Part 9: Loops

11 APRIL 2017

This is tutorial number 9 in Golang tutorial series.

A loop statement is used to execute a block of code repeatedly.

`for` is the only loop available in Go. Go doesn't have while or do while loops which are present in other languages like C.

for loop syntax

```
for initialisation; condition; post {
}
```

The initialisation statement will be executed only once. After the loop is initialised, the condition will be checked. If the condition evaluates to true, the body of loop inside the `{ }` will be executed followed by the post statement. The post statement will be executed after each successful iteration of the loop. After the post statement is executed, the condition will be rechecked. If its true, the loop will continue executing, else the for loop terminates.

All the three components namely initialisation, condition and post are optional in Go. Lets look at an example to understand for loop better.

Example

Lets write a program which uses for loop to print all numbers from 1 to 10.

```
package main

import (
    "fmt"
)

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

In the above program, `i` is initialised to 1. The conditional statement will checks if `i <= 10`. If the condition is true, the value of i is printed, else the loop is terminated. The post statement increments i by 1 at the end of each iteration. Once `i` becomes greater than 10, the loop terminates.

The above program will print `1 2 3 4 5 6 7 8 9 10`

The variables declared in a for loop are only available within the scope of the loop. Hence `i` cannot be accessed outside the body for loop.

break

The `break` statement is used to terminate the for loop abruptly before it finishes its normal execution and move the control to the line of code just after the for loop.

Lets write a program which prints numbers from 1 to 5 using break.

```
package main

import (
    "fmt"
)

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

In the above program, the value of i is checked during each iteration. If i is greater than 5 then `break` executes and the loop is terminated. The print statement just after the for loop is then executed. The above program will output,

```
1 2 3 4 5
line after for loop
```

Get the free Golang tools cheat sheet

continue

The `continue` statement is used to skip the current iteration of the for loop. All code present in a for loop after the continue statement will not be executed for the current iteration. The loop will move on to the next iteration.

Lets write a program to print all odd numbers from 1 to 10 using continue.

```
package main

import (
    "fmt"
)

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

In the above program the line `if i%2 == 0` checks if the reminder of dividing i by 2 is 0. If it is zero, then the number is even and `continue` statement is executed and the control moves to the next iteration of the loop. Hence the print statement after the continue will not be called and the loop proceeds to the next iteration. The output of the above program is `1 3 5 7 9`

more examples

Lets write some more code to cover all variations of `for` loop.

The program below prints all even numbers from 0 to 10.

```
package main

import (
    "fmt"
)

func main() {
    i := 0
    for ;i <= 10; { // initialisation and post are omitt
        fmt.Printf("%d ", i)
        i += 2
    }
}
```

As we already know all the three components of the for loop namely initialisation, condition and post are optional. In the above program, initialisation and post are omitted. `i` is initialised to `0` outside the for loop. The loop will be executed as long as `i <= 10`. i is increment by 2 inside the for loop. The above program outputs `0 2 4 6 8 10`.

The semicolons in the for loop of the above program can also be omitted. This format can be considered as an alternative for while loop. The above program can be rewritten as,

```
package main

import (
    "fmt"
)

func main() {
    i := 0
    for i <= 10 { //semicolons are ommitted and only cor
        fmt.Printf("%d ", i)
        i += 2
    }
}
```

It is possible to declare and operate on multiple variables in for loop. Lets write a program which prints the below sequence using multiple variable declaration.

```
10 * 1 = 10
11 * 2 = 22
12 * 3 = 36
13 * 4 = 52
14 * 5 = 70
15 * 6 = 90
16 * 7 = 112
17 * 8 = 136
18 * 9 = 162
19 * 10 = 190
```

```
package main

import (
    "fmt"
)

func main() {
    for no, i := 10, 1; i <= 10 && no <= 19; i, no = i+
        fmt.Printf("%d * %d = %d\n", no, i, no*i)
    }

}
```

In the above program `no` and `i` are declared and initialised to 10 and 1 respectively. They are incremented by 1 at the end of each iteration. The boolean operator `&&` is used in the condition to ensure that `i` is less than or equal to 10 and also `no` is less than or equal to 19.

infinite loop

The syntax for creating an infinite loop is,

```
for {
}
```

The following program will keep printing `Hello World` continuously without terminating.

```
package main

import "fmt"

func main() {
    for {
        fmt.Println("Hello World")
    }
}
```

If you try to run the above program in the go playground you will get error "process took too long". Please try running it in your local system to print "Hello World" infinitely.

There is one more construct **range** which can be used in `for` loops for array manipulation. We will cover this when we learn about arrays.

Thats it for loops. Hope you enjoyed reading. Please leave your valuable comments and feedback.

Next tutorial - switch statement

Get the free Golang tools cheat sheet

Naveen Ramanathan

iOS developer at dietco.de and Golang enthusiast.

Share this post