go 语言学习第一天

作者：zouchanglin
原文链接：https://ld246.com/article/1580362887128
来源网站：链滴

Golang的优势和特点

- 可直接编译成机器码，不依赖其他库，glibc 的版本有一-定要求，部署就是扔一个文件上去就完成了
- 静态类型语言，但是有动态语言的感觉，静态类型的语言就是可以在编译的时候检查出来隐藏的大数问题，动态语言的感觉就是有很多的包可以使用，写起来的效率很高。
- 语言层面支持并发，这个就是Go最大的特色，天生的支持并发。Go就是基因里面支持的并发，可充分的利用多核，很容易的使用并发。
- 内置runtime，支持垃圾回收，这属于动态语言的特性之一吧，虽然目前来说GC(内存垃圾回收机制不算完美，但是足以应付我们所能遇到的大多数情况，特别是Go1.1之后的GC
- 简单易学，Go语言的作者都有C的基因，那么Go自然而然就有了C的基因，那么Go关键字是25个但是表达能力很强大，几乎支持大多数你在其他语言见过的特性:继承、重载、对象等
- 丰富的标准库，Go目前已经内置了大量的库，特别是网络库非常强大。
- 内置强大的工具，Go语言里面内置了很多工具链，最好的应该是gofmt工具，自动化格式化代码，够让团队review变得如此的简单，代码格式一模一样，想不一样都很困难
- 跨平台编译，如果你写的Go代码不包含cgo，那么就可以做到window系统编译linux的应用，如何到的呢? Go引用了plan9的代码，这就是不依赖系统的信息。
- 内嵌C支持，Go里面也可以直接包含C代码，利用现有的丰富的C库。

Go适合用来做什么

- 服务器编程，以前你如果使用C或者C++做的那些事情，用Go来做很合适，例如处理日志、数据打、虚拟机处理、文件系统等。
- 分布式系统，数据库代理器等。
- 网络编程，这一块目前应用最广，包括Web应用、API 应用、下载应用。
- 内存数据库，如google开发的groupcache, couchbase 的部分组件。

● 云平台，目前国外很多云平台在采用Go开发，CloudFoundy的部分组件，前VMare的技术总监自出来搞的apcera云平台。

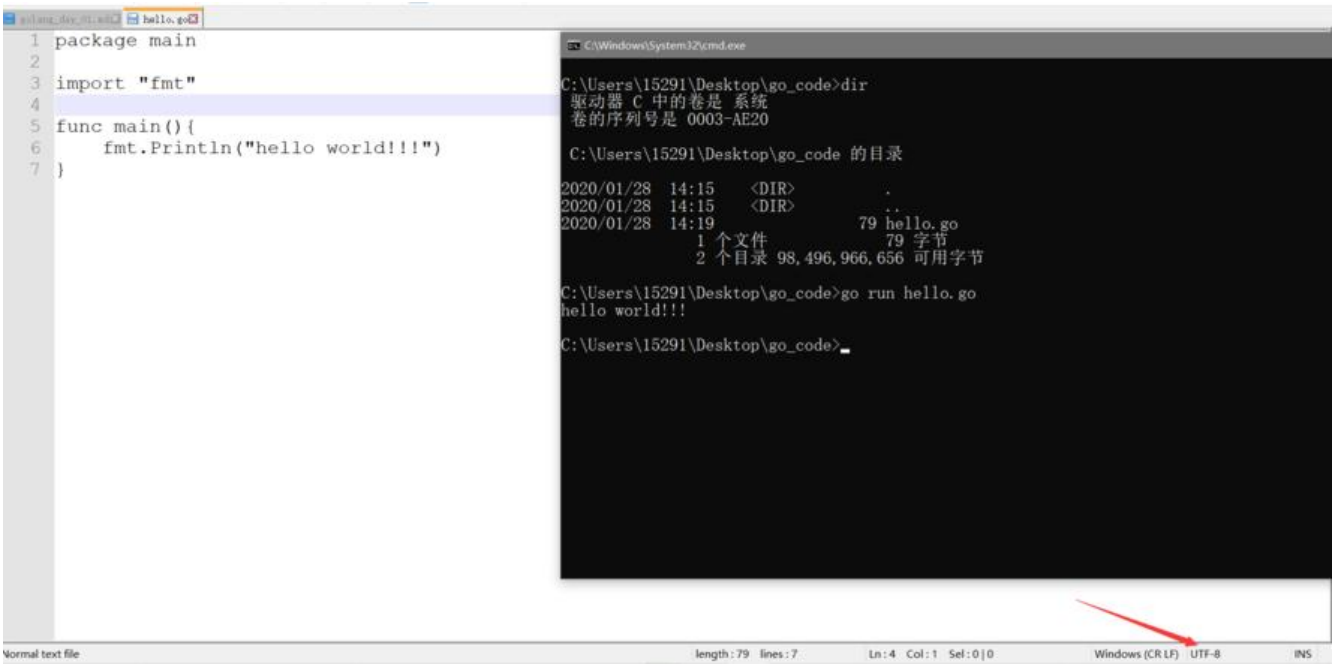
golang开发环境搭建

下载 go1.13.6.windows-amd64.msi 下载地址是https://dl.google.com/go/go1.13.6.windows-amd64.msi

一路Next安装别有中文路径即可

```
Microsoft Windows [版本 10.0.18363.592]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\15291>go version
go version go1.13.6 windows/amd64

C:\Users\15291>go env
set GO111MODULE=
set GOARCH=amd64
set GOBIN=
set GOCACHE=C:\Users\15291\AppData\Local\go-build
set GOENV=C:\Users\15291\AppData\Roaming\go\env
set GOEXE=.exe
set GOFLAGS=
set GOHOSTARCH=amd64
set GOHOSTOS=windows
set GONOPROXY=
set GONOSUMDB=
set GOOS=windows
set GOPATH=C:\Users\15291\go
set GOPRIVATE=
set GOPROXY=https://proxy.golang.org,direct
```

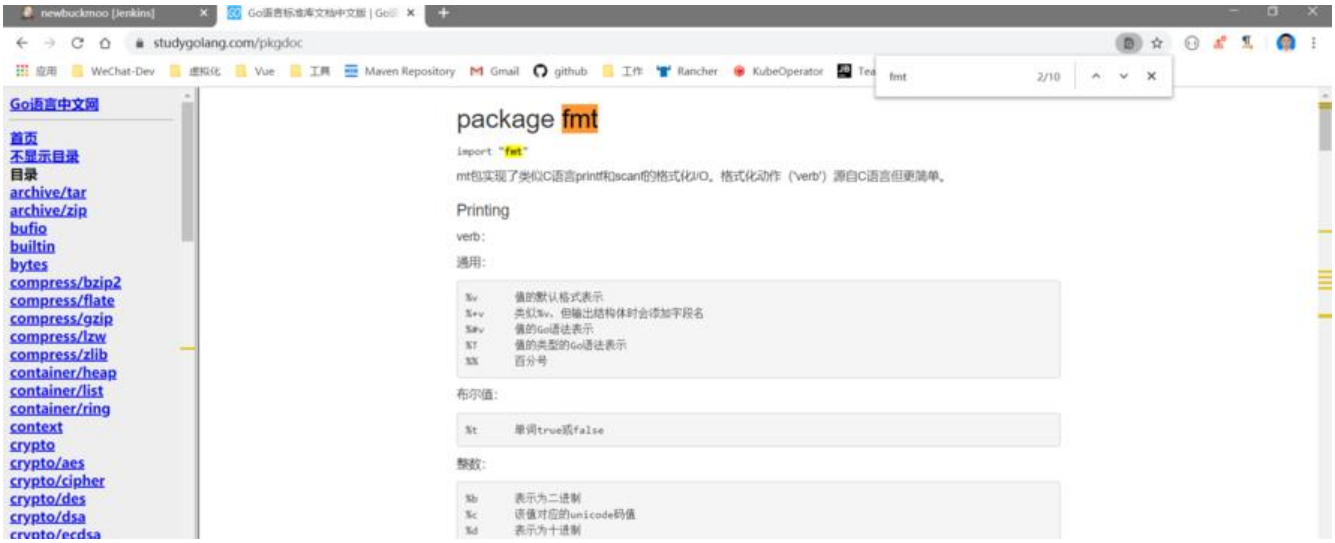
入门成功



另外还可以下载Go 的IDE，直接用LiteIDE就行了

https://studygolang.com/pkgdoc 这里是golang的文档

比如我们可以查找fmt包



golang简单语法

- 左括号和函数名称同行
- go语言以包作为管理单位，调用函数大部分需要导包
- 每个文件必须先声明包
- 程序必须有一个main包才能运行
- 注释和Java相同，　// 和/**/
- 导了包必须要使用，否则出错

golang的包管理方式

一个包（文件夹）下之恩那个有一个main函数，在IDE的情况下

如果是非要放在一个包，可以直接go run **.go

如果需要一个可执行程序，那么可以go build xxx.go

关键字和常量

Go语言中有25个关键字：

```
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
```

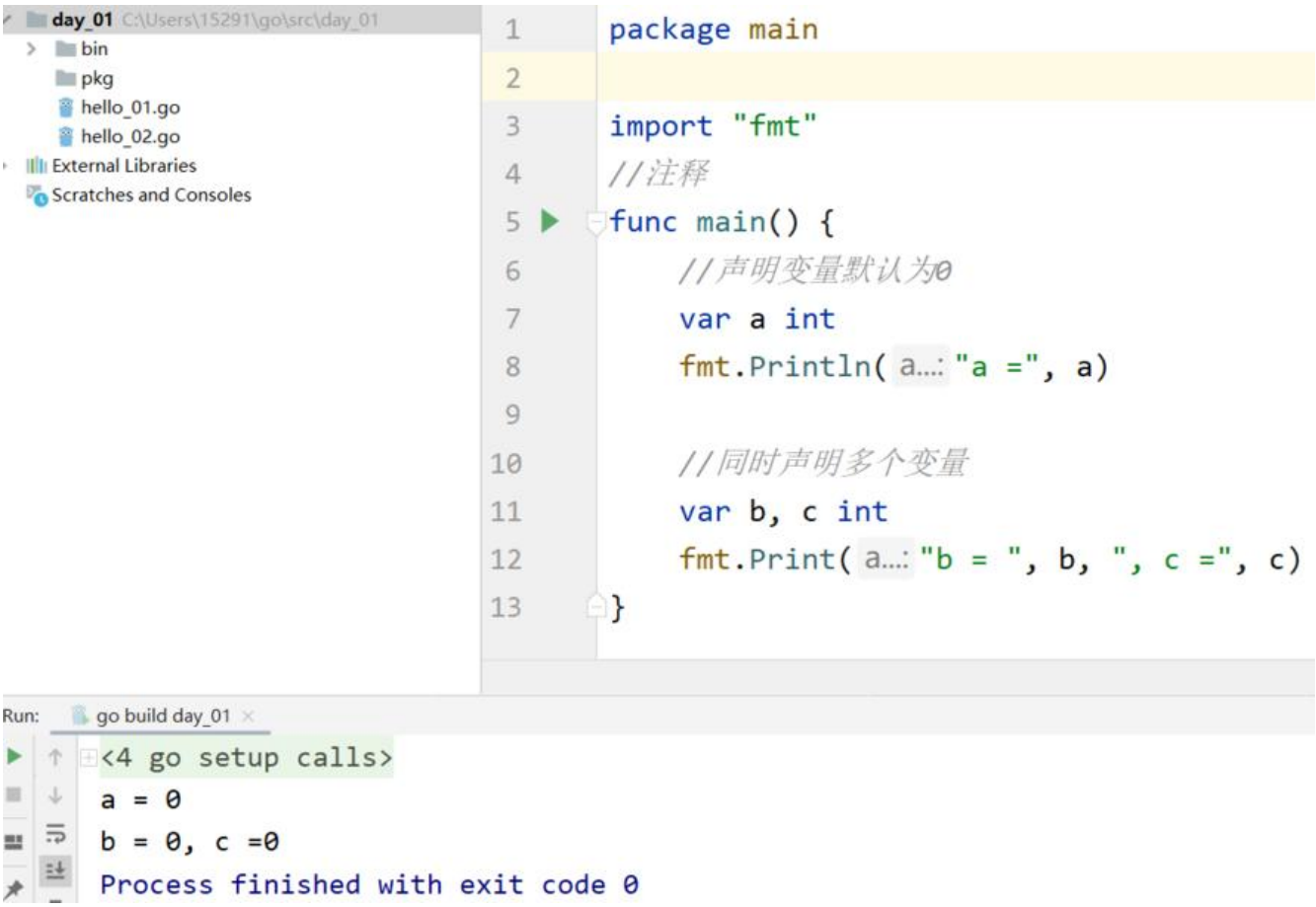
此外，Go语言中还有37个保留字。

```
Constants:    true  false  iota  nil                    内建常量

    Types:    int  int8  int16  int32  int64
              uint  uint8  uint16  uint32  uint64  uintptr   内建类型
              float32  float64  complex128  complex64
              bool  byte  rune  string  error

Functions:   make  len  cap  new  append  copy  close  delete
             complex  real  imag                            内建函数
             panic  recover
```

变量的使用

声明了变量必须要使用，只声明，没有初始化的变量默认为0

同一个{ }里，变量名是唯一的



直接看这段代码吧，比较好懂一点

```
package main

import "fmt"
//注释
func main() {
    //声明变量默认为0
    var a int
    fmt.Println("a =", a)

    //同时声明多个变量
    var b, c int
    fmt.Println("b =", b, ",c =", c)

    //声明时赋值
    var d int = 10
    fmt.Println("d =", d)

    //先声明，再赋值
    var e int
    e = 20
    fmt.Println("e =", e)

    //类型自动推导
    f := "I' am string"
    fmt.Println("f =", f)
```

```
    //%T用于打印变量的类型
    fmt.Println("Type is %T=", f)
    fmt.Printf("Type is %T\n", f)
    fmt.Printf("Type is %T\n", a)

    //类型自动推导只能用于初始化那一次
    g := 100
    fmt.Println("g =", g)

    //g:= "a str"     -> error
    //fmt.Println("g =", g)
    g = 200
    //g = "a str" 赋值时改变类型 -> error

    //和C语言的printf()一样的
    fmt.Printf("%d %d %s", a, g, f)
    //fmt.Println()只是简单的拼接，不能使用%T去打印类型之类的信息，但是fmt.Printf()却可以
}
```

下面是一个变量交换的例子

```
func main(){
    //-------------01----------
    //交换两个变量的值
    var a0 int = 10
    var b0 int = 20

    var tmp int = a0
    a0 = b0
    b0 = tmp
    fmt.Printf("a0=%d, b0=%d\n", a0, b0)

    //-------------02----------
    a1 := 10
    b1 := 20

    tmp1 := a1
    a1 = b1
    b1 = tmp1
    fmt.Printf("a1=%d, b1=%d\n", a1, b1)

    //-------------03----------
    a2, b2 := 10, 20
    a2, b2 = b2, a2
    fmt.Printf("a2=%d, b2=%d\n", a2, b2)

    //-------------04----------
    a3, b3, c3 := 10, 20, 30
    a3, b3, c3 = c3, a3, b3
    fmt.Printf("a3=%d, b3=%d, c3=%d\n", a3, b3, c3)
}
```

匿名变量

```go
//----------------------------------------
i := 10
j := 20
//匿名变量，丢弃数据不处理
tmp, _ = i, j
fmt.Printf( format: "i=%d, j=%d\n", i, j)

_ = 20
//匿名变量配置函数返回值使用才有优势
```

比如下面这样的

hello_03.go

```go
package main

import "fmt"

func test() (a, b, c, d int){
    return  a: 1, b: 2, c: 3,  d: 4
}

func main() {
    //比如你要拿到多个返回值中的两个
    _ , a, _ , d := test()
    fmt.Println( a...: "a =", a, "d =", d)
}
```

Run: go build day_01

```
<4 go setup calls>
a = 2 d = 4
```

常量的使用

变量声明为var、常量声明为const

```go
package main
import "fmt"

func main() {
    const a int = 10
    //a = 20 error
    fmt.Printf( format: "a = %d\n", a)

    //常量也支持类型推导，但是不是 :=
    const b = 20

    var c int   = 20
}
```

变量定义了必须使用

常量定义了可以不使用

注意常量的类型自动推导不能使用 :=

而且常量定义完了可以不使用，变量必须使用

多个变/常量的声明

```go
package main

import "fmt"

func main() {
    a, b := 10, 10.25

    fmt.Printf("a = %d, b = %f \n", a, b)
    fmt.Println("b =", b)

    //一次声明多个变量
    var (
        c int
        d float64
    )
    c = 10
    d = 99.99
    fmt.Printf("c = %d, d = %f \n", c, d)

    //一次声明多个变量并赋值
    var (
        e int = 10
        f float64 = 30.00
    )
    fmt.Printf("e = %d, f = %f \n", e, f)

    //一次声明多个常量
```

```
    const (
        g int = 10
        h int = 20
    )

    //一次声明多个常量并自动推导类型
    const (
        i = 10
        j = 20
    )
    fmt.Printf("g = %d, h = %d \n", g, h)
}
```

枚举的使用

```
package main

import "fmt"

func main() {

    //iota给常量赋值使用
    const (
        a = iota
        b = iota
        c = iota
    )

    fmt.Printf("a=%d, b=%d, c=%d", a, b, c) //a=0, b=1, c=2
}
```

iota 是一个常量自动生成器，每隔一行自动加一

iota遇到const，则重置为0

```
package main

import (
    "fmt"
)

func main() {

    const (
        a = iota
        b = iota
        c = iota
    )

    fmt.Printf("a=%d, b=%d, c=%d\n", a, b, c) // a=0, b=1, c=2

    //iota遇到const重置为0
    const d  = iota
    fmt.Printf("d=%d\n", d) // d=0
```

```
    //如果是同一行，值都一样
    const e, f, g = iota, iota, iota
    fmt.Printf("e=%d, f=%d, g=%d\n", e, f, g) // e=0, f=0, g=0

    const (
        h, i, j = iota, iota, iota
    )
    fmt.Printf("h=%d, i=%d, j=%d\n", h, i, j) // h=0, i=0, j=0

    const (
        k = iota
        l, m = iota, iota
        n = iota
    )
    fmt.Printf("k=%d, l=%d, m=%d, n=%d\n", k, l, m, n) // k=0, l=1, m=1, n=2
}
```

golang基本数据类型

Go 语言内置以下这些基础类型：

类型	名称	长度	零值	说明
bool	布尔类型	1	false	其值不为真即为家，不可以用数字代表 true 或 false
byte	字节型	1	0	uint8 别名
rune	字符类型	4	0	专用于存储 unicode 编码，等价于 uint32
int, uint	整型	4 或 8	0	32 位或 64 位
int8, uint8	整型	1	0	-128 ~ 127, 0 ~ 255
int16, uint16	整型	2	0	-32768 ~ 32767, 0 ~ 65535
int32, uint32	整型	4	0	-21 亿 ~21 亿, 0 ~ 42 亿
int64, uint64	整型	8	0	
float32	浮点型	4	0.0	小数位精确到 7 位
float64	浮点型	8	0.0	小数位精确到 15 位
complex64	复数类型	8		
complex128	复数类型	16		
uintptr	整型	4 或 8		足以存储指针的 uint32 或 uint64 整数
string	字符串		""	utf-8 字符串

```
package main

import "fmt"
```

```
func main() {

    //-----bool类型----------
    var a = false
    fmt.Println(a) //false

    b := true
    fmt.Println(b) //true

    var c bool
    fmt.Println(c) //false

    //-----浮点类型----------

    var d = 3.14 //这样赋值默认是float64
    fmt.Println(d)
    fmt.Printf("%T\n", d)

    var e float32
    fmt.Println(e)
    fmt.Printf("%T\n", e)

    f := 10.00 //这样赋值默认是float64
    fmt.Println(f)
    fmt.Printf("%T\n", f)

    //-----字符类型----------
    var a1 byte
    a1 = 'a'
    fmt.Println(a1) //97
    fmt.Printf("a1=%d,  a1=%c\n", a1, a1) //a1=97,  a1=a
    fmt.Printf("%T\n", a1)

    b1 := 'A'
    fmt.Printf("b1 = %c\n", b1)
    fmt.Printf("b1 = %c\n", b1 + ('a'-'A'))

    //-----字符串类型----------
    a2 := "Tim"
    fmt.Println(a2)
    // len()测字符串长度
    fmt.Println(len(a2))

    //打印字符串中的某一个字符
    fmt.Printf("a2[2] = %c\n", a2[2]) //a2[2] = m
    //fmt.Printf("a2[3] = %c", a2[3]) error 越界

    //-----复数类型----------
    a3 := 2.1 + 3i
    fmt.Println(a3) //(2.1+3i)

    var b3 complex128 = 2.5 + 3i
    fmt.Println(b3) //(2.5+3i)
```

```
    //通过内建函数取实部和虚部
    fmt.Println("实部 real(b3) =", real(b3), "虚部 imag(b3) =", imag(b3))
    //实部 real(b3) = 2.5 虚部 imag(b3) = 3
}
```

格式化输出

格式	含义
%%	一个%字面量
%b	一个二进制整数值(基数为 2)，或者是一个(高级的)用科学计数法表示的指数为 2 的浮点数
%c	字符型。可以把输入的数字按照 ASCII 码相应转换为对应的字符
%d	一个十进制数值(基数为 10)
%e	以科学记数法 e 表示的浮点数或者复数值
%E	以科学记数法 E 表示的浮点数或者复数值
%f	以标准记数法表示的浮点数或者复数值
%g	以%e 或者%f 表示的浮点数或者复数，任何一个都以最为紧凑的方式输出
%G	以%E 或者%f 表示的浮点数或者复数，任何一个都以最为紧凑的方式输出
%o	一个以八进制表示的数字(基数为 8)
%p	以十六进制(基数为 16)表示的一个值的地址，前缀为 0x,字母使用小写的 a-f 表示
%q	使用 Go 语法以及必须时使用转义，以双引号括起来的字符串或者字节切片

格式	含义
	[]byte，或者是以单引号括起来的数字
%s	字符串。输出字符串中的字符直至字符串中的空字符（字符串以'\0'结尾，这个'\0'即空字符）
%t	以 true 或者 false 输出的布尔值
%T	使用 Go 语法输出的值的类型
%U	一个用 Unicode 表示法表示的整型码点，默认值为 4 个数字字符
%v	使用默认格式输出的内置或者自定义类型的值，或者是使用其类型的 String() 方式输出的自定义值，如果该方法存在的话
%x	以十六进制表示的整型值(基数为十六)，数字 a-f 使用小写表示
%X	以十六进制表示的整型值(基数为十六)，数字 A-F 使用小写表示

键盘输入

```
package main

import "fmt"

func main() {
    var name string;
    fmt.Scanf("%s", &name) //手动输入格式
    fmt.Scan(&name) //自动匹配格式

    fmt.Printf("name=%s\n", name)
}
```

类型转换

Go语言中不允许隐式转换，所有类型转换必须显式声明，而且转换只能发生在两种相互兼容的类型间。

```
package main

import "fmt"

func main() {
    var a byte = 'A'

    //类型转换
    var b int = int(a)

    fmt.Printf("%T\n", a) //uint8
    fmt.Printf("%T\n", b) //int
}
```

类型别名

```
package main

import "fmt"


func main() {
    type bigint int64

    var a bigint = 100
    fmt.Printf("%T\n", a) //main.bigint

    //一次取多个别名
    type (
        long int64
        char byte
    )

    var b char = 'A'
```

```
    var c long = 100
    fmt.Printf("b=%c, type=%T\n", b, b) //b=A, type=main.char
    fmt.Printf("c=%d, type=%T\n", c, c) //c=100, type=main.long
}
```

golang的运算符

和c语言一样，*取值，&取地址

在go语言中，一元运算符拥有最高的优先级，二元运算符的运算方向均是从左至右。

优先级	运算符
7	^ !
6	* / % << >> & &^
5	+ - | ^
4	== != < <= >= >
3	<-
2	&&
1	||

golang流程控制

if if..else

```
package main

import "fmt"

func main() {

    //简单的if语句判断
    var name string
    name = "ABC"
    if name == "ABC" {
        fmt.Println("相等")
    }

    //if支持一个初始化语句
    if a := 10; a == 10{
        fmt.Println("a==10")
    }

    //if多分支
    name = "AAA"
    if name == "ABC" {
        fmt.Println("相等")
    }else {
        fmt.Println("不相等")
    }
```

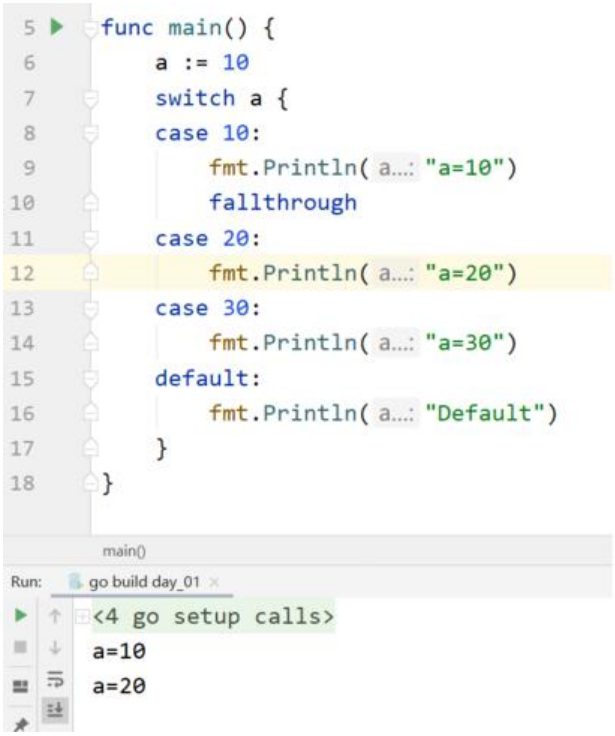
```
    name = "CCC"
    if name == "ABC" {
        fmt.Println("name=ABC")
    }else if name == "AAA" {
        fmt.Println("name=AAA")
    }else if name == "BBB" {
        fmt.Println("name=BBB")
    }else {
        fmt.Println("Other")
    }
}
```

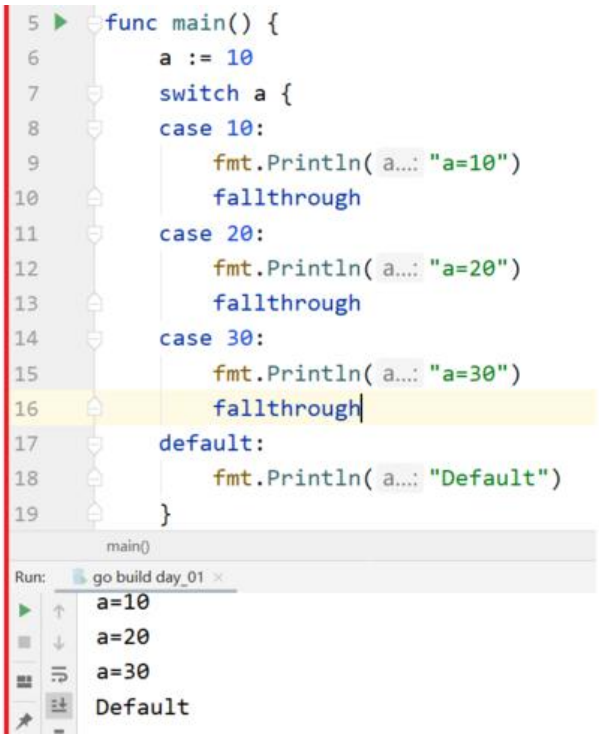
switch

```
package main

import "fmt"

func main() {
    a := 12
    switch a {
    case 10:
        fmt.Println("a=10")
    case 20:
        fmt.Println("a=20")
    case 30:
        fmt.Println("a=30")
    default:
        fmt.Println("Default")
    }
}
```

可以看出没有写break，go语言保留了break关键字，不写break，默认也包含break

fallthrough 关键字的作用：不跳出switch，还要执行紧随其后的一个分支






```
func main() {
    //同样的，switch也是支持一个初始化语句的
    switch a := 10; a {
    case 10:
        fmt.Println("a=10")
    case 20:
        fmt.Println("a=20")
    case 30:
        fmt.Println("a=30")
    default:
        fmt.Println("Default")
    }
}
```

switch可以没有条件

```
package main

import "fmt"

func main() {

    var score int
    //switch可以没有条件
    switch {
    case score > 90:
        fmt.Println("优秀")
    case score > 60 && score <= 90:
        fmt.Println("及格")
    default:
        fmt.Println("不及格")
    }
}
```

for

```
package main

import "fmt"

func main() {
    num := 0
    for i:=1; i<= 100; i++{
        num += i
    }
    fmt.Printf("num=%d\n", num) //5050

    //死循环的写法
    for{
        //TDDO...
    }
}
```

range

关键字range 会返回两个值，第一个返回值是元素的数组下标，第二个返回值是元素的值:

支持string、array、slice、map

```
package main

import "fmt"

func main() {

    str := "GoLand"
    //01 传统写法
    for i:=0; i<len(str); i++{
        fmt.Printf("%c ",str[i])
    }
    fmt.Println()

    //02 迭代写法
    for i := range str{
        fmt.Printf("%c ",str[i])
    }
    fmt.Println()

    //range返回两个值，一个是index、一个是元素本身
    //支持string、array、slice、map
    for i, data := range str{
        fmt.Printf("%d-%c\n",i, data)
    }
}
```

goto

和C语言的一样的：

```go
func main() {
    fmt.Println( a...: "start")
    goto End
    fmt.Println( a...: "run...")

End:
    fmt.Println( a...: "end")
}
```

main()

Run: go build day_01

```
<4 go setup calls>
start
end
```