优点：既能达到静态编译的安全和性能，又达到了动态语言开发维护的高效率

1. 继承了C的优点：表达式，语法，指针
2. 垃圾回收机制，内存自动回收，不需开发人员管理
3. 天然并发：从语言层面支持并发；goroutine轻量级线程高效利用多核；基于CPS并发模型
4. 吸收了管道通信机制：通过管道channel实现不同的goroute之间通信
5. 函数可以返回多个值
6. 创新：切片slice，延时执行defer

Go build编译

Go run 编译运行

注意事项

1. Main()为入口
2. 严格区分大小写
3. 行编译，一行只能一条语句
4. Go定义的变量必须使用到，import的包也必须使用到

转义符

1. \t 等于tab 制表位
2. \n 换行符
3. \\ 一个\
4. \” 一个“
5. \r 从字符串起始开始覆盖相应字符

注释

行注释//文字

块注释/\*文字\*/

Ctrl+/+选中可以整体注释

Tab可以实现缩进

Shift+tab 向左移动

标准库文档：

StudyGolang.com/pkgdoc

调用函数的方式：Import包

使用包中的函数：包名.函数名

变量的使用：声明变量（定义变量）、赋值、使用

变量表示内存中的一个存储区域，该区域有自己的名称（变量名）和类型（数据类型）

变量三种使用方式：

1. 基本语法： var 变量名 数据类型 var i int

指定变量类型，声明后若不赋值，使用默认值（int默认0）

1. 根据值自行判定变量类型（类型推导） var num = 10.11
2. 省略var 用**:=** 且左侧的变量不应是之前声明过的 num:= 10.11

多变量声明：

var n1, n2, n3 int 表示n1=n2=n3=0

var n1, name, n3 = 100, “tom”, 888 表示n1=100 name=tom n3=888

n1, name, n3 **:=** 100, “tom”, 888 表示n1=100 name=tom n3=888

一次性声明多个全局变量：

var n1 = 100

var n2 = 200

var name = “jack”

或者：

Var (

N3 = 300

N4 = 900

Name2 = “mary”

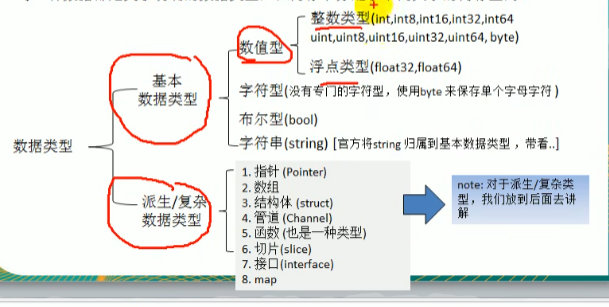
)

注意事项：

1. 该区域的数据值可以在同一类型范围内不断变化
2. 变量在同一个作用域（在一个函数或者代码块）内不能重名
3. 变量=变量名+值+数据类型
4. 没赋值的变量为默认值，int为0，string为空串

+号：

1. 两边都是数值做加法运算
2. 两边都是字符串做字符串拼接





Bit是最小存储单位，byte是基本存储单元，1byte=8bit

整形使用细节：

1. 分有符号和无符号，int uint的大小和系统有关
2. 整形默认声明为int型
3. 在程序中查看变量的字节大小和数据类型：fmt Printf (“x 的类型是 %T 字节数%d”, x, unsafe.Sizeof(x)
4. 保证程序运行正确的前提下占用空间小的数据类型

一次性引入多个包的写法：

Import(

“fmt”

“unsafe”

…

)

浮点数有符号 float 尾数可能有精度损失

Shift+alt+向下箭头=复制当前行

Go的字符串是由字节组成的

**%t输出类型(true or false) %d输出整数 %c输出字符 %f输出小数 %v输出原值 %q输出带引号的字符串**

Bool占用一个字节

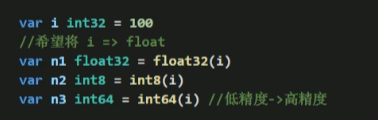
Go中字符串一旦赋值不可更改

反引号可直接引用大量字符串··

字符串拼接的加号留在上面一行

基本数据类型转string：ftm.Sprintf(数据类型，被转换的变量)

数据类型转换 T(v) 将v值转换为T

  
被转换的是一个副本，本身的类型没被转换，只转换n1.n2.n3

Printf是C++那样的输出 println是直接输出

获取地址的语法：&

指针类型的变量声明：Var ptr \*int = &i ptr存是i的地址

取ptr指向的值，也就是i的值：\*ptr

\_下划线是空标识符

首字母小写是私有的，大写是共有的变量

A/B得到小数还是整数取决于AB是不是整数

A%B=a-a/b\*b

++ --只能独立使用 只有后置没有前置

Go无三元运算符，改用IF ELSE实现

获取输入Scanln(&…) Scanf(型号)

布尔值 %t

十进制整数 %d

字符 %c

小数 %f

默认格式 %v

原码 反码（符号位0为正1为负） 补码（反码+1）

计算机以补码二进制运算

异或就是相同就取0不同取1

单分支和双分支 if语句都只会执行一个代码块

Else不能换行

Switch +值/表达式{

Case+值/表达式

后面跟的类型要一致

Break终止循环

Continue结束本轮循环开启下轮循环，可跳过指定label循环

不推荐用goto 容易造成流程混乱

Return跳出方法、函数

Fallthrough 穿透 继续执行

Func (函数名)(返回值列表){

执行语句

Return 返回值列表

}

返回值列表如果只有一个则不用括号，有多个必须用，隔开并且打括号，不想返回的用\_忽略

基本数据类型分配到栈区 调用函数的时候为其分配一个新的栈区 编译器自动区分其区域 函数执行完毕后该栈区自动被回收 如有return则直接返回return

引用数据类型到堆区

代码区

编译器存在逃逸分析

基本数据类型、数组都是值传递，只能浅拷贝

如果要深拷贝可以用&引用拷贝 main函数传入地址&，函数内用引用\*

函数也可作为形参

Type xxx int 自定义数据类型 但不完全一样

可变参数用args...

初始化函数用init

执行顺序 全局变量——init函数——main函数

**匿名函数：只使用一次的函数**

直接调用的话在反花括号后写形参，用变量调用的话在前面定义就好

全局匿名函数：首字母必须大写

闭包：返回的匿名函数和这个匿名函数引用的变量形成的整体，该变量不会被每次初始化

,用闭包的好处就是可以一直使用这个变量（累加器）

Defer：用defer栈（独立的栈）来延时执行（等defer处于的函数执行完毕之后先入后出）

栈的值同时会被压入，不会被改变  
主要用于及时释放函数创建的资源（打开了文件，获取了数据库链接，锁资源）

值类型（栈）：基本数据类型int，float，bool，string，struct

引用类型（堆）：指针，slice，map，chan，interface

赋值语句不能在函数体外

字符串遍历 r:=[]rune(str)

Go字符串函数：

1. 统计字符串长度。Len(str)
2. 字符串遍历。r:=[]rune(str)
3. 字符串转整数。N,err:=strconv.Atoi(“12”)
4. 整数转字符串 str=strconv.Itoa(12345)
5. 字符串转[]byte。Var bytes=[]byte(“hello go”)
6. []byte转字符串str=string([]byte{97,98,99})
7. 10进制转2，8，16进制。Str=strconv.FormatInt(123,2)
8. 查找子串是否在指定的字符串中 strings.Contains(“seafood“,”foo”\_)
9. 统计一个字符串有几个指定的字串 strings.Count(“ceheese”,”e“)
10. 不区分大小写的字符串比较（==是区分大小写的）fmt.Println(stings.EqualFold(“abc”,”Abc”)
11. 返回子串在字符串第一次出现的index值strings.Index(“nlt\_abc”,abc)
12. 返回字串最后一次出现的index。Strings.LastIndex()
13. 将指定的字串替换成另一个字串strings.Replace
14. 按照指定的某个字符，为分割表示，将字符串拆分成字符串组strings.Split
15. 字符串大小写转换 strings.ToLower strings.ToUpper
16. 将字符串左右两边的空格去掉 strings.TrimSpace
17. 将字符串左右两边指定的字符去掉 strings.Trim
18. 将字符串左边指定的字符去掉strings.TrimLeft
19. 将字符串右边指定的字符去掉strings.Trimright
20. 判断字符串是否以指定的字符串开头 strings.HasPrefix
21. 判断字符串是否以指定的字符串结束strings.HasSuffix

时间函数 now:=time.Now()

Sleep休眠函数 time.Sleep()

Time的Unix和UnixNano：

Time.Now().Unix Time.Now().UnixNano

**Go错误处理机制：抛出一个panic异常，defer中通过recover捕获**

defer func(){

if err:=recover(); err!=nil{

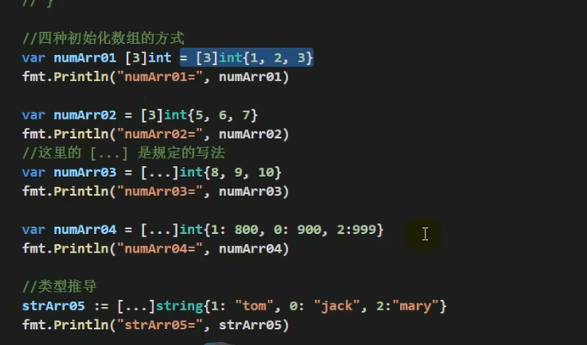
fmt(发生错误)

}()

Erros.new和panic内置函数来自定义错误

Int默认是64 占8个字节 也就是地址+8 int32占4

数组声明



For range遍历数组

Go的数组是值类型，值传递，值拷贝，数组之间互不影响

如果要在其它函数修改原来的数组，使用引用方式。

For 下标i,值v := range xxx

**切片slice**  
使用切片三种方式：

1. 从数组中引用 var slice=arr[x,x]
2. Make: var 切片名 []type=make([],len,[cap])
3. 定义+指定具体数组： var slice []int=[]int{x,x,x}

切片可由for和for range2种方式遍历

切片append 程序员不可见 底层创建一个新数组newArr 将slice原来包含的元素拷贝到新数组

String不可变，底层是一个byte数组，可以切片处理

如果要修改string，可以用[]rune转成切片，修改后重写成string

**Map （引用类型）**

声明+使用，只声明没办法用，必须make分配内存赋值后才能用

Var 变量名map[key][value]

常用：Map := make(map[int]int)

Map增删查：

增：直接加或者修改

删：delete某个，如果要删整个就直接make一个新的

查：val,finRes=map[]

Map只能用for range遍历

Map切片配合append使用

Map排序1map的key放入切片中2对切片排序3遍历切片按照key输出map

Map是引用类型 会修改原值

Map可以自动扩容

Map中的嵌套map要记得make

**Go的结构体是值类型**

type 名 类型

声明时的大小写和变量作用域一致

声明：var person Person

或者 var person Person=Person{}

Var person \*Person =new(Person)

Var person \*Person =&Person{}

可以直接用 结构体指针.字段名（底层自动转换）

结构体内存连续分布

结构体转换类型必须字段完全一致

结构体type重新定义取别名后是新的数据类型，不能直接=，但是可以带括号强转  
结构体字段可以带tag，通过反射机制获取，可用于序列化和反序列化

**方法**

作用于某个数据类型，自定义类型都可以有方法，并且方法是和数据类型绑定的，只能该数据类型调用

变量调用方法调用机制与函数一致

不同的是，调用时其变量也会作为一个参数传递给主方法（如果变量是值类型，则值拷贝，如果变量是引用类型，则地址拷贝）

方法的声明：

Func(receiver type)methodName(参数列表)(返回值列表){

方法体

Return返回值

}

方法中修改结构体的变量的值，一般让方法与结构体绑定

工厂模式的构造函数

Func 变量名（传参）\*结构体名{

Return &结构体名{

}

}

结构体可以访问匿名结构体所有的字段和方法

Golang通过匿名结构体实现继承

匿名结构体和结构体对于相同名字的方法是就近原则使用

嵌入两个匿名结构体时，访问方法必须带匿名结构体名字

多态通过接口interface实现（高内聚，低耦合）

必须全部实现接口内的方法才能使用接口

接口里只能声明方法

接口是一个指针（引用类型）

任何变量都可以赋给空接口

接口方法体不能有两个一样名字的方法

定义的时候用\* 则传入的时候用&

继承的价值：解决代码的复用性和可维护性

接口的价值：设计，设计各种方法和规范，让其他自定义类型去实现这些方法

接口更灵活，方便，实现代码解耦，继承是is-a，接口是like-a

接口体现多态的两种形式：多态参数、多态数组

**类型断言assert：之前空指针赋给一个变量后 重新使用变量.(变量之前的类型)**

If phone,ok :=usb.(phone) ; ok{

Call()

}

(items…interface{})

Switch x.(type)

File是一个指针

Go默认缓冲区是4096

Io.EOF表示文件末尾

Reader :=bufio.NewReader(file)

Str,err:=reader.ReaderString(‘\n’)

带缓冲的读取：os.Open, file.Close bufio.NewReader(file) reader.ReaderString

一次性读取：ioutil.ReadFile (content) 因为没有打开所以不用关闭 open和close被封装到函数内部了 适合打开一个小文件

Os.Open(filepath,读写操作,权限（windows可随意）)

Bufio.NewWriter调用WriterString方法时内容先写入缓存，需要调Flush方法真正写入文件

os.Args()获取命令行参数

json序列化：json.Marshal(指针类型)

data,err:= json.Marshal(指针类型)

json反序列化：err:=json.UnmMareshal([]byte(str),指针类型)

反序列化数据类型前后保持一致

Goroutine协程（协程是轻量级线程）

单核CPU进行多个线程叫并发（微观上只有一个任务运行）

多核CPU进行多个线程叫并行

协程有独立栈，共享堆空间

Go func()

Goroutine的调度模型是MPG M是内核OS线程，G是协程,P是上下文

设置运行CPU runtime.NumCPU()

**Channel管道（引用类型）**

是一个队列，默认可读可写。

只写Var 变量名 chan -< 数据类型

只读 Var 变量名 -< chan 数据类型

线程安全，多个协程操作同一个管道时不会发生资源竞争问题

Var 变量名 chan 数据类型 或者 变量名:=make(chan 数据类型，容量)

必须make了才能使用

管道的值是一个指向管道的地址

写入数据的方式：intChan<- num

读取数据 num=<-intChan

如果要存任意类型 可以传空接口

Channel关闭后只能读，不能写 Close(管道名)

遍历用 for range（遍历时，channel必须关闭，没关闭遍历会出现死锁）

管道只有写没有读就会阻塞，如果即使容量不够，只要有读就不会阻塞

Select解决从管道取数据阻塞的问题

select {

case v :=<-管道名:

}

**反射**

Reflect.TypeOf(变量名) 获取变量的类型

Refelct.ValueOf(变量名) 获取变量的值

变量、interface{}、reflect.Value可以相互转换

**网络编程**

TCP socket，基于tcp/ip协议。

b/s结构的http协议，http底层由tcp socket实现

0-1024是固定端口 一般不用

**Redis**

Redisdoc.com

对string：

添加key-val：set

查看当前所有key：keys \*

获取key对应val：get key

切换redis数据库 select index

查看当前数据库key-val数量：dbsize

清空当前和清空所有：flushdb flushall

String最大512M，一个key对应一个val

对Hash：

键值对集合，适合存储对象，是一个string类型的field和val映射表

CRUD：hset/hget/hgetall/hdel

一次性设置多个：hmset/hmget

对List：

类似于一个左右都是开口的管道

CRUD：lpush/rpush/lrange/lpop/rpop/del/llen key

对Set：

String类型的无序集合，元素值不重复

CRUD：sadd/smembers（取所有值）/sismember（判断是否存在）/srem（删除）

连接池：

Var pool \*redis.pool

Pool=&redis.pool{

MaxIdle:8//最大空闲连接数

MaxActive：0//表示和数据的最大连接数，0表示无限制

Idletimeout：100//最大空闲时间

Dail：func（）（redis.Conn, error）{ //初始化连接的代码，链接哪个ip的redis

Return redis.Dial(“tcp”,”localhost:6379”)

}

}

C:=pool.get()//从连接池取出一个链接

Pool.Close()//关闭连接池

Docker

Docs.docker.com

Docker.easydoc.net/doc/81170005

K8s.easydoc.net

strings.Split(s, sep)

s 待分割的字符串 字符串类型的参数

sep 分隔符 字符串类型的参数

返回一个字符串切片。

Strconv.Atoi()

Atoi()函数用于**将字符串类型的整数转换为int类型**

Strconv.Itoa()

Itoa()函数用于**将int类型数据转换为对应的字符串**表示

bufio.NewReader 等函数实现快读

input:=bufio.NewScanner(os.Stdin)

input.Scan()获取输入

此时输入的文本为input.Text()

res := strings.Split(input.Text(), " ")//按空格分割

strings.ToLower()函数接收一个字符串，并将字符串中的每个ASCII字符转换成小写，最后将新的字符串返回。

sort.Strings(input)//按首字母排序

Fmt.SCAN（）要传地址