链滴

Go 边看边练 -《Go 学习笔记》系列（四）

作者：88250

原文链接：https://ld246.com/article/1437984612418

来源网站：链滴

上一篇： https://hacpai.com/article/1437719712835

ToC

2.1 表达式

语言设计简练，保留字不多。

```
break default func interface select
case defer go map struct
chan else goto package switch
const fallthrough if range type
continue for import return var
```

2.2 运算符

全部运算符、分隔符，以及其他符号。

```
+ & += &= && == != ( )
- | -= |= || < <= [ ]
* ^ *= ^= <- > >= { }
/ << /= <<= ++ = := , ;
% >> %= >>= -- ! ... . :
&^ &^=
```

运算符结合律全部从左到右。

```
优先级 运算符 说明
-----------+---------------------------------------------+--------------------------
high * / & << >> & &^
+ - |" ^
== != < <= < >=
<- channel
&&
low ||
```

简单位运算演示。

```
0110 & 1011 = 0010 AND 都为 1。
0110 | 1011 = 1111 OR 至至少一一个为 1。
0110 ^ 1011 = 1101 XOR 只能一一个为 1。
0110 &^ 1011 = 0100 AND NOT 清除标志位。
```

标志位操作。

```
a := 0
a |= 1 << 2 // 0000100: 在 bit2 设置标志位。
a |= 1 << 6 // 1000100: 在 bit6 设置标志位
a = a &^ (1 << 6) // 0000100: 清除 bit6 标志位。
```

不支持运算符重载。尤其需要注意，"++"、"--" 是语句而非表达式。

```
n := 0
p := &n
// b := n++ // syntax error
// if n++ == 1 {} // syntax error
// ++n // syntax error
n++
*p++ // (*p)++
```

没有 "~"，取反运算也用 "^"。

```
x := 1
x, ^x // 0001, -0010
```

2.3 初始化

初始化复合对象，必须使用类型标签，且左大括号必须在类型尾部。

```
// var a struct { x int } = { 100 } // syntax error
// var b []int = { 1, 2, 3 } // syntax error
// c := struct {x int; y string} // syntax error: unexpected semicolon or newline
// {
// }
```

```
var a = struct{ x int }{100}
var b = []int{1, 2, 3}
```

初始化值以 "," 分隔。可以分多行，但最后一行必须以 "," 或 "}" 结尾。

```
a := []int{
    1,
    2 // Error: need trailing comma before newline in composite literal
}

a := []int{
    1,
    2, // ok
}

b := []int{
    1,
    2 } // ok
```

2.4 控制流

2.4.1 IF

很特别的写法：

- 可省略条件表达式括号。
- 支持初始化语句，可定义代码块局部变量。
- 代码块左大括号必须在条件表达式尾部。

例如：

```
x := 0

// if x > 10 // Error: missing condition in if statement
// {
// }

if n := "abc"; x > 0 { // 初始化语句未必就是定义变量，比如 println("init") 也是可以的。
    println(n[2])
} else if x < 0 { // 注意 else if 和 else 左大括号位置。
    println(n[1])
} else {
    println(n[0])
}
```

不支持三元操作符 "a > b ? a : b"。

2.4.2 For

支持三种循环方式，包括类 while 语法。

```
s := "abc"

for i, n := 0, len(s); i < n; i++ { // 常见的 for 循环，支持初始化语句。
    println(s[i])
}

n := len(s)
for n > 0 { // 替代 while (n > 0) {}
    println(s[n]) // 替代 for (; n > 0;) {}
    n--
}

for { // 替代 while (true) {}
    println(s) // 替代 for (;;) {}
}
```

不要期望编译器能理解你的想法，在初始化语句中计算出全部结果是个好主意。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/635bd3e676694ee8b8083057610b350.go?embed=true" width="99%" height="500"></iframe>

2.4.3 Range

类似迭代器操作，返回 (索引, 值) 或 (键, 值)。

```
1st value 2nd value
-----------------+------------------+-----------------+------------------
string index s[index] unicode, rune
array/slice index s[index]
map key m[key]
channel element
```

可忽略不想要的返回值，或用 "_" 这个特殊变量。

```
s := "abc"

for i := range s { // 忽略 2nd value，支持 string/array/slice/map。
    println(s[i])
}

for _, c := range s { // 忽略 index。
    println(c)
}

for range s { // 忽略全部返回值，仅迭代。
    ...
}

m := map[string]int{"a": 1, "b": 2}

for k, v := range m { // 返回 (key, value)。
    println(k, v)
```

```
}
```

注意，range 会复制对象。

```
a := [3]int{0, 1, 2}

for i, v := range a { // index、value 都是从复制品中取出。
    if i == 0 { // 在修改前，我们先修改原数组。
        a[1], a[2] = 999, 999
        fmt.Println(a) // 确认修改有效，输出 [0, 999, 999]。
    }

    a[i] = v + 100 // 使用复制品中取出的 value 修改原数组。
}

fmt.Println(a) // 输出 [100, 101, 102]。
```

建议改用引用类型，其底层数据不会被复制。

```
<iframe style="border:1px solid" src="https://wide.b3log.org/playground/dd9a9bdb73be4222159b5aa5cea9b78.go?embed=true" width="99%" height="500"></iframe>
```

另外两种引用类型 map、channel 是指针包装，而不像 slice 是 struct。

下一篇： https://hacpai.com/article/1438070631857
