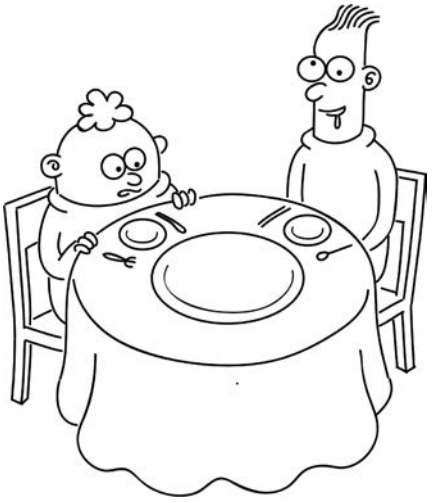
第 3 章

Go 语言的数据类型



在前面已经用到一些数据类型，如整数类型和字符串类型等，本章重点介绍基本数据类型。

微课视频

3.1 Go 语言的数据类型概述

Go 语言的数据类型可以分为基本数据类型、复合数据类型和自定义数据类型，如图 3-1 所示。

说明如下：

（1）基本数据类型是 Go 语言的内置数据类型，主要分为布尔类型、数值类型和字符串类型，其中数值类型又分为整数类型和浮点类型。

（2）复合数据类型是由基本数据类型组合出来的数据类型，如图 3-2 所示。复合数据类型将在第 5 章详细介绍，这里不再赘述。

（3）自定义数据类型是用户自定义的数据类型，需要通过 type 关键字定义。

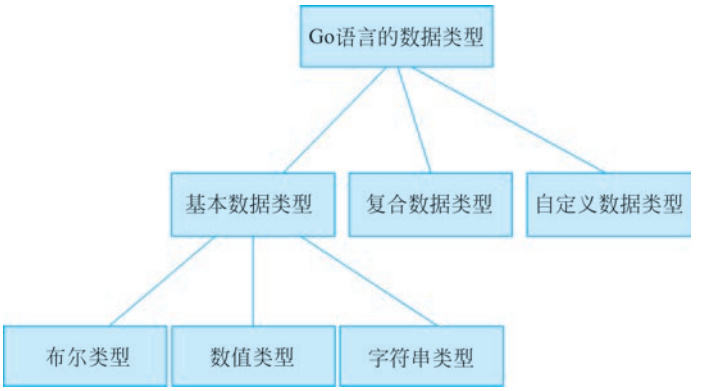


图 3-1　Go 语言的数据类型

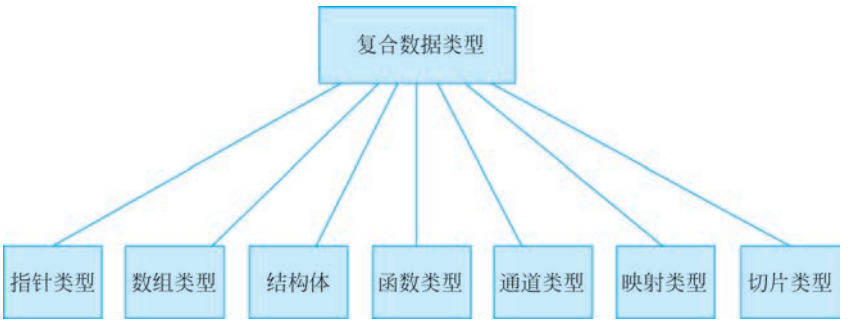


图 3-2　复合数据类型

3.2　整数类型


微课视频

Go 语言中整数类型分为以下两类。

（1）与平台无关整数类型：如图 3-3 所示，数据占用的内存空间与平台无关，占用空间分别为 8 位、16 位、32 位和 64 位，又可分为有符号整数和无符号整数。

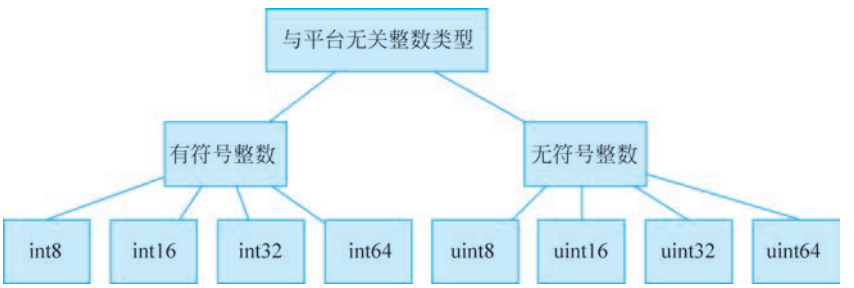


图 3-3　与平台无关整数类型

（2）与平台相关整数类型：如图 3-4 所示，数据类型占用的内存空间是由系统决定的。

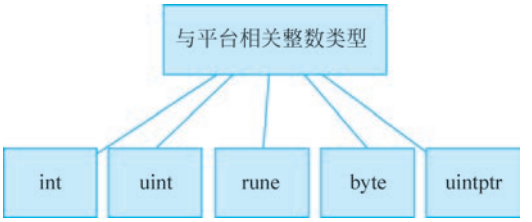


图 3-4　与平台相关整数类型

整数的默认类型是 int。

这些数据类型说明如表 3-1 所示。

表 3-1　数据类型说明

数据类型	占用空间（单位：位）	数据类型	占用空间（单位：位）
int8	8	uint16	16
int16	16	uint32	32
int32	32	uint64	64
uint	与平台相关	byte	等价于 uint8
int	与平台相关	uintptr	无符号的指针
int64	64	rune	等价于 int32
uint8	8		

Go 语言中的整数类型默认是 int，例如 19 表示的是十进制整数。其他进制（如二进制、八进制和十六进制）整数表示方式分别如下：

（1）二进制数：以 0b 或 0B 为前缀，注意 0 是阿拉伯数字，例如 0B10011 表示十进制数 19。

（2）八进制数：以 0o 或 0O 为前缀，第一个字符是阿拉伯数字 0，第二个字符是英文字母 o 或 O，例如 0O23 表示十进制数 19。

（3）十六进制数：以 0x 或 0X 为前缀，注意 0 是阿拉伯数字，例如 0X13 表示十进制数 19。

示例代码如下：

```
// 3.2 整数类型
package main

import "fmt"

func main() {

    // 声明变量

    var int0 = 19                    // 十进制表示的 19
```

```
    var int1 int8 = 0b10011              // 二进制表示的 19
    var int2 int16 = 0o23                // 八进制表示的 19
    var int3 int32 = 0x13                // 十六进制表示的 19

    fmt.Printf("十进制数 %d\n", int0)
    fmt.Printf("int0 的类型 : %T\n", int0)

    fmt.Printf("二进制数 %b\n", int1)
    fmt.Printf("int1 的类型 : %T\n", int1)

    fmt.Printf("八进制制数 %o\n", int2)
    fmt.Printf("int2 的类型 : %T\n", int2)

    fmt.Printf("十六进制制数 %x\n", int3)
    fmt.Printf("int3 的类型 : %T\n", int3)

}
```

上述代码执行结果如下：

```
十进制数 19
int0 的类型 : int
二进制数 10011
int1 的类型 : int8
八进制制数 23
int2 的类型 : int16
十六进制制数 13
int3 的类型 : int32
```

3.3 浮点类型


微课视频

浮点类型主要用来存储小数。Go 语言提供了两种精度的浮点数：32 位浮点数 float32 和 64 位浮点数 float64，默认类型是 float64。

浮点类型可以使用小数表示，也可以使用科学记数法表示，科学记数法中会使用大写或小写的 e 表示 10 的指数，如 e2 表示 10^2。

示例代码如下：

```
// 3.3 浮点类型
package main

import "fmt"

func main() {

    // 声明变量
    var float1 = 0.0                    // 浮点数 0
    var float2 float32 = 2.154327
    var float3 float64 = 2.1543276e2    // 科学记数法表示浮点数
```

```
    var float4 = 2.1543276e-2          // 科学记数法表示浮点数

    fmt.Printf("float1: %f\n", float1)
    fmt.Printf("float2 的类型 : %T\n", float2)

    fmt.Printf("float3 的类型 : %f\n", float3)
    fmt.Printf("float3 的类型 : %T\n", float3)

    fmt.Printf("float4 的类型 : %f\n", float4)
    fmt.Printf("float4 的类型 : %T\n", float4)
}
```

上述代码执行结果如下：

```
float1: 0.000000
float2 的类型 : float32
float3 的类型 : 215.432760
float3 的类型 : float64
float4 的类型 : 0.021543
float4 的类型 : float64
```


微课视频

3.4 复数类型

复数在数学中是非常重要的概念，无论是理论物理学，还是电气工程实践中都经常使用。很多计算机语言都不支持复数，而 Go 语言是支持复数的，这使得 Go 语言能够很好地用于科学计算。

Go 语言中的复数类型有两种：complex128（64 位实部和 64 位虚部）和 complex64（32 位实部和 32 位虚部），其中 complex128 为默认的复数类型。

示例代码如下：

```
// 3.4 复数类型
package main

import "fmt"

func main() {

    // 声明变量

    var complex1 complex128 = complex(2, -3)     // 声明实部为 2,虚部为 - 3 的复数
    var complex2 complex64 = complex(9, 2)       // 通过 complex()函数创建复数,该函数第
                                                 // 1 个参数是实部,第 2 个参数是虚部

    fmt.Println(complex1)
    fmt.Println(complex2)

}
```

上述代码执行结果如下：

```
(2 - 3i)
(9 + 2i)
```

3.5　布尔类型


微课视频

Go 语言中的布尔类型为 bool，它只有两个值：true 和 false。

示例代码如下：

```
// 3.5 布尔类型
package main

import "fmt"

func main() {

	// 声明变量
	var male bool = true
	var female = false

	fmt.Println(male)
	fmt.Println(female)

	fmt.Printf("male 的类型 : %T\n", male)
	fmt.Printf("female 的类型 : %T\n", female)
```

上述代码执行结果如下：

```
true
false
male 的类型 : bool
female 的类型 : bool
```

3.6　类型转换


微课视频

学习了前面的数据类型后，读者可能会思考一个问题：数据类型之间是否可以相互转换呢？数据类型的转换情况比较复杂。在基本数据类型中，数值类型之间可以互相转换，但字符类型和布尔类型不能与数值类型相互转换。

> 注意　Go 语言中不存在隐式类型转换，所有类型都必须显式类型转换。

显式类型转换语法格式如下：

```
目标数据类型(表达式)
```

显式转换示例代码如下：

```
// 3.6 类型转换
package main

import "fmt"

func main() {

    // 声明变量
    var sum int = 17
    var count int = 5
    var mean float32

    mean = float32(sum) / float32(count)                                    ①
    fmt.Printf("mean : %f\n", mean)

    var long1 int64 = 999999999999
    var int1 int32 = int32(long1)   // 将 int64 类型强制转换为 int32 类型,精度丢失   ②
    fmt.Println(int1)               // -727379969

    var float1 = 3.456
    var int2 = int32(float1)        // 小数部分被截掉                          ③
    fmt.Println(int2)

}
```

上述代码第①行将 sum 变量转换为 float32 数据类型，将 count 变量转换为 float32 数据类型。

需要注意的是，代码第②行经过类型转换后，原本的 999999999999 变成了负数。当字节长度大的数值转换为字节长度小的数值时，大数值的高位被截掉，这样就会导致数据精度丢失。

代码第③行浮点数转换为整数时，小数部分会被截掉。

上述代码执行结果如下：

```
mean : 3.400000
-727379969
```

3.7 字符串类型

一个字符串是一个不可改变的字节序列，其中每个字符均采用 UTF-8 编码，UTF-8 编码可以表示世界上存在的任何语言的字符。


微课视频

3.7.1 字符串表示

Go 语言中的字符串是用双引号("")包裹起来表示的。

示例代码如下：

```
// 3.7.1 字符串表示
package main

import "fmt"

func main() {

	// 声明字符串变量
	var s1 = "Hello"
	// 字符串拼接
	var s2 = s1 + "World."
	// 采用 Unicode 编码表示字符串
	var s3 = "\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u0064"
	//汉字字符串
	const s4 = "世界你好"

	fmt.Println(s2)
	fmt.Println(s3)
	fmt.Println(s4)

	// 通过 len()函数获得字符串长度
	fmt.Printf("s1 字符串长度: %d\n", len(s1))
	fmt.Printf("%c\n", s1[0])

}
```

上述代码执行结果如下：

```
HelloWorld.
Hello World
世界你好
s1 字符串长度:5
```

3.7.2 字符转义


微课视频

如果想在字符串中包含一些特殊的字符，例如换行符、制表符等，在普通字符串中则需要转义，前面要加上反斜线“\”，这称为字符转义。用于转义的字符称为转义符，表 3-2 所示是常用的转义符。

表 3-2 常用的转义符

转 义 符	Unicode 编码	说 明
\t	\u0009	水平制表符 tab
\n	\u000a	换行
\r	\u000d	回车
\"	\u0022	双引号
\'	\u0027	单引号
\\	\u005c	反斜线

示例代码如下：

```
// 3.7.2 字符转义
package main

import "fmt"

func main() {
	// 声明变量
	var s1 = "\"世界\"你好!"          // 转义双引号
	var s2 = "Hello\t World"          // 转义制表符
	var s3 = "Hello\\ World"          // 转义反斜线
	var s4 = "Hello\n World"          // 转义换行符

	fmt.Println("s1", s1)
	fmt.Println("s2", s2)
	fmt.Println("s3", s3)
	fmt.Println("s4", s4)
}
```

上述代码执行结果如下：

```
s1 "世界"你好!
s2 Hello World
s3 Hello\ World
s4 Hello
World
```


微课视频

3.7.3 原始字符串

如果字符串中有很多特殊字符都需要转义，使用转义符就非常麻烦，也不美观。这种情况下可以使用原始字符串(rawstring)表示。原始字符串使用反引号(`)包裹起来，其中的特殊字符不需要转义，按照字符串的本来“面目”呈现。例如在 Windows 系统中，tony 用户 Documents 文件夹下面的\readme.txt 文件的路径表示如下：

```
C:\Users\tony\Documents\readme.txt
```

由于文件路径分隔用反斜线表示，在程序代码中用普通字符串表示时需要将反斜线进行转义，而路径中有很多反斜线，所以很麻烦。如果采用原始字符串，就比较简单了，示例代码如下：

```
// 3.7.3 原始字符串
package main

import "fmt"

func main() {
	// 声明变量
	// 采用普通字符串表示文件路径，其中的反斜线需要转义
	const filepath1 = "C:\\Users\\tony\\Documents\\readme.txt"
```

```
    // 采用原始字符串表示文件路径,其中的反斜线不需要转义
    const filepath2 = `C:\Users\tony\Documents\readme.txt`

    fmt.Println("路径 1:", filepath1)
    fmt.Println("路径 2:", filepath2)

    // 声明长字符串 s1

    const s1 = `
        «将进酒»
君不见黄河之水天上来,    奔流到海不复回。
君不见高堂明镜悲白发,    朝如青丝暮成雪。
人生得意须尽欢,    莫使金樽空对月。
天生我材必有用,    千金散尽还复来。
烹羊宰牛且为乐,    会须一饮三百杯。
岑夫子,丹丘生,    将进酒,杯莫停。
与君歌一曲,  请君为我倾耳听。
钟鼓馔玉不足贵,    但愿长醉不复醒。
古来圣贤皆寂寞,    惟有饮者留其名。
陈王昔时宴平乐,    斗酒十千恣欢谑。
主人何为言少钱,    径须沽取对君酌。
五花马,千金裘,    呼儿将出换美酒,
与尔同销万古愁。
`

    fmt.Println("长字符串 s1:", s1)

}
```

3.7.4　操作字符串的常用函数


微课视频

Go 语言的 strings 包提供了很多操作字符串的函数,下面介绍几个常用的操作字符串函数。

(1) func Contains(s, substr string) bool：判断字符串 s 中是否包含字符串 substr。

(2) func Replace(s, old, new string, n int) string：用 string 替换字符串 s 中的 old 字符串,并返回替换后的字符串,其中参数 n 是指定替换的个数。

(3) func ToUpper(s string) string：将字符串 s 中的所有字母转换为大写字符。

(4) func ToLower(s string) string：将字符串 s 中的所有字母转换为小写字符。

(5) func Split(s, sep string) []string：将字符串 s 按照 sep 进行分割,返回字符切片。

```
// 3.7.4 操作字符串的常用函数
package main

import (
    "fmt"
    "strings"                                    // 导入字符串包
)

func main() {
```

```
    // 短变量声明字符串
    string1 := "Hello"
    string2 := "hello"
    result := string1 == string2                                    ①
    fmt.Println(result)                          //输出 false

    // 短变量声明字符串
    text1 := "Go Programming"
    substring1 := "Go"
    // 在 text1 中查找是否包含 substring1 字符串
    result = strings.Contains(text1, substring1)                    ②
    fmt.Println(result)                          // 输出 true

    text2 := "car"
    fmt.Println("旧字符串:", text2)

    // 把 r 替换为 t
    replacedText := strings.Replace(text2, "r", "t", 1)             ③

    fmt.Println("新字符串:", replacedText)    // 输出 cat

    text3 := "I Love Golang"

    // 转换为大写
    text4 := strings.ToUpper(text3)                                 ④

    fmt.Println(text4)                           // 输出 I LOVE GOLANG
    // 转换为小写
    var text5 = strings.ToLower(text3)                              ⑤
    fmt.Println(text5)                           // 输出 i love golang

    // 用空格分割字符串" "
    splittedString := strings.Split(text3, " ")                     ⑥
    fmt.Println(splittedString)                  // 输出[I Love Golang]
}
```

上述代码第①行通过“==”比较字符串是否相等。

代码第②行通过字符串的 Contains()方法判断是否包含子字符串。

代码第③行通过字符串的 Replace()方法替换字符。

代码第④行将字符串中的所有字母转换为大写，代码第⑤行将字符串中的所有字母转换为小写。

代码第⑥行通过字符串的 Split()方法分割字符串，其中第 1 个参数是要被分割的字符串，第 2 个参数是分割字符串，返回字符串数组。

上述代码执行结果如下：

```
false
true
旧字符串: car
新字符串: cat
```

```
I LOVE GOLANG
i love golang
[I Love Golang]
```

3.8　动手练一练

选择题

(1) 下面哪行代码在编译时不会出现警告或错误信息？(　　)

A. var f = 1.3　　B. var f int = 13　　C. f:= 13　　D. var f int = 13

(2) int8 的取值范围是(　　)。

A. −128～127　　B. −256～256

C. −255～256　　D. 依赖于计算机本身硬件

(3) 下列选项中不是 Go 语言的基本数据类型是(　　)。

A. short　　B. int16　　C. int　　D. float

(4) 下列选项中关于原始字符串的说法正确的是(　　)。

A. 需使用单引号(')包裹起来　　B. 需使用双引号(")包裹起来

C. 需使用三重单引号(''')包裹起来　　D. 需使用反引号(`)包裹起来