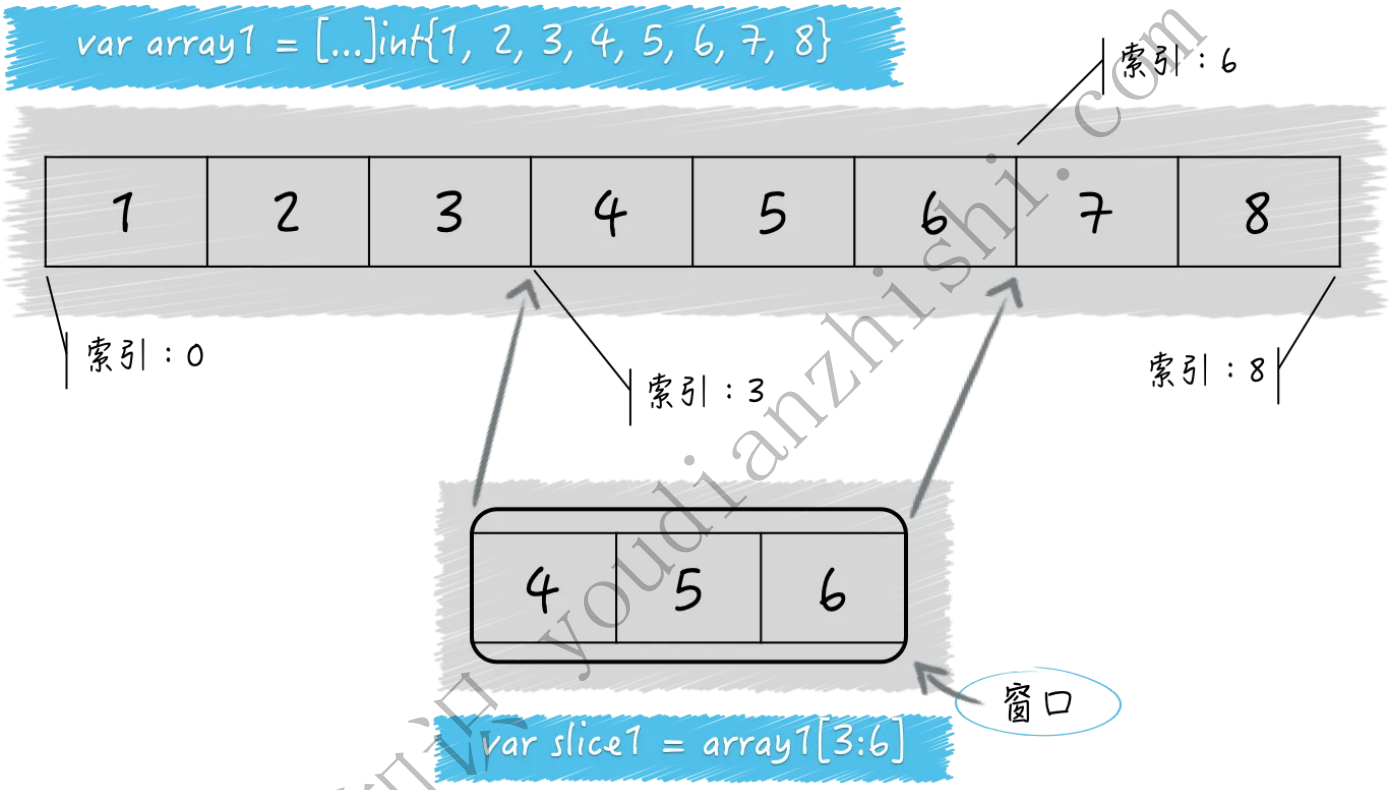
```
package main
import (
    "fmt"
    "strconv"
)
func main() {
    num := 100
    strNum := strconv.Itoa(num)
    fmt.Printf("num: %T %v \n", num, num)
    fmt.Printf("strNum: %T %v \n", strNum, strNum)
    intNum, _ := strconv.Atoi(strNum)
    fmt.Printf("intNum: %T %v \n", intNum, intNum)
}
/*
num: int 100
strNum: string 100
intNum: int 100
*/
```

七、数组

7.1 数组介绍

1. Array介绍

- 数组是指一系列 同一类型数据的集合 。
- 数组中包含的每个数据被称为数组元素(element)，这种类型可以是任意的原始类型，比如 int、string 等
- 一个数组包含的元素个数被称为数组的长度。
- 在 Golang 中数组是一个长度固定的数据类型，数组的长度是类型的一部分，也就是说 [5]int 和[10]int 是两个不同的类型 。
- Golang中数组的另一个特点是占用内存的连续性，也就是说数组中的元素是被分配到连续的内存地址中的，因而索引数组元素的速度非常快。
- 和数组对应的类型是 Slice（切片），Slice 是可以增长和收缩的动态序列，功能也更灵活，但是想要理解 slice 工作原理的话需要先理解数组，所以本节主要为大家讲解数组的使用。



2. 数组定义

```
var 数组变量名 [元素数量]类型
```

- 比如: var a [5]int， 数组的长度必须是常量，并且长度是数组类型的一部分， [5]int 和[4]int 是不同的类型。
- 一旦定义，长度不能变，数组中的元素是可以变。

```
package main
import "fmt"
func main() {
    // 定义一个长度为 3 元素类型为 int 的数组 a
    var a [5]int
    // 定义一个长度为 3 元素类型为 int 的数组 b 并赋值
    var b [3]int
    b[0] = 80
    b[1] = 100
    b[2] = 96
    fmt.Println(a)   // [0 0 0 0 0]
    fmt.Print(b)   // [80 100 96]
}
```

- 数组属于在进行数据传递时，是值传递，而非引用传递

```
package main
import "fmt"
func main(){
  var arr = [3]int{1,2,3}
  arr2 := arr
  arr2[0] = 3
  fmt.Println(arr,arr2) //[1 2 3] [3 2 3]
}
```

7.2 数组的遍历

1. 普通遍历数组

```
package main
import "fmt"
func main() {
    var a = [...]string{"北京", "上海", "深圳"}
    for i := 0; i < len(a); i++ {
      fmt.Println(a[i])
    }
}
/*
北京
上海
深圳
*/
```

2. k,v遍历数组

```
package main
import "fmt"
func main() {
    var a = [...]string{"北京", "上海", "深圳"}
    for index, value := range a {
      fmt.Println(index, value)
    }
}
/*
0 北京
1 上海
2 深圳
*/
```

八、切片

8.1 切片基础

1. 切片的定义

- 切片（Slice）是一个拥有相同类型元素的可变长度的序列。
- 它是基于数组类型做的一层封装，它非常灵活，支持自动扩容。
- 切片是一个引用类型，它的内部结构包含地址、长度和容量。
- 声明切片类型的基本语法如下:

```
// var name []T
// 1、name:表示变量名
// 2、T:表示切片中的元素类型
```

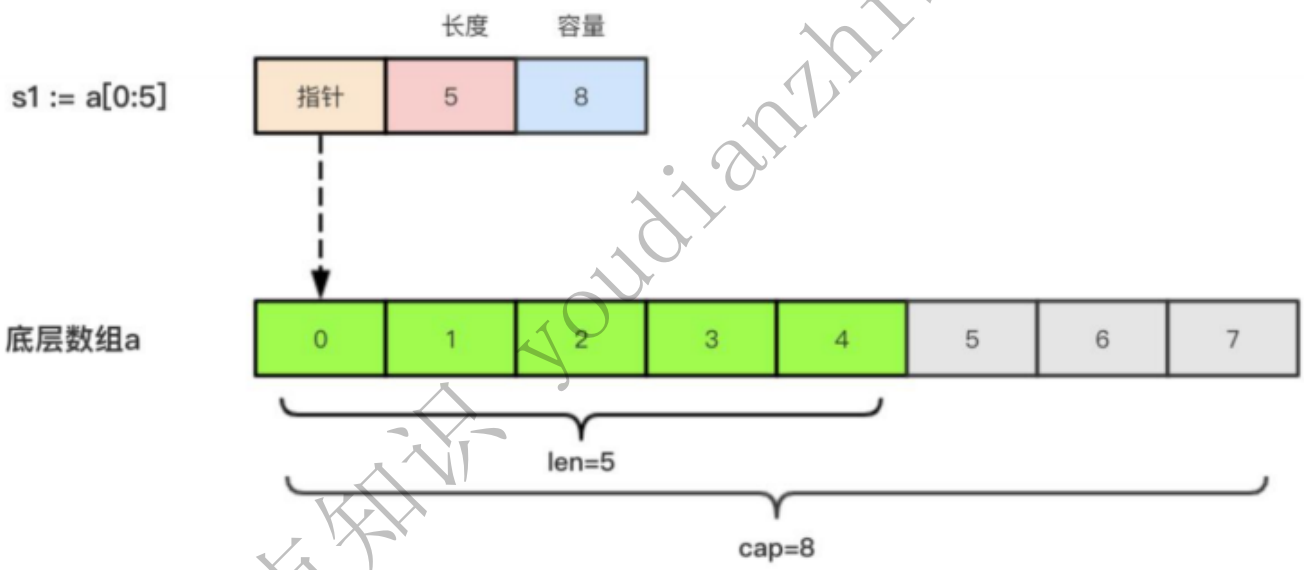
```
package main

import "fmt"
func main() {
    // 切片是引用类型，不支持直接比较，只能和 nil 比较
    var a []string    //声明一个字符串切片
    fmt.Println(a)    //[]
    fmt.Println(a == nil)    //true
    var b = []int{}    //声明一个整型切片并初始化
    fmt.Println(b)     //[]
    fmt.Println(b == nil)    //false
    var c = []bool{false, true}    //声明一个布尔切片并初始化
    fmt.Println(c)        //[false true]
    fmt.Println(c == nil)     //false
}
```

```
package main

import "fmt"
func main(){
  var slice = []int{1,2,3}
  slice2 := slice
  slice[0] = 3
  fmt.Println(slice,slice2)
}
```

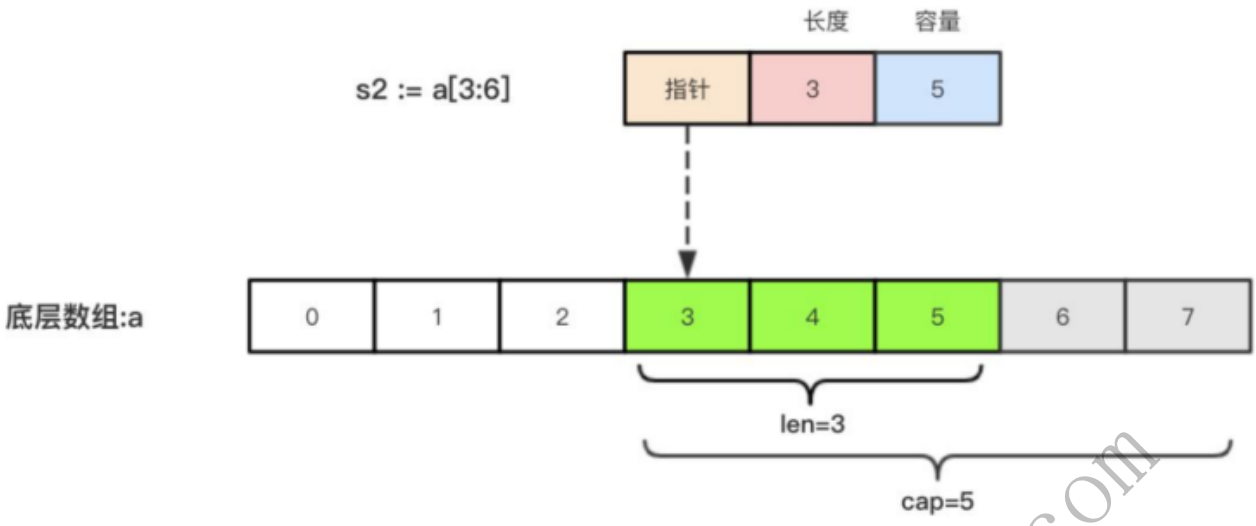
- 切片之间是不能比较的，我们不能使用==操作符来判断两个切片是否含有全部相等元素。
- 切片唯一合法的比较操作是和 nil 比较。 一个 nil 值的切片并没有底层数组，一个 nil 值的切片的长度和容量都是 0。
- 但是我们不能说一个长度和容量都是 0 的切片一定是 nil

2. 切片的本质

- 切片的本质就是对底层数组的封装，它包含了三个信息：底层数组的指针、切片的长度（len）和切片的容量（cap）。
- 举个例子，现在有一个数组 a := [8]int{0, 1, 2, 3, 4, 5, 6, 7}，切片 s1 := a[:5]，相应示意图如下。



- 切片 s2 := a[3:6]，相应示意图如下



3. 切片的长度和容量

- 切片拥有自己的长度和容量，我们可以通过使用内置的 len()函数求长度，使用内置的 cap()函数求切片的容量。
- 切片的长度就是它所包含的元素个数。
- 切片的容量是从它的第一个元素开始数，到其底层数组元素末尾的个数。
- 切片 s 的长度和容量可通过表达式 len(s) 和 cap(s) 来获取

```
package main

import "fmt"

func main(){
  a := [8]int{0, 1, 2, 3, 4, 5, 6, 7}
  b := a[3:6]
  fmt.Printf("值=%d, 长度=%d, 容量=%d\n", b, len(b), cap(b)) //值=[3 4 5], 长度=3, 容量=5
  c := b[:cap(b)]
  fmt.Printf("值=%d, 长度=%d, 容量=%d", c, len(c), cap(c)) //值=[3 4 5 6 7], 长度=5, 容量=5
}
```

8.2 切片循环

- 切片的循环遍历和数组的循环遍历是一样的

1. 基本遍历

```go
package main
import "fmt"
func main() {
    var a = []string{"北京", "上海", "深圳"}
    for i := 0; i < len(a); i++ {
      fmt.Println(a[i])
    }
}
/*
北京
上海
深圳
*/
```

2. k, v遍历

```go
package main
import "fmt"
func main() {
    var a = []string{"北京", "上海", "深圳"}
    for index, value := range a {
      fmt.Println(index, value)
    }
}
/*
0 北京
1 上海
2 深圳
*/
```

8.3 append()

- Go 语言的内建函数 append()可以为切片动态添加元素，每个切片会指向一个底层数组
- 这个数组的容量够用就添加新增元素。
- 当底层数组不能容纳新增的元素时，切片就会自动按照一定的策略进行“扩容”，此时该切片指向的底层数组就会更换。
- “扩容”操作往往发生在append()函数调用时，所以我们通常都需要用原变量接收 append 函数的返回值

1. append添加

```
package main
import "fmt"
func main() {
    // append()添加元素和切片扩容
    var numSlice []int
    for i := 0; i < 10; i++ {
        numSlice = append(numSlice, i)
        fmt.Printf("%v len:%d cap:%d ptr:%p\n", numSlice, len(numSlice), cap(numSlice),
numSlice)
    }
}
```

2. append添加多个

```
package main
import "fmt"
func main() {
    var citySlice []string
    citySlice = append(citySlice, "北京") // 追加一个元素
    citySlice = append(citySlice, "上海", "广州", "深圳") // 追加多个元素
    a := []string{"成都", "重庆"}
    citySlice = append(citySlice, a...)    // 追加切片
    fmt.Println(citySlice) //[北京 上海 广州 深圳 成都 重庆]
}
```

3. 切片中删除元素

- Go 语言中并没有删除切片元素的专用方法，我们可以使用切片本身的特性来删除元素

```
package main
import "fmt"
func main() {
    a := []int{30, 31, 32, 33, 34, 35, 36, 37}
    a = append(a[:2], a[3:]...)     // 要删除索引为 2 的元素
    fmt.Println(a)          //[30 31 33 34 35 36 37]
}
```

4. 切片合并

```
package main
import "fmt"
func main() {
    arr1 := []int{2,7,1}
    arr2 := []int{5,9,3}
    fmt.Println(arr2,arr1)
    arr1 = append(arr1, arr2...)
    fmt.Println(arr1)  // [2 7 1 5 9 3]
}
```

九、Map

9.1 Map介绍

- map 是一种无序的基于 key-value 的数据结构，Go 语言中的 map 是引用类型，必须使用make初始化才能使用。
- Go 语言中 map 的定义语法如下： map[KeyType]ValueType
- 其中:
  - KeyType:表示键的类型。
  - ValueType:表示键对应的值的类型。
  - map 类型的变量默认初始值为 nil，需要使用 make()函数来分配内存。
- 其中 cap 表示 map 的容量，该参数虽然不是必须的。
- 注意：获取 map 的容量不能使用 cap，cap 返回的是数组切片分配的空间大小，根本不能用于map。
- 要获取 map 的容量，可以用 len 函数。

9.2 Map定义

```
package main
import (
  "fmt"
)
func main() {
    userInfo := map[string]string{
        "username": "root",
        "password": "123456",
    }
  fmt.Println(userInfo) // map[password:123456 username:IT 营小王子]
}
```

9.3 Map基本使用

1. 判断某个键是否存在

```
package main
import (
  "fmt"
)
func main() {
    userInfo := map[string]string{
        "username": "zhangsan",
        "password": "123456",
    }
  v, ok := userInfo["username"]
    if ok {
      fmt.Println(v)    // zhangsan
```

```
    }else {
      fmt.Println("map中没有此元素")
    }
}
```

2. delete()函数

- 使用 delete()内建函数从 map 中删除一组键值对，delete()函数的格式如下：delete(map 对象, key)
- 其中，
  - map 对象:表示要删除键值对的 map 对象
  - key:表示要删除的键值对的键

```
package main
import (
  "fmt"
)
func main() {
    userInfo := map[string]string{
        "username": "root",
        "password": "123456",
    }
    delete(userInfo, "password") //将 password从 map 中删除
    fmt.Println(userInfo)     // map[username:root]
}
```

9.4 map遍历

1. 遍历key和value

```
package main
import (
  "fmt"
)
func main() {
    scoreMap := map[string]int{
        "zhangsan": 24,
        "lisi":   26,
        "wangwu":  24,
    }
    for k, v := range scoreMap {
      fmt.Println(k, v)
    }
}
/*
zhangsan 24
lisi 26
wangwu 24
*/
```

2. 只遍历Key

- 注意： 遍历 map 时的元素顺序与添加键值对的顺序无关

```
package main
import (
  "fmt"
)
func main() {
    scoreMap := map[string]int{
        "zhangsan": 24,
        "lisi":   26,
        "wangwu":  24,
    }
    for k := range scoreMap {
      fmt.Println(k)
    }
}
/*
zhangsan
lisi
wangwu
*/
```

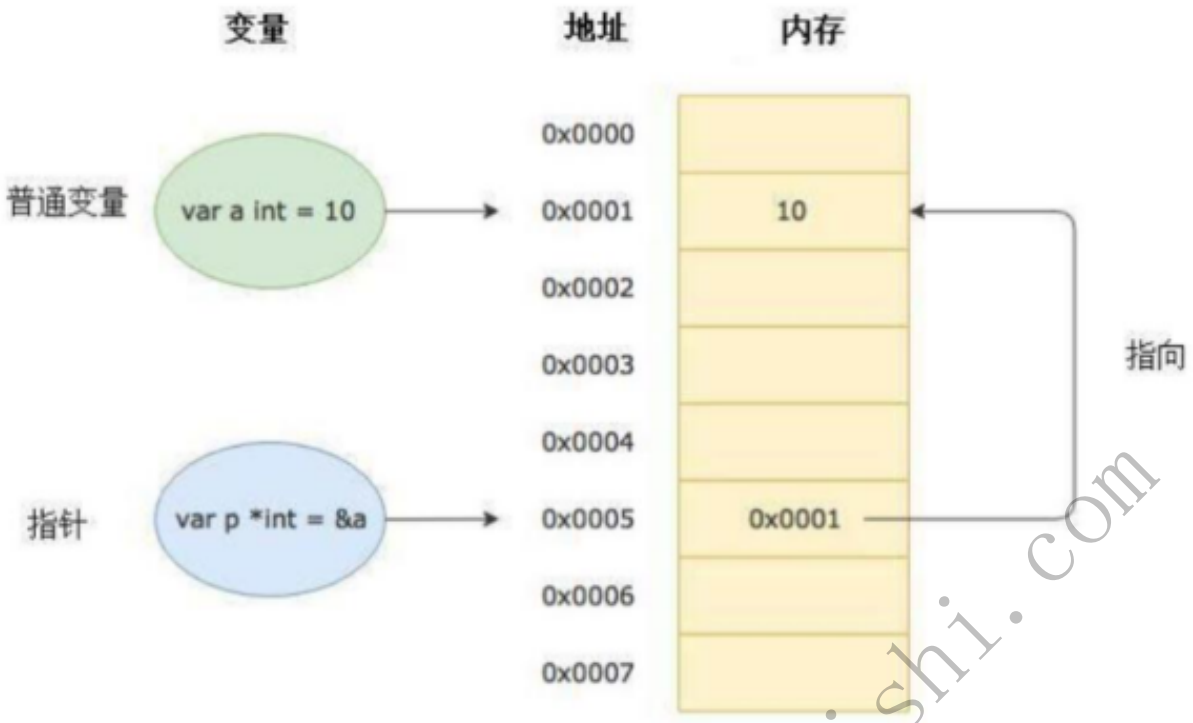

十、指针

10.1 关于指针

要搞明白 Go 语言中的指针需要先知道 3 个概念： 指针地址、指针类型、指针取值

- 指针地址 (&a)
- 指针取值 (*&a)
- 指针类型 (&a) -> *int 改变数据传指针
- 变量的本质是给存储数据的内存地址起了一个好记的别名。比如我们定义了一个变量 a := 10 ,这个时候可以直接通过 a 这个变量来读取内存中保存的 10 这个值。
- 在计算机底层 a 这个变量其实对应了一个内存地址。
- 指针也是一个变量，但它是一种特殊的变量，它存储的数据不是一个普通的值，而是另一个变量的内存地址。
- Go 语言中的指针操作非常简单，我们只需要记住两个符号：&（取地址）和 *（根据地址取值

```
package main
import "fmt"
func main() {
    var a = 10
    fmt.Printf("%d \n",&a)      // &a 指针地址 (824633761976)
    fmt.Printf("%d \n",*&a)     // *&a 指针取值 (10)
    fmt.Printf("%T \n",&a)      // %T 指针类型 (*int )
}
```

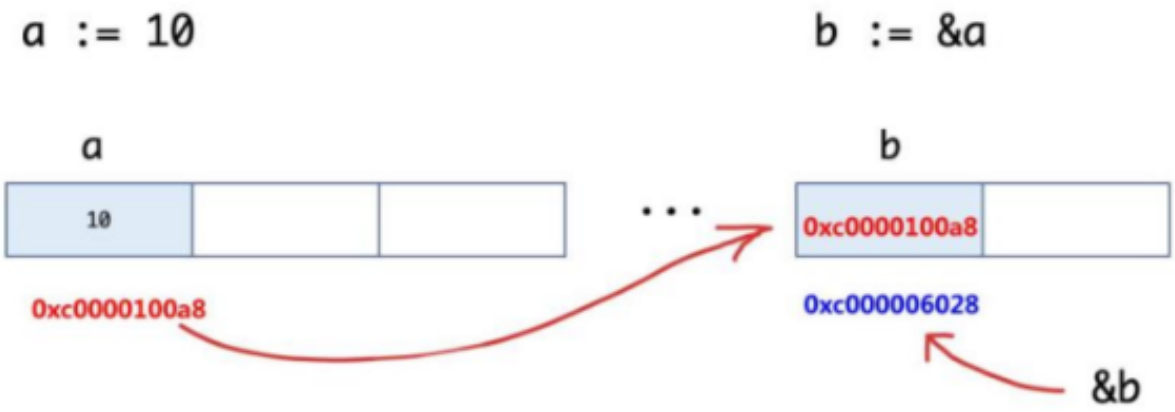


10.2 &取变量地址

1. &符号取地址操作

```
package main
import "fmt"
func main() {
    var a = 10
    var b = &a
    var c = *&a
    fmt.Println(a) // 10 a的值
    fmt.Println(b) // 0xc00001e060 a变量的内存地址
    fmt.Println(c) // 10 *内存地址 取值
}
```

2. b := &a 的图示



10.3 new 和 make

1. 执行报错

- 执行下面的代码会引发 panic，为什么呢?
- 在 Go 语言中对于引用类型的变量，我们在使用的时候不仅要声明它，还要为它分配内存空间，否则我们的值就没办法存储。
- 而对于值类型的声明不需要分配内存空间，是因为它们在声明的时候已经默认分配好了内存空间。
- 要分配内存，就引出来今天的 new 和 make。
- Go 语言中 new 和 make 是内建的两个函数，主要用来分配内存。

```
package main
import "fmt"
func main() {
    var userinfo map[string]string
    userinfo["username"] = "张三"
    fmt.Println(userinfo)
}
/*
panic: assignment to entry in nil map
*/
```

2. make和new比较

- new 和 make 是两个内置函数，主要用来创建并分配类型的内存。
- make和new区别
  - make 关键字的作用是创建于 slice、map 和 channel 等内置的数据结构
  - new 的作用是为类型申请一片内存空间，并返回指向这片内存的指针

```go
package main
import "fmt"
func main() {
    a := make([]int, 3, 10)    // 切片长度为 1，预留空间长度为 10
    a = append(a,1)
    fmt.Printf("%v--%T \n",a,a)  // [0 0 0]--[]int  值----切片本身
    var b = new([]int)
    //b = b.append(b,2)      // 返回的是内存指针，所以不能直接 append
    *b = append(*b, 3)      // 必须通过 * 指针取值，才能进行 append 添加
    fmt.Printf("%v--%T",b,b)    // &[]--*[]string 内存的指针---内存指针
}
```

3. new函数

- 系统默认的数据类型，分配空间

```go
package main
import "fmt"
func main() {
    // 1.new实例化int
    age := new(int)
    *age = 1

    // 2.new实例化切片
    li := new([]int)
    *li = append(*li, 1)

    // 3.实例化map
    userinfo := new(map[string]string)
    *userinfo = map[string]string{}
    (*userinfo)["username"] = "张三"
    fmt.Println(userinfo)   // &map[username:张三]
}
```

- 自定义类型使用 new 函数来分配空间

```go
package main
import "fmt"
func main() {
    var s *Student
    s = new(Student)     //分配空间
    s.name ="zhangsan"
    fmt.Println(s)      // &{zhangsan 0}
}
type Student struct {
    name string
    age int
}
```

4. make函数

- make 也是用于内存分配的，但是和 new 不同，它只用于 chan、map 以及 slice 的内存创建而且它返回的类型就是这三个类型本身，而不是他们的指针类型。
- 因为这三种类型就是引用类型，所以就没有必要返回他们的指针了

```
package main
import "fmt"
func main() {
    a := make([]int, 3, 10)    // 切片长度为 1，预留空间长度为 10
    b := make(map[string]string)
    c := make(chan int, 1)
    fmt.Println(a,b,c)      // [0 0 0] map[] 0xc0000180e0
}
```

- 当我们为slice分配内存的时候，应当尽量预估到slice可能的最大长度
- 通过给make传第三个参数的方式来给slice预留好内存空间
- 这样可以避免二次分配内存带来的开销，大大提高程序的性能。