Part 10: Switch Statement

15 APRIL 2017

This is tutorial number 10 in Golang tutorial series.

A switch is a conditional statement which evaluates an expression and compares it against a list of possible matches and executes blocks of code according to the match. It can be considered as an idiomatic way of writing multiple `if else` clauses.

An example program is worth a hundred words. Lets start with a simple example which will take a finger number as input and outputs the name of that finger :) . For example 0 is thumb, 1 is index and so on.

```
package main

import (
    "fmt"
)

func main() {
    finger := 4
    switch finger {
    case 1:
        fmt.Println("Thumb")
    case 2:
        fmt.Println("Index")
    case 3:
        fmt.Println("Middle")
    case 4:
        fmt.Println("Ring")
    case 5:
        fmt.Println("Pinky")

    }
}
```

In the above program `switch finger` compares the value of `finger` with each of the `case` statements. The cases are evaluated from top to bottom and the first case which matches the expression is executed. In this case finger has a value of 4 and hence `Ring` is printed.

Duplicate cases with the same constant value are not allowed. If you try to run the program below, the compiler will complain `duplicate case 4 in switch`

```
package main

import (
    "fmt"
)

func main() {
    finger := 4
    switch finger {
    case 1:
        fmt.Println("Thumb")
    case 2:
        fmt.Println("Index")
    case 3:
        fmt.Println("Middle")
    case 4:
        fmt.Println("Ring")
    case 4://duplicate case
        fmt.Println("Another Ring")
    case 5:
        fmt.Println("Pinky")

    }
}
```

default case

We have only 5 fingers in our hand. What will happen if we input a incorrect finger number. This is where the default case comes into picture. The default case will be executed when none of the other cases match.

```
package main

import (
    "fmt"
)

func main() {
    switch finger := 8; finger {//finger is declared in
    case 1:
        fmt.Println("Thumb")
    case 2:
        fmt.Println("Index")
    case 3:
        fmt.Println("Middle")
    case 4:
        fmt.Println("Ring")
    case 5:
        fmt.Println("Pinky")
    default: //default case
        fmt.Println("incorrect finger number")
    }
}
```

In the above program `finger` is `8` and it does not match any of the cases and hence `incorrect finger number` is printed. It's not necessary that default should be the last case in a switch statement. It can be present anywhere in the switch.

You might also have noticed a small change in the declaration of `finger`. It is declared in the switch itself. A switch can include an optional statement which is executed before the expression is evaluated. In this line `switch finger := 8; finger` finger is first declared and also used in the expression. The scope of `finger` in this case is limited to the switch block.

Get the free Golang tools cheat sheet

multiple expressions in case

It is possible to include multiple expressions in a case by separating them with comma.

```
package main

import (
    "fmt"
)

func main() {
    letter := "i"
    switch letter {
    case "a", "e", "i", "o", "u": //multiple expressions
        fmt.Println("vowel")
    default:
        fmt.Println("not a vowel")
    }
}
```

The above program checks whether `letter` is a vowel of not. The line `case "a", "e", "i", "o", "u":` matches all the vowels. This program outputs `vowel`.

expressionless switch

The expression in a switch is optional and it can be omitted. If the expression is omitted, the switch is considered to be `switch true` and each of the `case` expression is evaluated for truth and the corresponding block of code is executed.

```
package main

import (
    "fmt"
)

func main() {
    num := 75
    switch { // expression is omitted
    case num >= 0 && num <= 50:
        fmt.Println("num is greater than 0 and less than
    case num >= 51 && num <= 100:
        fmt.Println("num is greater than 51 and less tha
    case num >= 101:
        fmt.Println("num is greater than 100")
    }

}
```

In the above program the expression is absent in switch and hence it is considered as true and each of the case is evaluated. `case num >= 51 && num <= 100:` is true and the program outputs `num is greater than 51 and less than 100`. This type of switch can be considered as an alternative to multiple `if else` clauses.

fallthrough

In Go the control comes out of the switch statement immediately after a case is executed. A `fallthrough` statement is used to transfer control to the first statement of the case that is present immediately after the case which has been executed.

Lets write an program to understand fallthrough. Our program will check whether the input number is lesser than 50, 100 or 200. For instance if we input 75, the progam will print that 75 is less than both 100 and 200. We will achieve this output using fallthrough.

```
package main

import (
    "fmt"
)

func number() int {
        num := 15 * 5
        return num
}

func main() {

    switch num := number(); { //num is not a constant
    case num < 50:
        fmt.Printf("%d is lesser than 50\n", num)
        fallthrough
    case num < 100:
        fmt.Printf("%d is lesser than 100\n", num)
        fallthrough
    case num < 200:
        fmt.Printf("%d is lesser than 200", num)
    }

}
```

Switch and case expressions need not be only constants. They can be evaluated at runtime too. In the above program `num` is initialised to the return value of the function `number()`. The control comes inside the switch and the cases are evaluated. `case num < 100:` is true and the program prints `75 is lesser than 100`. The next statement is `fallthrough`. When `fallthrough` is encountered the control moves to the first statement of the next case and also prints `75 is lesser than 200`. The output of the program is

```
75 is lesser than 100
75 is lesser than 200
```

`fallthrough` should be the last statement in a `case`. If it present somewhere in the middle, the compiler will throw error `fallthrough statement out of place`

Thats if for switch. There is one more type of switch called **type switch**. We will look into this when we learn about interfaces.

Please share your valuable comments and feedback:).

Next tutorial - Arrays and Slices

Get the free Golang tools cheat sheet

Naveen Ramanathan

iOS developer at dietco.de and Golang enthusiast.

Share this post