第三章 Go语言流程控制

本章学习Go语言的流程控制语句及循环语句。流程控制包括if条件判断语句、switch分支语句。循环语句包括for循环、for嵌套循环以及与循环语句相关的语句——break、continue、goto。

本章重点为大家介绍如下的内容：

* Go语言流程控制概述
* if条件判断语句
* switch分支语句
* Go语言中循环语句概述
* for循环语句及其多种语法形式
* for嵌套循环语句
* 循环控制语句（break、continue、goto）

3.1 Go 语言流程控制

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，流程控制是一门语言的经脉；流程控制语句包含条件判断语句、条件分支语句及循环语句。

Go语言的基本流程控制语句包括：

* if条件判断语句
* switch分支语句
* for循环语句
* goto跳转语句及break和continue循环控制语句

Go 语言提供了以下几种条件判断、分支语句

|  |  |
| --- | --- |
| 语句 | 描述 |
| if 语句 | if 语句由一个布尔表达式后紧跟一个或多个语句组成。 |
| if...else 语句 | if 语句后可以使用可选的else 语句, else 语句中的表达式在布尔表达式为 false 时执行。 |
| if 嵌套语句 | 你可以在if或else if语句中嵌入一个或多个if或else if语句。 |
| switch 语句 | switch语句用于基于不同条件执行不同动作。 |
| select 语句 | select语句类似于switch语句，但是select会随机执行一个可运行的case。如果没有case可运行，它将阻塞，直到有case可运行。 |

表 3.1

Go 语言提供了以下几种循环语句

|  |  |
| --- | --- |
| 循环类型 | 描述 |
| for循环 | 重复执行语句块 |
| 循环嵌套 | 在 for 循环中嵌套一个或多个 for 循环 |

表 3.2

GO 语言支持以下几种循环控制语句

|  |  |
| --- | --- |
| 控制语句 | 描述 |
| break语句 | 经常用于中断当前 for 循环或跳出 switch 语句 |
| continue语句 | 跳过当前循环的剩余语句，然后继续进行下一轮循环。 |
| goto语句 | 将控制转移到被标记的语句。 |

表 3.3

3.2 if条件判断语句

Go 编程语言中 if 语句的语法如下：

if 布尔表达式 {

/\* 在布尔表达式为 true 时执行 \*/

}

if 在布尔表达式为 true 时，其后紧跟的语句块执行，如果为 false 则不执行。

Go 编程语言中 if...else 语句的语法如下：

if 布尔表达式 {

/\* 在布尔表达式为 true 时执行 \*/

} else {

/\* 在布尔表达式为 false 时执行 \*/

}

if 在布尔表达式为 true 时，其后紧跟的语句块执行，如果为 false 则执行 else 语句块。

Go 编程语言中 if...else if ... else 语句的语法如下：

if 布尔表达式1 {

/\* 在布尔表达式1为 true 时执行 \*/

} else if 布尔表达式2 {

/\* 在布尔表达式2为 true 时执行 \*/

} else {

/\* 在不满足以上条件时执行 \*/

}

if语句中的注意细节如下：

* 不需使用括号将条件包含起来;
* 大括号{}必须存在，即使只有一行语句;
* 左括号必须在if或else的同一行;
* 在if之后，条件语句之前，可以添加变量初始化语句，使用"；"进行分隔。

例：

1.用 if 语句判断数据奇数偶数

package main

import "fmt"

func main() {

num := 20

if num%2 == 0 {

fmt.Println(num, "偶数")

} else {

fmt.Println(num, "奇数")

}

}

2.判断学生平均成绩。有优、良、中等、及格、不及格等五档。

package main

import "fmt"

func main() {

score := 88

if score >= 90 {

fmt.Println("优秀")

} else if score >= 80 {

fmt.Println("良好")

} else if score >= 70 {

fmt.Println("中等")

} else if score >= 60 {

fmt.Println("及格")

} else if score < 60 {

fmt.Println("不及格")

}

}

if语句还有一个变体。它的语法是：

if statement; condition {

//代码块

}

例：

判断一个数是奇数还是偶数？

package main

import "fmt"

func main() {

if num := 10; num%2 == 0 {

fmt.Println(num, "偶数")

} else {

fmt.Println(num, "奇数")

}

}

需要注意的是，num的定义在if里，那么只能够在该if..else语句块中使用，否则编译器会报错的。

3.3 if嵌套语句

可以在 if 或 else if 语句中嵌入一个或多个 if 或 else if 语句。

Go 编程语言中 if...else 语句的语法如下：

if 布尔表达式 1 {

/\* 在布尔表达式 1 为 true 时执行 \*/

if 布尔表达式 2 {

/\* 在布尔表达式 2 为 true 时执行 \*/

}

}

可以以同样的方式在 if 语句中嵌套 else if...else 语句

例：

判断学生平均成绩。有优、良、中等、及格、不及格等五档。

package main

import "fmt"

func main() {

if score := 98; score >= 60 {

if score >= 70 {

if score >= 80 {

if score >= 90 {

fmt.Println("优")

} else {

fmt.Println("良")

}

} else {

fmt.Println("中等")

}

} else {

fmt.Println("及格")

}

} else {

fmt.Println("不及格")

}

}

3.4 switch分支语句

Go 编程语言中 switch 语句的语法如下：

switch var1 {

case val1:

...

case val2:

...

default:

...

}

switch 语句执行的过程自上而下，直到找到case匹配项，匹配项中无需使用break，因为Go语言中的switch默认给每个case自带break，因此匹配成功后不会向下执行其他的case分支，而是跳出整个switch。

变量 var1 可以是任何类型，而 val1 和 val2 则可以是同类型的任意值。类型不被局限于常量或整数，但必须是相同类型或最终结果为相同类型的表达式。

case后的值不能重复。

可以同时测试多个符合条件的值，也就是说case后可以有多个值，这些值之间使用逗号分割，例如：case val1, val2, val3。

Go语言中switch后的表达式可以省略，那么默认是switch true。

Go语言中的switch case因为自带break，所以匹配某个case后不会自动向下执行其他case，如需贯通后续的case，可以添加fallthrough（中文含义是：贯穿），强制执行后面的case分支。

fallthrough必须放在case分支的最后一行。如果它出现在中间的某个地方，编译器就会抛出错误（fallthrough statement out of place，含义是fallthrough不在合适的位置）。



图 3.1

例：

1.判断学生平均成绩。有优、良、中等、及格、不及格等五档。

package main

import "fmt"

func main() {

/\* 定义局部变量 \*/

grade := ""

score := 78.5

//思考：以下代码逻辑错误在哪里？

//switch {

//case score < 60:

// grade = "E"

//case score >= 60:

// grade = "D"

//case score >= 70:

// grade = "C"

//case score >= 80:

// grade = "B"

//case score >= 90:

// grade = "A"

//}

switch { //switch后面省略不写，默认相当于：switch true

case score >= 90:

grade = "A"

case score >= 80:

grade = "B"

case score >= 70:

grade = "C"

case score >= 60:

grade = "D"

default:

grade = "E"

}

fmt.Printf("你的等级是: %s\n", grade)

fmt.Print("最终评价是: ")

switch grade {

case "A":

fmt.Printf("优秀!\n")

case "B":

fmt.Printf("良好\n")

case "C":

fmt.Printf("中等\n")

case "D":

fmt.Printf("及格\n")

default:

fmt.Printf("差\n")

}

}

2.判断某年某月的天数

package main

import "fmt"

func main() {

/\* 定义局部变量:年、月、日 \*/

year := 2008

month := 2

days := 0

switch month {

case 1, 3, 5, 7, 8, 10, 12:

days = 31

case 4, 6, 9, 11:

days = 30

case 2:

if (year%4 == 0 && year%100 != 0) || year%400 == 0 {

days = 29

} else {

days = 28

}

default:

days = -1

}

fmt.Printf("%d 年 %d 月的天数为：%d\n", year, month, days)

}

switch 语句还可以被用于 type-switch 来判断某个 interface 变量中实际存储的变量类型。

语法结构如下：

switch x.(type){

case type:

statement(s);

case type:

statement(s);

/\* 你可以定义任意个数的case \*/

default: /\* 可选 \*/

statement(s);

}

例：

package main

import "fmt"

func main() {

var x interface{}

switch i := x.(type) {

case nil:

fmt.Printf(" x 的类型 :%T",i)

case int:

fmt.Printf("x 是 int 型")

case float64:

fmt.Printf("x 是 float64 型")

case func(int) float64:

fmt.Printf("x 是 func(int) 型")

case bool, string:

fmt.Printf("x 是 bool 或 string 型" )

default:

fmt.Printf("未知型")

}

}

结果

x 的类型 :<nil>

3.5 Go循环语句

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。

循环处理语句有：

|  |  |
| --- | --- |
| 循环类型 | 描述 |
| for 循环 | 重复执行语句块 |
| 循环嵌套 | 在 for 循环中嵌套一个或多个 for 循环 |

表 3.4

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 | 描述 |
| break语句 | 经常用于中断当前 for 循环或跳出 switch 语句 |
| continue语句 | 跳过当前循环的剩余语句，然后继续进行下一轮循环。 |
| goto语句 | 将控制转移到被标记的语句。 |

表 3.5

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。

3.5.1 for循环语法结构

1.基本for循环语法结构：

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。

例：

for i := 0; i <= 10; i++ {

fmt.Printf("%d ", i)

}

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。

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。

2.for关键字后只有一个条件表达式：

for 循环条件condition { }

效果类似其它编程语言中的while循环

例：

var i int

for i <= 10 {

fmt.Println(i)

i++

}

3.for关键字后无表达式：

for { }

效果与其它编程语言的for(;;) {}一致，此时for执行无限循环

例：

var i int

for {

if (i > 10) {

break

}

fmt.Println(i)

i++

}

4.for ... range：

for 循环的 range 格式

主要用于对字符串、slice、数组、map等进行迭代循环

for key, value := range oldMap {

newMap[key] = value

}

例：

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 ，字符是一

第 12 位的ASCII值=19969 ，字符是丁

第 15 位的ASCII值=19970 ，字符是丂

例：

遍历切片中元素

func traverseSlice() {

arr := []int{100, 200, 300}

for i, value := range arr {

fmt.Println(i, ":", value)

}

}

3.5.2 for循环使用案例

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)

}

3.6 for嵌套循环语句

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

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

statement(s);

}

statement(s);

}

例：

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()

}

}

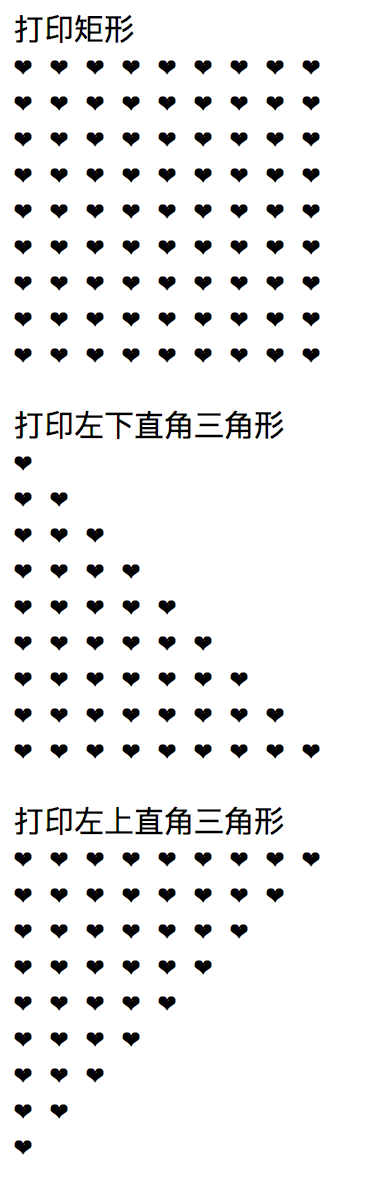


图 3.2

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()

}

}

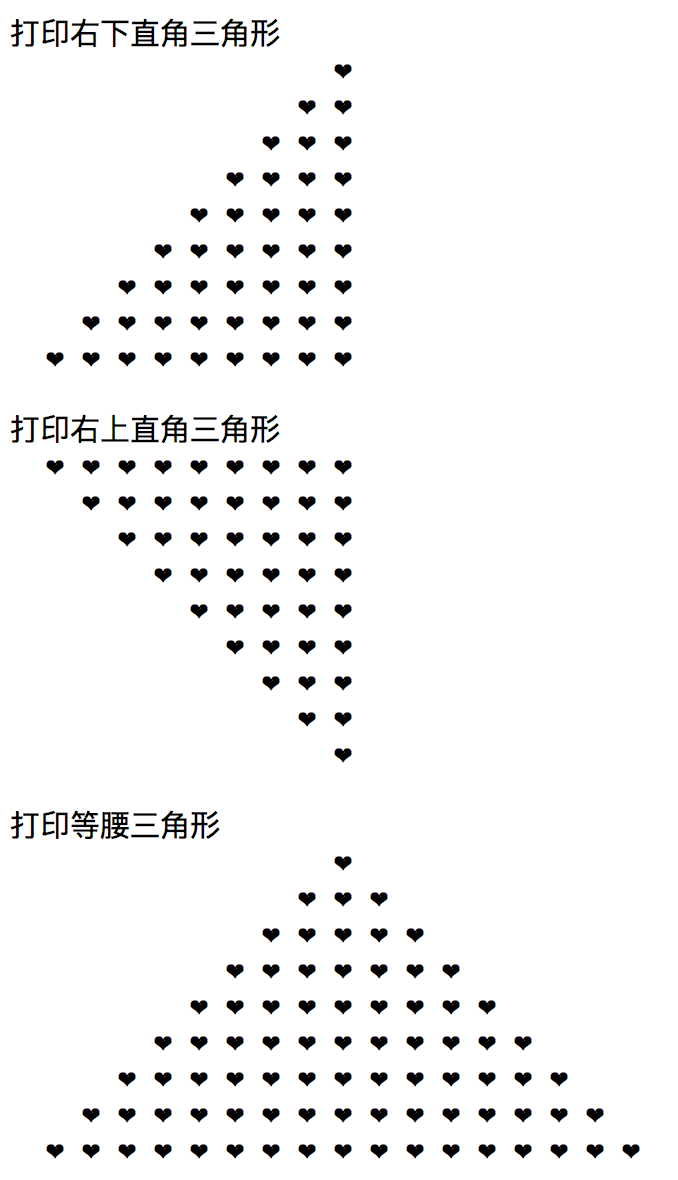


图 3.3

3.打印九九乘法表

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()

}

}

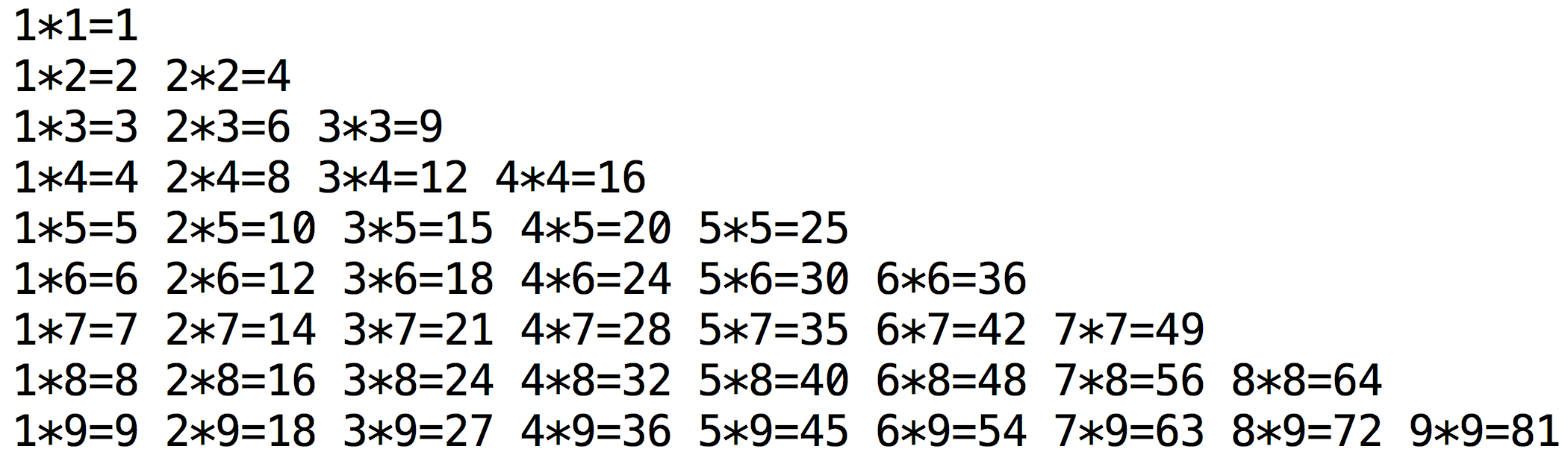


图 3.4

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)

}

}

}

3.7 循环控制语句

3.7.1 break语句

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。

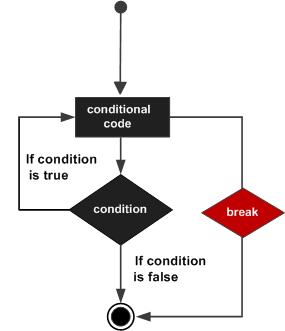


图 3.5

例：

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")

}

3.7.2 continue语句

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。

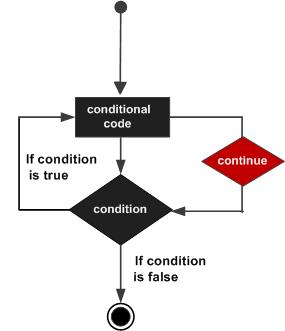


图 3.6

例：

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。

// break终止循环

for i := 0; i < 10; i++ {

if i == 5 {

break

}

fmt.Print(i)

}

//结果是：01234

//continue跳过某次循环

for i := 0; i < 10; i++ {

if i == 5 {

continue

}

fmt.Print(i);

}

//结果是：012346789

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)

}

}

3.7.3 goto语句

Go 语言的 goto 语句允许无条件地转移到程序指定的行。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。

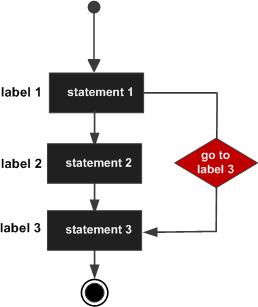


图 3.7

goto 语法格式如下：

LABEL: statement

goto LABEL

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)

}

}

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)

}

}