0x01 变量

Go语言为静态类型语言，变量是有明确类型的，编译器会检测变量类型正确与否

声明变量一般使用var关键词

```
var name type
```

name为变量名，type为变量类型

注意Go语言的变量类型是在变量名称之后的

Go语言中声明int型指针类型的变量，格式如下

```
var c, d *int
```

1.1 变量默认值、命名

在一个变量被声明时，默认值如下

```
int      --->  0
float    --->  0.0
bool     --->  false
string   --->  空
指针      --->  nil
```

1.2 变量声明的格式

(1)标准格式

```
var 变量名 变量类型
```

(2)批量声明

```
var(
  age int
  name string
  phone string
)
```

(3)简短格式

简短的变量定义和初始化语法

```
名字 := 表达式
```

简短格式限制

- 只能用来定义变量，同时会显式初始化
- 不能提供数据类型
- 只能用在函数内部，即不能用来声明全局变量

简短格式变量声明语句也可以声明和初始化一组变量

```
name ,hobby := "J0o1ey", “Hacking”
```

简短格式简洁灵活的特点使其广泛应用于局部变量的声明与初始化

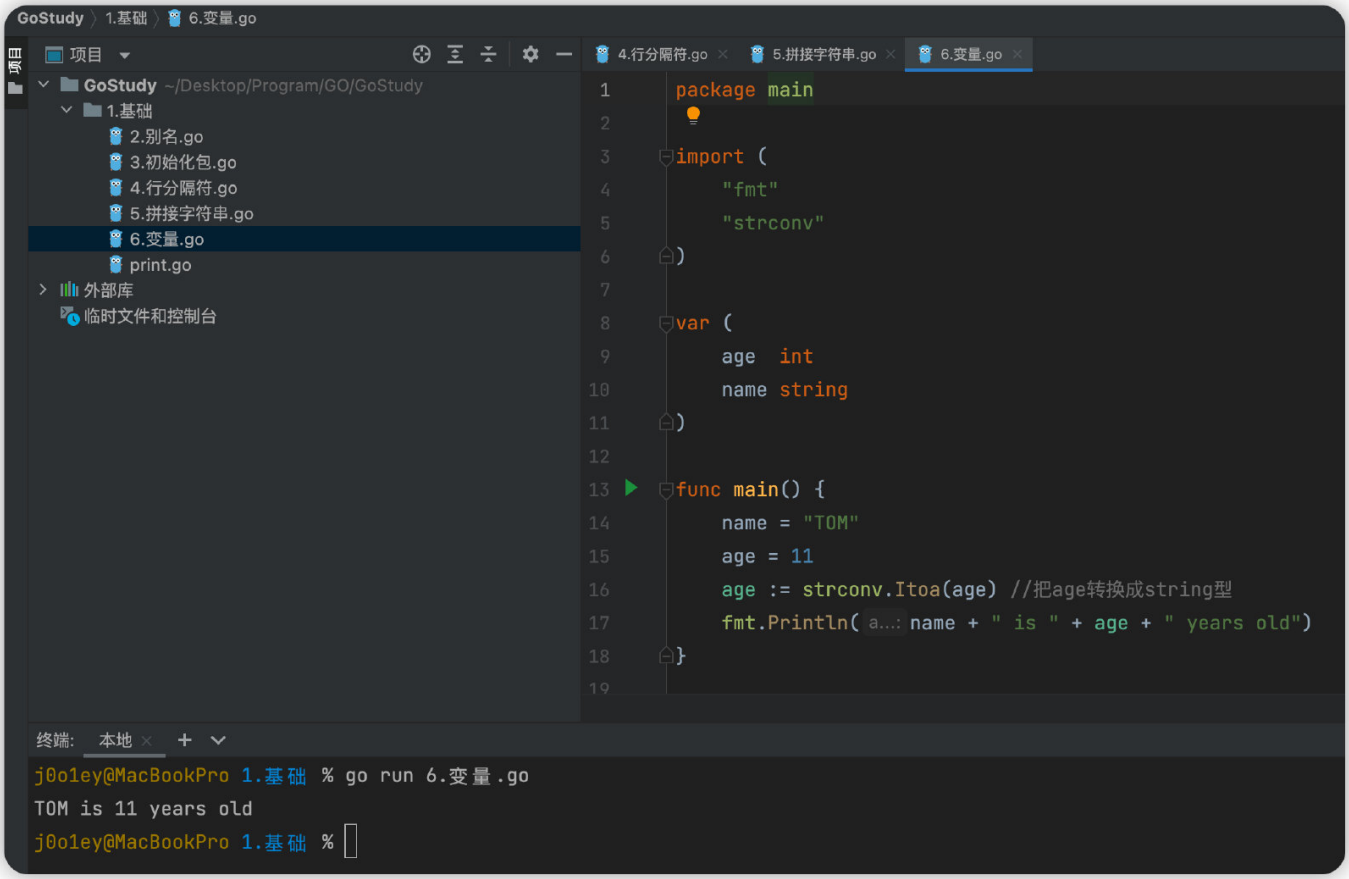
(4)以上变量声明知识的应用实例

```
package main

import (
  "fmt"
  "strconv"
)

var (
  age  int
  name string
)

func main() {
  name = "TOM"
  age = 11
  age := strconv.Itoa(age) //把age转换成string型
  fmt.Println(name + " is " + age + " years old")
}
```



1.3 赋值

(1)单个变量赋值

```
var 变量名 类型 = 变量值
```

此时如果不想声明变量类型，则可以省略

```
var language string = "Go"

//等效于

var language = "Go"

//等效于

language := "Go"
```

(2)给多个变量赋值

多行形式

```
var (
  变量名1 (变量类型1) = 变量值1
  变量名2 (变量类型2) = 变量值
  //  other var...
)
```

单行形式，中间用英文","隔开

```
var name1,name2,name3 = value1,value2,value3
```

(3)声明一个用户的信息，用如下三种等效方法赋值

多行形式

```
var(
  age      int = 20
  name     string = "J0o1ey"
  hobby    string = "Hacking"
  money    float32 = "99.99"
)
```

单行形式

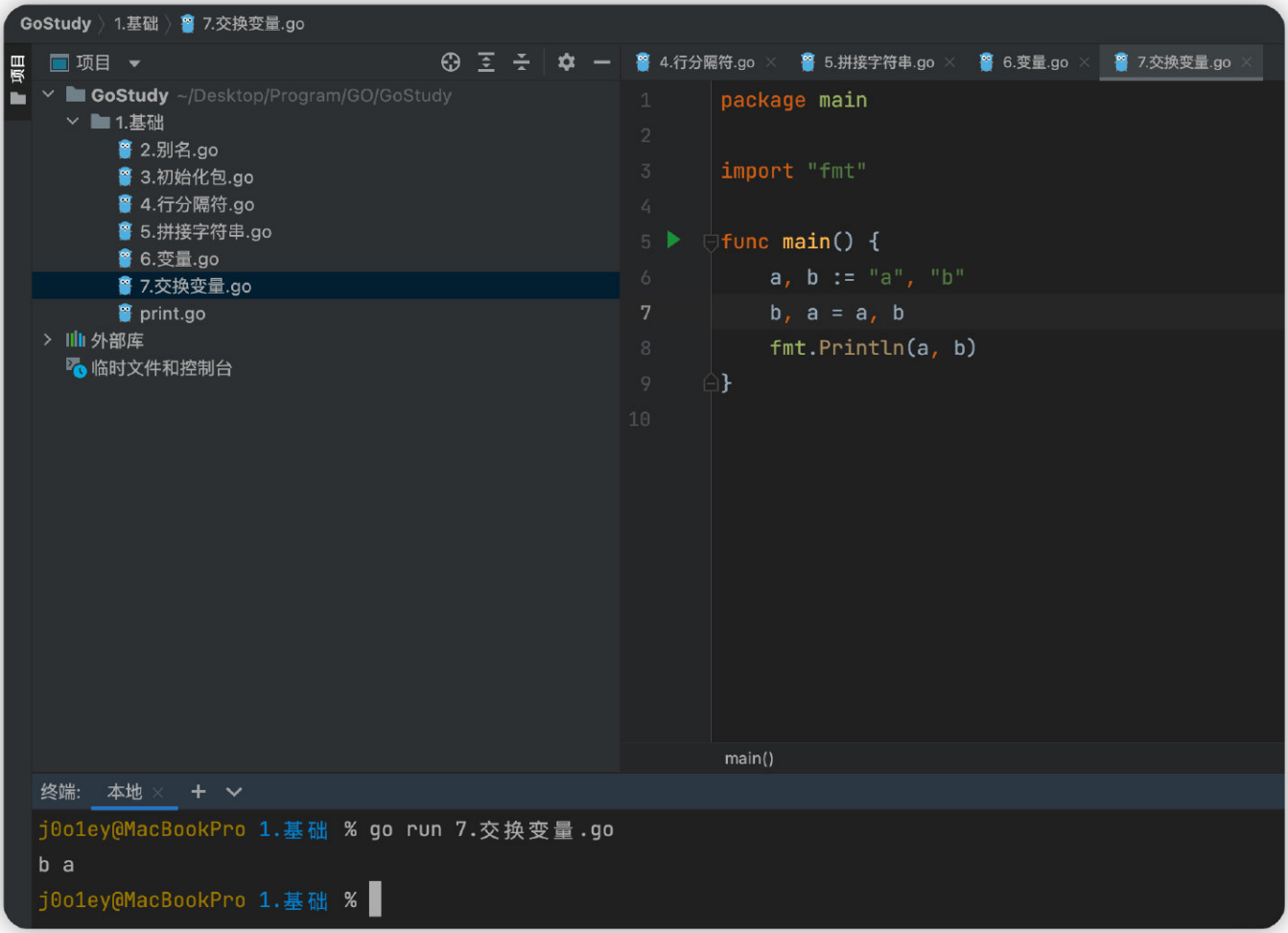
```
var age.name,hobby,money = 20,"J0o1ey","Hacking",99.99
```

简短格式

```
age,name,hobby,money := 20,"J0o1ey","Hacking",99.99
```

(4)交换变量

```
a, b := "a","b"
b, a =  a,b
```



1.4 变量的作用域

Go语言中的变量可以分为局部变量和全局变量

(1)局部变量

函数体内声明的变量为"局部变量"，其作用域只在函数体内，参数和返回值变量也是局部变量

如下代码声明了三个局部变量

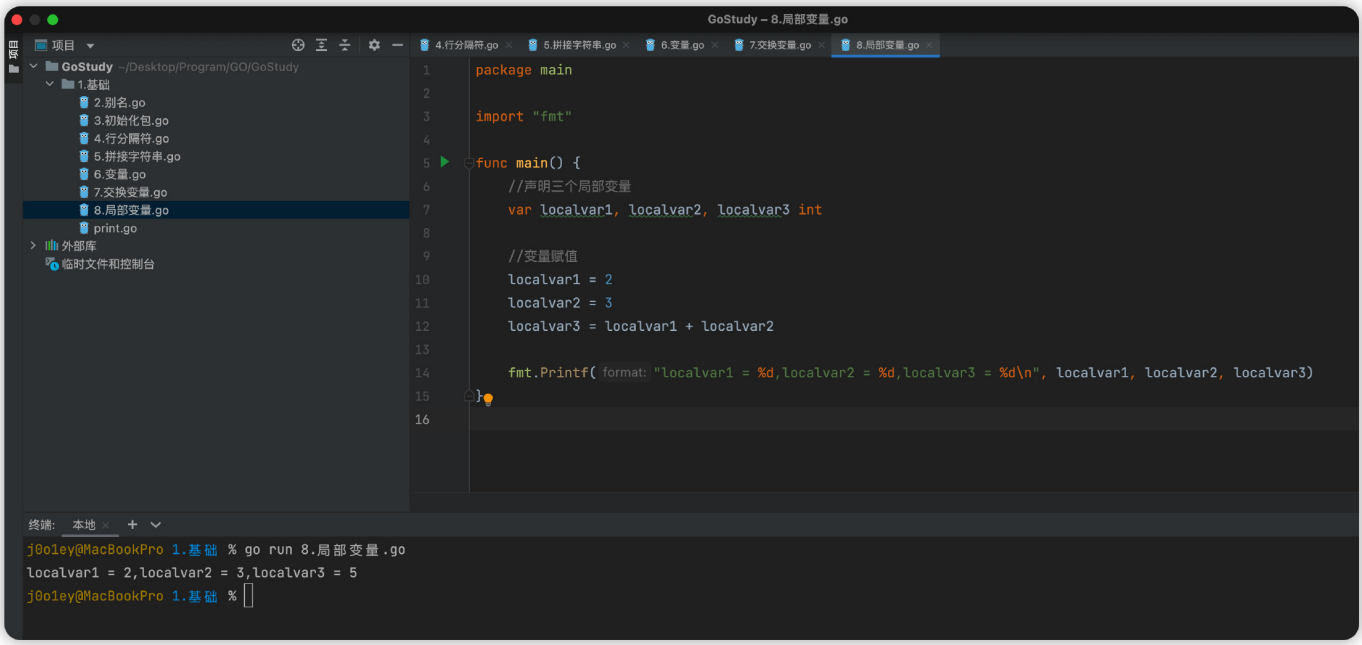
```
package main

import "fmt"

func main() {
  //声明三个局部变量
  var localvar1, localvar2, localvar3 int

  //变量赋值
  localvar1 = 2
  localvar2 = 3
  localvar3 = localvar1 + localvar2
```

```
  fmt.Printf("localvar1 = %d,localvar2 = %d,localvar3 = %d\n", localvar1, localvar2,
localvar3)
}
```



(2)全局变量

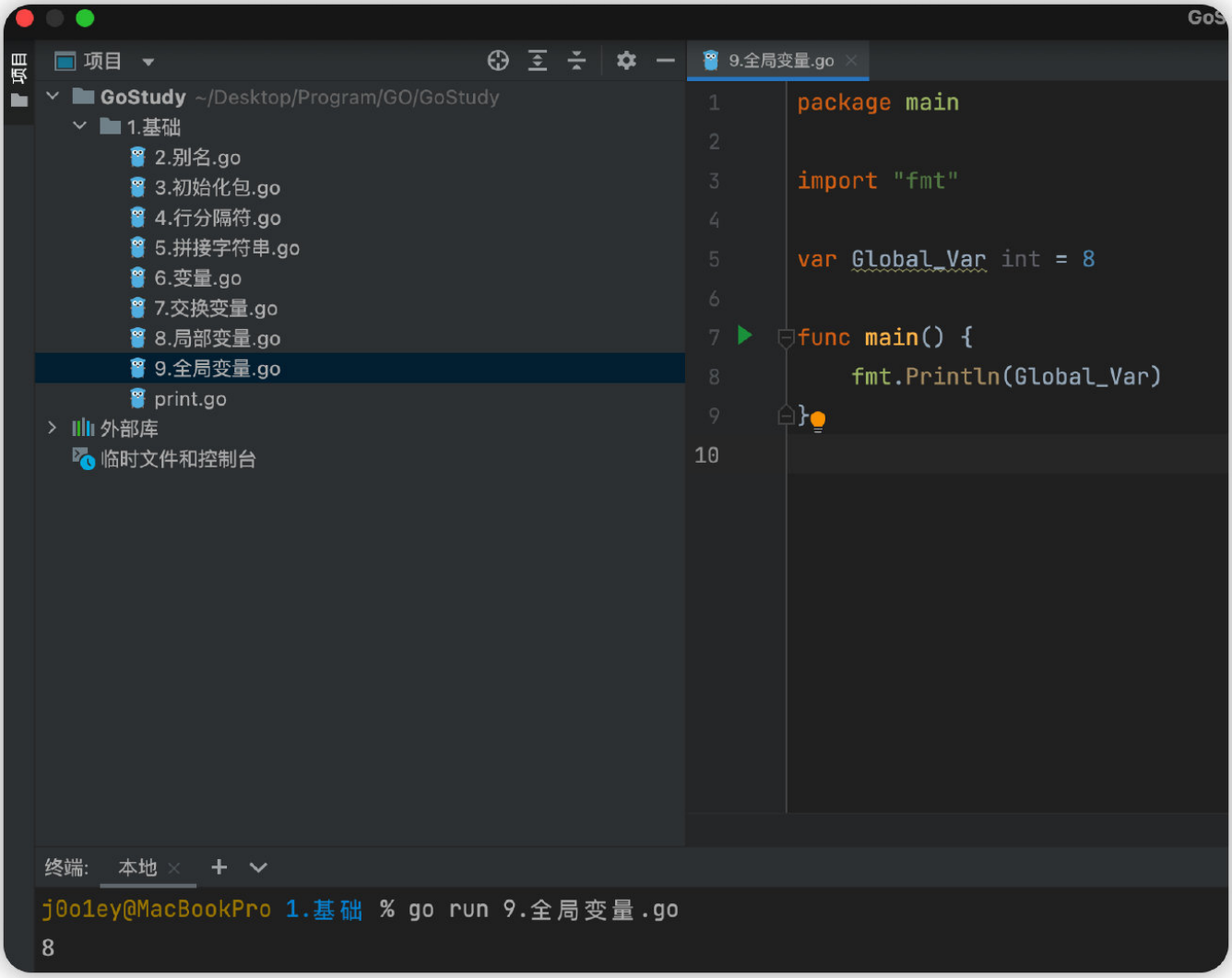
函数体外声明的变量称为“全局变量”。全局变量可以在整个包、被导出的外部包中使用，也可以在任意函数中使用，示例如下

```
package main

import "fmt"

var Global_Var int = 8

func main() {
  fmt.Println(Global_Var)
}
```



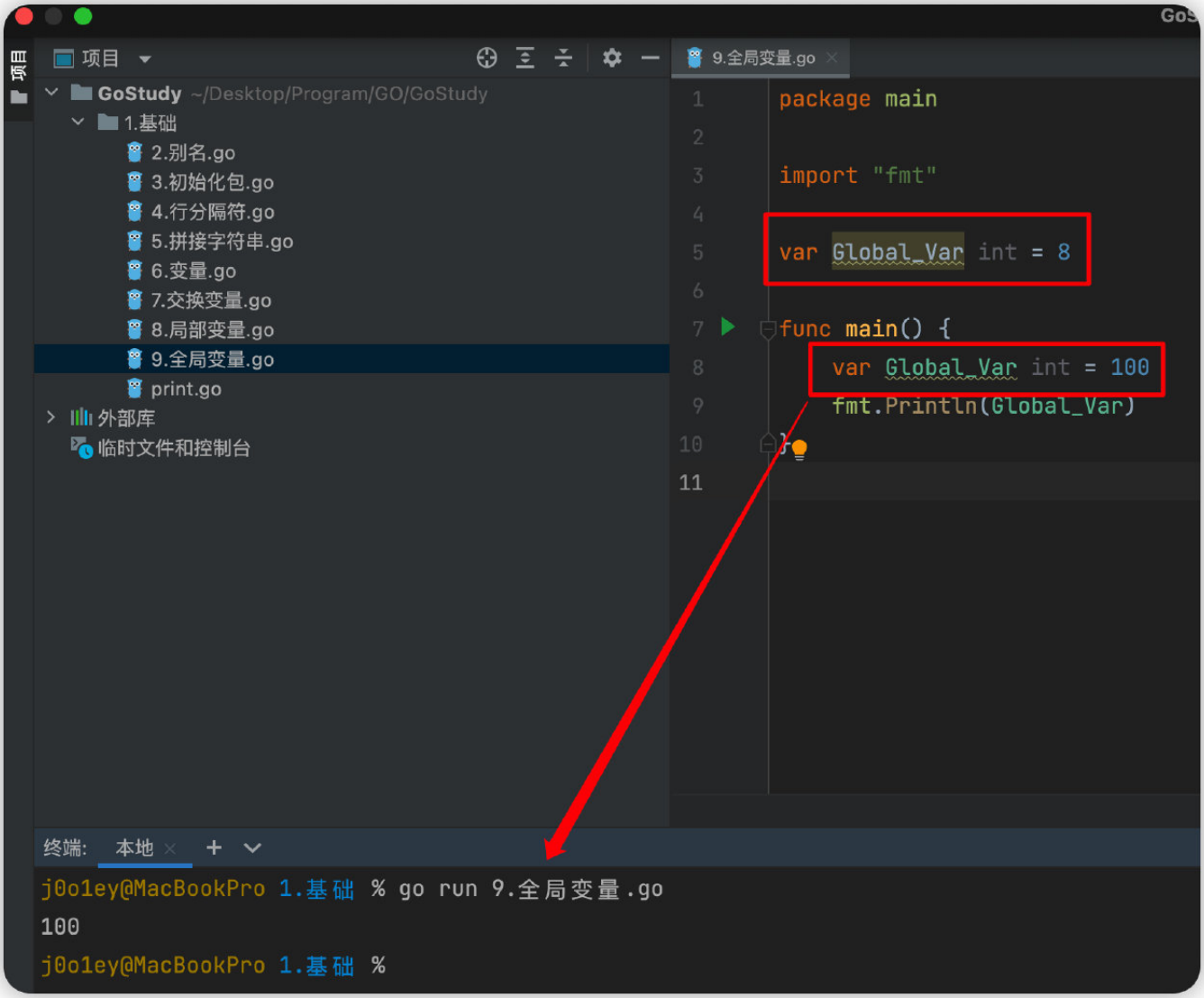
在Golang中全局变量与局部变量名称可以想通，函数内的局部变量会被优先考虑

```
package main

import "fmt"

var Global_Var int = 8

func main() {
  var Global_Var int = 100
  fmt.Println(Global_Var)
}
```



0x02 常量

Go语言中声明常量使用关键词const声明。常量储存不会改变的数据，只能是布尔型，数字型，字符串型、

由于编译的限制，声明常量的表达式必须为“能被编译器求值的常量表达式”

常量的声明格式如下

```
const 常量名称 类型 = 常量值
```

例如声明pi

```
const pi float32 = 3.1415926
```

也可以直接

```
const pi = 3.1415926
```

Notice:

①在Go语言中，可以省略说明符“类型”，编译器会根据变量值判断其类型

②常量的值必须是能够在编译时可被确定的，可以在其赋值表达式中涉及计算，**但是计算期间的值必须在编译期间能获得**

```
const result = 5/2  //正确常量

const url = os.GetEnv("url")  //错误常量
```

2.1 批量声明常量

```
const (
  e = 2.7182818
  pi = 3.1415926
)
```

常量的运算都是在编译期间完成的，方便代码的编译优化，同时一些运行的错误也可以在编译期间被发现，比如整数除零、字符串索引越界，导致无效浮点数操作...

2.2 常量特性

常量间所有的算术运算，逻辑运算，比较运算，返回结果都是常量

对常量使用类型转换，或调用len()，cap()，imag()，unsafe.Sizeof()等函数，返回的结果依然是常量，因为他们的值在编译期间是确定的，因此常量可以是构成类型的一部分，例如用于指定数组类型的长度。

如下例部分代码，用常量IPV4Len来指定数组p的长度

```
const IPv4Len = 4

//parseIPv4解析一个IPv4地址(xxx.xxx.xxx.xxx)
func parseIPv4(s string)  {
  var p = [IPv4Len]byte
  // ......
}
```

2.3 常量生成器iota

声明常量可以使用常量生成器iota初始化，用于生成一组以相似规则初始化的常量，不用每一行都写一遍初始化表达式

在一个const声明语句中，在第1个声明的常量的所在行，iota会被置为0，之后的每一个有常量声明的行都会被加1

示例如下

```
package main

type Direction int    //定义一个Direction命名类型
const(
  North Direction = iota
  East
  West
  South
)
//为东南西北各自定义了一个常量，从北方0开始，在其他语言，这种类型被称为“枚举类型”
//North的值为1，East的值为2，West值为3....
```

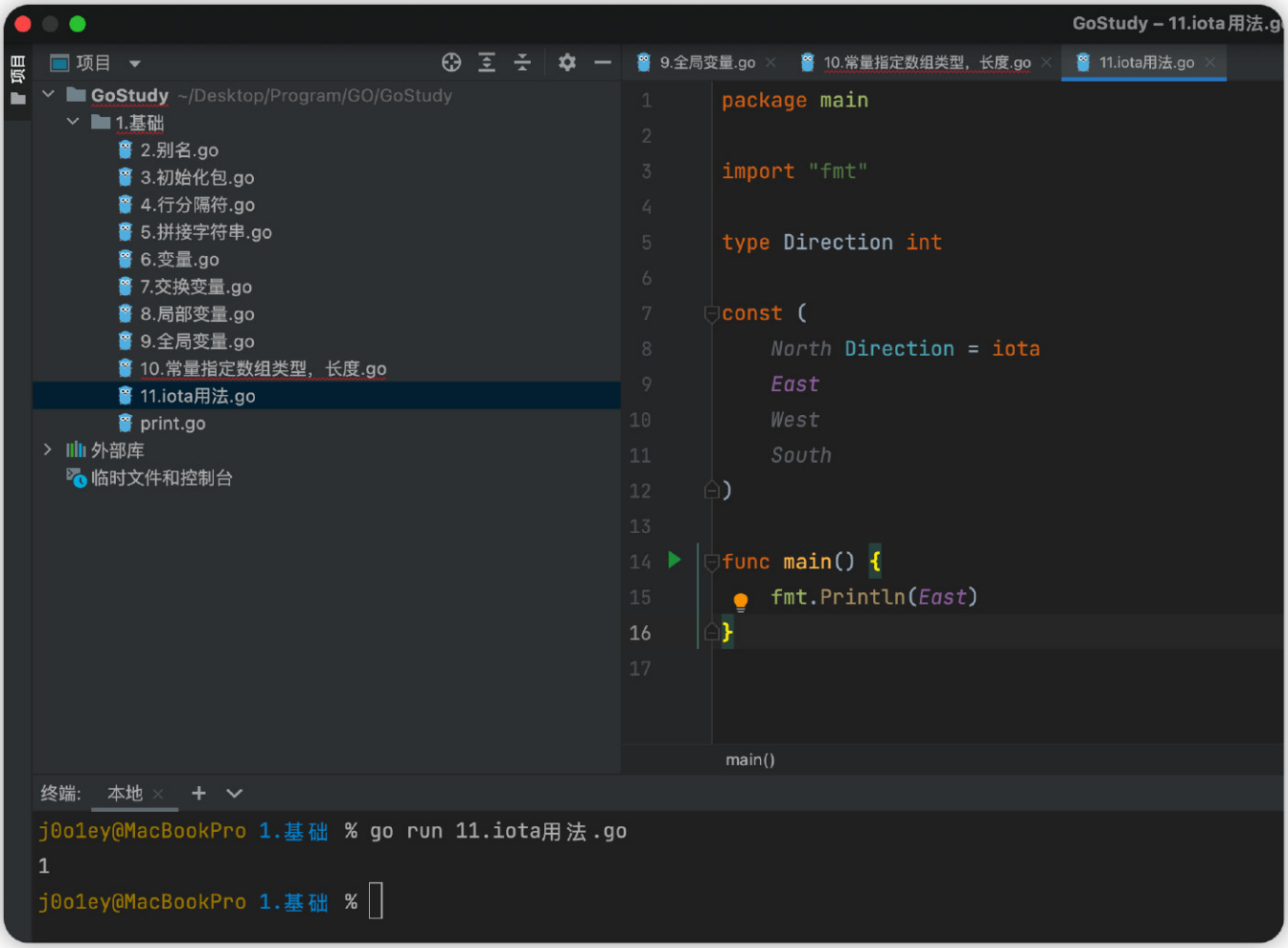
```
package main

import "fmt"

type Direction int

const (
  North Direction = iota
  East
  West
  South
)

func main() {
  fmt.Println(East)
}
```



2.4 延迟明确常量的具体类型

Go语言中虽然一个常量可以有任意一个确定的基础类型(int，float64，string...)，但很多常量没有确定的基础类型，编译器为这些没有明确基础类型的数字常量，提供比基础类型更高精度的算术运算

Go语言6种未明确类型的常量：
①无类型的布尔型
②无类型的整数
③无类型的字符
④无类型的浮点数
⑤无类型的复数
⑥无类型的字符串

延迟却明确常量的具体类型，不仅可以提供更高的运算精度，还可以用于更多表达式而不需要显式的类型转换

示例如下(math.Pi为无类型的浮点数常量)：

```
var a float32 = math.Pi
var b float64 = math.Pi
var c complex128 = math.Pi
```

如果math.Pi被设置为特定类型(下例为float64)，那么结果精度可能会有出入。

同时需要在用到float32，complex128等类型时对其进行明确的强制类型转换

```
const Pi64 float64 = math.Pi
var a float32 = float32(Pi64)
var b float64 = float64(Pi64)
var c complex128 = complex128(Pi64)
```

2.5 其他

对于常量的值，不同的写法会有不同的类型

示例如下（以下常量值相同）

```
0         //无类型的整数

0.0       //无类型的浮点数

0i        //无类型的复数

\u0000    //无类型的字符
```

```
true
//布尔型或者无类型的字符串型

false
```