

## 0x01 switch-case使用

---

Go语言中switch case非常灵活，表达式的值不必为常量，甚至不必为整数

case从上到下进行求值，直至找到匹配项

可以将多个if-else改成switch-case语句，可读性更好

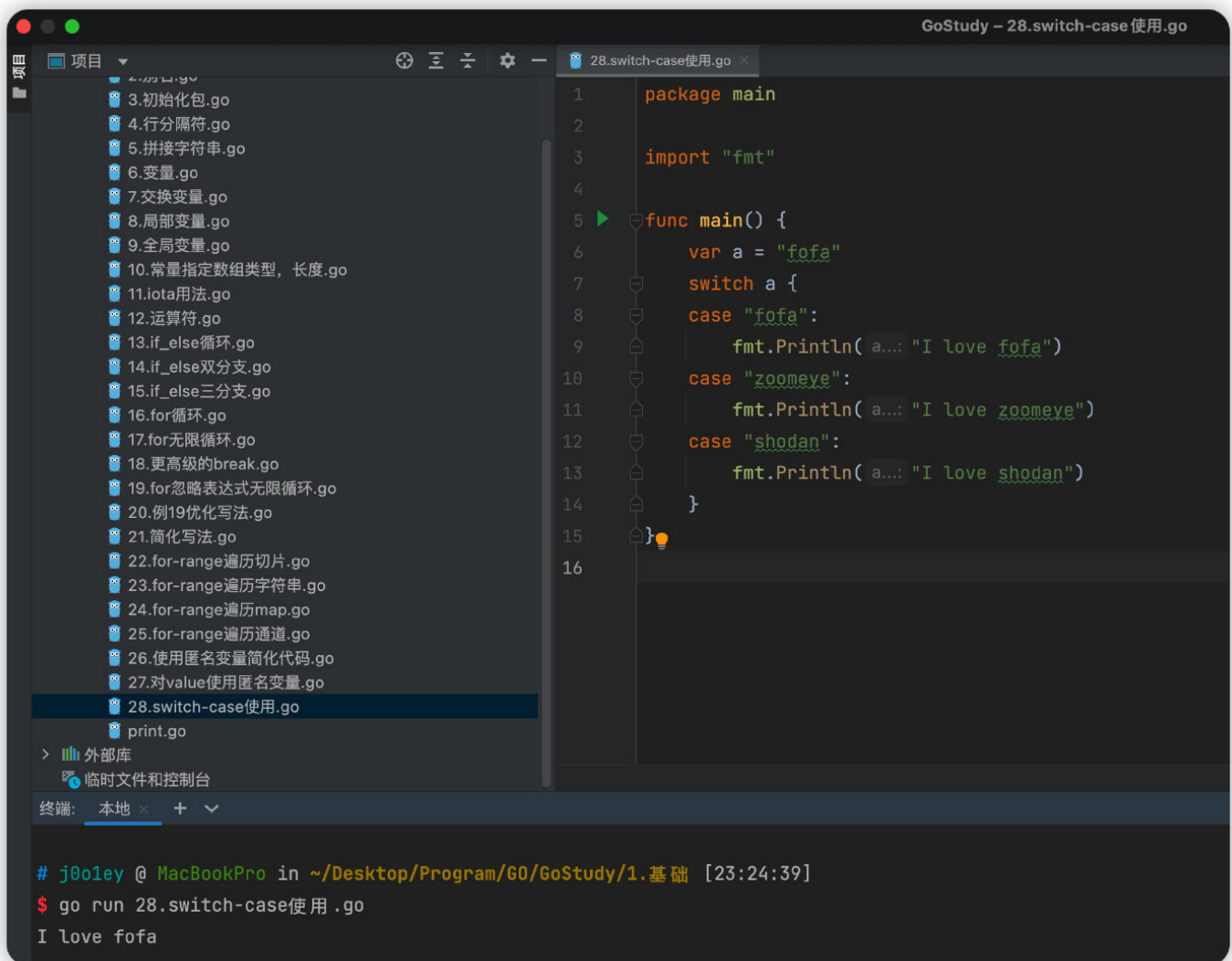
### Notice:

在Go语言中，case和case之间是独立的代码块，不需要通过break语句跳出当前的case代码块，避免执行到下一行

```
package main

import "fmt"

func main() {
    var a = "fofa"
    switch a {
    case "fofa":
        fmt.Println("I love fofa")
    case "zoomeye":
        fmt.Println("I love zoomeye")
    case "shodan":
        fmt.Println("I love shodan")
    }
}
```



上述例子，每一个case都是后接字符串，且使用了default分支。Go语言规定每个switch只能有一个default分支  
同时Go语言的case也支持一个分支多个值，多个表达式

## 1.1 一个分支多个值

如果要将多个case放在一起，可以使用如下写法

```
package main

import "fmt"

func main() {
    var language = "python"
    switch language {
    case "golang", "java", "python": //多个值
        fmt.Println("I love", language)
    }
}
```

The screenshot shows the GoStudy IDE interface. On the left, a file explorer lists various Go files, with '29.switch-分支多值.go' selected. The main editor displays the following Go code:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var language = "python"
7     switch language {
8     case "go", "java", "python":
9         fmt.Println("I love", language)
10    }
11 }
```

Below the editor, a terminal window shows the command execution:

```
# j0o1ey @ MacBookPro in ~/Desktop/Program/GO/GoStudy/1.基础 [0:17:07]
$ go run 29.switch-分支多值.go
I love python
```

在一个分支多个值的case表达式中，使用逗号分隔值

## 1.2 分支表达式

case语句后可以使用常量，也可以添加表达式

在这种情况下switch后不需要加用于判断的变量

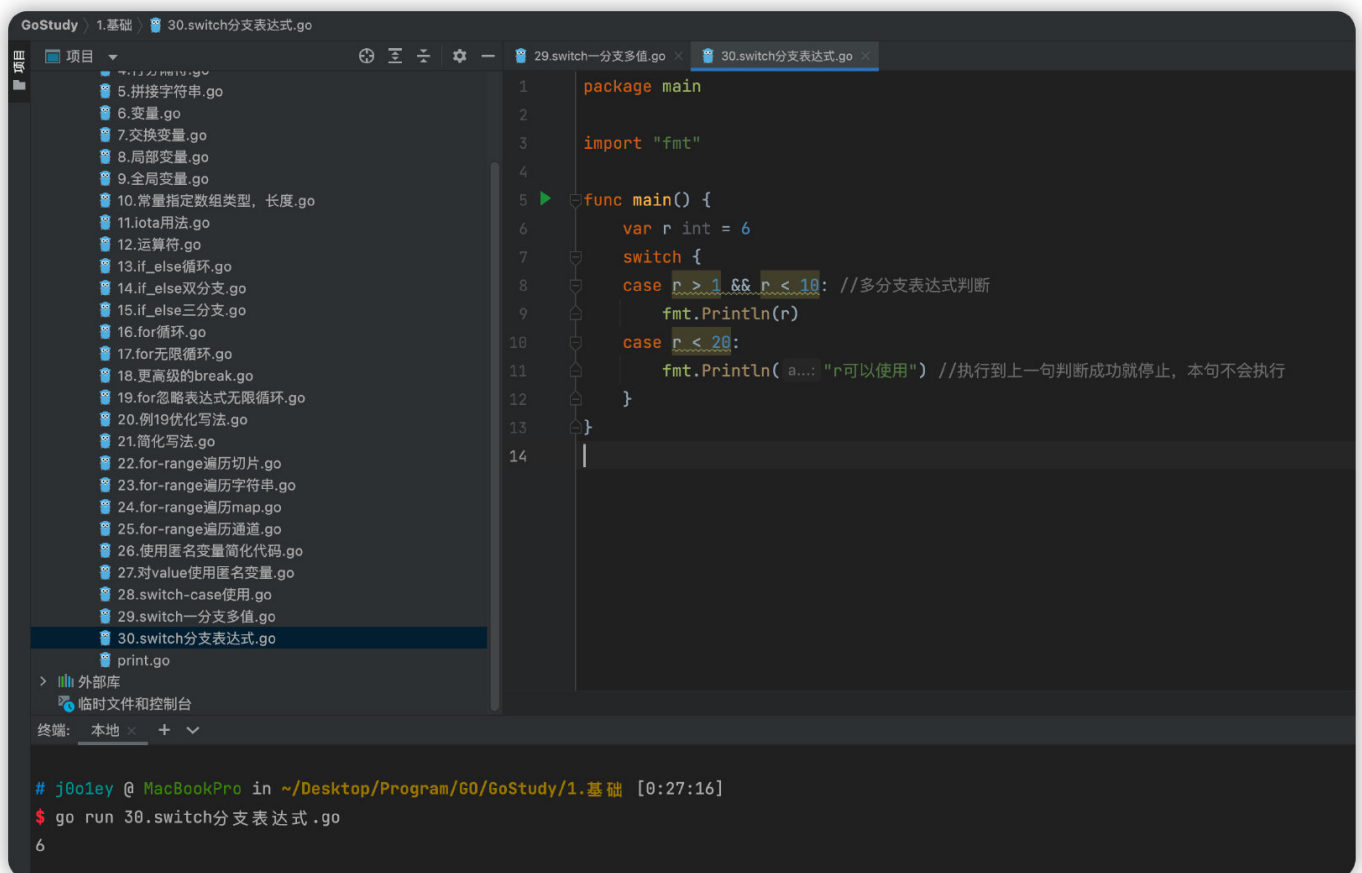
```

package main

import "fmt"

func main() {
    var r int = 6
    switch {
    case r > 1 && r < 10: //多分支表达式判断
        fmt.Println(r)
    case r < 20:
        fmt.Println("r可以使用") //执行到上一句判断成功就停止，本句不会执行
    }
}

```



The screenshot shows the GoStudy IDE with the following components:

- File Explorer (Left):** A list of files including '5.拼接字符串.go', '6.变量.go', '7.交换变量.go', '8.局部变量.go', '9.全局变量.go', '10.常量指定数组类型、长度.go', '11.iota用法.go', '12.运算符.go', '13.if\_else循环.go', '14.if\_else双分支.go', '15.if\_else三分支.go', '16.for循环.go', '17.for无限循环.go', '18.更高级的break.go', '19.for忽略表达式无限循环.go', '20.例19优化写法.go', '21.简化写法.go', '22.for-range遍历切片.go', '23.for-range遍历字符串.go', '24.for-range遍历map.go', '25.for-range遍历通道.go', '26.使用匿名变量简化代码.go', '27.对value使用匿名变量.go', '28.switch-case使用.go', '29.switch-分支多值.go', '30.switch分支表达式.go' (highlighted), and 'print.go'.
- Editor (Center):** Displays the Go code from the previous block, with line numbers 1 through 14. The code is identical to the one shown in the first block.
- Terminal (Bottom):** Shows the command prompt with the following text:
 

```

# j0o1ey @ MacBookPro in ~/Desktop/Program/G0/GoStudy/1.基础 [0:27:16]
$ go run 30.switch分支表达式.go
6

```

## 0x02 select-case使用

select 是 Go 中的一个控制结构，类似于用于通信的 switch 语句。每个 case 必须是一个通信操作，要么是发送要么是接收。

select语句是一种仅能用于channel发送和接收消息的专用语句，此语句运行期间是阻塞的；当select中没有case语句的时候，会阻塞当前的goroutine。

```

select {
    case communication clause :
        statement(s);
    case communication clause :
        statement(s);
    /* 你可以定义任意数量的 case */
    default : /* 可选 */
        statement(s);
}

```

## 0x02 goto语句

在Go语言中，可以通过goto语句跳转到标签，进行代码间无条件的跳转。

同时goto在快速跳出循环，避免重复退出等方面也有作用，使用goto可以简化一些代码的实现

例子如下：

在满足条件时，如果需要连续退出两层循环，传统的Coding代码如下

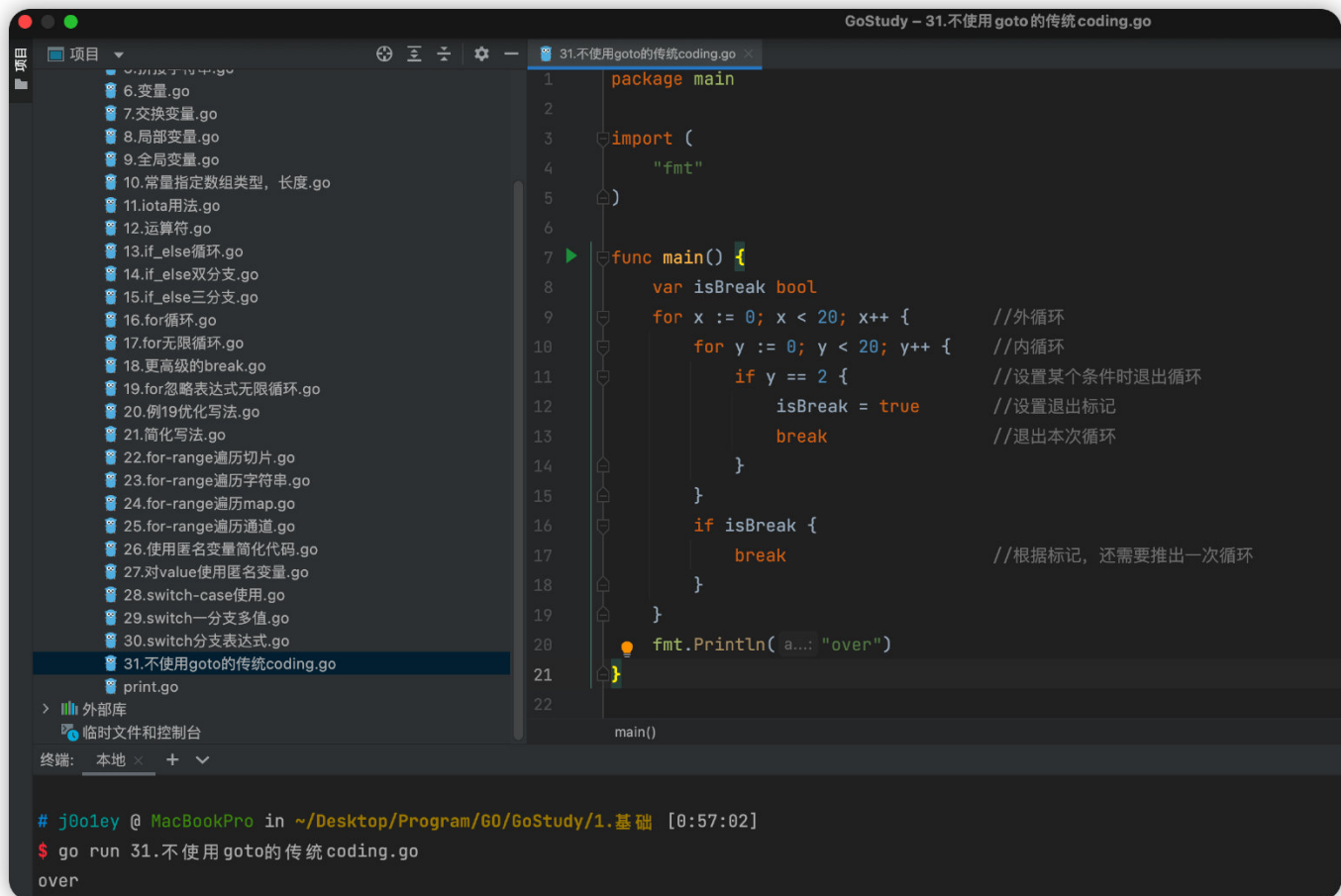
```

package main

import (
    "fmt"
)

func main() {
    var isBreak bool
    for x:=0; x<20 ; x++ {           //外循环
        for y := 0; y<20 ; y++ {     //内循环
            if y == 2 {              //设置某个条件时退出循环
                isBreak = true       //设置退出标记
                break                //退出本次循环
            }
        }
        if isBreak {                 //根据标记，还需要推出一次循环
            break
        }
    }
    fmt.Println("over")
}

```

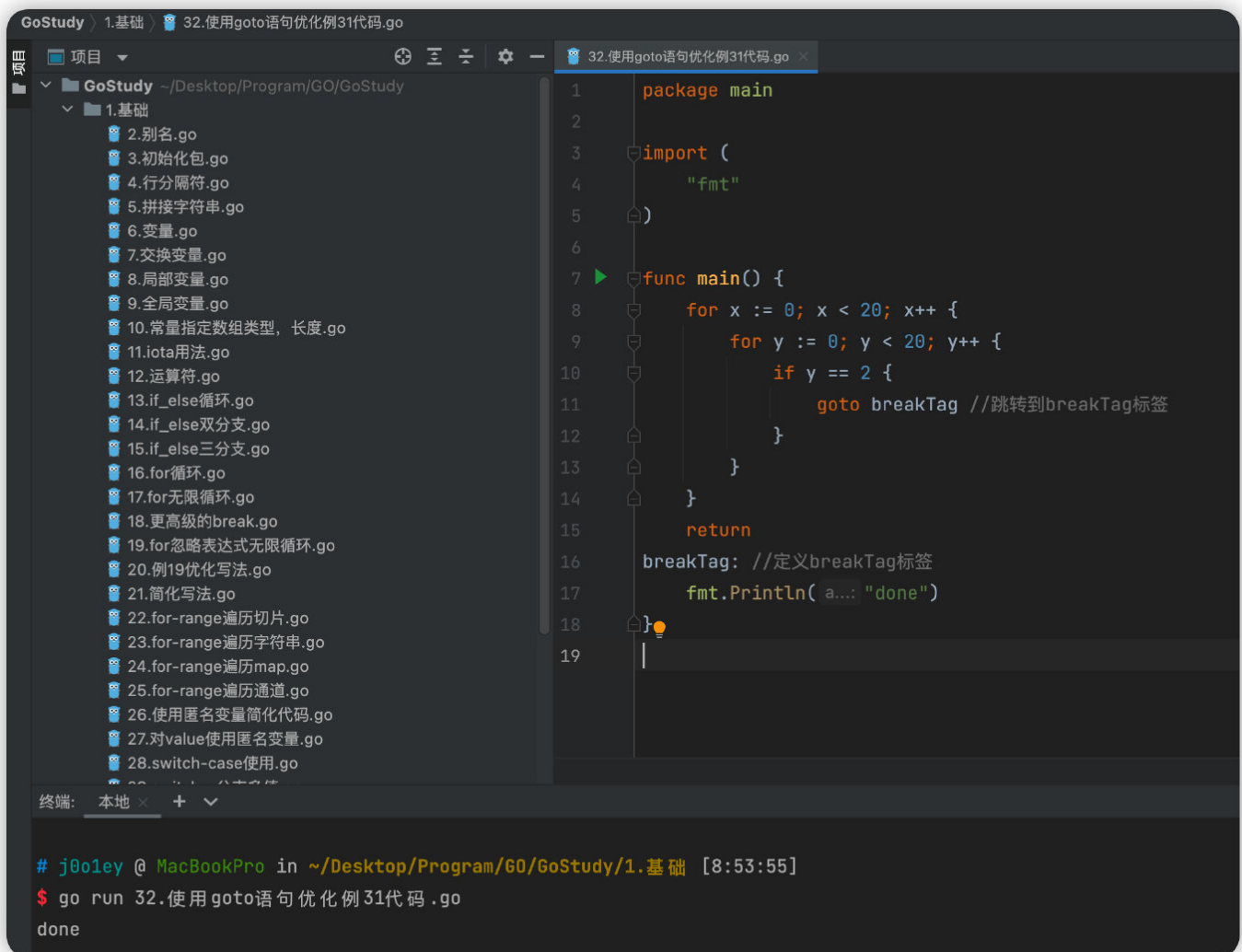


使用goto语句优化如上代码

```
package main

import (
    "fmt"
)

func main() {
    for x := 0; x < 20; x++ {
        for y := 0; y < 20; y++ {
            if y == 2 {
                goto breakTag //跳转到breakTag标签
            }
        }
    }
    return
breakTag: //定义breakTag标签
    fmt.Println("done")
}
```



如上代码使用goto语句 "goto breakTag"来跳转到指明的标签。

其中的标签只能被goto使用，不影响代码执行流程。

在定义breakTag标签之前有return语句，此处如果不手动返回，则会在不满足条件时执行breakTag代码

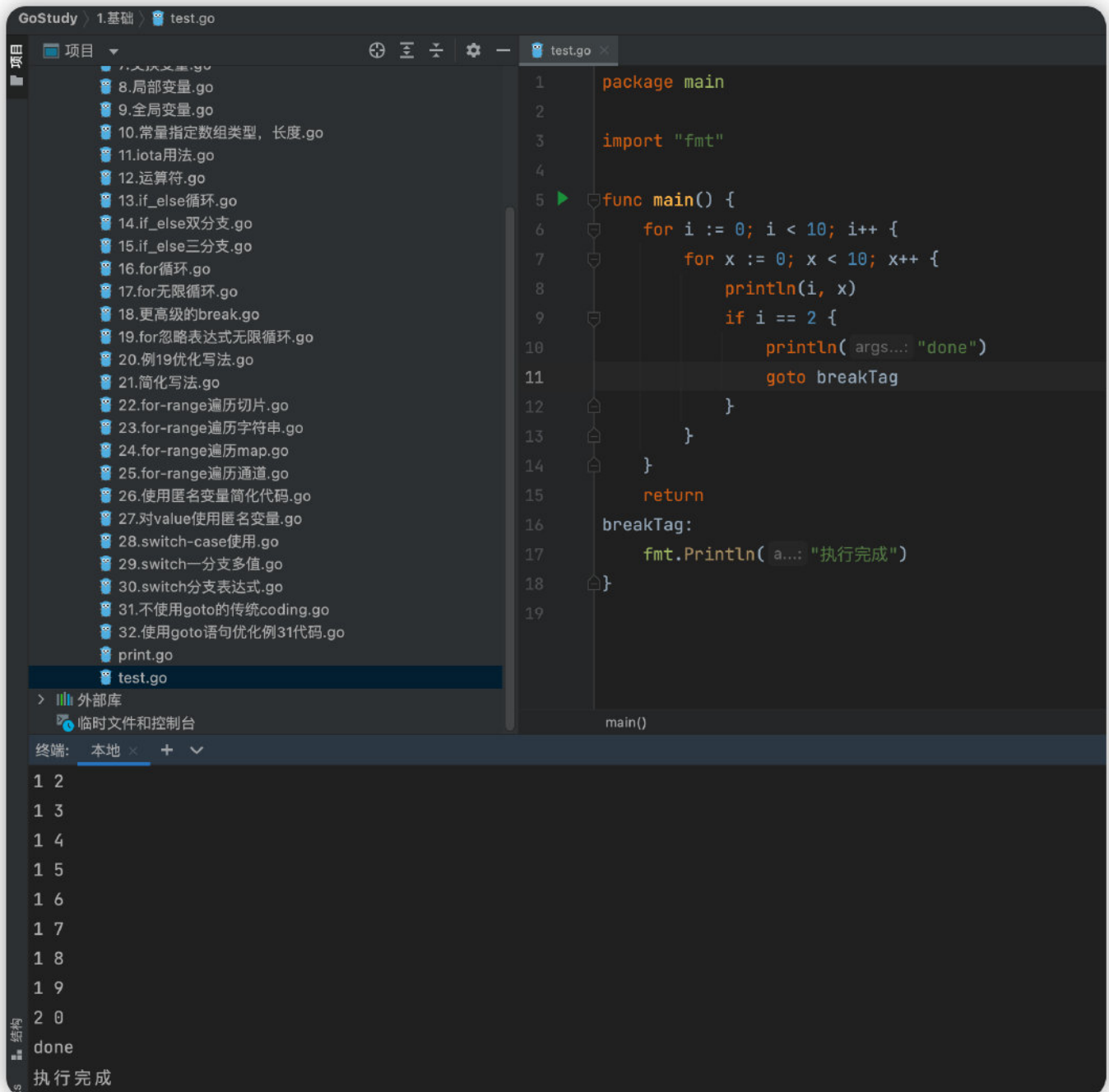
demo2

```
package main

import "fmt"

func main() {
    for i := 0; i < 10; i++ {
        for x := 0; x < 10; x++ {
            println(i, x)
            if i == 2 {
                println("done")
                goto breakTag
            }
        }
    }
    return
}
```

```
breakTag:
    fmt.Println("执行完成")
}
```



## 2.1 多错误处理

在日常开发时，经常会遇到多错误处理问题，在多处理问题中常常会遇到代码重复问题

```
package main

import "fmt"
```



```
func main() {
    //.....省略前面代码
    err := getUserInfo()
    if err != nil {
        fmt.Println(err)
        exitProcess()
        return
    }
    err = getEmail()
    if err != nil {
        fmt.Println(err)
        exitProcess()
        return
    }
    fmt.Println("over")
}
```

上述代码中有一部分是重复的代码，如果后期要加入更多的判断条件很容易造成错误，此时可以使用goto来进行优化

```
package main

import "fmt"

func main() {
    err := getUserInfo()
    if err != nil {
        goto doExit          //将跳转错误标签 onExit
    }
    err = getEmail()
    if err != nil {
        goto doExit          //将跳转错误标签 onExit
    }
    fmt.Println("over")
    return
doExit:                      //汇总所有流程进行错误打印并退出进程
    fmt.Println(err)
    exitProcess()
}
```

上述代码发生错误，将统一跳转到错误标签doExit，汇总所有流程，进行错误打印并退出进程

## 0x03 break语句

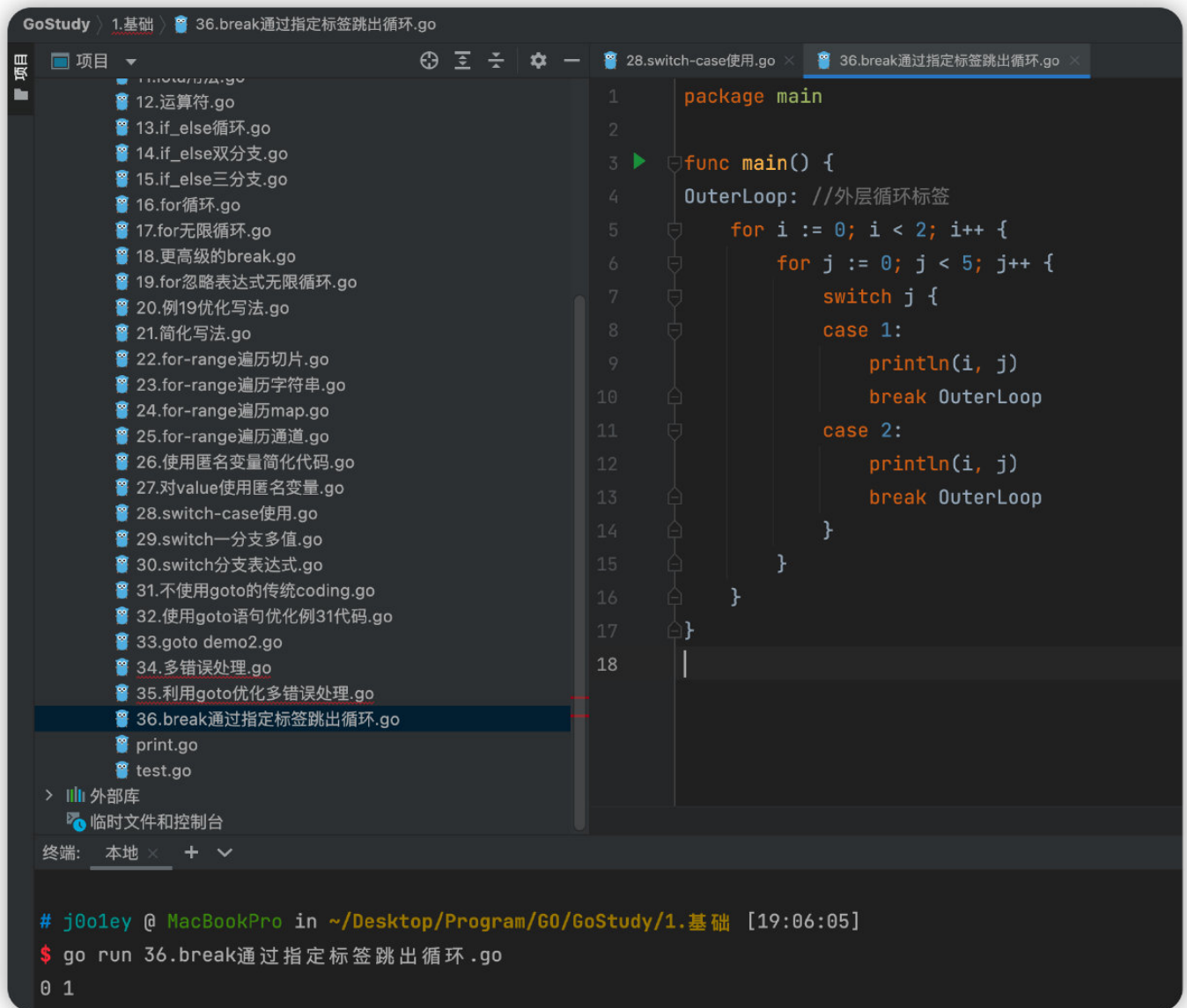
Go语言中的break可以结束for，switch和select的代码块。另外，还可以在break语句后加上标签，表示退出某个标签对应的代码块。

添加的标签必须定义在对应的for，switch和select代码块上

利用标签跳出循环

```
package main

func main() {
OuterLoop: //外层循环标签
    for i := 0; i < 2; i++ {
        for j := 0; j < 5; j++ {
            switch j {
            case 1:
                println(i, j)
                break OuterLoop
            case 2:
                println(i, j)
                break OuterLoop
            }
        }
    }
}
```



## 0x04 continue语句

在Go中，continue语句用于结束当前循环，并开始下一次的循环迭代过程

它仅限在for循环中使用

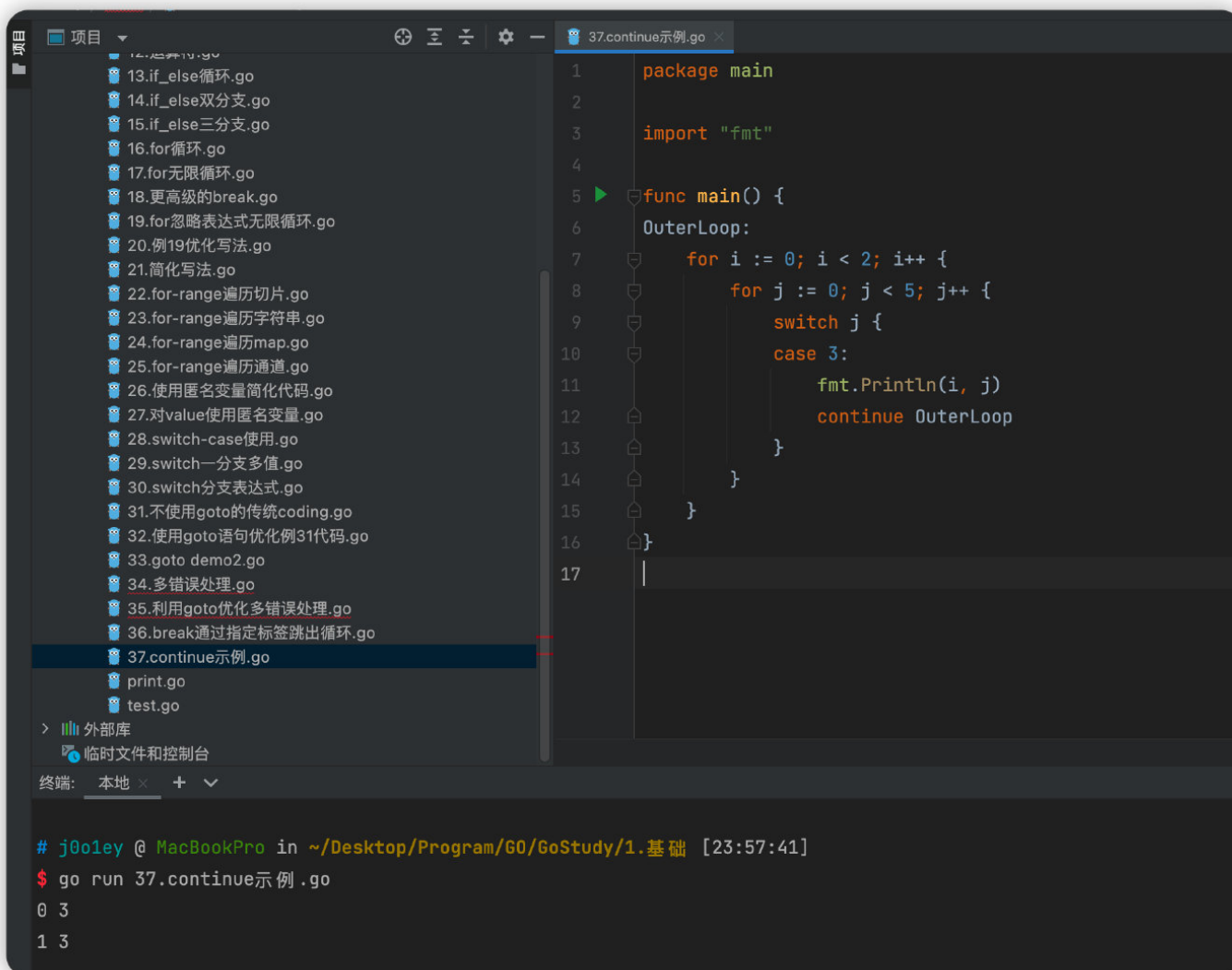
在continue语句后添加标签，表示结束标签对应语句的当前循环，并开启下一层循环，示例如下

```
package main

import "fmt"

func main() {
    OuterLoop:
    for i := 0; i < 2; i++ {
        for j := 0; j < 5; j++ {
            switch j {
            case 3:
                continue OuterLoop
            }
        }
    }
}
```

```
    fmt.Println(i, j)
    continue OuterLoop //结束当前循环，开启下一次的外层循环
}
}
}
```



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 various Go files, with '37.continue示例.go' selected. The code editor displays the following Go code:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     OuterLoop:
7         for i := 0; i < 2; i++ {
8             for j := 0; j < 5; j++ {
9                 switch j {
10                    case 3:
11                        fmt.Println(i, j)
12                        continue OuterLoop
13                    }
14                }
15            }
16        }
17    }
```

The terminal at the bottom shows the command to run the program and its output:

```
# j0o1ey @ MacBookPro in ~/Desktop/Program/G0/GoStudy/1.基础 [23:57:41]
$ go run 37.continue示例.go
0 3
1 3
```