Go语言基础之流程控制

2017年6月18日 | Golang |10351 阅读

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。

if else(分支结构)

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。

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")
    }
}
```

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：

```
func ifDemo2() {
    if score := 65; 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？

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。

```
func forDemo() {
    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：

```
func forDemo2() {
    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：

```
func forDemo3() {
    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。

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。

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。

switch case

使用 `switch` 语句可方便地对大量的值进行条件判断。

```
func switchDemo1() {
    finger := 3
    switch finger {
    case 1:
        fmt.Println("大拇指")
    case 2:
        fmt.Println("食指")
    case 3:
        fmt.Println("中指")
    case 4:
        fmt.Println("无名指")
    case 5:
        fmt.Println("小拇指")
    default:
        fmt.Println("无效的输入！")
    }
}
```

Go语言规定每个`switch`只能有一个`default`分支。

一个分支可以有多个值，多个case值中间使用英文逗号分隔。

```
func testSwitch3() {
    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)
    }
}
```

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：

```
func switchDemo4() {
    age := 30
    switch {
    case age < 25:
        fmt.Println("好好学习吧")
    case age > 25 && age < 35:
        fmt.Println("好好工作吧")
    case age > 60:
        fmt.Println("好好享受吧")
    default:
        fmt.Println("活着真好")
    }
}
```

`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。

```go
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
```

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。 例如双层嵌套的for循环要退出时：

```go
func gotoDemo1() {
    var breakFlag bool
    for i := 0; i < 10; i++ {
        for j := 0; j < 10; j++ {
            if j == 2 {
                // 设置退出标签
                breakFlag = true
                break
            }
            fmt.Printf("%v-%v\n", i, j)
        }
        // 外层for循环判断
        if breakFlag {
            break
        }
    }
}
```

使用 `goto` 语句能简化代码：

```
func gotoDemo2() {
    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")
}
```

break(跳出循环)

`break` 语句可以结束 `for` 、 `switch` 和 `select` 的代码块。

`break` 语句还可以在语句后面添加标签，表示退出某个标签对应的代码块，标签要求必须定义在对应的 `for` 、 `switch` 和 `select` 的代码块上。 举个例子:

```
func breakDemo1() {
BREAKDEMO1:
    for i := 0; i < 10; i++ {
        for j := 0; j < 10; j++ {
            if j == 2 {
                break BREAKDEMO1
            }
            fmt.Printf("%v-%v\n", i, j)
        }
    }
    fmt.Println("...")
}
```

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。

在 `continue` 语句后添加标签时，表示开始标签对应的循环。例如:

```
func continueDemo() {
forloop1:
    for i := 0; i < 5; i++ {
        // forloop2:
        for j := 0; j < 5; j++ {
            if i == 2 && j == 2 {
                continue forloop1
            }
            fmt.Printf("%v-%v\n", i, j)
        }
    }
}
```

练习题

1. 编写代码打印9*9乘法表。