链滴

Go 边看边练 -《Go 学习笔记》系列（一）

作者：88250

原文链接：https://ld246.com/article/1437497122181

来源网站：链滴

相信大家在看编程语言学习教程的时候都会难免觉得有点枯燥：

- 自己看书，遇到问题不方便求助
- 书上的代码片段、示例是静态的
- 虽然有时书上会给出运行输出，但是不够直观
- 自己拷贝代码到 IDE 里运行是也比较麻烦
- 即使运行了也要来回在 IDE 和书之间切换，还是麻烦
- 总之，　程序员都是懒人

如果能 **边看边练** 就会舒服很多，书上的示例代码经过自己实际验证/微调后也更容易理解并消化。

基于这个出发点，黑客派决定给大家上干货中的干货：

- 从内容上：选择了雨痕的《Go 学习笔记》作为教程，该书言简意赅，讲解的都是 Go 关键点
- 从技术上：社区整合了 Go 黑科技——　Wide，在技术上实现了 **边看边练** ！

即使你以前看过该书，我们也强烈建议你在这里再看一遍，因为这将会带来完全不一样的体验~

光说不练假把式，上货！

ToC

1.1 变量

Go 是静态类型语言，不能在运行期改变变量类型。

使用关键字 var 定义变量，自动初始化为零值。如果提供初始化值，可省略变量类型，由编译器自动断。

```
var x int
var f float32 = 1.6
var s = "abc"
```

在函数内部，可用更简略的 ":=" 方式定义变量。

```
func main() {
    x := 123    // 注意检查，是定义新局部变量，还是修改全局变量。该方式容易造成错误。
}
```

可一次定义多个变量。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/a19ff7e5335b49c94293e421f16530a.go?embed=true" width="99%" height="500"></iframe>

多变量赋值时，先计算所有相关值，然后再从左到右依次赋值。

```
data, i := [3]int{0, 1, 2}, 0
i, data[i] = 2, 100              // (i = 0) -> (i = 2), (data[0] = 100)
```

特殊只写变量 "_"，用于忽略值占位。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/72d626d489db0c1b0bedb350455e495.go?embed=true" width="99%" height="400"></iframe>

编译器会将未使用的局部变量当做错误。

```
var s string      // 全局变量没问题。

func main() {
    i := 0        // Error: i declared and not used。(可使用 "_ = i" 规避)
}
```

注意重新赋值与定义新同名变量的区别。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/6e98455f94b87d0bf4fc2e8b80578a5.go?embed=true" width="99%" height="450"></iframe>

1.2 常量

常量值必须是编译器可确定的数字、字符串、布尔值。

```
const x, y int = 1, 2 // 多常量初始化
const s = "Hello, World!" // 类型推断

const ( // 常量组
    a, b = 10, 100
    c bool = false
```

```
)

func main() {
    const x = "xxx" // 未使用用局部常量不会引发编译错误。
}
```

不支持 1UL、2LL 这样的类型后缀。

在常量组中，如不提供类型和初始化值，那么视作与上一常量相同。

```
const (
    s = "abc"
    x // x = "abc"
)
```

常量值还可以是 len、cap、unsafe.Sizeof 等编译器可确定结果的函数返回值。

```
const (
    a = "abc"
    b = len(a)
    c = unsafe.Sizeof(b)
)
```

如果常量类型足以存储初始化值，那么不会引发溢出错误。

```
const (
    a byte = 100 // int to byte
    b int = 1e20 // float64 to int, overflows
)
```

枚举

关键字 iota 定义常量组中从 0 开始按行计数的自增枚举值。

```
const (
    Sunday    = iota // 0
    Monday           // 1，通常省略后续行行表达式。
    Tuesday          // 2
    Wednesday        // 3
    Thursday         // 4
    Friday           // 5
    Saturday         // 6
)

const (
    _         = iota             // iota = 0
    KB int64 = 1 << (10 * iota) // iota = 1
    MB                           // 与 KB 表达式相同，但 iota = 2
    GB
    TB
)
```

在同一常量组中，可以提供多个 iota，它们各自增长。

```
const (
    A, B = iota, iota << 10 // 0, 0 << 10
    C, D                    // 1, 1 << 10
)
```

如果 iota 自增被打断，须显式恢复。

```
const (
    A = iota // 0
    B        // 1
    C = "c"  // c
    D        // c，与上一一行行相同。
    E = iota // 4，显式恢复。注意计数包含了 C、D 两行行。
    F        // 5
)
```

可通过自定义类型来实现枚举类型限制。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/72946aeff7a4a690c4015dcbef47d63.go?embed=true" width="99%" height="650"></iframe>

下一篇： https://hacpai.com/article/1437558810339
