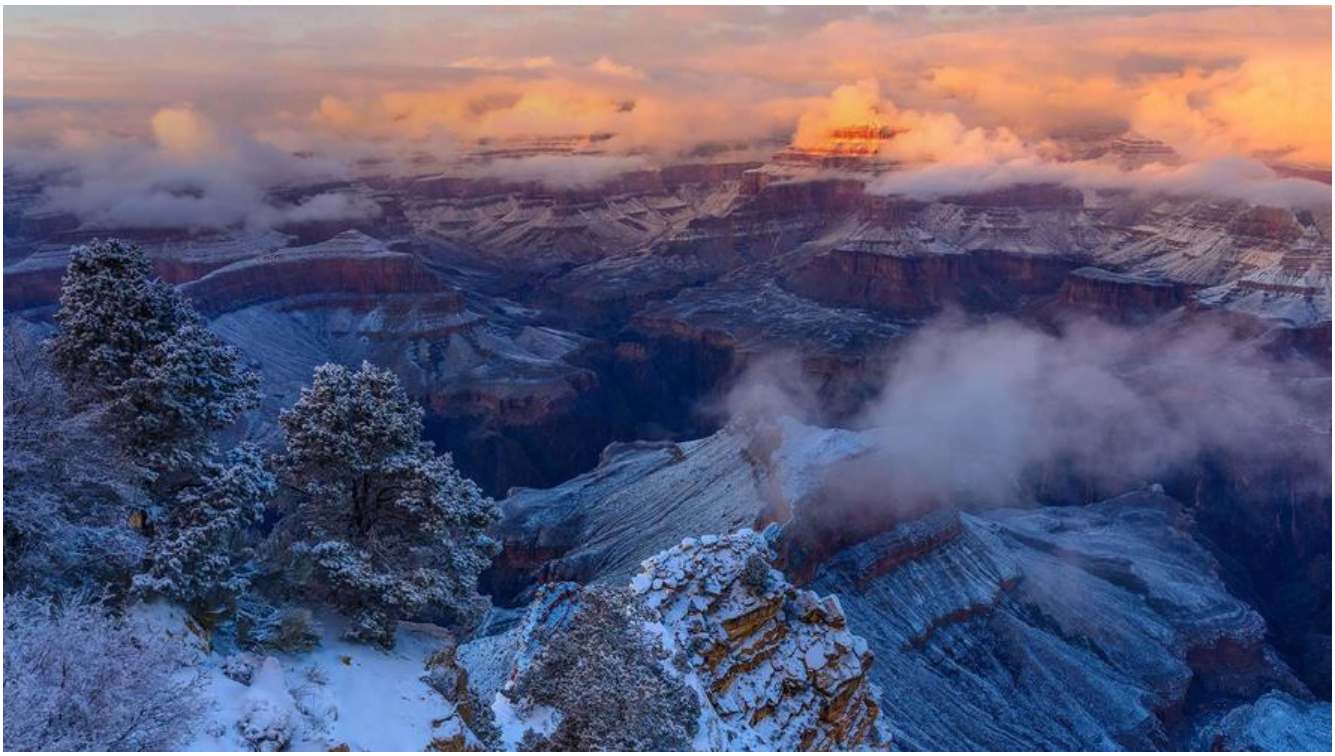
链滴

【极简 Golang 入门】控制结构

作者：Allenxuxu

原文链接：https://ld246.com/article/1564915428991

来源网站：链滴



控制结构

Go 程序和大多数编程语言一样从 main() 函数开始执行，然后按顺序执行该函数体中代码。代码中必需要进行条件判断，Go 中提供如下分支结构：

- if-else
- switch
- select

Go 中同样有循环结构来重复执行某段代码：

- for(range)

if-else 结构

if 是用于测试某个条件（布尔型或逻辑型）的语句，如果该条件成立，则会执行 if 后由大括号括起来代码块。else 这个代码块中的代码只有在 if 条件不满足时才会执行。if 和 else 后的两个代码块是相独立的分支，永远只会执行其中一个。

```
if condition {
    // do something
} else {
    // do something
}
```

如果需要 增加更多分支选择，可以使用 else if 。else-if 分支的数量是没有限制的，但是当选择条件多时，应该使用 switch 。

```
if condition1 {
```

```
    // do something
} else if condition2 {
    // do something else
} else {
    // catch-all or default
}
```

if 可以包含一个初始化语句，常用于 err 的条件判断。

```
if initialization; condition {
    // do something
}
```

例如：

```
if err := fun(); err != nil {
    // do something
}
```

switch 结构

相比较 C/C++ 等其他语言而言，Go 语言中的 switch 结构使用非常灵活, 并且不需要 breake 语句跳出。

```
switch value {
    case v1:
        ...
    case v2:
        ...
    default:
        ...
}
```

value 变量可以是任何类型，v1 和 v2 是同类型的任意值或者是最终结果为相同类型的表达式，但不于常量和整数。

同一个 case 可以匹配多个可能符合条件的值，通过逗号分割：

```
switch i {
    case 0:
    case 1,2,3:
        f() // 当 i == 1 或者 i == 2 或者 i == 3 则执行 f()
}
```

switch 语句可以不提供任何被判断的值，然后在每个 case 分支中进行测试不同的条件。当任一分支测试结果为 true 时，该分支的代码会被执行。这看起来非常像链式的 if-else 语句。

```
switch {
    case i < 0:
        f1()
    case i == 0:
        f2()
    case i > 0:
        f3()
}
```

switch 语句还可以包含一个初始化语句：

```
switch result := calculate(); {
    case result < 0:
        ...
    case result > 0:
        ...
    default:
        // 0
}

switch a, b := x[i], y[j]; {
    case a < b: t = -1
    case a == b: t = 0
    case a > b: t = 1
}
```

因为 Go 的 switch 相当于每个case最后都自带一个 break， 匹配成功后就不会向下执行其他 case 所以如果需要接着执行下一个 case 的可以使用 fallthrough 关键字。

```
    i:= 2
    switch i {
    case 1:
            fmt.Println("1")
            fallthrough
    case 2:
            fmt.Println("2")
            fallthrough
    case 3:
            fmt.Println("3")
            fallthrough
    case 4:
            fmt.Println("4")
            fallthrough
    default:
            fmt.Println("default")

    }
```

输出：

```
2
3
4
default
```

fallthrough 只会强制执行下一个 case 。

```
    i:= 2
    switch i {
    case 1:
            fmt.Println("1")
            fallthrough
    case 2:
            fmt.Println("2")
```

```
        fallthrough
    case 3:
        fmt.Println("3")
        // fallthrough
    case 4:
        fmt.Println("4")
        fallthrough
    default:
        fmt.Println("default")

    }
```

输出：

```
2
3
```

Go 中的 switch 还可以用来做类型判断。

```
package main

import "fmt"

func Type(v interface{}) {
    switch v.(type) {
    case bool:
        fmt.Println("bool")
    case float64:
        fmt.Println("float64")
    case int:
        fmt.Println("int")
    case nil:
        fmt.Println("nil")
    case string:
        fmt.Println("string")
    default:
        fmt.Println("default")
    }
}

func main() {
    Type(1)
    Type("")
    Type(true)
    Type(nil)
    i := 1
    Type(&i)    // *int
}
```

输出：

```
int
string
bool
nil
```

```
default
```

for 结构

Go 中循环结构只有 for 语句，并没有 while 语句。

for 语句基本用法和其他语言无异：

```
for i := 0; i < 5; i++ {
    // do
}
```

支持多个变量控制循环：

```
for i, j := 0, 1; i < j; i, j = i+1, j-1 {
    // do
}
```

for 语句实现 while 语句功能：

```
var i int = 3

for i >= 0 {
    i--
    // do
}
```

无限循环：

```
for {

}

for i := 0; ; i++ {

}

for ; ; {

}
```

Go 中还提供一个关键字用于循环结构 range ，它可以迭代任何一个集合（数组，map）。

```
for k, v := range s {
    // do
}
```

需要注意的是，v 对于元素的值拷贝，任何对 v 的修改都不会影响集合 s 。

break 和 continue

和其他语言一样，break 用于跳出整个循环，continue 用于跳出当前循环继续下一次循环。

```
for i := 0; i < 3; i++ {
```

```
    if i == 1 {
        continue
    }
    println(i)
}
```

输出：

```
0
2
```

```
for i := 0; i < 3; i++ {
    if i == 1 {
        break
    }
    println(i)
}
```

输出：

```
0
```