Go语言基本语法

2019-03-15

Go语言虽然在语言设计上不被王垠看好，但它如此简洁的代码结构确实让人着迷。

语句

Go语言语句结尾不需要;。

变量和常量

使用var声明变量。当变量需要初始化时，可以使用赋值符号:=代替=以省略var关键字。

```
var a int
var b string

var c int = 10
var d = "golang"  // 编译器自动推断类型
d := 10
```

与C语言或Java不同，Go语言的类型声明在变量右侧。需要注意的是，如果程序中声明的变量未经使用，程序将无法通过编译。Go语言是一种工程化的语言，因此它的一些特性让人感觉不可理喻，但又会在实际工程中提高效益。

Go语言的变量赋值支持一些炫酷的写法，比如要交换变量x和y的值，可以使用这种违反直觉的写法：

```
x, y = y, x
```

Go语言中使用const定义常量，true、false和iota是预定义常量。其中iota稍显特殊，iota会在每一个const关键字出现时重置为0，然后在下一次const出现前，每出现一次iota，iota的值加1。

```
const a = iota   // 0
const b = iota   // 0
const (
  c = iota       // 0
  d = iota       // 1
)
```

数组和切片

声明一个元素个数为3的数组，并初始化：

```
array := [3]int{0, 1, 2}
array[0] = 3
fmt.Println(array)
```

和其他语言一样，Go语言在声明数组后并不能改变数组的大小。所以Go语言提供了像Python一样的切片。切片可以从数组中产生，也可以使用make()函数新建。

```
array := [3]int{0, 1, 2}
slice1 := array[:2]       // 从数组中创建

slice2 := make([]int, 3)  // 直接创建

fmt.Println(slice1)       // [0 1]
fmt.Println(slice2)       // [0 0 0]
```

除切片外，映射也是使用make函数创建，映射的类型全称是var myMap map[string] int，意为

声明变量myMap，key为string，value为int。

流程控制

Go语言允许if-else语句的条件表达式不加小括号，当然加上也无妨。

```
a := 1
if a == 1 {
  print(1)
} else if (a == 2) {
  print(2)
} else {
  print(3)
}
```

选择语句的条件表达式同样不需要小括号，另外也不需要break，其他匹配项并不会执行，这一点和Scala相同。对选择语句的优化貌似已经是不约而同的做法。

```
i := 0
switch i {
case 0:
  print(0)
case 1:
  print(1)
}
```

循环结构的条件表达式依然不需要小括号。Go语言只支持for循环。同时对无限循环的场景也做了优化，不再需要for(;;)的写法。

```
for {
  print(1)
}
```

函数

Go语言诞生自C语言的派系，因此Go语言从一开始就不是OOP或FP的语言，没有类、对象等概念。函数是程序中的一等公民。和C语言相同，（main包下的）main函数是整个程序的入口。

```
func add(a int, b int) (int, int) {
  return a + b, a - b
}

func main() {
  x, y := add(1, 2)
  print(x, y)
}
```

Go语言的语句简洁高效，函数名后的第一个括号为入参，第二个括号是出参。函数支持多返回值。如果参数类型相同，可以将类型声明合并到一起，如`(a, b int)`。

结构体

刚才提到Go语言没有类、对象等概念，但是Go语言有类似C语言的结构体，并且能力强大。这里定义一个Person结构体，包含两个属性name和age，并为Person添加一个方法getInfo，用于输出Person对象的信息：

```
type Person struct {
  name string
  age int
}

func (p Person) getInfo() {
  print(p.name, p.age)
```

```
}

func main() {
  smallyu := new(Person)
  smallyu.name = "smallyu"
  smallyu.age = 1
  smallyu.getInfo()
}
```

用OOP的思想理解这样的程序并不违和。除了结构体，Go语言还保留有指针的概念。Java程序员对指针可能稍感陌生，关于指针在结构体方法中的应用，可以通过一个简单的例子来了解：

```
type Person struct {
  name string
}

func (p Person) setName() {
  p.name = "set name"
}

func (p *Person) setName2() {
  p.name = "set name"
}

func main() {
  smallyu := &Person{"smallyu"}
  smallyu.setName()
  fmt.Println(smallyu)        // &{smallyu}

  bigyu := &Person{"bigyu"}
  bigyu.setName2()
  fmt.Println(bigyu)          // &{set name}
}
```

使用值类型定义的结构体方法，入参为形参；使用引用类型定义的结构体方法，入参为实参。`&{}`是初始化对象的方法之一，等同于`new()`。

匿名结合

Go语言中匿名结合的概念，相当于OOP语言的继承。一个结构体可以继承另一个结构体的属性和方法，大致是这样。

```
type Father struct {
  name string
}

func (f Father) getName() {
  print(f.name)
}

type Son struct {
  Father
}

func main() {
  smallyu := &Son{}
  smallyu.name = "smallyu"
  smallyu.getName()       // smallyu
}
```

Son并没有定义name属性，也没有定义getName()方法，它们均继承自Father。

接口

Go语言的接口是非侵入式的，结构体只要实现了接口中的所有方法，程序就会认为结构体实现

了该接口。

```
type IPerson interface {
  getName()
}

type Person struct {
  name string
}

func (p Person) getName() {
  print(p.name)
}

func main() {
  var smallyu IPerson = &Person{"smallyu"}
  smallyu.getName()
}
```

协程

使用协程的关键字是`go`，从命名就能看出协程对于Go语言的重要性、协程是轻量级的线程，启动一个协程非常简单：

```
func f(msg string) {
  println(msg)
}

func main() {
  f("直接调用方法")
  go f("协程调用方法")
}
```

运行程序，你会发现程序只打印出”直接调用方法”几个字。这种情况是不是似曾相识？go启用了另一个”线程”来打印消息，而main线程早已结束。在程序末尾加上`fmt.Scanln()`阻止main线程的结束，就能看到全部的打印内容。

通道

通道即协程之间相互通信的通道。

```
func main() {
  message := make(chan string)

  go func() {
    message <- "ping"
  }()

  msg := <-message
  println(msg)
}
```

make函数返回一个chan string类型的通道，在匿名函数中将字符串”ping”传入通道，之后将通道中的数据输出到变量msg，最后打印出msg的值为”ping”。

错误处理

Go语言在错误处理部分有两个函数较为常用，panic函数和defer函数。panic函数会打印错误消息，并终止整个程序的执行，类似Java的Throw Exception；defer函数会在当前上下文环境执行结束前再执行，类似try catch后的finally；panic函数虽然会终止整个程序，但不会终止defer函数的执行，可以将defer函数用于打印日志。这是一个简单的例子：

```
func main() {
```

```
  println("beginning")
  defer func() {
    println("defer")
  } ()
  println("middle")
  panic("panic")
  println("ending")
}
```

来分析一下程序的执行结果。首先beginning被打印；然后遇到defer，暂不打印；middle在defer之前被打印；遇到panic，程序将终止，打印defer和panic。

这里要注意，defer是在程序结束前执行，而不是在其他语句结束后执行，这是有区别的。就像这里，panic函数引起了当前程序的结束，所以defer会在panic函数前执行，而不是panic后。程序的执行结果如下：

```
beginning
middle
defer
panic: panic

goroutine 1 [running]:
main.main()
    D:/go/src/awesomeProject/main.go:12 +0x7f
```

其他

除此之外Go语言还有很多语言特性，也提供了非常多实用的工具包。Go语言是一种值得我们尝试去使用的语言。关于协程和通道，后续会单独探讨这一重要特性。

参考

- 《Go语言编程》
- Go by Example