第二章 变量与常量

本章内容讲解变量与常量，其实变量无非就是值会变化的变量，比如人的年龄，杯盘里面盛放的东西，都是可以改变的，而常量其实就是我们常说的1,2,3,4,5 等一些固定的不会变化的数据值。接下来，我们先去学习一下变量的使用。

2.1变量

2.1.1什么是变量

1. 变量是计算机语言中储存数据的抽象概念。变量的功能是存储数据。变量通过变量名访问；
2. 变量的本质是计算机分配的一小块内存，专门用于存放指定数据，在程序运行过程中该数值可以发生改变；
3. 变量的存储往往具有瞬时性，或者说是临时存储，当程序运行结束，存放该数据的内存就会释放，而该变量就会消失；
4. Go 语言的变量名由字母、数字、下划线组成，首个字符不能为数字;
5. Go语法规定，定义的局部变量若没有被调用则编译错误。

变量相当于是对一块数据存储空间的命名，程序可以通过定义一个变量来申请一块数据存储空间，之后可以通过引用变量名来使用这块存储空间。

如何理解这个概念？需要去思考一下，当一个杯子用来装水那就是水杯，用来打人那就是凶器，用来放笔那就是个笔筒，用来装饰桌面，那就是个装饰品，你会发现，杯子依然是那个杯子，因为我们对这个杯子的使用不同，对这个杯子的称呼也不同。

以上的举例中，杯子就是概念中所说的那一块数据存储空间，我们可以重复的使用这个杯子，杯子里的东西可以变化，也就意味着，变量所代表的那一块空间一样可以重复使用，而这个空间中存储的数据，也就是变量的值是可以改变的；

综合以上，变量具有以下特点：

a.变量所代表的内存空间可以重复使用

b.变量的值可以改变

2.1.2变量的声明

a.为了方便变量的使用，我们会给变量定义名字

b.变量所代表的内存空间要存放数据，我们会给变量赋值

c.变量有自己的所属类型

使用以上三点，来完成对杯子的描述，杯子属于什么？它是容器的一种，存放的若是水，我们便给这个容器取个名字叫水杯；

1、未初始化的标准格式

var变量名 变量类型

那么用代码去定义我们的水杯的话可以这样定义：

var 水杯 容器 = 水

用变量来表示一个人的年龄可以定义为：

var age int = 19;

用变量来表示一个人的名字可以定义为：

var name string = ”张三”

为了不重复的写var这个关键字，我们可以将若干个需要声明的变量放在一起，如下

var(

age int = 19

name string = “张三”

)

例：2.1

**func** main() {

    //年龄变量

**var** age int = 19

    //姓名变量

**var** name string = "张三"

**var** (

        age2 int = 18

        name2 string = "李四"

    )

    fmt.Println(age, name, age2, name2)

}

以上代码演示结果：

19 张三 18 李四

● 未初始化变量的默认值：

○ 整形和浮点型变量默认值：0

○ 字符串默认值为空字符串

○ 布尔型默认值为false

○ 函数、指针变量初始值为nil

2.1.3变量的初始化

变量的初始化也就给声明的变量赋值，

初始化变量的标准格式

var 变量名 类型 = 表达式

var v1 int = 10 // 正确的使用方式1

4、初始化变量的编译器自动推断类型格式

var 变量名 = 表达式

var v2 = 10 // 正确的使用方式，编译器可以自动推导出v2的类型

5、初始化变量的简短声明格式（短变量声明格式）

变量名 := 表达式

v3 := 10 // 正确的使用方式，编译器可以自动推导出v3的类型

使用 := 赋值操作符，:= 可以高效地创建一个新的变量，称之为初始化声明。

声明语句省略了var 关键字

变量类型将由编译器自动推断

这是声明变量的首选形式，但是它只能被用在函数体内，而不可以用于全局变量的声明与赋值

该变量名必须是没有定义过的变量，若定义过，将发生编译错误

例如：2.2

**func main() {**

**var i int**

**i := 99**

**fmt.println(i)**

**}**

导致一下编译错误：

no new variables on left side of :=

以上错误的根本原因在于：var i int意味着声明了一个名称为i的变量，i:=99同样是声明了一个名称为i的变量，同一代码片段中出现了两个同名的变量，就会导致调用时的混乱，因此不允许出现该情况，就像我们为了区分两个孩子不会让两个孩子使用同一个名字同理。

注意：同一代码片段中，不可声明重名变量。

在多个短变量声明和赋值中，至少有一个新声明的变量出现在左侧中，那么即便有其它变量名可能是重复声明的，编译器也不会报错。

Go语言中，变量类型之所以可以不必指定，是因为go的编译器可以从初始化表达式的右值推导出该变量应声明为哪种类型，这让go语言看上去有点像弱类型语言，尽管go语言实际上是强类型语言。

2.1.4变量的赋值

变量的赋值过程，分别如下：例2.3

**func** main() {

**var** i = 9 //直接赋值

**var** j int //没有直接赋值，go语言机制默认使其持有默认值0

     j = 19 //显式为上一行中的j变量赋值为19

     fmt.Println(i, j)

}

结果如下：

9 19

多个变量直接赋值：例2.4

**func** main() {

**var** a, b, c, d = 1, 2, 3, 4 //则依次赋值为a=1,b=2,c=3,d=4，简洁明了

     fmt.Println(a, b, c, d)

}

结果如下：

1 2 3 4

那么变量之间互换持有值的话，该如何做？

var a =1

var b = 2

我们希望a和b的持有值互换，在以往的编写过程中，我们需要声明一个第三方变量，

如下：例2.5

**func** main() {

**var** a = 2

**var** b = 3

**var** t = a //t=a=1连等关系

    a = b //a = b = 2

    b = t //b = t = 1

    fmt.Println("a=", a, "b=", b)

}

结果如下：

a= 3 b= 2

以上代码就完成了两个变量之间值的互换，是其他语言常用的方式，在go语言中，我们大可不必如此麻烦，一行代码就可以搞定：

例2.6

**func** main() {

**var** a = 2

**var** b = 3

     a, b = b, a

     fmt.Println("a=", a, "b=", b)

}

结果如下：

a= 3 b= 2

多重赋值的特性在Go语言库的实现中也被使用得相当充分，在介绍函数的多重返回值时， 将对其进行更加深入的介绍。总而言之，多重赋值功能让Go语言与C/C++，java等语言相比可以非常明显地减少代码行数。

2.1.5匿名变量

匿名变量，简而言之就是没有名字的变量，那么大家就需要去思考没有名字的变量怎么使用？使用一个变量，不是为了方便使用才需要去给定义名字的吗，没有名字还怎么使用呢？难道是我不用这个变量了？既然不用这个变量，我为什么还要定义这个变量呢？这不是自相矛盾？

大家思考一个问题，当一件事有两个结果，而我只需要其中一个的时候，是不是我只要拿到我需要拿到的结果就可以了？那么偏偏有两个，那个不需要的结果我不用做处理，但是偏偏需要给它留一个位置出来，这个我不想拿到不想处理的结果，就可以用这个匿名变量去表示，当别人看得到这个匿名变量的存在，就知道我不想要这个结果了。

假设GetName()函数的定义如下，它返回3个值，分别为firstName、lastName和 nickName：

func GetName() (firstName, lastName, nickName string) {

return "May", "Chan", "Chibi Maruko"

}

若只想获得nickName，则函数调用语句可以用如下方式编写： \_, \_, nickName := GetName() 这种用法可以让代码非常清晰，基本上屏蔽掉了可能混淆代码阅读者视线的内容，从而大幅降低沟通的复杂度和代码维护的难度。

具体代码实现如下：例2.7

**func** main() {

    \_, \_, nickName := GetName()

    fmt.Println(nickName)

}

**func** GetName() (firstName, lastName, nickName string) {

**return** "May", "Chan", "Chibi Maruko"

}

结果如下：

Chibi Maruko

也就是说，我们不想处理的结果可以用”\_”下划线来表示，这就是一个匿名变量。

2.2常量

常量也就是值是固定的，不会改变的量，常量可以是数值类型（包括整型、 浮点型和复数类型）、布尔类型、字符串类型等。

2.2.1字面常量

所谓字面常量（literal），是指程序中硬编码的常量，

如：

-3.14159265358979323846 // 浮点类型的常量

3.2+12i // 复数类型的常量

false // 布尔类型的常量

"go" // 字符串常量

123//整型的常量

对于这些常量的类型，大家会在go语言的数据类型中进行详细的学习。

2.2.2 常量定义

定义常量需要使用一个关键字：const，而格式就是：

const 常量名 [类型] = 值；

可以省略类型说明符 [类型]，因为编译器可以根据变量的值来自动推断其类型

如下：

const Pi float64 = 3.14159265358979323846

const zero = 0.0 // 无类型浮点常量

const (

size int64 = 1024

eof = -1 // 无类型整型常量

)

const u, v float32 = 0, 3 // u = 0.0, v = 3.0，常量的多重赋值

const a, b, c = 3, 4, "foo"

// a = 3, b = 4, c = "foo", 无类型整型和字符串常量

Go的常量定义可以限定常量类型，但不是必需的。如果定义常量时没有指定类型，那么它与字面常量一样，是无类型常量。

常量定义的右值也可以是一个在编译期运算的常量表达式，比如 const result = 2 << 5 由于常量的赋值是一个编译期行为，所以右值不能出现任何需要运行期才能得出结果的表达式，比如试图以如下方式定义常量就会导致编译错误： const Home = os.GetEnv("HOME") 原因很简单，os.GetEnv()只有在运行期才能知道返回结果，在编译期并不能确定，所以无法作为常量定义的右值。

2.2.3 预定义常量

Go语言预定义了这些常量：true、false和iota。 iota比较特殊，可以被认为是一个可被编译器修改的常量，在每一个const关键字出现时被重置为0，然后在下一个const出现之前，每出现一次iota，其所代表的数字会自动增1.

案例如下：例2.8

**func** main() {

**const** ( // iota被重设为0

        a = iota // a == 0

        b = iota // b == 1

        c = iota // c == 2

    )

**const** (

        x = iota \* 12 // x == 0

        y float64 = iota \* 12 // y == 12.0

        z = iota \* 12 // z == 24

    )

**const** j = iota // j == 0 (因为iota主要遇到const就会被重设为0)

**const** k = iota //k == 0 (同上)

    fmt.Println(a, b, c)

    fmt.Println(x, y, z)

    fmt.Println(j, k)

}

结果如下：

0 1 2

0 12 24

0 0

如果两个const的赋值语句的表达式是一样的，那么可以省略后一个赋值表达式。因此，上

面的第一个const语句可简写为：例2.9

**func** main() {

**const** ( // iota被重设为0

        a = iota // c0 == 0

        b // c1 == 1

        c // c2 == 2

    )

    fmt.Println(a, b, c)

}

结果如下：

0 1 2

5、示例一

const (

i = 1<<iota

j = 3<<iota

k

l

)

func main() {

fmt.Println("i=",i)

fmt.Println("j=",j)

fmt.Println("k=",k)

fmt.Println("l=",l)

}

● 打印输出结果：

○ i= 1

○ j= 6

○ k= 12

○ l= 24

6、示例二

const (

a1 = '一'

b1

c1 = iota

d1

)

func main() {

fmt.Println(a1, b1, c1, d1)

}

● 打印输出结果：

○ 19968 19968 2 3

2.2.4 枚举

枚举指一系列相关的常量，比如关于一个星期中每天的定义。通过上一节的例子，我们看到可以用在const后跟一对圆括号的方式定义一组常量，这种定义法在Go语言中通常用于定义枚举值。Go语言并不支持众多其他语言明确支持的enum关键字。

下面是一个常规的枚举表示法，其中定义了一系列整型常量：例2.10

**func** main() {

**const** (

        Sunday = iota

        Monday

        Tuesday

        Wednesday

        Thursday

        Friday

        Saturday

    )

    fmt.Println(Sunday, Monday)

}

结果如下：

0 1

以上枚举中的常量定义，首字母大写是为了在其他包中可以进行调用，若为小写，则在其他包不能调用。

枚举的作用多数则是为了限制某一变量或者是属性的持有值，比如性别的赋值多数为男或女，使用枚举则固定了对性别值的限制，不必再去判断该值是否符合现实逻辑。

以上便是变量和常量的内容，不必担心对这两个知识点的掌握情况，因为它们属于我们编写代码过程中没有机会遗忘的内容，熟能生巧，多练多用，是最好的掌握方式。