# 一 Go安装

<https://golang.org/dl/>

windows下不需要进行环境变量配置；

Linux下环境变量配置：

export GOROOT=$PATH:/path/to/go/

export PATH=$PATH:$GOROOT/bin/

export GOPATH=/home/user/project/go

调试工具delve安装：

<https://github.com/derekparker/delve/tree/master/Documentation/installation>

mac: brew install go-delve/delve/delve

linux&windows: go get github.com/derekparker/delve/cmd/dlv

命令： go version 查看版本

go run hello.go 运行软件

go build user 编译user文件夹为二进制文件

编译命令使用步骤：

1-为项目新建一个用户环境变量：GOPATH， 值为 项目根目录路径

2-项目下包含src文件，所有的go源码文件、文件夹都在此目录下

3-在项目根目录下运行 go build user，这个命令会自动寻找根目录下名称为 src 下的 user文件夹，并将user文件夹编译为二进制文件，并生成在根目录下：user.exe。

4-此时我们直接在命令行输入 user.exe 将会直接执行该文件，并得到结果。

# 二 Go语言特性

1. 垃圾回收

内存自动回收，再也不需要开发人员管理内存，开发人员专注业务实现，降低了心智负担，只需要new分配内存，不需要释放。

1. 天然并发

从语言层面支持并发，非常简单，goroute，轻量级线程，创建成千上万个goroute成为可能，基于CSP（Communicating Sequential Process）模型实现，

1. channel

管道，类似unix/linux中的pipe，多个goroute之间通过channel进行通信，支持任何类型

1. 多返回值

异步变成实例

**package** main  
  
**import** (  
 **"fmt"**)  
  
**func** main(){  
 **for** i := 0; i < 100; i++ {  
 *//不会执行到这个地方，因为main是主线程，内部的代码异步后，main函数直接执行完退出* **go** fmt.Print(i)  
 }  
}

# 三 helloworld

**package** main   
**import** (  
 **"fmt"**)  
**func** main(){  
 fmt.Println(**"hi"**)  
}

注意：Go文件是以.go为后缀。

package main main包用来将文件生成可执行程序，每个程序都只有一个main包

import 导入包 import(“fmt”); 通常写为案例中的样子：

import (

“fmt”

“os”

)

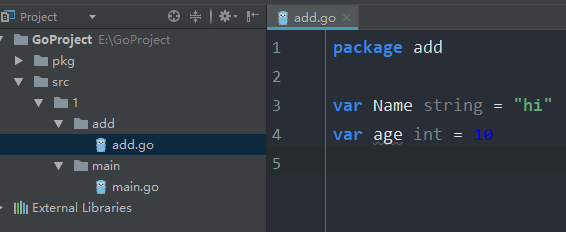
func main() func声明一个函数main，main被称为主函数，有且只有一个！

fmt.Println() 同一个包中的函数直接调用，不同包的函数，调用需要使用 包名.函数名

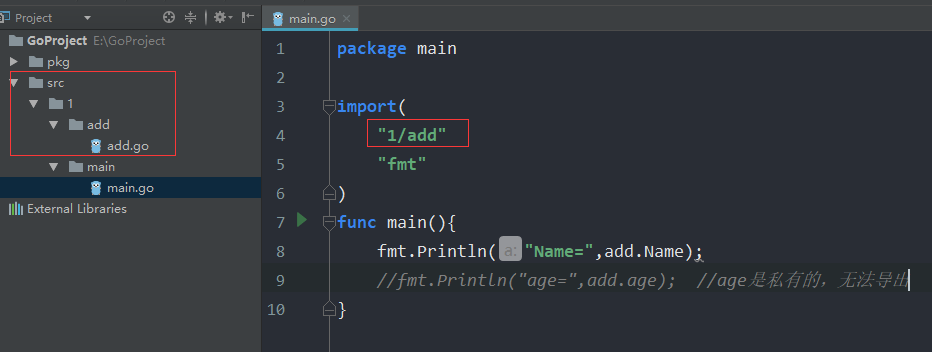
# 四 包

包的使用案例：

第一步：定义一个包add，且暴露出去一个属性Name



第二步：使用包，导入时是不把src包含在内的



包内如果没有main函数，那么编译后生成的是一个lib库，如果包内有main函数，那么编译成一个exe可执行程序。

包访问控制规则：

大写意味着这个函数/变量是可导出的

小写意味着这个函数/变量是私有的，包外部不能访问

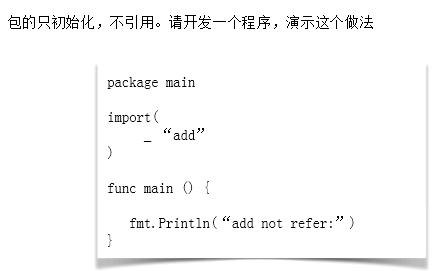
包的别名：

import(

a “1/add”

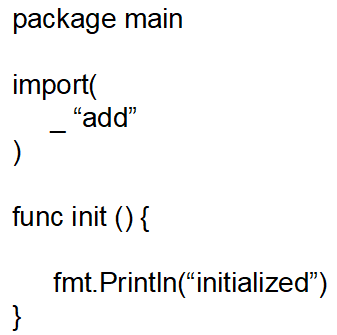
)

如果我们只需要导入一个包，但是不使用这个包，可以在包内这样处理：



# 五 init函数

每个源文件都可以包含一个init函数，这个init函数自动被go运行框架调用。



init函数会在main函数执行前执行，一般我们用这个函数来进行初始化。

# 五 数据

## 0 标识符

以字母或者下划线开头，大小写敏感。

\_ 是一个特殊的标识符，用来忽略结果，比如返回多个返回值，不需要的返回值使用 \_

## 1 常量

常量使用const 修饰，代表永远是只读的，不能修改。const 只能修饰boolean，number（int相关类型、浮点类型、complex）和string。

语法：const identifier [type] = value，其中type可以省略

普通写法：

const PI = 3.14

优雅写法：

const(

a = 0

b = 1

c = 2

)

专业写法：

const(

a = iota

b // 1 会自动+1

c // 2

)

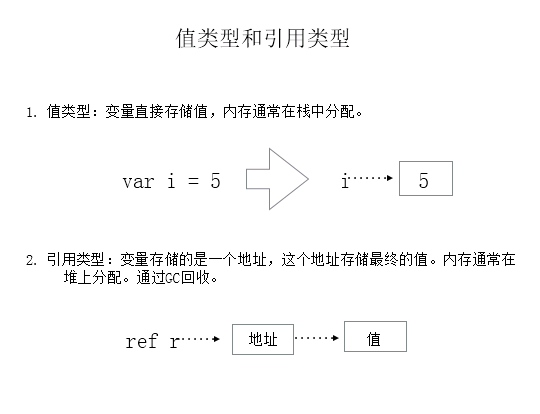
示例：

**package** main  
  
**import**(  
 **"fmt"  
 "time"**)  
**const**(  
 Man = 1  
 Feman = 2  
)  
**func** main(){  
 **for** {  
 second := time.Now().Unix()  
 **if**(second % Feman == 0){  
 fmt.Println(Man)  
 } **else** {  
 fmt.Println(Feman)  
 }  
 }  
}

## 2变量

**package** main  
  
**import "fmt"  
  
var**(  
a int *//默认为0*b string *//默认为""*c bool *//默认为false*d int = 8  
e string = **"hello world"**)  
  
**func** main(){  
 fmt.Print(a)  
}

## 3 值类型与引用类型



值类型：基本数据类型int、float、bool、string以及数组和struct。

引用类型：指针、slice、map、chan等都是引用类型。

bool类型，只能存true和false

数字类型，主要有int、int8、int16、int32、int64、uint8、uint16、uint32、uint64、float32、float64

字符串表示两种方式： 1）双引号 2）`` （反引号）

演示案例：

a := 100  
b := make(**chan** int,1)  
fmt.Println(**"a="**,a) *//100*fmt.Println(**"b="**,b) *//0xc04207c000*

演示指针案例

**package** main  
  
**import "fmt"  
func** modify1(a int){  
 a = 10  
}  
**func** modify2(a \*int){  
 \*a = 10  
}  
**func** main(){  
 a := 100  
 modify1(a)  
 fmt.Println(a) *//100* modify2(&a)  
 fmt.Println(a) *//10*}

&a 取出a的指针地址

\*int 接收变量的指针地址

# 六 函数

## 1 声明

函数声明格式： unc 函数名字 (参数列表) (返回值列表）{}

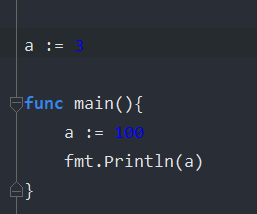
## 2 变量作用域

在函数内部声明的变量叫做局部变量，生命周期仅限于函数内部。

在函数外部声明的变量叫做全局变量，生命周期作用于整个包，如果是大写的，

则作用于整个程序。

在全局中，只能声明，不能赋值：

比如：这种写法是错误的，其等价于  
var a

a = 3

在全局中不能这样赋值，只能使用 var a = 3