Golang 入门笔记 -02-Go 语言基本语法和结构

作者：zyk
原文链接：https://ld246.com/article/1599959629136
来源网站：链滴

1. 命名规范

1.1 文件

Go 语言中，源文件以 .go 格式保存，例如 main.go，且文件名只能由**小写字母**组成，若有多个单词可以用下划线 _ 进行拼接，例如 my_file.go。

1.2 标识符

Go 语言中，标识符是区分**大小写**的，标识符必须以字符或 _ 开头，后面可以跟 0 个或多个字符（字、数字以及其他符号），例如 _，_abc，a1，b_2。同时标识符不能是 Go 语言中的**保留关键字**，不以**数字开头**，也不能有**运算符**。

以下这些标识符是错误的：

- 2c（不能以数字开头）
- switch（不能是保留关键字）
- m+n（不能出现运算符）

_ 是一个特殊的标识符，叫做**空白标识符**。它既可以被**赋值**或**定义**（任意类型的数据），但它接收的将被抛弃，无法在代码中继续使用。有时候函数会返回多个值，我们可能不需要使用某些值，便可以用 _ 来接收，提高代码的灵活性。

用 _ 接收的变量统称为**匿名变量**。

1.3 保留关键字

以下是 Go 语言中的保留关键字:

保留关键字仅 25 个，还是比较少的，有利于加快编译速度。

```
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
```

1.4 预定义标识符

以下是 Go 语言中 36 个预定义标识符，在后面我们会用到：

```
append    bool      byte    cap     close   complex
complex64 complex128 uint16 copy    false   float32
float64   imag      int     int8    int16   int32
int64     uint      uint8   uint32  uint64  uintptr
itoa      len       make    new     nil     panic
print     println   real    recover string  true
```

2. 语法规范

2.1 基本规范

Go 语言的语法和 C 语言类似，都用 { } 来表示代码块，用 && 表示逻辑与，用 || 表示逻辑或，// 表单行注释，/**/ 表示多行注释。

Go 语言中可以用 ; 来表示语句结束，但**不建议**这样做。Go 语言是可以省略 ; 的，**直接换行**来表明语结束，Go 编译器会帮我们添加分号。

C 语言中经常会使用 () 来表示条件，而在 Go 语言中，() 必须省略，例如：

```
if a == 1 {
    print("success")
}
```

当然在 Go 语言中，() 仍可以用来表示优先级，例如：

```
if (a == 1 && b == 2) || (c == 3) {
    print("success")
}
```

在 C 语言中，{ 可以换行也可以不换行，但在 Go 语言中，{ 强制不能换行。

这样的写法在 Go 语言中是**错误**的：

```
if a == 1
{
    print("error")
}
```

正确的写法是：

```
if a == 1 {
    print("success")
}
```

3. 代码结构

为了加深对 go 语言的理解，我们将创建并运行第一个 go 程序。

先创建 一个 go 源文件 main.go，在其中写入：

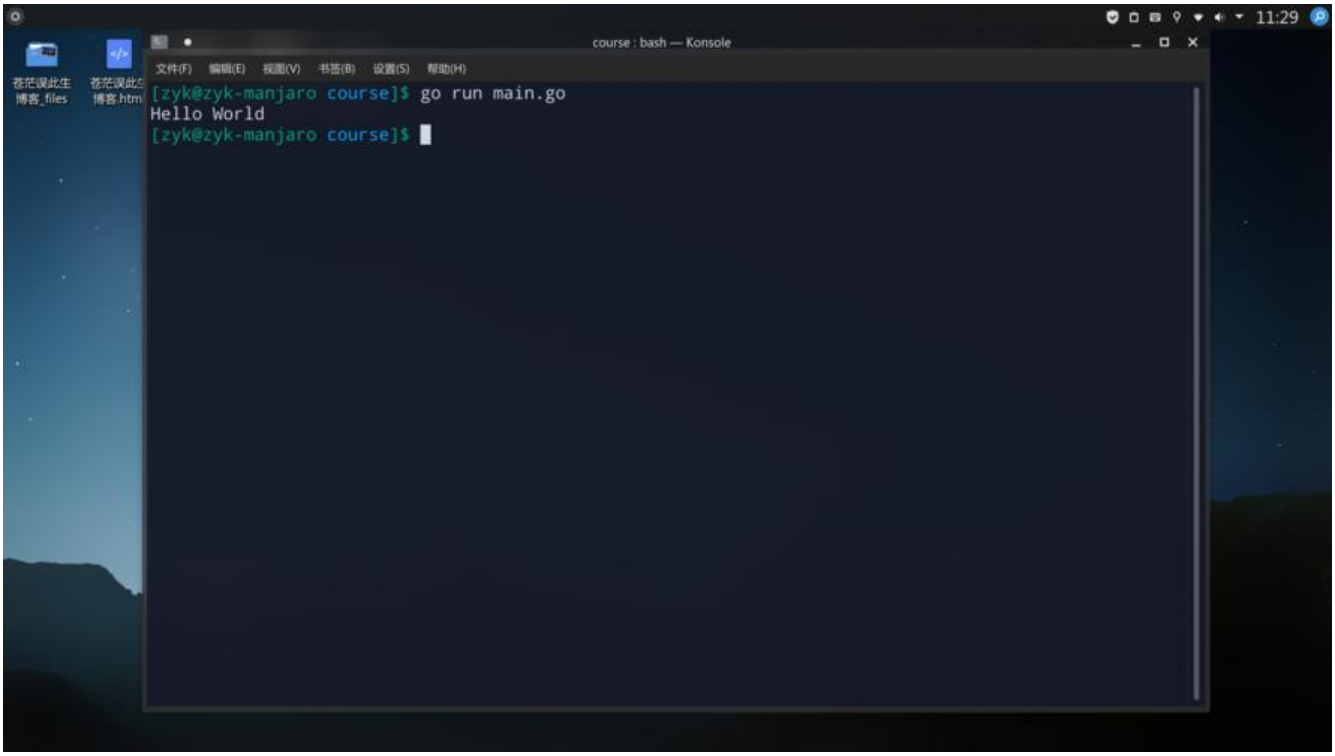
```
package main

import "fmt"

func main() {
    fmt.Println("Hello World")
}
```

接着打开终端（如果是 Windows 系统则在 CMD 或 PowerShell 下），执行以下命令：

注意：该命令需在 main.go 所在目录下执行。go run 命令将会直接编译并运行程序，不会生成二进文件。

```
go run main.go
```

执行成功后，观察终端输出结果。



3.1 包

● 包的基本概念

包（package）由一个或多个保存在同一目录下（不含子目录）的 go 文件组成，用于结构化代码。般包名称和目录名称相同，虽然 go 语言不强制要求包名和目录名相同，但建议相同，这样结构更清。

每一个 go 文件都属于且**仅属于**一个包，必须在源文件**第一行有效代码**中声明文件所属包。例如上述码中的 package main 表明这是应用程序入口包。

包名都应使用**小写字母**。

● 导入包

可以在 go 文件中导入其他包，例如上述代码中的 import "fmt"，导入了名为 fmt 的包，fmt包中包格式化输入输出的功能函数。

导入包的关键词为 import，包名需要用 "" 包裹起来。

注意： import 语句一般放在源文件包声明的**下方**。

导入单个包：

```
import "fmt"
```

导入多个包：

```
import "fmt"
import "path"
```

还有更简便的多包导入方式：

```
import (
    "fmt"
    "path"
)
```

- 导入方式
    - 全路径导入

全路径导入：导入 $GOROOT/src/ 或 $GOPATH/src/ 目录下存放的包。Go 标准库中的包只能通过**路径导入。**

GOROOT 是 Go 的安装路径，GOPATH 是 Go 项目的工作目录。

例如：

```
import "fmt" // 导入 $GOROOT/src/fmt
import "models" // 导入 $GOPATH/src/models
```

    - 相对路径导入

相对路径导入：只能导入 $GOPATH 目录下的包。

例如：

```
/*
当前文件所在路径为 $GOPATH/src/lab/models
*/
package models
import "../utils" // 相对路径导入包 utils，包 utils 所在目录为 $GOPATH/src/lab/utils
```

也可以用全路径导入：

```
/*
当前文件所在路径为 $GOPATH/src/lab/models
*/
package models
import "lab/utils" // 全路径导入包 utils，包 utils 所在目录为 $GOPATH/src/lab/utils
```

- 包的引用

以 fmt 包为例，依次讲解。

    - 标准引用

标准引用：直接用 "" 包裹包名。在源文件中以 fmt. 调用 fmt包中的函数。

```
package main

import "fmt"

func main() {
    fmt.Println("标准引用")
}
```

    - 自定义引用

自定义引用：给 fmt 包起一个别名 f。在源文件中以 f. 调用 fmt 包中的函数。

```
package main
```

```
import f "fmt"

func main() {
    f.Println("自定义引用")
}
```

- 省略引用

省略引用：相当于将 fmt 包中内容直接与当前包合并，不需要 fmt. 前缀即可调用 fmt 包中的函数。

```
package main

import . "fmt"

func main() {
    Println("省略引用")
}
```

- 匿名引用

匿名引用：用 _ 表示匿名引用，匿名引用的包同其他方式引用的包一起，仍然会被编译到可执行文件。匿名引用不会导入包中的内容，但若包中包含 init 初始化函数，那么匿名引用该包时，会执行 init 数。

```
import _ "fmt"
```

init 函数用于**初始化包**（初始化包中的变量等），不能被其他函数调用，init函数会先于 main 函数动执行，不同包中的 init 函数按照包的依赖关系顺序执行。

一个源文件中只能有**一个** init 函数。

一个包中可以包含**多个** init 函数，但是 Go 编译器无法保证 init 函数的执行顺序，建议将多个初始化作放到一个 init 函数中。

- 注意：
  - 导入的包　**必须使用**，否则编译无法通过（为了加快编译速度）。
  - 包的引用　**不能出现重名**，若出现重名，可通过自定义引用方式，给包起一个别名，来避免包的用冲突。
  - 包　**不能循环引用**，例如 a 包中导入了 b 包，b 包中又导入了 a 包，编译会出现错误。

3.2 变量

- 类比 C 语言

Go 语言的变量声明方式和 C 语言大相径庭，C 语言中我们声明变量时，习惯将数据类型放在前面，如：

```
int m;
```

这将导致一个问题，当我们在一行中声明多个变量时，会有歧义，例如：

```
int* m, n;
```

上述代码中变量 m 的类型为指针，n 的类型并非指针，而是整型。

Go 语言中可以很轻松地声明多个指针型变量：

```
var m, n *int
```

var 是声明变量的**关键字**，m 和 n 为变量名称，*int 为变量类型。

- 一般形式

Go 语言中声明变量的一般形式为：var "变量名称" "变量类型"。

刚从 类 C 语言转过来的小伙伴可能会有点不适应这种写法，但这种写法在一定程度上能避免歧义，左向右的阅读方式也更符合逻辑。

声明多行变量：

```
var m int
var s string
var b bool
```

也可以这样声明：

```
var (
    m int
    s string
    b bool
)
```

Go 语言中，变量一旦被定义，就已经被初始化，int 类型的初始值为 0，string 类型的初始值为 ""，ool 类型的初始值为 false，指针类型的初始值为 nil。

- 自动推断形式

我们还可以通过 var "变量名称 = "值" 的方式来声明变量：

```
var m = 1
var s = ""
var b = false

var (
    m = 1
    s = ""
    b = false
)
```

Go 编译器会通 = 右边的值自动进行类型推导。

- 短变量形式

在函数内部，可以通过 "变量名称" := "值" 的形式来定义**局部变量**，Go 编译器会自动根据值来推导量类型。

注意：短变量命名方式只能用在**函数内部**，不能用于**全局变量**的声明。

```
m := 1
s := ""
b := false
```

也可以在一行中进行声明：

```
m, s, b := 1, "", false
```

Go 语言**不允许重复声明变量**，以下的声明方式是**错误**的：

```
s := ""
```

```
s := "abc"
```

但当**声明新变量**和**声明重复变量**同时发生时，这个重复变量将会被**重新赋值**：

```
m, s := 1, ""
b, s := false, "abc" // s 被重新赋值为 "abc"
```

- 赋值

单行赋值：

```
m, s, b = 1, "", false
```

多行赋值：

```
m = 1
s = ""
b = false
```

- 作用域

局部变量只能作用于**函数体内部**，而全局变量作用于**整个源文件**。

若局部变量和全局变量同名，则在函数内部，局部变量优先级**高于**全局变量。

例如：

```
package main

import "fmt"

var m = 1 // 全局变量 m

func showValue() {
    m := 2 // 局部变量 m
    fmt.Println("showValue -> m: ", m)
}

func main() {
    showValue()
    fmt.Println("main -> m: ", m)
}
```

运行结果为：

```
showValue -> m:  2
main -> m:  1
```

- 注意

  - Go 语言中的变量命名遵循 **驼峰命名方式**，多个单词拼接时，从第二个单词开始，每个单词首母大写，例如myFileName。
  - **全局变量**允许声明不使用；**局部变量**声明了，必须使用，否则编译不通过。
  - 变量一般不允许重复声明（除了 _ 匿名变量）。
  - **小写字母开头**的变量不可以被外部包使用，**大写字母开头**的变量可以被外部包使用（大写字母头的变量可作为**全局变量**使用）。
  - Go 是强类型语言，只有 **相同类型**的变量才能赋值。

3.3 函数

- 基本概念

函数是基本的代码块。我们可以按照功能和任务把代码划分为不同的函数，每一个函数对应一个功能可以反复调用，以提高代码的复用率。

函数执行到最后的 } 之前或者遇到 return，将会停止运行。

Go 语言中，除了 init 和 main 函数，函数可以接收 0 个或者多个参数，也可以返回 0 个或者多个。

- 函数定义

Go 语言中用关键字 func 声明一个函数，基本格式为：

```
func 函数名称(形参) (返回参数) {
    return 值
}
```

例如，实现两数相加功能函数 add，在 main 函数中调用：

```
package main

import "fmt"

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

func main() {
    a, b := 1, 2
    result := add(a, b)
    fmt.Println("a + b = ", result)
}
```

运行结果为：

```
a + b = 3
```

- 参数传递

上述代码中，在 main 函数中的 a 和 b 叫做**实参**，main 函数中通过 add(a, b) 来调用 add 函数，传实参 a 与 b；

然后 add 函数接收两个参数 a 和 b，我们称为**形参**，**实参**与**形参**一一对应，调用 add 函数时 Go 编器会把**实参**的值拷贝给**形参**。

虽然这里形参和实参的名称相同，其实他们是完全不同的变量，形参的作用域仅在 add 函数内部，参的作用域仅在 main 函数内部。

对于同类型的形参，可以简写：

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

- 返回值
    - 常规返回

返回一个参数：

返回一个参数可以省略 ()

```
func one() int {
    return 1
}
```

返回多个参数，例如我们需要返回三个参数，类型分别为 int, string, bool，可以这样定义：

返回多个参数不可以省略 ()

```
func three() (int, string, bool) {
    return 1, "abc", true
}
```

- 参数返回

我们可以给返回值指定参数，然后将这些参数返回。

```
func three() (a int, s string, b bool) {
    a, s, b = 1, "abc", true
    return a, s, b
}
```

在 Go 语言中，如果给返回值指定了参数，return 语句后可以省略返回参数，Go 编译器会帮我们处
。

```
func three() (a int, s string, b bool) {
    a, s, b = 1, "abc", true
    return // 等价于 return a, s, b
}
```

- 参数接收

我们可以通过定义变量或赋值的方式来接收函数的返回参数：

```
package main

func three() (a int, s string, b bool) {
    a, s, b = 1, "abc", true
    return // 等价于 return a, s, b
}

func main() {
    a, s, b := three()
}
```

有时，可能我们不需要使用某些返回参数，可以用 _ 匿名变量来接收：

```
package main

func three() (a int, s string, b bool) {
    a, s, b = 1, "abc", true
    return // 等价于 return a, s, b
}

func main() {
```

```
    a, _, _ := three()
}
```

_ 匿名变量接收的参数将被抛弃，也无法使用。

- 注意
  - 调用函数时，注意实参和形参 **位置**和**类型**要一一对应。
  - 函数也可以作为参数传递给另一个函数，但前提是作为参数的函数 **返回值类型**与**形参类型**一致。
  - Go 语言不支持函数重载（编写函数名相同，但形参或返回值不同的函数，编译无法通过）。