# 1 介绍

Google

编译型语言

二十一世纪的C语言

支持交叉编译，编译速度快

开发效率高

执行性能好

天生支持高并发

跨平台编译（windows平台的源码经过跨平台编译生成linux环境执行文件）

## 1.1 应用

服务端开发：

日志处理

文件系统

监控系统

容器虚拟化技术：

Docker

K8s

Docker Swarm

存储：

Etcd

TDB

Groupcache

Web开发：

Net/http

Gin

Echo

区块链：

以太坊

Fabric

## 1.2 学习方法

对比学习

多写代码

实际工作中使用

自主学习的能力

## 1.3 hello world

1.14版本后，采用go mod管理依赖环境。代码不再强制必须写在GOPATH下。

修改GOPROXY

go env -w GOPROXY=https://goproxy.cn,direct

```golang

package main  // 声明 main 包，表明当前是一个可执行程序

import "fmt"  // 导入内置 fmt 包

func main(){  // main函数，是程序执行的入口

fmt.Println("Hello World!")  // 在终端打印 Hello World!

}

```

直接运行：

go run hello.go

go build 项目名

go build hello

go编译器会去 GOPATH的src目录下找要编译的hello项目

go build -o heiheihei.exe

Windows下编译Mac平台64位可执行程序：

SET CGO\_ENABLED=0  // 禁用CGO，使用了cgo的代码是不支持跨平台编译的

SET GOOS=linux  // 目标平台是linux

SET GOARCH=amd64  // 目标处理器架构是amd64

go build

Linux 下编译 Mac 和 Windows 平台64位可执行程序：

CGO\_ENABLED=0 GOOS=darwin GOARCH=amd64 go build

CGO\_ENABLED=0 GOOS=windows GOARCH=amd64 go build

# 2 变量和常量

变量命名规则和java一致。

变量必须经过声明才能开始使用。

变量声明后必须使用。

var 变量名 变量类型

var name string

var age int

var isOk bool

批量声明

var (

    a string

    b int

    c bool

    d float32

)

变量的初始化：

变量声明后，会自动对变量的内存区域进行初始化操作。

int，float --> 0

string --> ''

bool --> false

slice, func, 指针 --> nil

var 变量名 类型 = 表达式

一次初始化多个变量：

var name, age = "Q1mi", 20

类型推导：

var name = "Q1mi"

短变量声明：

m := 200

仅在函数内部使用

匿名变量：

下划线

匿名变量不占用命名空间，不会分配内存。

注意事项：

函数外的每个语句都必须以关键字开始（var、const、func等）

:=不能使用在函数外。

\_多用于占位，表示忽略值。

常量在定义的时候必须赋值。

const pi = 3.1415

const e = 2.7182

多个常量一起声明：

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

)

iota是go语言的常量计数器，只能在常量的表达式中使用。

iota在const关键字出现时将被重置为0。const中每新增一行常量声明将使iota计数一次(iota可理解为const语句块中的行索引)。 使用iota能简化定义，在定义枚举时很有用。

```golang

package main

import "fmt"

// 函数外的每个语句都必须以关键字开始（var、const、func等）

// var a int

// a = 1000

// :=不能使用在函数外

// b := 100

// 全局变量m

var m = 100

// 常量定义

const e = 2.7182

const (

    pi = 3.1415

    e1 = 2.7182

)

const (

    n11 = 100

    n2

    n3

)

func main() {

    fmt.Println(m)

    // 变量必须经过声明才能开始使用。

    // fmt.Print(b)

    // 变量声明后必须使用。

    // var a;

    // fmt.Println(a)

    var a = 1

    fmt.Println(a)

    // 类型推导

    var a1 = 2

    fmt.Println(a1)

    // 声明多个变量

    var b1, b2 = "b1", 222

    fmt.Println(b1, b2)

    fmt.Println(e)

    fmt.Println(pi)

    fmt.Println(e1)

    fmt.Println(n11)

    fmt.Println(n2)

    fmt.Println(n3)

    const (

        n1 = iota //0

        n2        //1

        n3        //2

        n4        //3

    )

    fmt.Println(n1)

    fmt.Println(n2)

    fmt.Println(n3)

    fmt.Println(n4)

    fmt.Println("================================")

    const (

        n12 = iota //0

        n22        //1

        \_

        n42 //3

    )

    fmt.Println(n12)

    fmt.Println(n22)

    fmt.Println(n42)

    fmt.Println("================================")

    const (

        n13 = iota //0

        n23 = 100  //100

        n33 = iota //2

        n43        //3

    )

    const n5 = iota //0

    fmt.Println(n13)

    fmt.Println(n23)

    fmt.Println(n33)

    fmt.Println(n43)

    fmt.Println(n5)

    const (

        \_  = iota

        KB = 1 << (10 \* iota)

        MB = 1 << (10 \* iota)

        GB = 1 << (10 \* iota)

        TB = 1 << (10 \* iota)

        PB = 1 << (10 \* iota)

    )

    fmt.Println(KB)

    fmt.Println(MB)

    fmt.Println(GB)

    fmt.Println(TB)

    fmt.Println(PB)

    const (

        aa, bb = iota + 1, iota + 2 //1,2

        c, d                        //2,3

        e, f                        //3,4

    )

    fmt.Println(aa, bb, c, d, e, f)

}

```

# 3 基本数据类型

基本的整型、浮点型、布尔型、字符串

数组、切片、结构体、函数、map、通道（channel）等。

整型分为以下两个大类： 按长度分为：int8、int16、int32、int64

对应的无符号整型：uint8、uint16、uint32、uint64

其中，uint8就是byte型，int16对应C语言中的short型，int64对应C语言中的long型。

uint8   无符号 8位整型 (0 到 255)

uint16  无符号 16位整型 (0 到 65535)

uint32  无符号 32位整型 (0 到 4294967295)

uint64  无符号 64位整型 (0 到 18446744073709551615)

int8    有符号 8位整型 (-128 到 127)

int16   有符号 16位整型 (-32768 到 32767)

int32   有符号 32位整型 (-2147483648 到 2147483647)

int64   有符号 64位整型 (-9223372036854775808 到 9223372036854775807)

特殊整型

uint    32位操作系统上就是uint32，64位操作系统上就是uint64

int 32位操作系统上就是int32，64位操作系统上就是int64

uintptr 无符号整型，用于存放一个指针

注意：

在使用int和 uint类型时，不能假定它是32位或64位的整型，而是考虑int和uint可能在不同平台上的差异。

注意事项：

获取对象的长度的内建len()函数返回的长度可以根据不同平台的字节长度进行变化。

实际使用中，切片或 map 的元素数量等都可以用int来表示。

在涉及到二进制传输、读写文件的结构描述时，为了保持文件的结构不会受到不同编译目标平台字节长度的影响，不要使用int和 uint。

数字字面量语法：

以二进制、八进制或十六进制浮点数的格式定义数字

v := 123\_456 // 123456

两种浮点型数：float32和float64

math.MaxFloat32

math.MaxFloat64

复数

complex64和complex128

复数有实部和虚部，complex64的实部和虚部为32位，complex128的实部和虚部为64位

不允许将整型强制转换为布尔型

布尔型无法参与数值运算，也无法与其他类型进行转换。

Go语言中的字符串以原生数据类型出现。

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。

字符串转义符

\r  回车符（返回行首）

\n  换行符（直接跳到下一行的同列位置）

\t  制表符

\'  单引号

\"  双引号

\\  反斜杠

多行字符串

反引号

字符串的常用操作：

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

字符用单引号

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。

当需要处理中文、日文或者其他复合字符时，则需要用到rune类型。rune类型实际是一个int32。