

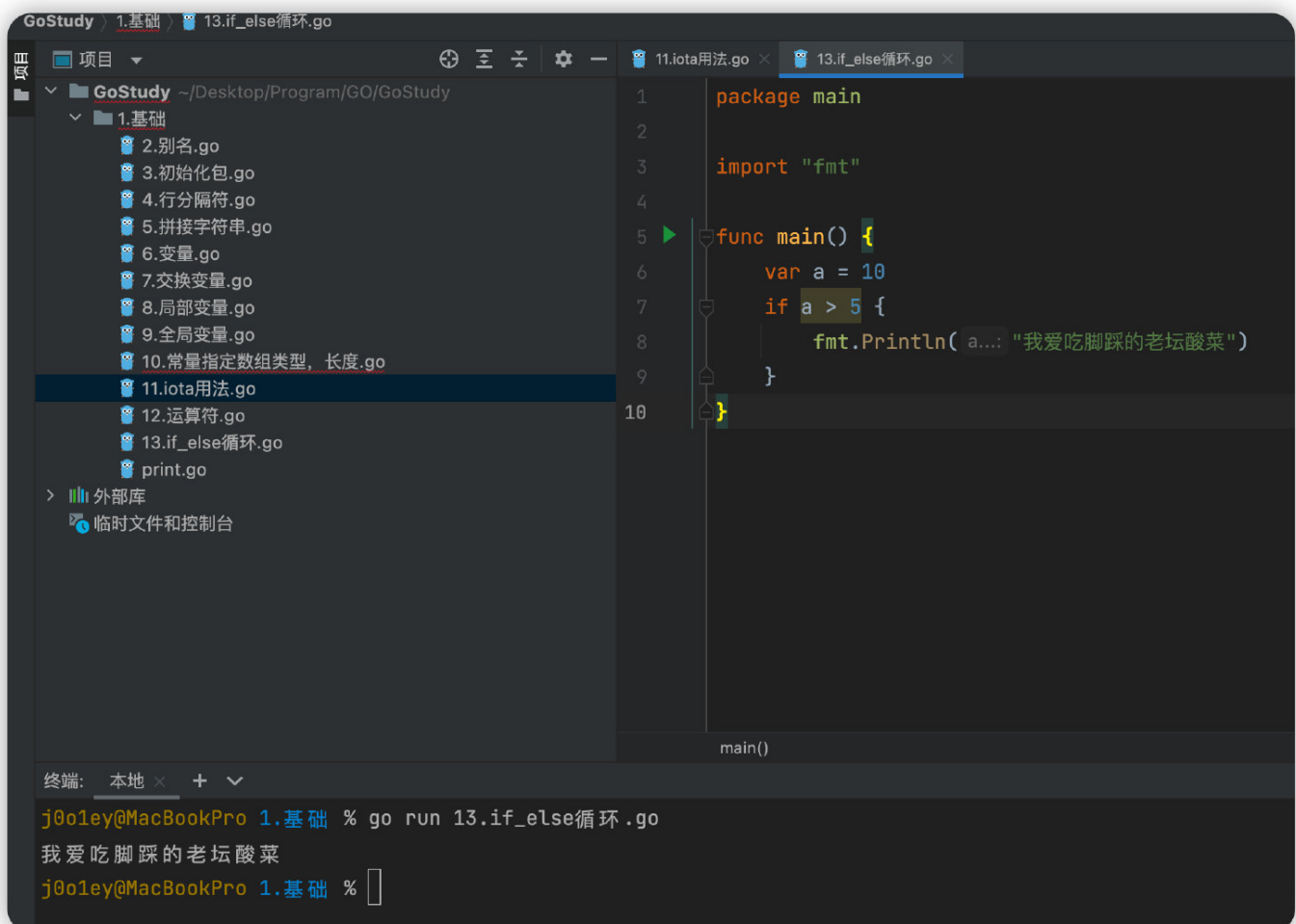
## 0x01 if-else分支

Golang中，关键词if通常用于判断某个条件(布尔型或逻辑型)。如果条件成立则执行if后大括号( {} )包裹的代码，否则就忽略该部分，继续执行后续的代码。

```
package main

import "fmt"

func main() {
    var a = 10
    if a > 5 {
        fmt.Println("我爱吃脚踩的老坛酸菜")
    }
}
```



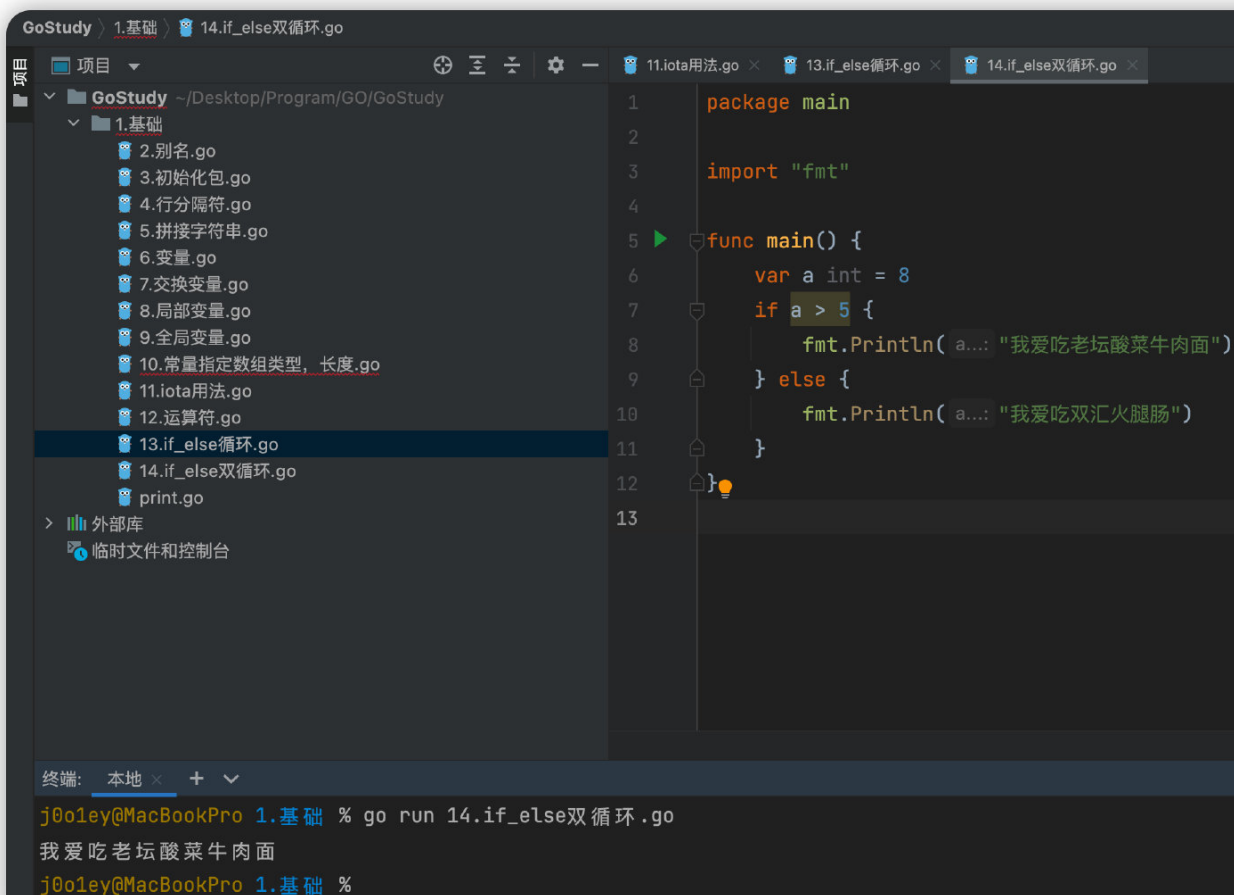
## 1.1 两个分支

else部分代码块在if条件不满足时执行，if{}和else{}两个代码块相互独立，两者只能执行其中一个

```
package main

import "fmt"

func main() {
    var a int = 8
    if a > 5 {
        fmt.Println("我爱吃老坛酸菜牛肉面")
    } else {
        fmt.Println("我爱吃双汇火腿肠")
    }
}
```



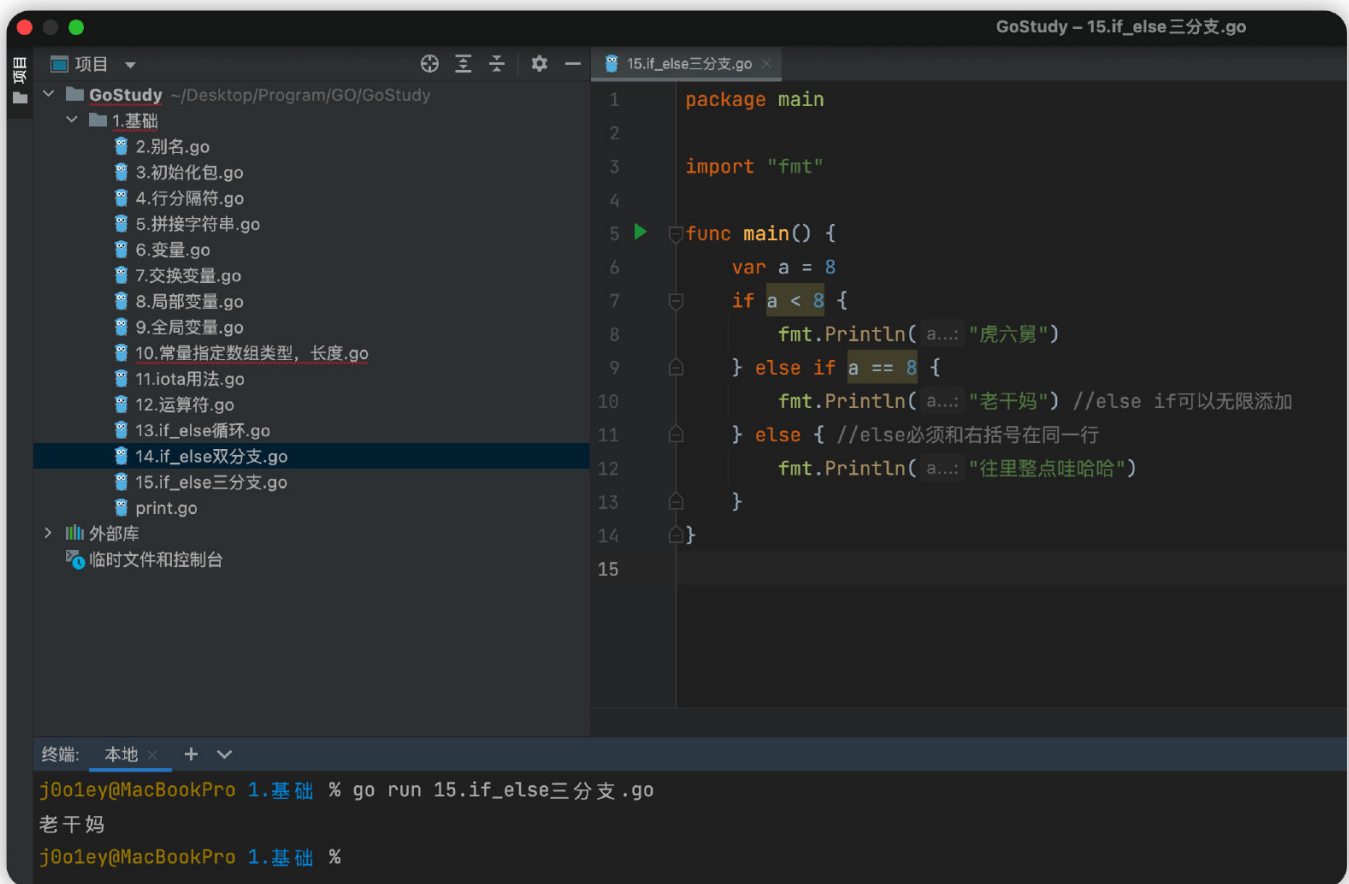
## 1.2 三个分支

```
package main

import "fmt"
```

```
func main() {
    var a = 8
    if a < 8 {
        fmt.Println("虎六舅")
    } else if a == 8 {
        fmt.Println("老干妈") //else if可以无限添加
    } else { //else必须和右括号在同一行
        fmt.Println("往里整点哇哈哈")
    }
}
```

else if可以无限添加，为了代码可读性，最好不要加太多，else-if结构中右大括号"}"必须和else if在同一行



## 0x02 for循环

Go语言中循环语句只支持for关键词，不支持while与do-while，关键词for的用法如下

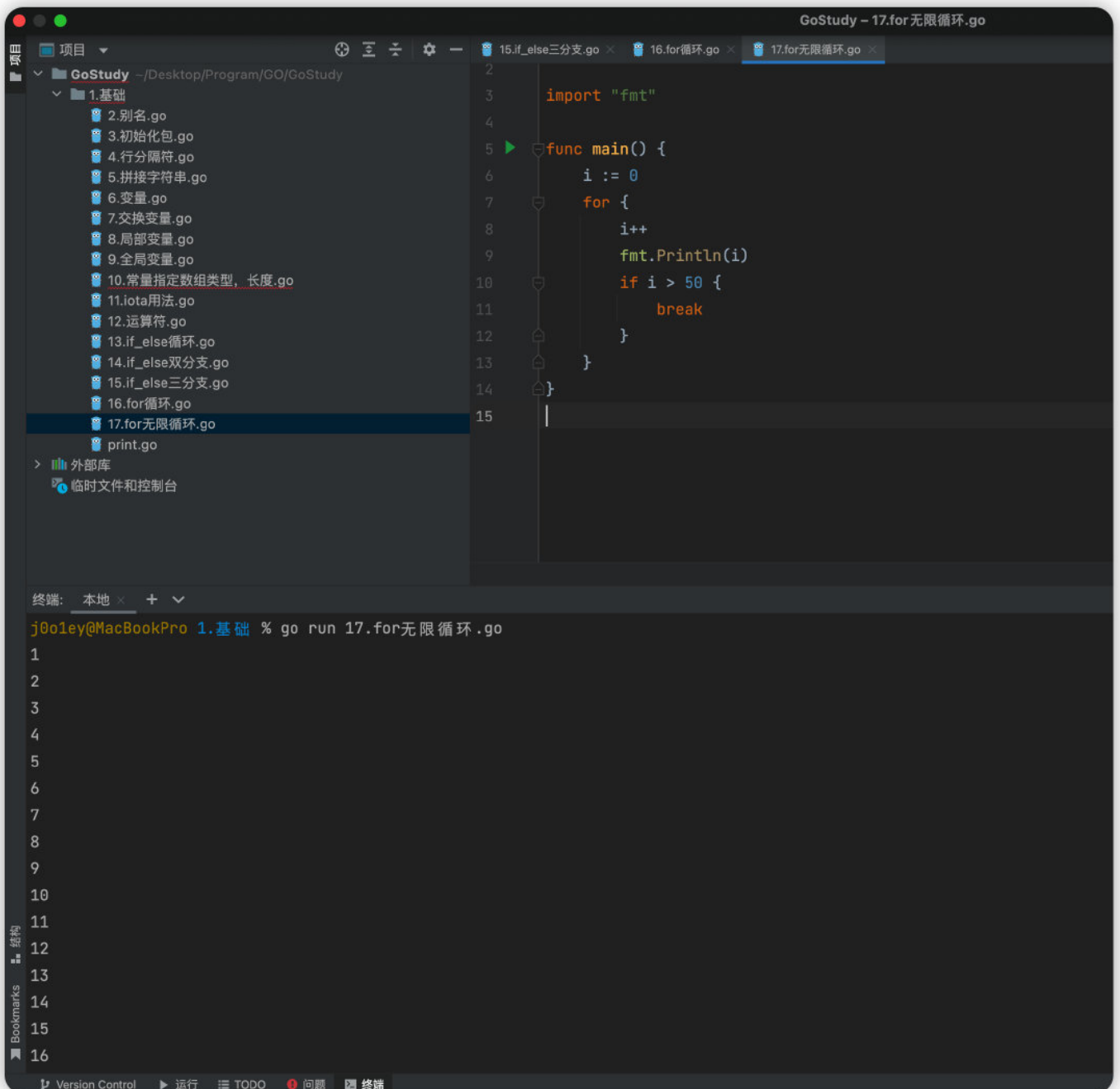
```
id := 1
for i := 1; i < 5; i++ { //for后面的表达式不需要用括号括起来
    id *= i
}
```

无限循环场景，简化写法如下

```
package main

import "fmt"

func main() {
    i := 0
    for {
        i++
        fmt.Println(i)
        if i > 50 {
            break
        }
    }
}
```



## Notice:

- 1.左括号和for关键词在同一行
- 2.Go不支持以逗号为间隔的多个赋值语句，必须用平行赋值方式赋值多个变量
- 3.Go语言的for循环支持使用continue和break来控制循环

## 2.1更高级的break

Go在for循环中提供了更高级的break，如下例，可以选择中断哪一个循环

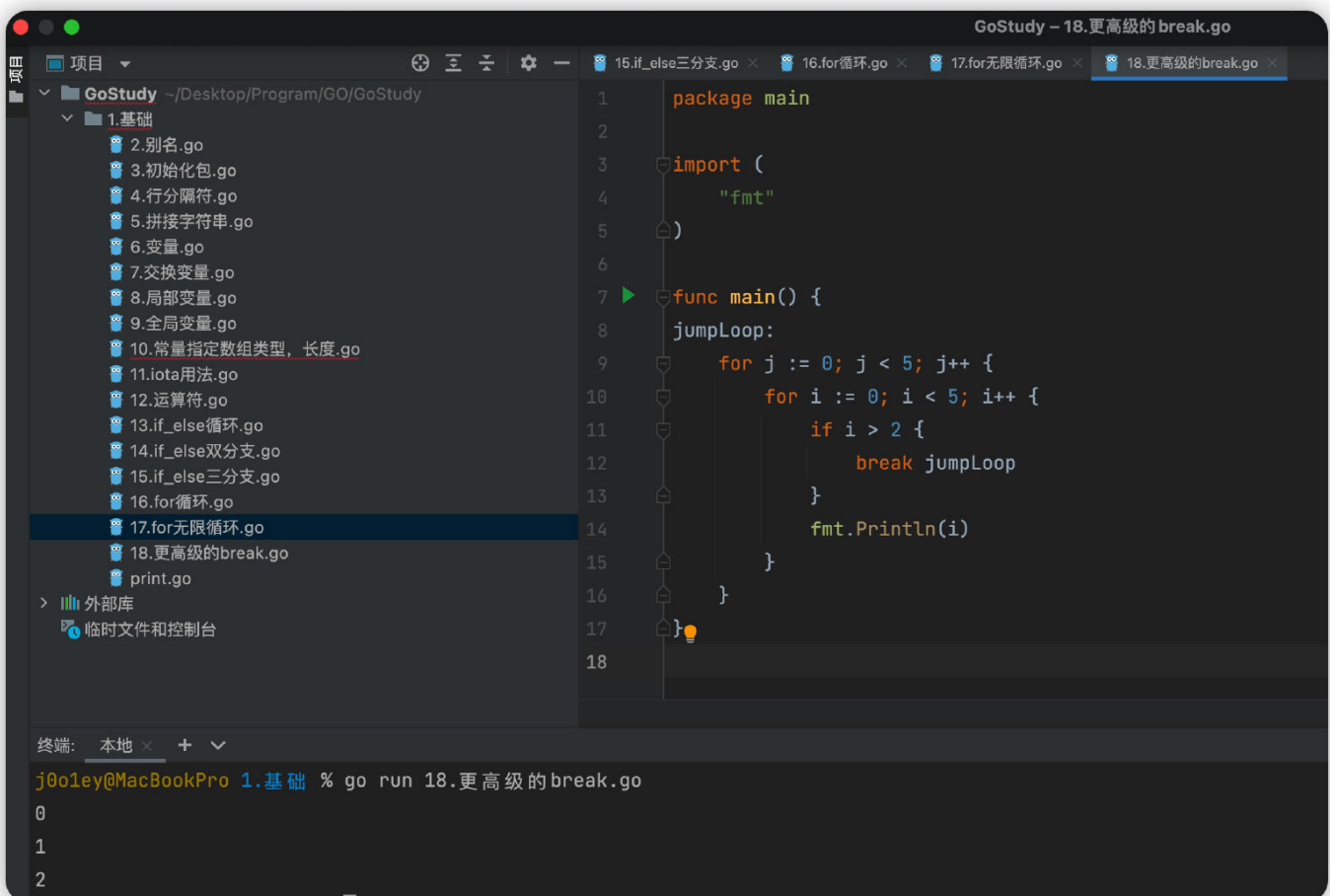
```
package main
```

```

import (
    "fmt"
)

func main() {
jumpLoop:
    for j := 0; j < 5; j++ {
        for i := 0; i < 5; i++ {
            if i > 2 {
                break jumpLoop //break中止的是Jumploop标签对应的for循环
            }
            fmt.Println(i)
        }
    }
}

```



## 2.2 for循环的多种表达写法

for中的初始语句是在第一次循环前执行的语句，一般使用初始化语句进行变量初始化。

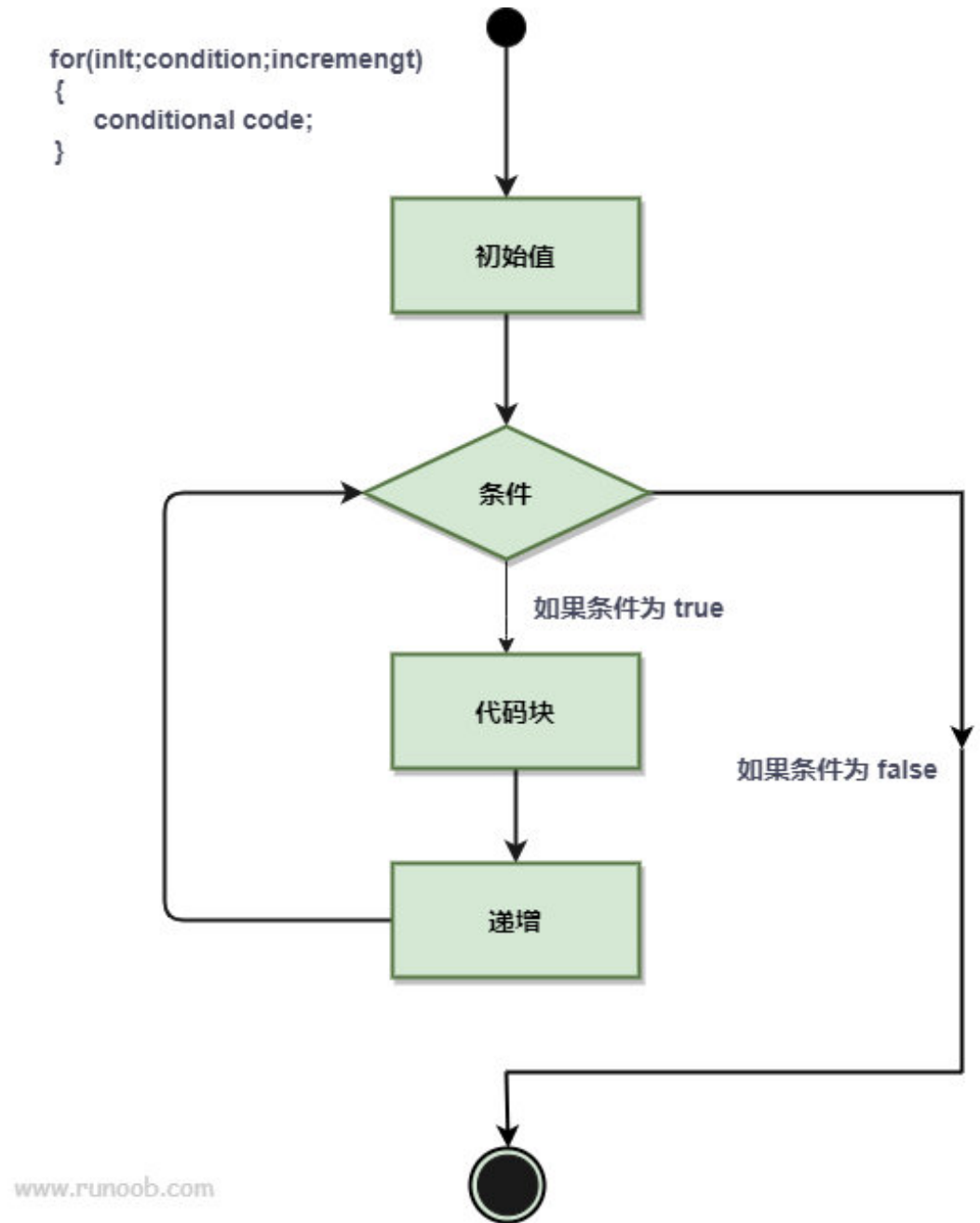
但如果变量在for循环前被声明，则其作用域只是这个for的范围

初始语句可以省略，但初始语句之后的分号必须要写，示例如下

```
j := 2
for ; j>0 ; j--{
    fmt.println(j)
}
```

上述代码中，将j放在for前进行初始化，for中没有初始语句，此时j的作用域，比初始语句中直接声明的j作用域要大

for中的表达式是控制是否循环的开关，每次循环开始前，都会判断条件表达式，如果表达式为true，则继续循环，否则就停止循环



条件表达式可以被忽略，忽略表达式后默认形成无限循环

如下代码会忽略条件表达式，形成无限循环



```

package main

func main() {
    var i int
JumpLoop:
    for ; ; i++ {    //for没有设置i的初始值，也没有设置i的循环条件表达式，此时循环会一直持续下去
        println(i)
        if i >= 10 {
            break JumpLoop    // i大于10的时候通过break跳出JumpLoop标签对应的for循环
        }
    }
}

```

The screenshot shows an IDE with a file explorer on the left listing various Go files. The main editor displays the Go code from the previous block. The terminal at the bottom shows the command `go run 19.for忽略表达式无限循环.go` being executed, and the output is a sequence of numbers from 0 to 10, each on a new line.

更美观的写法如下

```

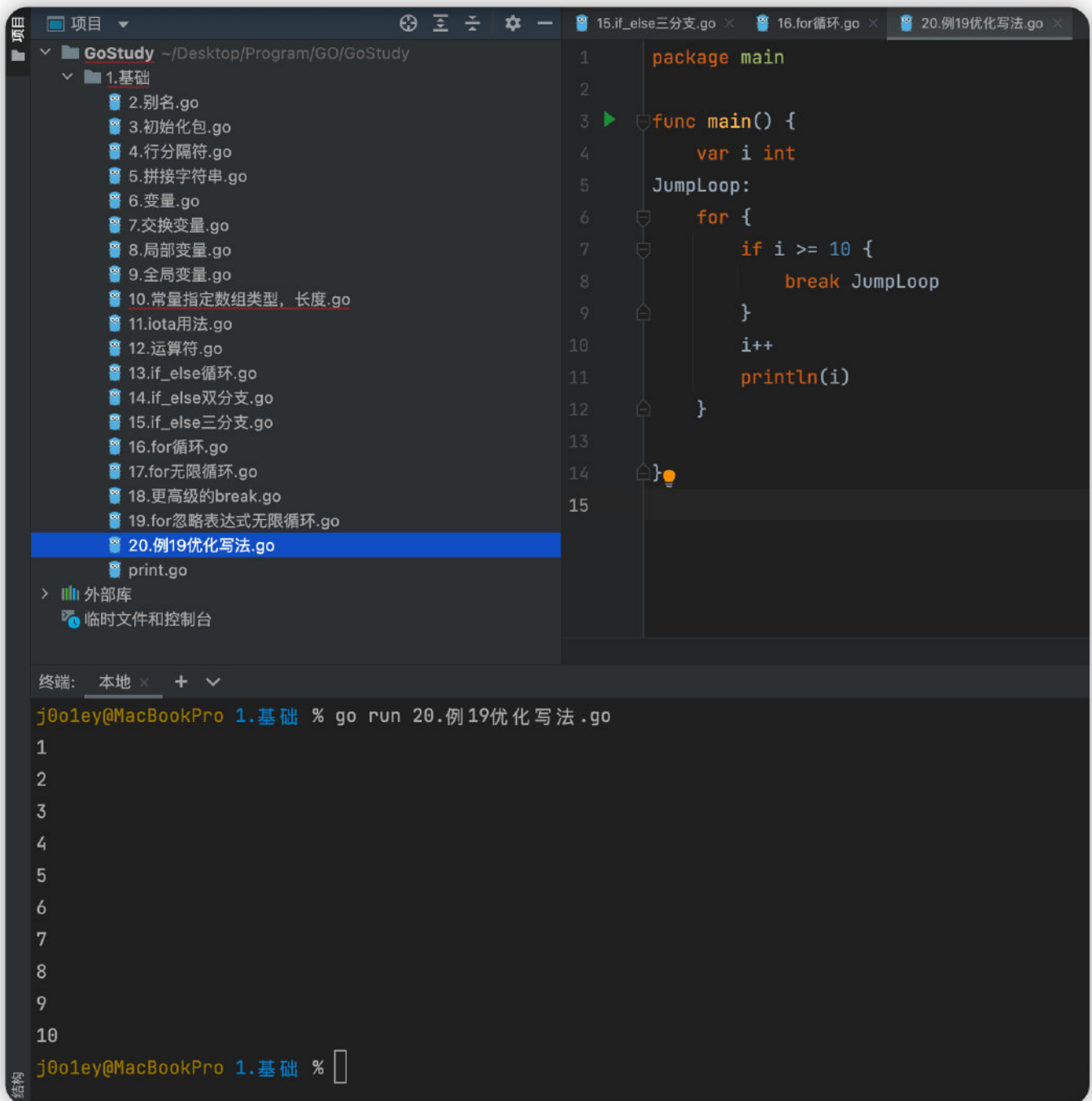
package main

func main() {
    var i int
JumpLoop:
    for {            //忽略for后面的变量与分号，此时for无限循环
        if i >= 10 {
            break JumpLoop
        }
    }
    i++              //从for的结束句放到函数体末尾，两者是等效的
}

```

```
println(i)
}

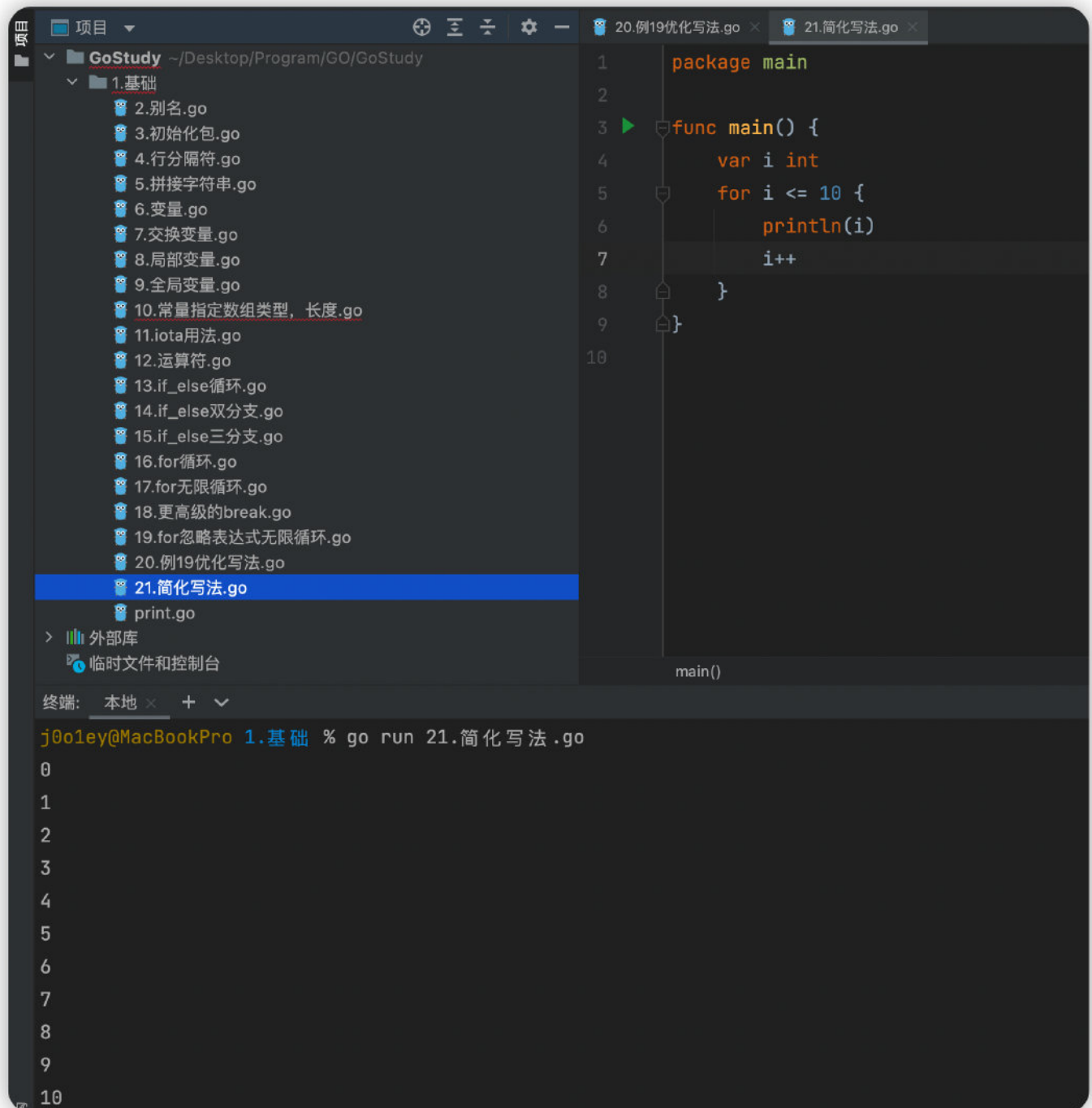
}
```



进一步简化，将if整合到for中，代码如下

```
package main

func main() {
    var i int
    for i <= 10 {
        println(i)
        i++
    }
}
```



Other:在for循环中，如果循环被break，goto，return，panic等语句强制退出，则之后的语句不会继续执行

## 0x03 for-range循环

for-range循环结构是Go语言特有的一种迭代结构，应用十分广泛

for-range可以遍历数组、切片、字符串、map、通道

语法类似于PHP的foreach语句，示例如下：

```
for key, val := range 复合变量值{
    //...逻辑语句
}
```

**Notice:**

**val**始终作为集合中对应索引的一个复制值。因此，它一般只有“只读”属性，对它所做的任何修改都不会影响集合中原有的值

一个字符串是Unicode编码的字符(或称为rune)，也可以用它来迭代字符串

```
for position, char := range str{
    //...逻辑语句
}
```

每个rune字符和索引在for-range中循环的值是一一对应的，他能自动根据UTF-8规则识别Unicode编码的字符

通过**for-range**遍历的返回值如下

- 数组，切片，字符串返回索引和值
- map返回键和值
- 通道只返回通道内的值

具体示例如下

### 3.1 遍历数组、切片

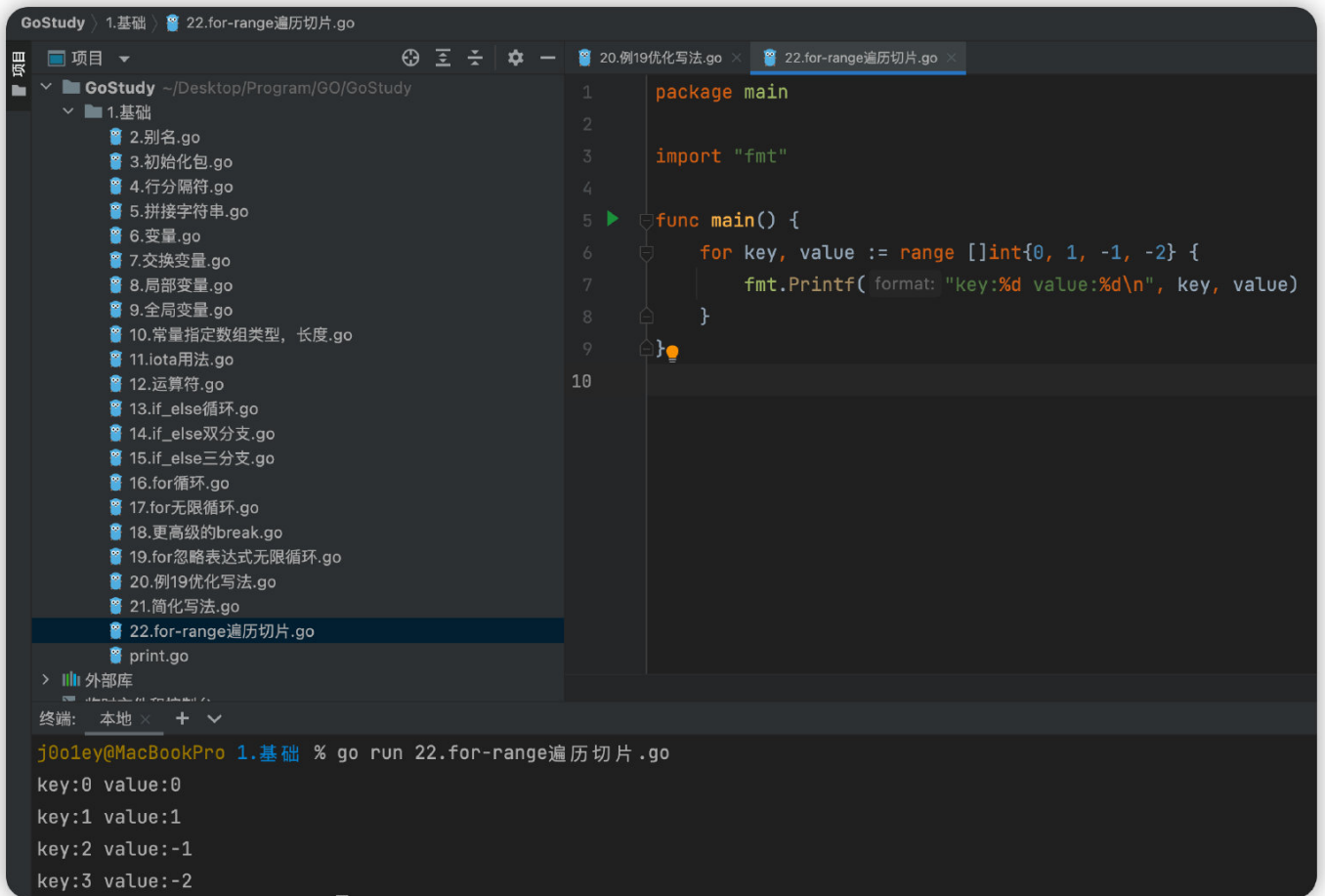
在遍历代码中，key和value分别代表切片的下标以及下标对应的值

如下代码展现的是如何遍历切片

```
package main

import "fmt"

func main() {
    for key, value := range []int{0, 1, -1, -2} {
        fmt.Printf("key:%d value:%d\n", key, value)
    }
}
```



## 3.2 遍历字符串

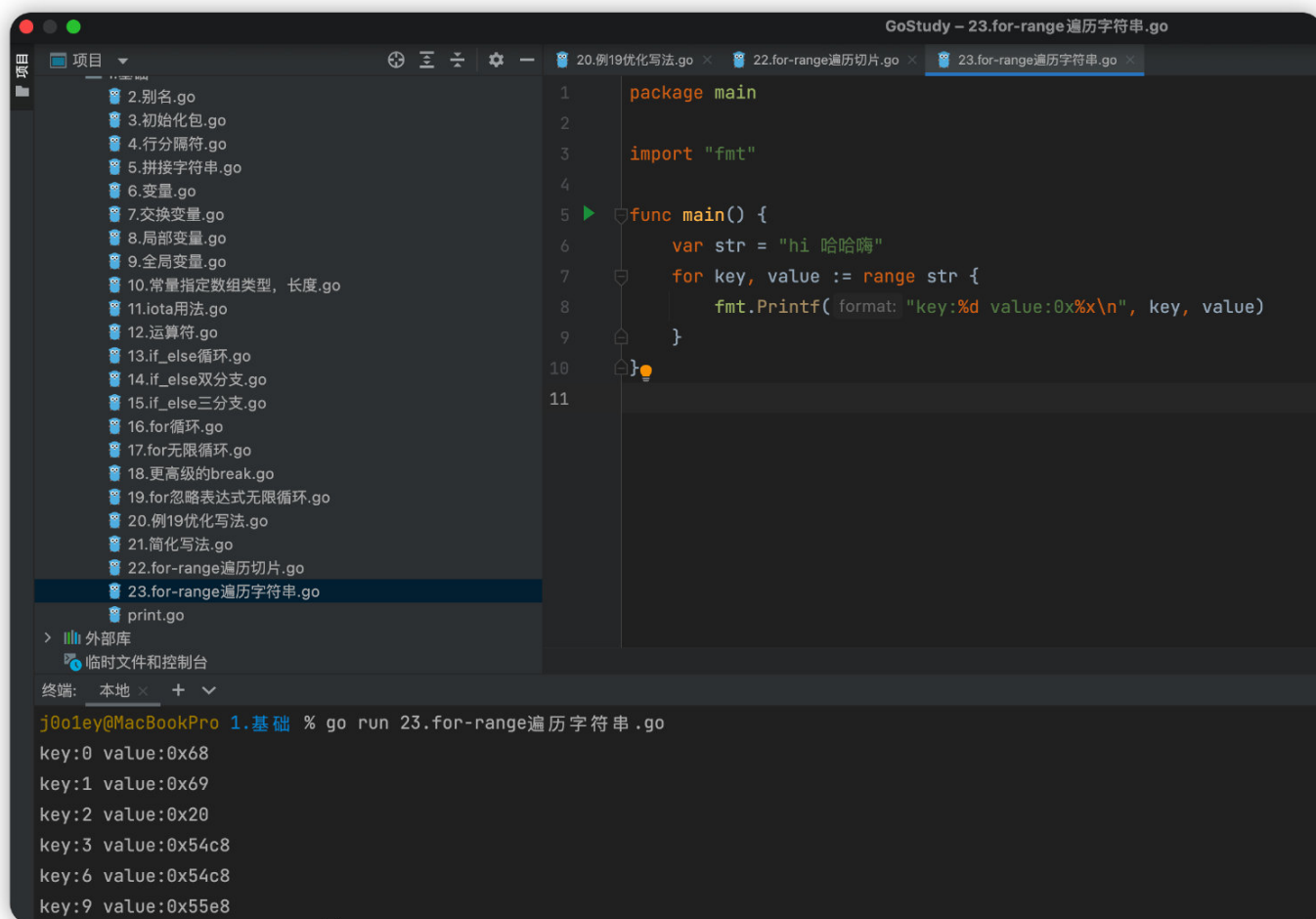
Go语言中可以通过for-range的组合对字符串进行遍历。在遍历时，key和value分别代表字符串的索引和字符串的一个字符

```
package main

import "fmt"

func main() {
    var str = "hi 哈哈嗨"
    for key, value := range str {
        fmt.Printf("key:%d value:0x%x\n", key, value)
    }
}
```

代码中的变量value的类型为rune类型，以16进制打印出来字符的编码



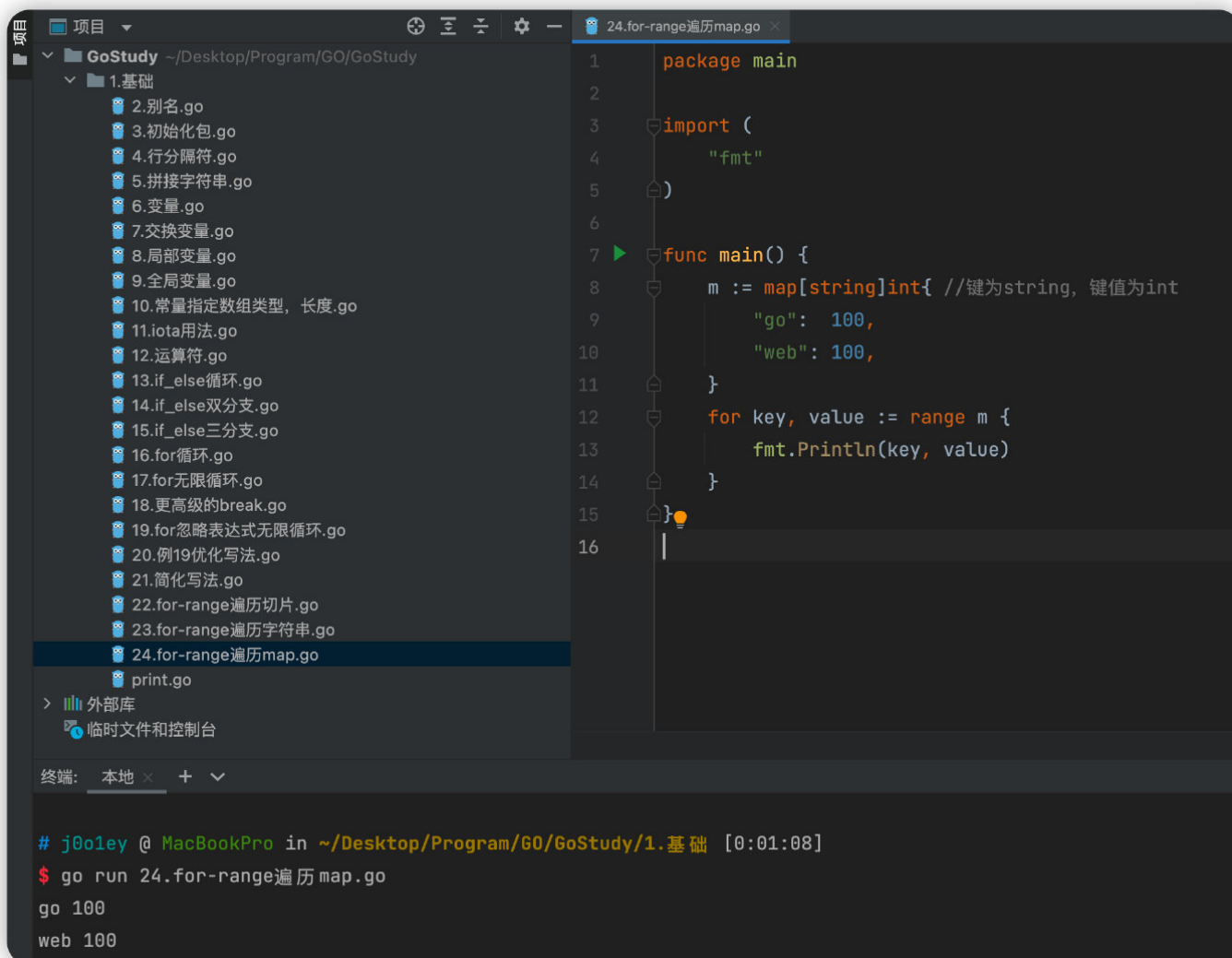
## 3.3 遍历map

对于map类型，for-range在遍历时，key和value代表map的索引键key和索引键对应的值

```
package main

import (
    "fmt"
)

func main() {
    m := map[string]int{ //键为string, 键值为int
        "go" : 100,
        "web" : 100,
    }
    for key,value := range m{
        fmt.Println(key, value)
    }
}
```



### 3.4 遍历通道(channel)

通道可以使用for range进行遍历，与map不同，遍历通道时只输出一个值，即通道内类型对应的数据

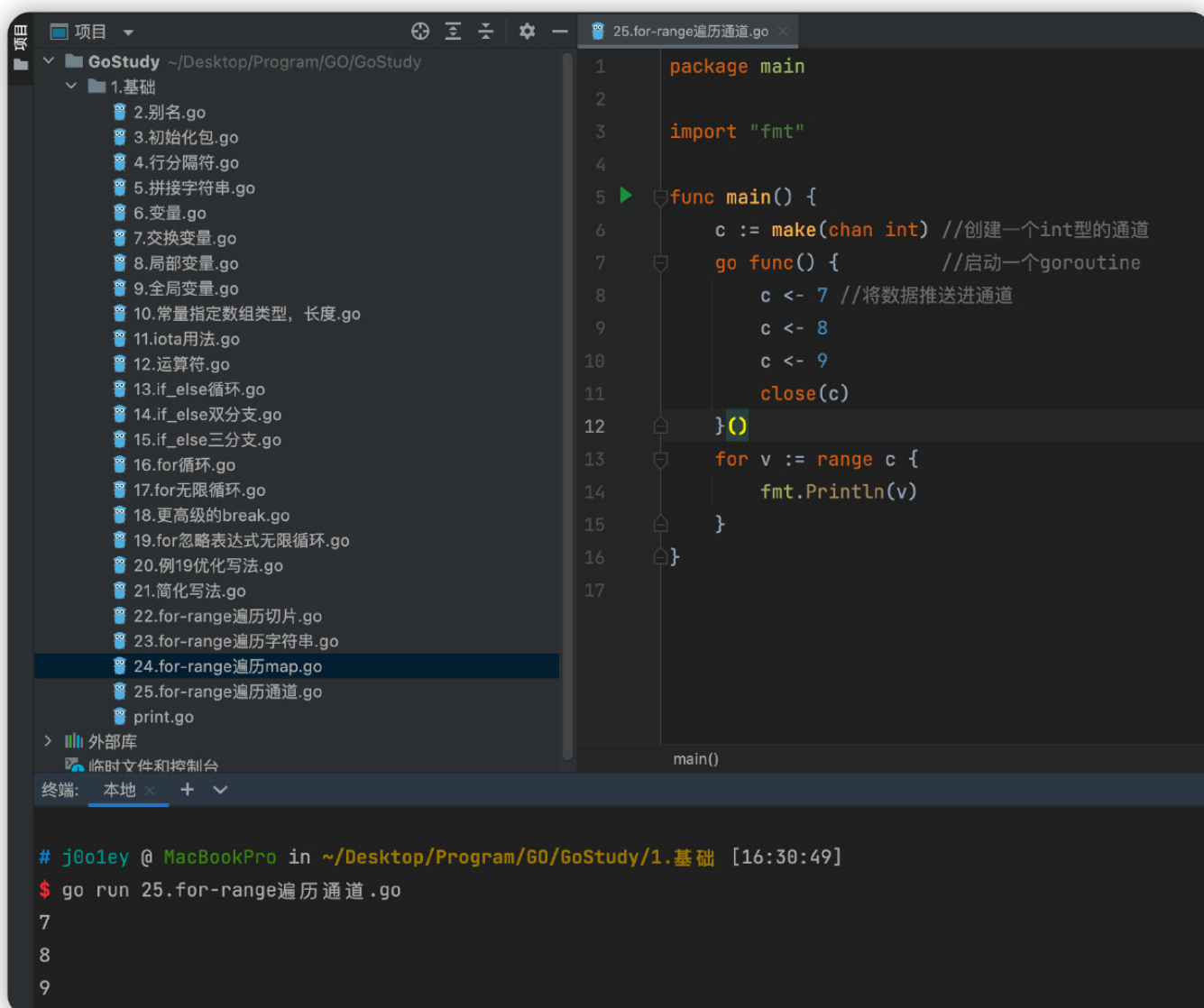
下面代码展示了通道的遍历方法

```
package main

import "fmt"

func main() {
    c := make(chan int) //创建一个int型的通道
    go func() {          //启动一个goroutine
        c <- 7           //将数据推进进通道
        c <- 8
        c <- 9
        close(c)
    }()
    for v := range c {
        fmt.Println(v)
    }
}
```

```
}  
}
```



The screenshot shows an IDE with a project named 'GoStudy' at the path '~/Desktop/Program/GO/GoStudy'. The file explorer on the left lists various Go files, with '25.for-range遍历通道.go' selected. The editor displays the following Go code:

```
1 package main  
2  
3 import "fmt"  
4  
5 func main() {  
6     c := make(chan int) //创建一个int型的通道  
7     go func() {         //启动一个goroutine  
8         c <- 7 //将数据推送进通道  
9         c <- 8  
10        c <- 9  
11        close(c)  
12    }()  
13    for v := range c {  
14        fmt.Println(v)  
15    }  
16 }  
17
```

The terminal at the bottom shows the command execution:

```
# j001ey @ MacBookPro in ~/Desktop/Program/GO/GoStudy/1.基础 [16:30:49]  
$ go run 25.for-range遍历通道.go  
7  
8  
9
```

代码逻辑为下

- 创建一个整型类型的通道并实例化
- 通过关键词go启动了一个goroutine
- 将数字传入通道，向通道推送数据7、8、9
- 结束并关闭隧道
- 使用for-range语句对通道c进行遍历，即不断地从通道中接收数据直至通道关闭

## 3.5 使用匿名变量

在使用for-range遍历某个对象时，往往不会同时使用key和value的值，而是只需要其中的一个值

此时可以通过匿名变量让代码变得更加简单



```

package main

import "fmt"

func main() {
    m := map[string]int{
        "fofa": 100,
        "shodan": 90,
    }
    for _, value := range m {
        fmt.Println(value)
    }
}

```

The screenshot shows an IDE with a project explorer on the left, a code editor in the center, and a terminal at the bottom. The project explorer lists files under '1. 基础' (Basic), with '26.使用匿名变量简化代码.go' (Using anonymous variables to simplify code.go) selected. The code editor displays the same Go code as the first block. The terminal shows the command 'go run 26.使用匿名变量简化代码.go' and its output: '100' and '90'.

```

package main

import "fmt"

func main() {
    m := map[string]int{
        "fofa": 100,
        "shodan": 90,
    }
    for _, value := range m {
        fmt.Println(value)
    }
}

```

```

# j0o1ey @ MacBookPro in ~/Desktop/Program/GO/GoStudy/1.基础 [16:55:33]
$ go run 26.使用匿名变量简化代码.go
100
90

```

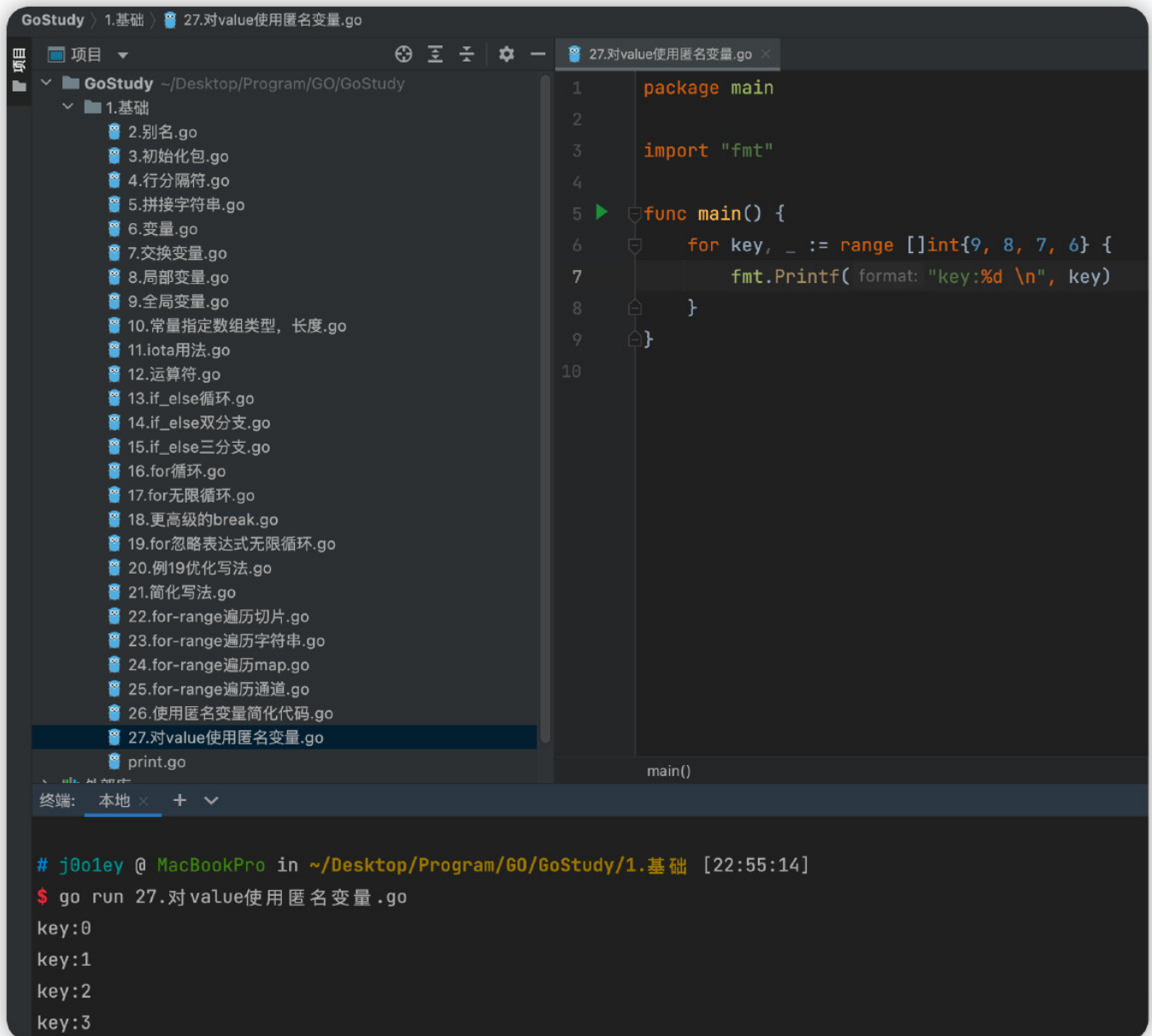
For-range中可以对key使用匿名变量，也可以对value使用匿名变量

Demo如下

```
package main

import "fmt"

func main() {
    for key, _ := range []int{9, 8, 7, 6} {
        fmt.Printf("key:%d \n ", key)
    }
}
```



The screenshot shows the GoStudy IDE interface. The left sidebar displays a project tree with a folder named '1.基础' containing various Go files. The file '27.对value使用匿名变量.go' is selected and open in the main editor. The code in the editor matches the code block above. The bottom terminal window shows the command 'go run 27.对 value使用匿名变量 .go' being executed, resulting in the output: 'key:0', 'key:1', 'key:2', and 'key:3'.

```
GoStudy > 1.基础 > 27.对value使用匿名变量.go
```

```
package main

import "fmt"

func main() {
    for key, _ := range []int{9, 8, 7, 6} {
        fmt.Printf("key:%d \n ", key)
    }
}
```

```
# j001ey @ MacBookPro in ~/Desktop/Program/GO/GoStudy/1.基础 [22:55:14]
$ go run 27.对 value使用匿名变量 .go
key:0
key:1
key:2
key:3
```