Go 语言中的流程控制

主讲教师：（大地）
合作网站：www.itying.com （IT 营）
我的专栏：https://www.itying.com/category-79-b0.html

1、Golang 中的流程控制

流程控制是每种编程语言控制逻辑走向和执行次序的重要部分，流程控制可以说是一门语言的“经脉”。

Go 语言中最常用的流程控制有 if 和 for，而 switch 和 goto 主要是为了简化代码、降低重复代码而生的结构，属于扩展类的流程控制。

2、if else(分支结构)

1、if 条件判断基本写法

Go 语言中 if 条件判断的格式如下：

```
if 表达式 1 {
    分支 1
} else if 表达式 2 {
```

```
    分支 2
} else{
    分支 3
}
```

当表达式 1 的结果为 true 时，执行分支 1，否则判断表达式 2，如果满足则执行分支 2，都不满足时，则执行分支 3。 if 判断中的 else if 和 else 都是可选的，可以根据实际需要进行选择。

注意：Go 语言规定与 if 匹配的左括号{必须与 if 和表达式放在同一行，{放在其他位置会触发编译错误。 同理，与 else 匹配的{也必须与 else 写在同一行，else 也必须与上一个 if 或 else if 右边的大括号在同一行。

举个例子：

```
func ifDemo1() {
    score := 65
    if score >= 90 {
        fmt.Println("A")
    } else if score > 75 {
        fmt.Println("B")
    } else {
        fmt.Println("C")
    }
}
```

2、if 条件判断特殊写法

if 条件判断还有一种特殊的写法，可以在 if 表达式之前添加一个执行语句，再根据变量值进行判断，举个例子：

```
if score := 56; score >= 90 {
    fmt.Println("A")
} else if score > 75 {
    fmt.Println("B")
} else {
    fmt.Println("C")
}
```

思考题： 上下两种写法的区别在哪里？

```
package main
import "fmt"
func main() {
    //这里的 score 是局部作用域
    if score := 56; score >= 90 {
        fmt.Println("A")
    } else if score > 75 {
        fmt.Println("B")
    } else {
        fmt.Println("C")
    }
    fmt.Println(score) //undefined: score
}
```

练习：求两个数的最大值 (注意 go 语言中没有三目运算)

```
    var n1 int = 20
    var n2 int = 55
    var max int
    if n1 > n2 {
        max = n1
    } else {
        max = n2
    }
    fmt.Println("max=", max)
```

练习：判断一个人的成绩 小于 60 分不及格 大于等于 60 小于 80 及格，大于 80 优秀

3、for(循环结构)

Go 语言中的所有循环类型均可以使用 for 关键字来完成。

for 循环的基本格式如下：

```
for 初始语句;条件表达式;结束语句{
    循环体语句
}
```

条件表达式返回 true 时循环体不停地进行循环，直到条件表达式返回 false 时自动退出循环。

```
for i := 0; i < 10; i++ {
        fmt.Println(i)
}
```

for 循环的初始语句可以被忽略，但是初始语句后的分号必须要写，例如：

```
i := 0
for ; i < 10; i++ {
    fmt.Println(i)
}
```

for 循环的初始语句和结束语句都可以省略，例如：

```
i := 0
for i < 10 {
    fmt.Println(i)
    i++
}
```

这种写法类似于其他编程语言中的 while，在 while 后添加一个条件表达式，满足条件表达式时持续循环，否则结束循环。

注意：Go 语言中是没有 while 语句的，我们可以通过 for 代替

for 无限循环

```
for {
    循环体语句
}
```

for 循环可以通过 break、goto、return、panic 语句强制退出循环。

```
    k := 1
    for { // 这里也等价 for ; ; {
        if k <= 10 {
            fmt.Println("ok~~", k)
        } else {
            break //break 就是跳出这个 for 循环
        }
        k++
    }
```

练习：打印 0-50 所有的偶数

```
for i := 0; i < 50; i++ {
    if i%2 == 0 {
        fmt.Println(i)
    }
}
```

练习：打印 1~100 之间所有是 9 的倍数的整数的个数及总和

```
count := 0
sum := 0
for i := 1; i < 100; i++ {
    if i%9 == 0 {
        sum += i
        count++
    }
}
fmt.Println("count=",count)
fmt.Println("sum=",sum)
```

练习：求 1+2+3+4 +...100 的和

```
sum := 0
for i := 0; i <= 100; i++ {
    sum += i
}
fmt.Println("sum=", sum)
```

练习：计算 5 的阶乘 (12345 n 的阶乘 12……n)

```
var n = 5
sum := 1
for i := 1; i <= n; i++ {
    sum *= i
}
fmt.Println("sum=", sum)
```

练习： 打印一个矩形

```
    for i := 1; i <= 12; i++ {

        fmt.Print("*")
        if i%4 == 0 {
            fmt.Println()
        }
    }
```

嵌套循环解决这个问题

```
    line := 4
    num := 8
    for i := 1; i <= line; i++ {
        for j := 0; j < num; j++ {
            fmt.Print("*")
        }
        fmt.Println()
    }
```

练习： 打印一个三角形

```
*
**
***
****
*****
```

```
    line := 5
    for i := 1; i <= line; i++ {
        for j := 0; j < i; j++ {
            fmt.Print("*")
        }
        fmt.Println()
    }
```

练习：打印出九九乘法表

```
1x1=1
2x1=2    2x2=4
3x1=3    3x2=6    3x3=9
4x1=4    4x2=8    4x3=12   4x4=16
5x1=5    5x2=10   5x3=15   5x4=20   5x5=25
6x1=6    6x2=12   6x3=18   6x4=24   6x5=30   6x6=36
7x1=7    7x2=14   7x3=21   7x4=28   7x5=35   7x6=42   7x7=49
8x1=8    8x2=16   8x3=24   8x4=32   8x5=40   8x6=48   8x7=56   8x8=64
9x1=9    9x2=18   9x3=27   9x4=36   9x5=45   9x6=54   9x7=63   9x8=72   9x9=81
```

```
for i := 1; i <= 9; i++ {
    for j := 1; j <= i; j++ {
        fmt.Printf("%vx%v=%v \t", i, j, i*j)
    }
    fmt.Println()
}
```

4、for range(键值循环)

Go 语言中可以使用 for range 遍历数组、切片、字符串、map 及通道（channel）。 通过 for range 遍历的返回值有以下规律：

1. 数组、切片、字符串返回索引和值。
2. map 返回键和值。
3. 通道（channel）只返回通道内的值。

```
    str := "abc 上海"
    for index, val := range str {
        fmt.Printf("index=%d, val=%c \n", index, val)
    }
    str := "abc 上海"
    for _, val := range str {
        fmt.Printf("val=%c \n", val)
    }
```

5、switch case

使用 switch 语句可方便地对大量的值进行条件判断。

练习：判断文件类型,如果后缀名是.html 输入 text/html，如果后缀名.css 输出 text/css ,如果后缀名是.js 输出 text/javascript

Go 语言规定每个 switch 只能有一个 default 分支。

```
extname := ".a"
switch extname {
    case ".html":
        fmt.Println("text/html")
        break
    case ".css":
        fmt.Println("text/css")
        break
    case ".js":
        fmt.Println("text/javascript")
        break
    default:
        fmt.Println("格式错误")
        break
}
```

Go 语言中每个 case 语句中可以不写 break，不加 break 也不会出现穿透的现象 **如下例子：**

```
extname := ".a"
    switch extname {
    case ".html":
        fmt.Println("text/html")
    case ".css":
        fmt.Println("text/css")
    case ".js":
        fmt.Println("text/javascript")
    default:
        fmt.Println("格式错误")
    }
```

一个分支可以有多个值，多个 case 值中间使用英文逗号分隔。

```
    n := 2
    switch n {
    case 1, 3, 5, 7, 9:
        fmt.Println("奇数")
    case 2, 4, 6, 8:
        fmt.Println("偶数")
    default:
        fmt.Println(n)
    }
```

另一种写法：

```
switch n := 7; n {
    case 1, 3, 5, 7, 9:
        fmt.Println("奇数")
    case 2, 4, 6, 8:
        fmt.Println("偶数")
    default:
        fmt.Println(n)
    }
```

注意： 上面两种写法的作用域

分支还可以使用表达式，这时候 switch 语句后面不需要再跟判断变量。例如：

```
age := 56
    switch {
    case age < 25:
        fmt.Println("好好学习吧！")
    case age > 25 && age <= 60:
        fmt.Println("好好工作吧！")
    case age > 60:
        fmt.Println("好好享受吧！")
    default:
        fmt.Println("活着真好！")
}
```

switch 的穿透 fallthrought

fallthrough`语法可以执行满足条件的 case 的下一个 case，是为了兼容 C 语言中的 case 设计的。

```
func switchDemo5() {
    s := "a"
    switch {
    case s == "a":
        fmt.Println("a")
        fallthrough
    case s == "b":
        fmt.Println("b")
    case s == "c":
        fmt.Println("c")
    default:
        fmt.Println("...")
    }
}
```

输出：

```
a
b
```

```
var num int = 10
    switch num {
    case 10:
        fmt.Println("ok1")
        fallthrough //默认只能穿透一层
    case 20:
        fmt.Println("ok2")
        fallthrough
    case 30:
        fmt.Println("ok3")
    default:
        fmt.Println("没有匹配到..")
    }
```

输出：

```
ok1
ok2
ok3
```

6、break(跳出循环)

Go 语言中 break 语句用于以下几个方面：

- 用于循环语句中跳出循环，并开始执行循环之后的语句。
- break 在 switch（开关语句）中在执行一条 case 后跳出语句的作用。
- 在多重循环中，可以用标号 label 标出想 break 的循环。

1、 switch（开关语句）中在执行一条 case 后跳出语句的作用。

```
extname := ".a"
switch extname {
    case ".html":
        fmt.Println("text/html")
        break
    case ".css":
        fmt.Println("text/css")
        break
    case ".js":
        fmt.Println("text/javascript")
        break
    default:
        fmt.Println("格式错误")
        break
}
```

2、 for 循环中默认 break 只能跳出一层循环

```
package main
import "fmt"

func main() {

    for i := 0; i < 2; i++ {
```

```
        for j := 0; j < 10; j++ {
            if j == 2 {
                break
            }
            fmt.Println("i j 的值", i, "-", j)
        }
    }
}
```

```
    k := 1
    for { // 这里也等价 for ; ; {
        if k <= 10 {
            fmt.Println("ok~~", k)
        } else {
            break //break 就是跳出这个 for 循环
        }
        k++
    }
```

3、 在多重循环中，可以用标号 label 标出想 break 的循环。

```
    package main

    import "fmt"

    func main() {
    lable2:
        for i := 0; i < 2; i++ {
            for j := 0; j < 10; j++ {
                if j == 2 {
                    break lable2
                }
                fmt.Println("i j 的值", i, "-", j)

            }

        }

    }
```

7、continue(继续下次循环)

continue 语句可以结束当前循环，开始下一次的循环迭代过程，仅限在 for 循环内使用。

```
package main
import "fmt"
func main() {
    for i := 0; i < 2; i++ {
        for j := 0; j < 4; j++ {
            if j == 2 {
                continue

            }
            fmt.Println("i j 的值", i, "-", j)
        }
    }
}
```

输出：

```
d:\golang\src\demo01>go run main.go

i j 的值 0 - 0

i j 的值 0 - 1

i j 的值 0 - 3

i j 的值 1 - 0

i j 的值 1 - 1

i j 的值 1 - 3
```

在 continue 语句后添加标签时，表示开始标签对应的循环。例如：

```
package main
```

```
import "fmt"

func main() {
here:
    for i := 0; i < 2; i++ {
        for j := 0; j < 4; j++ {
            if j == 2 {
                continue here
            }
            fmt.Println("i j 的值", i, "-", j)
        }

    }

}


d:\golang\src\demo01>go run main.go

i j 的值 0 - 0

i j 的值 0 - 1

i j 的值 1 - 0

i j 的值 1 - 1
```

8、goto(跳转到指定标签)

goto 语句通过标签进行代码间的无条件跳转。goto 语句可以在快速跳出循环、避免重复退出上有一定的帮助。Go 语言中使用 goto 语句能简化一些代码的实现过程。

```
package main
import "fmt"
func main() {
    var n int = 30
    fmt.Println("ok1")
    if n > 20 {
        goto label1
    }
```

```
        fmt.Println("ok2")
        fmt.Println("ok3")
        fmt.Println("ok4")

    label1:

        fmt.Println("ok5")
        fmt.Println("ok6")
        fmt.Println("ok7")

    }
```

输出结果：

```
d:\golang\src\demo01>go  run  main.go
ok1
ok5
ok6
ok7
```

使用 goto 语句能简化代码：

```
package  main

import  "fmt"

func  main() {
    for  i  :=  0;  i  <  10;  i++  {
        for  j  :=  0;  j  <  10;  j++  {
            if  j  ==  2  {
                //  设置退出标签
                goto  breakTag
            }
            fmt.Printf("%v-%v\n",  i,  j)
        }
    }
    return
    //  标签
breakTag:
    fmt.Println("结束 for 循环")
}
```

输出结果：

```
d:\golang\src\demo01>go  run  main.go
0-0
0-1
```