

Go流程控制——循环语句

目录：

- 1. Go语言中循环语句概述
- 2. for循环语句及其多种语法形式
- 3. for嵌套循环语句
- 4. 循环控制语句（break、continue、goto）

一、Go 语言循环语句

（一）、概述

1、在不少实际问题中有许多具有规律性的重复操作，因此在程序中就需要重复执行某些语句。循环语句包括循环处理语句及循环控制语句。

2、循环处理语句有：

循环类型	
for 循环	重复执行语句块
循环嵌套	在 for 循环中嵌套一个

3、循环控制语句可以控制循环体内语句的执行过程。循环控制语句有：

控制语句	
break 语句	经常用于中断当前 for
continue 语句	跳过当前循环的剩余语
goto 语句	将控制转移到被标记的

二、for循环语句

- 循环语句表示条件满足，可以反复的执行某段代码。
- for是Go语言中唯一的循环语句，Go没有while、do...while循环。
- 按语法结构来分，Go语言的for循环有4种形式，只有其中第一种使用分号。
- for循环中for关键字后不能加小括号。

(一)、语法形式一（for关键字后有三个表达式——基本for循环语法结构）

1、语法结构：

```
for 初始语句init; 条件表达式condition; 结束语句post {  
    //循环体代码  
}
```

- 三个组成部分，即初始化、条件表达式和post都是可选的。
- 因此这种基本的for循环语法结构又能演化出四种略有不同的写法。

2、示例代码

```
for i := 0; i <= 10; i++ {  
    fmt.Printf("%d ", i)  
}
```

3、语法解释

(1)、初始语句init：

- 初始语句是在第一次循环前执行的语句，一般为赋值表达式，给控制变量赋初始值。
- 如果控制变量在此处被声明，其作用域将被局限在这个for的范围内；在for循环中声明的变量仅在循环范围内可用。
- 初始语句可以省略不写，但是初始语句之后的分号必须要写。

```
i := 0  
for ; i <= 10; i++ {  
    fmt.Printf("%d ", i)  
}
```

(2)、条件表达式condition:

- 条件表达式是控制循环与否的开关;
- 如果表达式为true, 则循环继续, 否则结束循环;
- 条件表达式可以省略不写, 之后的分号必须要写;
- 省略条件表达式默认形成无限循环。

```
i := 0
for;; i++ {
    if (i > 20) {
        break
    }
    fmt.Printf("%d ", i)
}
```

(3)、结束语句post:

- 一般为赋值表达式, 给控制变量递增或者递减;
- post语句将在循环的每次成功迭代之后执行。

4、for语句执行过程如下:

- 先执行初始化语句, 对控制变量赋初始值。初始化语句只执行一次。
- 其次根据控制变量判断条件表达式的返回值, 若其值为true, 满足循环条件, 则执行循环体内语句, 之后执行 post语句, 开始下一次循环。
- 执行post语句之后, 将重新计算条件表达式的返回值, 如果是true, 循环将继续执行, 否则循环终止。然后执行循环体外语句。

(二)、语法形式二 (for关键字后只有一个条件表达式)

1、语法结构

- for 循环条件condition { }
- 效果类似其它编程语言中的while循环

2、示例代码

```
var i int
for i <= 10 {
    fmt.Println(i)
    i++
}
```

```
}
```

(三)、语法形式三 (for关键字后无表达式)

1、语法结构

- for { }
- 效果与其它编程语言的for(;;) {}一致，此时for执行无限循环

2、示例代码

```
var i int
for {
    if (i > 10) {
        break
    }
    fmt.Println(i)
    i++
}
```

(四)、语法形式四 (for ... range)

1、for 循环的 range 格式

- 对字符串、slice、数组、map等进行迭代循环

2、语法结构

```
for key, value := range oldMap {
    newMap[key] = value
}
```

3、案例：遍历字符串，获得字符

//遍历字符串，获得字符

```
func traverseString() {
    str := "123ABCabc一丁丂"
    for i, value := range str {
        fmt.Printf("第 %d 位的ASCII值=%d , 字符是%c \n", i, value ,value)
    }
}
```

```
}
```

打印结果：

第 0 位的ASCII值=49，字符是1
第 1 位的ASCII值=50，字符是2
第 2 位的ASCII值=51，字符是3
第 3 位的ASCII值=65，字符是A
第 4 位的ASCII值=66，字符是B
第 5 位的ASCII值=67，字符是C
第 6 位的ASCII值=97，字符是a
第 7 位的ASCII值=98，字符是b
第 8 位的ASCII值=99，字符是c
第 9 位的ASCII值=19968，字符是—
第 12 位的ASCII值=19969，字符是丁
第 15 位的ASCII值=19970，字符是丂

4、案例：遍历切片中元素

```
func traverseSlice() {  
    arr := []int{100, 200, 300}  
    for i, value := range arr {  
        fmt.Println(i, ":", value)  
    }  
}
```

（五）、for循环案例代码：

1、求1-100 的和

```
func summation() {  
    sum := 0  
    for i := 1; i <= 100; i++ {  
        sum += i;  
    }  
    fmt.Println(sum)  
}
```

2、求1-100之间3的倍数的和

```
func summation2() {  
    i := 1
```

```

sum := 0
for i <= 100 {
    if i%3 == 0 {
        sum += i
        fmt.Print(i)
        if i < 99 {
            fmt.Print("+")
        } else {
            fmt.Printf(" = %d \n", sum)
        }
    }
    i++
}
}

```

3、截竹竿。32米竹竿，每次截1.5米，最快截几次之后能小于4米？

```

func cutBamboo() {
    count := 0;
    for i := 32.0; i >= 4; i -= 1.5 {
        count++;
    }
    fmt.Println(count)
}

```

三、for嵌套循环语句

- Go 语言允许在循环体内使用循环。

（一）、语法结构

```

for [condition | ( init; condition; increment ) | Range] {
    for [condition | ( init; condition; increment ) | Range] {
        statement(s);
    }
    statement(s);
}

```

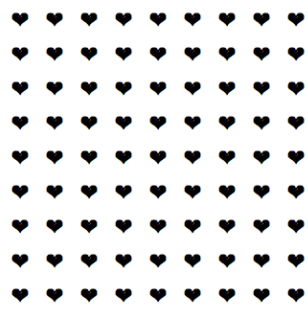
```
}
```

(二)、案例代码

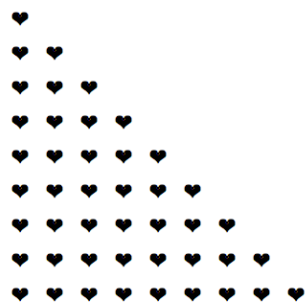
1、打印直角三角形

```
func printRightTriangle() {  
    // 定义行数  
    lines := 8  
    for i := 0; i < lines; i++ {  
        for n := 0; n < 2*i+1; n++ {  
            fmt.Print("* ");  
        }  
        fmt.Println()  
    }  
}
```

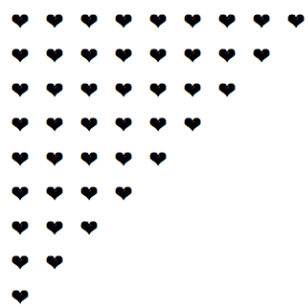
打印矩形



打印左下直角三角形



打印左上直角三角形



打印右下直角三角形

```

      ♥
     ♥♥
    ♥♥♥
   ♥♥♥♥
  ♥♥♥♥♥
 ♥♥♥♥♥♥
♥♥♥♥♥♥♥

```

打印右上直角三角形

```

♥♥♥♥♥♥♥♥♥♥
 ♥♥♥♥♥♥♥♥♥
  ♥♥♥♥♥♥♥♥
   ♥♥♥♥♥♥♥
    ♥♥♥♥♥♥
     ♥♥♥♥♥
      ♥♥♥♥
       ♥♥♥
        ♥♥
         ♥
          ♥

```

打印等腰三角形

```

      ♥
     ♥♥♥
    ♥♥♥♥♥
   ♥♥♥♥♥♥♥
  ♥♥♥♥♥♥♥♥♥
 ♥♥♥♥♥♥♥♥♥♥♥
♥♥♥♥♥♥♥♥♥♥♥♥♥

```

2、打印等腰三角形

```
func printTriangle() {
    for i := 1; i <= 10; i++ {
        //控制每一行符号前的空格的数量
        for m := 10; m > i; m-- {
            fmt.Print(" ")
        }
        //控制每一行符号的数量
        for j := 1; j <= 2*i-1; j++ {
            fmt.Print("♥ ")
        }
        fmt.Println()
    }
}
```

```

    }
}

```

3、打印九九乘法表

```

1*1=1
1*2=2 2*2=4
1*3=3 2*3=6 3*3=9
1*4=4 2*4=8 3*4=12 4*4=16
1*5=5 2*5=10 3*5=15 4*5=20 5*5=25
1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81

```

```

func multiply99() {
    for i := 1; i <= 9; i++ { // i 控制行数，是乘法中的第二个数。
        for j := 1; j <= i; j++ { // j 控制每行的列数，是乘法中的第一个数。
            fmt.Printf("%d*%d=%d ", j, i, i*j);
        }
        fmt.Println()
    }
}

```

4、使用循环嵌套来输出 2 到 100 间的素数：

```

func printPrimeNumber() {
    /* 定义局部变量 */
    fmt.Print("1-100的素数：")
    var a, b int
    for a = 2; a <= 100; a++ {
        for b = 2; b <= (a / b); b++ {
            if a%b == 0 {
                break // 如果发现因子，则不是素数
            }
        }
        if b > (a / b) {
            fmt.Printf("%d\t", a)
        }
    }
}

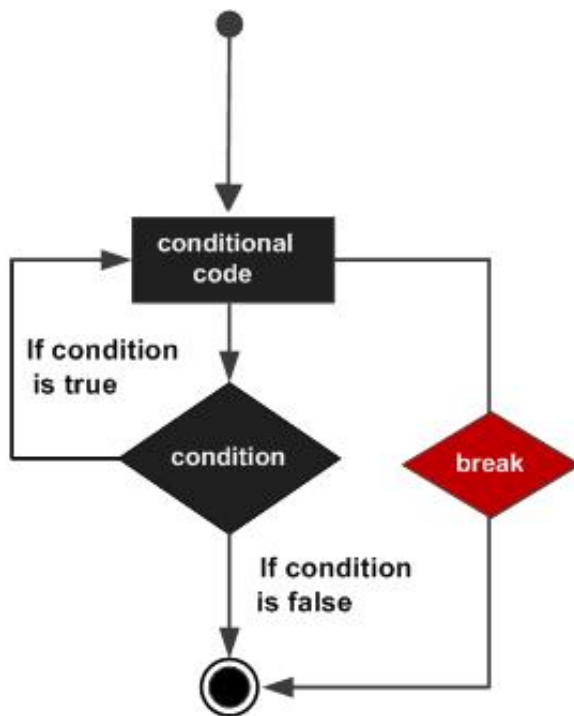
```

```
}  
}
```

四、循环控制语句

(一)、break语句

1、break：跳出循环体。break语句用于在结束其正常执行之前突然终止for循环，并开始执行循环之后的语句。

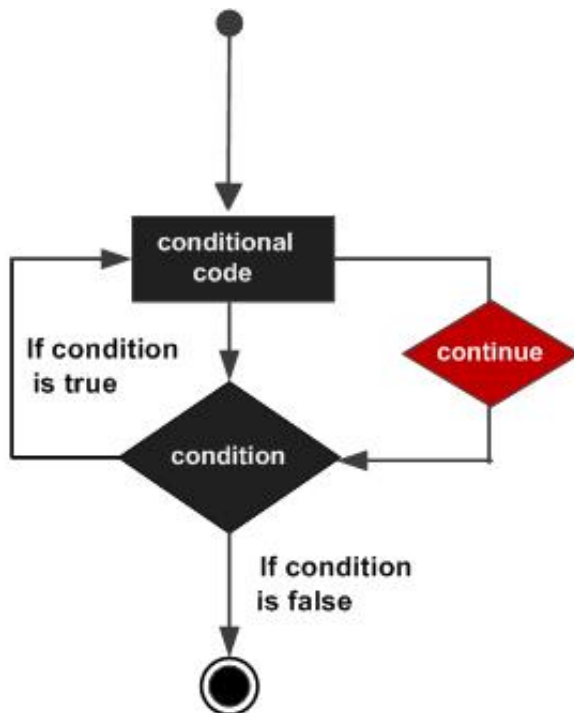


2、示例代码：

```
func main() {  
    for i := 1; i <= 10; i++ {  
        if i > 5 {  
            break // 如果i > 5，则循环终止 (loop is terminated)  
        }  
        fmt.Printf("%d ", i)  
    }  
    fmt.Printf("\nline after for loop")  
}
```

(二)、continue语句

1、Go 语言的 continue 语句 有点像 break 语句。但是 continue 不是跳出循环，而是跳过当前循环执行下一次循环语句。for 循环中，执行 continue 语句会触发 for 增量语句的执行。换句话说，continue 语句用于跳过 for 循环的当前迭代，循环将继续到下一个迭代。



2、示例代码：

```
func main() {  
    for i := 1; i <= 10; i++ {  
        if i%2 == 0 {  
            continue  
        }  
        fmt.Printf("%d ", i)  
    }  
}
```

【备注】： *break*, *continue*的区别：

- *break* 语句将无条件跳出并结束当前的循环， 然后执行循环体后的语句;

- `continue`语句是跳过当前的循环， 而开始执行下一次循环。

// 1、`break`终止循环

```
for i := 0; i < 10; i++ {  
    if i == 5 {  
        break  
    }  
    fmt.Print(i)  
}
```

//结果是: 01234

// 2、`continue`跳过某次循环

```
for i := 0; i < 10; i++ {  
    if i == 5 {  
        continue  
    }  
    fmt.Print(i);  
}
```

//结果是: 012346789

3、案例：输出1-50之间所有不包含数字4的数（`continue`实现）

```
func eludeFour() {  
    fmt.Println("\n输出1-50之间所有不包含数字4的数")  
    //定义局部变量  
    num := 0  
    //循环开始  
    for num < 50 {  
        num++  
        /* 跳过迭代 */  
        if num%10 == 4 || num/10%10 == 4 {  
            continue  
        }  
        fmt.Printf("%d\t", num)  
    }  
}
```

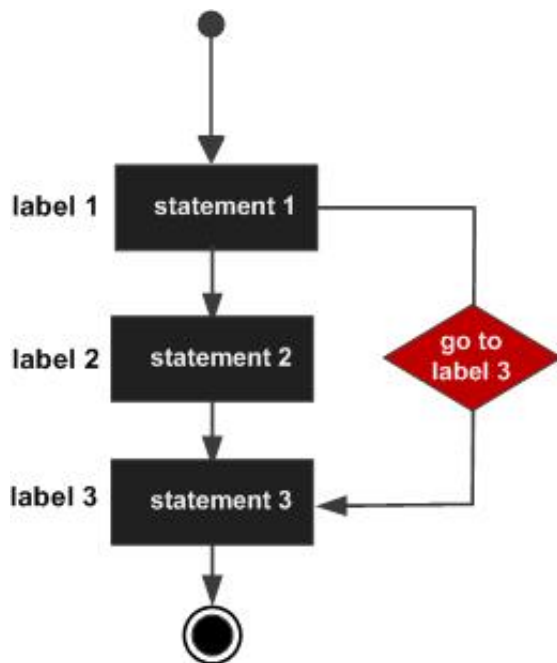
}

(三)、goto语句

1、Go 语言的 goto 语句可以无条件地转移到程序指定的行。

2、goto语句通常与条件语句配合使用。可用来实现条件转移， 构成循环， 跳出循环体等功能。

但是，在结构化程序设计中一般不主张使用goto语句， 以免造成程序流程的混乱，使理解和调试程序都产生困难。



3、goto 语法格式如下：

LABEL: statement

goto **LABEL**

4、案例：输出1-50之间不包含4的数（goto实现）

```
func gotoTest() {
```

```
    //定义局部变量
```

```
    num := 0
```

```
    /* 跳过迭代 */
```

```
LOOP:
```

```
    for num < 50 {
```

```
        num++
```

```
        if num%10 == 4 || num/10%10 == 4 {
```

```

        goto LOOP
    }
    fmt.Printf("%d\t", num)
}
}

```

5、案例：求1-100的素数（借助goto跳转）

```

func printPrimeNumberGoto() {
    var C, c int //声明变量
    C = 1        /*这里不写入FOR循环是因为For语句执行之初会将C的值变为1，当我们goto A时for语句会重新执行（不是重新一轮循环）*/
    LOOP:
    for C < 100 {
        C++ //C=1不能写入for这里就不能写入
        for c = 2; c < C; c++ {
            if C%c == 0 {
                goto LOOP //若发现因子则不是素数
            }
        }
        fmt.Printf("%d\t", C)
    }
}

```