02 数据类型：你必须掌握的数据类型有哪些？

上节课的思考题是打印出自己的名字，这个作业比较简单，属于文本的替换，你只需要把我示例中的"Hello 世界"修改成自己的名字即可，比如以我的名字为例，替换为“飞雪无情”。

经过上一节课的学习，你已经对 Go 语言的程序结构有了初步了解，也准备好了相应的开发环境。但是一个完整的项目需要更复杂的逻辑，不是简单的“Hello 世界”可相比的。这些逻辑通过变量、常量、类型、函数方法、接口、结构体组成，这节课我就将带你认识它们，让你的 Go 语言程序变得更加生动。

变量声明

变量代表可变的数据类型，也就是说，它在程序执行的过程中可能会被一次甚至多次修改。

在 Go 语言中，通过 var 声明语句来定义一个变量，定义的时候需要指定这个变量的类型，然后再为它起个名字，并且设置好变量的初始值。所以 var 声明一个变量的格式如下：

```
var 变量名 类型 = 表达式
```

现在我通过一个示例来演示如何定义一个变量，并且设置它的初始值：

ch02/main.go

```
package main
import "fmt"
func main() {
    var i int = 10
    fmt.Println(i)
}
```

观察上面例子中 main 函数的内容，其中 var i int = 10 就是定义一个类型为 int（整数）、变量名为 i 的变量，它的初始值为 10

这里为了运行程序，我加了一行 fmt.Println(i)，你在上节课中就见到过它，表示打印出变量 i 的值。

这样做一方面是因为 Go 语言中定义的变量必须使用，否则无法编译通过，这也是 Go 语言比较好的特性，防止定义了变量不使用，导致浪费内存的情况；另一方面，在运行程序的时候可以查看变量 i 的结果。

通过输入 go run ch02/main.go 命令回车运行，即可看到如下结果：

```
$ go run ch02/main.go
10
```

打印的结果是10，和变量的初始值一样。

因为 Go 语言具有类型推导功能，所以也可以不去刻意地指定变量的类型，而是让 Go 语言自己推导，比如变量 i 也可以用如下的方式声明：

```
var i = 10
```

这样变量 i 的类型默认是 int 类型。

你也可以一次声明多个变量，把要声明的多个变量放到一个括号中即可，如下面的代码所示：

```
var (
    j int= 0
    k int= 1
)
```

同理因为类型推导，以上多个变量声明也可以用以下代码的方式书写：

```
var (
    j = 0
    k = 1
)
```

这样就更简洁了。

其实不止 int 类型，我后面介绍的 float64、bool、string 等基础类型都可以被自动推导，也就是可以省略定义类型。

演示项目目录结构

为了让你更好地理解我演示的例子，这里我给出演示项目的目录结构，以后的所有课时都会按照这个目录进行演示。

我的演示项目结构如下所示：

```
gotour
├── ch01
│   └── main.go
├── ch02
│   └── main.go
└── go.mod
```

其中 gotour 是演示项目的根目录，所有 Go 语言命令都会在这里执行，比如 go run。

ch01、ch02 这些目录是按照课时命名的，每一讲都有对应的目录，便于查找相应的源代码。具体的 Go 语言源代码会存放到对应的课时目录中。

基础类型

任何一门语言都有对应的基础类型，这些基础类型和现实中的事物一一对应，比如整型对应着 1、2、3、100 这些整数，浮点型对应着 1.1、3.4 这些小数等。Go 语言也不例外，它也有自己丰富的基础类型，常用的有：整型、浮点数、布尔型和字符串，下面我就为你详细介绍。

整型

在 Go 语言中，整型分为：

- **有符号整型**：如 int、int8、int16、int32 和 int64。
- **无符号整型**：如 uint、uint8、uint16、uint32 和 uint64。

它们的差别在于，有符号整型表示的数值可以为负数、零和正数，而无符号整型只能为零和正数。

除了有用“位”（bit）大小表示的整型外，还有 int 和 uint 这两个没有具体 bit 大小的整型，它们的大小可能是 32bit，也可能是 64bit，和硬件设备 CPU 有关。

在整型中，如果能确定 int 的 bit 就选择比较明确的 int 类型，因为这会让你的程序具备很好的移植性。

在 Go 语言中，还有一种字节类型 byte，它其实等价于 uint8 类型，可以理解为 uint8 类型的别名，用于定义一个字节，所以字节 byte 类型也属于整型。

浮点数

浮点数就代表现实中的小数。Go 语言提供了两种精度的浮点数，分别是 float32 和 float64。项目中最常用的是 float64，因为它的精度高，浮点计算的结果相比 float32 误差会更小。

下面的代码示例定义了两个变量 f32 和 f64，它们的类型分别为 float32 和 float64。

ch02/main.go

```
var f32 float32 = 2.2

var f64 float64 = 10.3456

fmt.Println("f32 is",f32,",f64 is",f64)
```

运行这段程序，会看到如下结果：

```
$ go run ch02/main.go

f32 is 2.2 ,f64 is 10.3456
```

特别注意：在演示示例的时候，我会尽可能地贴出演示需要的核心代码，也就是说，会省略 package 和 main 函数。如果没有特别说明，它们都是放在main函数中的，可以直接运行。

布尔型

一个布尔型的值只有两种：true 和 false，它们代表现实中的“是”和“否”。它们的值会经常被用于一些判断中，比如 if 语句（以后的课时会详细介绍）等。Go 语言中的布尔型使用关键字 bool 定义。

下面的代码声明了两个变量，你可以自己运行，看看打印输出的结果。

ch02/main.go

```
var bf bool =false

var bt bool = true

fmt.Println("bf is",bf,",bt is",bt)
```

布尔值可以用于一元操作符 !，表示逻辑非的意思，也可以用于二元操作符 &&、||，它们分别表示逻辑和、逻辑或。

字符串

Go 语言中的字符串可以表示为任意的数据，比如以下代码，在 Go 语言中，字符串通过类型 string 声明：

ch02/main.go

```
var s1 string = "Hello"
var s2 string = "世界"
fmt.Println("s1 is",s1,",s2 is",s2)
```

运行程序就可以看到打印的字符串结果。

在 Go 语言中，可以通过操作符 + 把字符串连接起来，得到一个新的字符串，比如将上面的 s1 和 s2 连接起来，如下所示：

ch02/main.go

```
fmt.Println("s1+s2=",s1+s2)
```

由于 s1 表示字符串“Hello”，s2 表示字符串“世界”，在终端输入 go run ch02/main.go 后，就可以打印出它们连接起来的结果“Hello世界”，如以下代码所示：

```
s1+s2= Hello世界
```

字符串也可以通过 += 运算符操作，你自己可以试试 s1+=s2 会得到什么新的字符串。

零值

零值其实就是一个变量的默认值，在 Go 语言中，如果我们声明了一个变量，但是没有对其进行初始化，那么 Go 语言会自动初始化其值为对应类型的零值。比如数字类的零值是 0，布尔型的零值是 false，字符串的零值是 "" 空字符串等。

通过下面的代码示例，就可以验证这些基础类型的零值：

ch02/main.go

```
var zi int
var zf float64
```

```
var zb bool

var zs string

fmt.Println(zi,zf,zb,zs)
```

变量

变量简短声明

有没有发现，上面我们演示的示例都有一个 var 关键字，但是这样写代码很烦琐。借助类型推导，Go 语言提供了变量的简短声明 :=，结构如下：

```
变量名:=表达式
```

借助 Go 语言简短声明功能，变量声明就会非常简洁，比如以上示例中的变量，可以通过如下代码简短声明：

```
i:=10

bf：=false

s1:="Hello"
```

在实际的项目实战中，如果你能为声明的变量初始化，那么就选择简短声明方式，这种方式也是使用最多的。

指针

在 Go 语言中，指针对应的是变量在内存中的存储位置，也就说指针的值就是变量的内存地址。通过 & 可以获取一个变量的地址，也就是指针。

在以下的代码中，pi 就是指向变量 i 的指针。要想获得指针 pi 指向的变量值，通过*pi这个表达式即可。尝试运行这段程序，会看到输出结果和变量 i 的值一样。

```
pi:=&i

fmt.Println(*pi)
```

赋值

在讲变量的时候，我说过变量是可以修改的，那么怎么修改呢？这就是赋值语句要做的事情。最常用也是最简单的赋值语句就是 =，如下代码所示：

```
i = 20
```

```
fmt.Println("i的新值是",i)
```

这样变量 i 就被修改了，它的新值是 20。

常量

一门编程语言，有变量就有常量，Go 语言也不例外。在程序中，常量的值是指在编译期就确定好的，一旦确定好之后就不能被修改，这样就可以防止在运行期被恶意篡改。

常量的定义

常量的定义和变量类似，只不过它的关键字是 const。

下面的示例定义了一个常量 name，它的值是“飞雪无情”。因为 Go 语言可以类型推导，所以在常量声明时也可以省略类型。

ch02/main.go

```
const name =  "飞雪无情"
```

在 Go 语言中，只允许布尔型、字符串、数字类型这些基础类型作为常量。

iota

iota 是一个常量生成器，它可以用来初始化相似规则的常量，避免重复的初始化。假设我们要定义 one、two、three 和 four 四个常量，对应的值分别是 1、2、3 和 4，如果不使用 iota，则需要按照如下代码的方式定义：

```
const(
    one = 1
    two = 2
    three =3
    four =4
)
```

以上声明都要初始化，会比较烦琐，因为这些常量是有规律的（连续的数字），所以可以使用 iota 进行声明，如下所示：

```
const(
    one = iota+1
```

```
    two
    three
    four
)
fmt.Println(one,two,three,four)
```

你自己可以运行程序，会发现打印的值和上面初始化的一样，也是 1、2、3、4。

iota 的初始值是 0，它的能力就是在每一个有常量声明的行后面 +1，下面我来分解上面的常量：

1. one=(0)+1，这时候 iota 的值为 0，经过计算后，one 的值为 1。
2. two=(0+1)+1，这时候 iota 的值会 +1，变成了 1，经过计算后，two 的值为 2。
3. three=(0+1+1)+1，这时候 iota 的值会再 +1，变成了 2，经过计算后，three 的值为 3。
4. four=(0+1+1+1)+1，这时候 iota 的值会继续再 +1，变成了 3，经过计算后，four 的值为 4。

如果你定义更多的常量，就依次类推，其中 () 内的表达式，表示 iota 自身 +1 的过程。

字符串

字符串是 Go 语言中常用的类型，在前面的基础类型小节中已经有过基本的介绍。这一小结会为你更详细地介绍字符串的使用。

字符串和数字互转

Go 语言是强类型的语言，也就是说不同类型的变量是无法相互使用和计算的，这也是为了保证Go 程序的健壮性，所以不同类型的变量在进行赋值或者计算前，需要先进行类型转换。涉及类型转换的知识点非常多，这里我先介绍这些基础类型之间的转换，更复杂的会在后面的课时介绍。

以字符串和数字互转这种最常见的情况为例，如下面的代码所示：

ch02/main.go

```
i2s:=strconv.Itoa(i)
s2i,err:=strconv.Atoi(i2s)
fmt.Println(i2s,s2i,err)
```

通过包 strconv 的 Itoa 函数可以把一个 int 类型转为 string，Atoi 函数则用来把 string 转为 int。

同理对于浮点数、布尔型，Go 语言提供了 strconv.ParseFloat、strconv.ParseBool、strconv.FormatFloat 和 strconv.FormatBool 进行互转，你可以自己试试。

对于数字类型之间，可以通过强制转换的方式，如以下代码所示：

```
i2f:=float64(i)

f2i:=int(f64)

fmt.Println(i2f,f2i)
```

这种使用方式比简单，采用“类型（要转换的变量）”格式即可。采用强制转换的方式转换数字类型，可能会丢失一些精度，比如浮点型转为整型时，小数点部分会全部丢失，你可以自己运行上述示例，验证结果。

把变量转换为相应的类型后，就可以对相同类型的变量进行各种表达式运算和赋值了。

Strings 包

讲到基础类型，尤其是字符串，不得不提 Go SDK 为我们提供的一个标准包 strings。它是用于处理字符串的工具包，里面有很多常用的函数，帮助我们对字符串进行操作，比如查找字符串、去除字符串的空格、拆分字符串、判断字符串是否有某个前缀或者后缀等。掌握好它，有利于我们的高效编程。

以下代码是我写的关于 strings 包的一些例子，你自己可以根据strings 文档自己写一些示例，多练习熟悉它们。

ch02/main.go

```
//判断s1的前缀是否是H

fmt.Println(strings.HasPrefix(s1,"H"))

//在s1中查找字符串o

fmt.Println(strings.Index(s1,"o"))

//把s1全部转为大写

fmt.Println(strings.ToUpper(s1))
```

总结

本节课我讲解了变量、常量的声明、初始化，以及变量的简短声明，同时介绍了常用的基础类型、数字和字符串的转换以及 strings 工具包的使用，有了这些，你就可以写出功能更

强大的程序。

在基础类型中，还有一个没有介绍的基础类型——复数，它不常用，就留给你来探索。这里给你一个提示：复数是用 complex 这个内置函数创建的。

本节课的思考题是：如何在一个字符串中查找某个字符串是否存在？提示一下，Go 语言自带的 strings 包里有现成的函数哦。

上一页 下一页