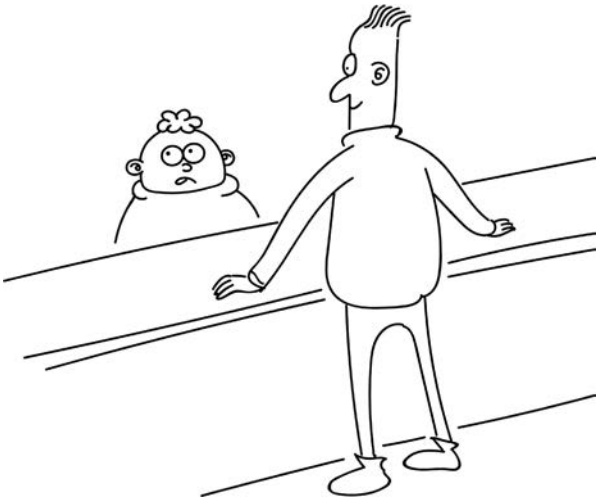
第 5 章

复合数据类型



第 3 章介绍了基本数据类型，这些数据类型只能存储单个数据值。由基本数据类型组合而成的数据类型称复合数据类型，复合数据类型又分为指针、数组、切片、映射、结构体、函数和管道等类型，本章重点介绍指针、数组、切片、映射等复合数据类型。

微课视频

5.1 指针

指针是用来保存其他变量内存地址的变量。一个变量初始化后，如 x 变量被赋值为 100 后，计算机会为该变量分配内存空间，假设变量 x 的内存地址是 0x61ff08，声明一个指针变量 ptr，它将保存变量 x 的内存地址 0x61ff08，如图 5-1 所示。

5.1.1 声明指针变量

声明指针变量的语法格式如下：

```
var 变量名 *变量类型
```

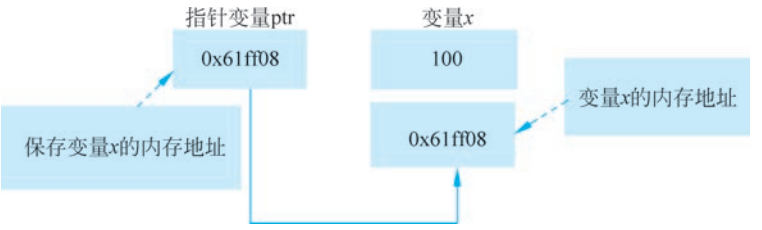


图 5-1　指针

注意　*间接寻址运算符与数据类型及变量名之间可以间隔任意多个空格或制表符，但一般推荐间隔一个空格。

声明指针变量示例代码如下：

```
// 5.1.1 声明指针变量
package main

import "fmt"

func main() {

    // 声明变量 x
    var x int = 100
    fmt.Printf("变量 x 的内存地址：%x\n", &x)              ①

    // 声明并初始化指针变量 ptr
    var ptr *int = &x                                    ②

    fmt.Printf("指针变量 ptr 的值是：%d\n", *ptr)          ③
}
```

上述代码第①行中表达式 &x 获取变量 x 的内存地址，其中“&”是取地址运算符。代码第②行声明并初始化指针变量 ptr，它保存了变量 x 的内存地址。代码第③行打印指针变量 ptr，访问指针变量需要使用 *ptr 表达式，即要在指针变量 ptr 前加上“*”。

上述代码执行结果如下：

```
变量 x 的内存地址:c000122058
指针变量 ptr 的值是:100
```

5.1.2　空指针


微课视频

如果指针变量没有初始化，那么它所指向的内存地址为 0，表示变量没有分配内存空间，这就是空指针。

示例代码如下：

```
// 5.1.2 空指针
```

```
package main

import "fmt"

func main() {

    var ptr * int
    fmt.Printf("指针 ptr 的值是：% x\n", ptr)
    // 判断 ptr 是否为空指针
    if ptr == nil {                          ①
       fmt.Println(" ptr 是空指针")
    }
}
```

上述代码第①行判断指针是否为空，其中 nil 是空指针值。

上述代码执行结果如下：

```
指针 ptr 的值是:0
ptr 是空指针
```


微课视频

5.1.3 二级指针

二级指针就是指向指针的指针。如图 5-2 所示，变量 x 的内存地址是 0x61ff08；指针变量 ptr 保存变量 x 的内存地址，ptr 也会占用内存空间，也有自己的内存地址 0x61ff10；指针变量 pptr 保存了指针变量 ptr 的内存地址，pptr 是指向 ptr 指针的指针，即二级指针。

图 5-2 二级指针

示例代码如下：

```
// 5.1.3 二级指针
package main

import "fmt"

func main() {

    var x int              // 声明整数变量 x
    var ptr * int          // 声明指针变量
    var pptr ** int        // 声明二级指针变量              ①
    x = 300                // 初始化变量 x

    ptr = &x               // 获取变量 x 的内存地址
    pptr = &ptr            // 获取指针变量 ptr 的内存地址    ②

    fmt.Printf("x: % d\n", x)
    fmt.Printf(" * ptr =  % d\n", * ptr)
```

```
    fmt.Printf("** pptr = %d\n", **pptr)                ③
}
```

上述代码第①行声明二级指针变量，用两个星号表示。代码第②行获取指针变量 ptr 的内存地址。代码第③行通过 **pptr 参数访问指针变量的内容。

5.2 数组


微课视频

数组(Array)具有如下特性：

(1) 一致性：数组只能保存相同数据类型的元素。

(2) 有序性：数组中的元素是有序的，通过下标访问。

(3) 不可变性：数组一旦初始化，则长度(数组中元素的个数)不可变。

5.2.1 声明数组

使用数组之前首先要声明数组，声明数组可以采用下面两种形式。

(1) 标准形式声明：采用 var 关键字声明，语法格式如下：

```
var 数组变量名 = [length]datatype{values}      // 指定数组长度
```

或

```
var 数组变量名 = [...]datatype{values}         // 数组长度根据元素个数推断出来
```

其中 length 是数组的长度；datatype 是数组中元素的数据类型；values 是数组中的元素列表，元素之间用逗号“,”分隔。

(2) 采用短变量形式声明：即使用“:=”符号声明，语法格式如下：

```
数组变量名 := [length]datatype{values}         // 指定数组长度
```

或

```
数组变量名 := [...]datatype{values}            // 数组长度根据元素个数推断出来
```

声明数组的示例代码如下：

```
// 5.2.1 声明数组
package main

import "fmt"

func main() {
    // 采用标准格式声明 3 个元素的 int 类型数组
    var arr1 = [3]int{1, 2, 3}

    // 采用标准格式声明 3 个元素的 float32 类型数组
    var arr3 = [...]float32{1.2, 2.6, 3.6}

    // 采用短变量形式声明 5 个元素的 int 类型数组
```

```
    arr2 := [5]int{4, 5, 6, 7, 8}
    // 采用短变量形式声明 5 个元素的 float32 类型数组
    arr4 := [...]float32{4, 5, 6.3, 7.6, 5.8}

    fmt.Printf("arr1 = %d\n", arr1)
    fmt.Printf("arr2 = %d\n", arr2)
    fmt.Printf("arr3 = %f\n", arr3)
    fmt.Printf("arr4 = %f\n", arr4)
}
```

上述代码执行结果如下：

```
arr1 = [1 2 3]
arr2 = [4 5 6 7 8]
arr3 = [1.200000 2.600000 3.600000]
arr4 = [4.000000 5.000000 6.300000 7.600000 5.800000]
```

5.2.2 访问数组元素

数组的下标是从 0 开始的，事实上，很多计算机语言的数组下标都是从 0 开始的。Go 语言中的数组下标访问运算符是中括号，如 intArray[0]表示访问 intArray 数组的第 1 个元素，其中 0 是第 1 个元素的下标。

访问数组元素示例代码如下：

```
// 5.2.2 访问数组元素
package main

import "fmt"

func main() {
    arr1 := [4]string{"沃尔沃", "宝马", "福特", "奔驰"}
    arr2 := [...]int{1, 2, 3, 4, 5, 6}

    fmt.Println(len(arr1))                                    ①
    fmt.Println(len(arr2))

    fmt.Println(arr1[0])              // 打印第 1 个元素       ②
    fmt.Println(arr1[len(arr1) - 1])  // 打印最后一个元素

    // 声明循环变量
    var i, j int
    //声明 10 个元素 int 类型数组
    var n [10]int

    //遍历数组，设置数组元素
    for i = 0; i < 10; i++{
       n[i] = i + 100                 // 设置元素
    }
```

```
    //遍历数组,打印数组元素
    for j = 0; j < 10; j++{
       fmt.Printf("Element[ %d] = %d\n", j, n[j])
    }
}
```

上述代码第①行中的 len()函数可以获得数组的长度,代码第②行通过下标索引访问数组第 1 个元素。

上述代码执行结果如下:

```
4
6
沃尔沃
奔驰
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

5.3 切片


微课视频

在实际编程时,数组使用得并不多,这主要是因为数组是不可变的,在实际编程时常常会使用可变数组数据获得数据——切片(Slice)。

5.3.1 声明切片

若要声明切片,可以将其声明为数组,只是无须指定其大小。也可使用 make()函数创建切片,make()函数语法格式如下:

```
make([] T, len, cap)
```

其中,T 是切片元素数据类型;len 是切片的长度;cap 是切片的容量,预先分配元素数量,该参数是可选的。

示例代码如下:

```
// 5.3.1 声明切片
package main

import "fmt"

func main() {
```

```
    // 声明字符串切片
    strSlice1 := []string{"沃尔沃", "宝马", "福特", "奔驰"}
    // 声明 int 类型切片
    intSlice1 := []int{1, 2, 3, 4, 5, 6}
    // 使用 make()函数创建 int 切片
    var intSlice2 = make([]int, 10)              ①
    // 使用 make()函数创建字符串切片
    var strSlice2 = make([]string, 10, 20)       ②

    fmt.Println(intSlice1)
    fmt.Println(intSlice2)
    fmt.Println(strSlice1)
    fmt.Println(strSlice2)

}
```

上述代码第①行使用 make()函数创建切片，切片的长度和容量相同。代码第②行也是通过 make()函数创建切片，切片的长度和容量不同，容量 20 表示预先分配 20 个元素的空间，而长度 10 表示只使用了 10 个元素。

上述代码执行结果如下：

```
[1 2 3 4 5 6]
[0 0 0 0 0 0 0 0 0 0]
[沃尔沃 宝马 福特 奔驰]
[ ]
```


微课视频

5.3.2 使用切片操作符

Go 语言提供了一种操作符，用于实现从切片中切分出小的子切片，这种操作符称为切片操作符。切片操作符不仅适用于切片，也适用于数组。切片操作符语法格式如下：

```
切片名[startIndex:endIndex]
```

注意 使用切片操作符切分出的子切片中的元素不包括 endIndex。如果省略 startIndex，则从 0 开始切片，如果省略 endIndex，则切分到切片最后一个元素。

示例代码如下：

```
// 5.3.2 使用切片操作符

package main

import "fmt"

func main() {
    // 声明 9 个元素的 int 数组
    numbers := []int{0, 1, 2, 3, 4, 5, 6, 7, 8}                    ①
```

```
    // 打印原始切片 numbers
    fmt.Println("numbers == ", numbers)
    //切分出从索引 1 开始到索引 3 的子切片
    fmt.Println("numbers[1:4] == ", numbers[1:4])    // [1 2 3]

    //切分出从索引 0 开始到索引 2 的子切片
    fmt.Println("numbers[:3] == ", numbers[:3])      // [0 1 2]          ②

    //切分出从索引 4 开始到最后一个元素的子切片
    fmt.Println("numbers[4:] == ", numbers[4:])      //[4 5 6 7 8]

    // 声明字符串,字符串也是字符的切片
    a := "Hello"                                                         ③

    fmt.Println("a[4:] == ", a[4:])                  // o                ④
    fmt.Println("a[0:3] == ", a[0:3])                // Hel
    fmt.Println("a[0:5] == ", a[0:5])                // Hello
    fmt.Println("a[:] == ", a[:])                    // Hello
}
```

上述代码第①行声明一个 int 类型数组。

注意代码第②行是对 numbers 数组进行切片操作,其中的 startIndex 省略了,即从 0 开始切片。

代码第③行声明一个字符串变量 a,字符串本质上也是切片类型,所以也可以进行切片操作。

代码第④行 endIndex 省略了,即切分到切片最后一个元素。

上述代码执行结果如下:

```
numbers == [0 1 2 3 4 5 6 7 8]
numbers[1:4] == [1 2 3]
numbers[:3] == [0 1 2]
numbers[4:] == [4 5 6 7 8]
a[4:] == o
a[0:3] == Hel
a[0:5] == Hello
a[:] == Hello
```

5.3.3　添加切片元素


微课视频

添加切片元素需使用 append()函数,该函数的语法格式如下:

```
slice = append(slice, elem1, elem2, ...)
```

其中,第 1 个参数 slice 是要添加元素的切片,从第 2 个参数开始是要追加的元素。该函数返回值是追加完成后的切片。

示例代码如下:

```
// 5.3.3 添加切片元素

package main

import "fmt"

func main() {
    // 创建一个空的 int 类型切片
    var slice []int                    ①
    // 打印切片
    printSlice(slice)

    //追加 1 个元素
    slice = append(slice, 0)           ②
    printSlice(slice)
    //再次追加 1 个元素
    slice = append(slice, 1)           ③
    printSlice(slice)
    //一次追加多个元素
    slice = append(slice, 2, 3, 4)     ④
    printSlice(slice)

    //声明切片
    slice2 := []int{10, 20, 30}
    //把 slice2 追加到 slice 后
    slice = append(slice, slice2...)   ⑤
    printSlice(slice)

}

// 自定义打印切片函数
func printSlice(s []int) {
    fmt.Printf("length = %d %d\n", len(s), s)
}
```

上述代码第①行创建一个空的切片，该切片是 int 类型；代码第②行和第③行追加 1 个元素；代码第④行追加多个元素；代码第⑤行把 slice2 追加到 slice 后，注意“...”表示对 slice 切片进行解包(即拆开切片)，然后再重新追加元素到切片 slice。

上述代码执行结果如下：

```
length = 0 []
length = 1 [0]
length = 2 [0 1]
length = 5 [0 1 2 3 4]
length = 8 [0 1 2 3 4 10 20 30]
```

5.4 映射

映射(Map)表示一种非常复杂的集合,允许按照某个键访问元素。映射是由两个集合构成的,一个是键(key)集合,另一个是值(value)集合。键集合不能有重复的元素,而值集合可以有重复的元素。映射中的键和值是成对出现的。

图 5-3 所示是 Map 类型的国家代号集合,其键是国家代号,不能重复。

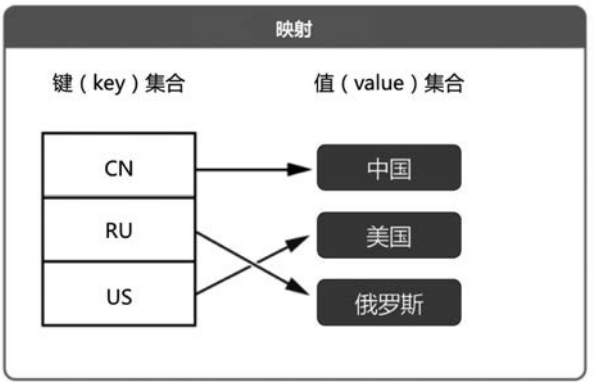


图 5-3 国家代号集合

提示 映射适用于通过键快速访问值,就像查英文字典一样,键就是要查的英文单词,而值是英文单词的翻译和解释等。有时,一个英文单词会对应多个翻译和解释,这是与 Map 集合特性对应的。

5.4.1 声明映射


微课视频

使用映射之前首先要声明映射,声明映射可以采用两种形式。

(1) 使用 map 关键字声明,语法格式如下:

```
var 映射变量名 = map[key_data_type]value_data_type{key:value...}
```

其中,key_data_type 是键的数据类型,value_data_type 是值的数据类型,key_data_type 和 value_data_type 之间可以有空格。{key: value...}是键-值对,放在大括号中,键-值对之间用逗号分隔。

(2) 使用 make()函数创建映射,语法格式如下:

```
映射变量名 = make(map[key_data_type]value_data_type)
```

其中,key_data_type 是键的数据类型,value_data_type 是值的数据类型。

示例代码如下:

```
// 5.4.1 声明映射
```

```
package main

import "fmt"

func main() {
    // 1.通过 map 关键字声明映射
    var countryCodeMap = map[string]string{"CN": "中国",                    ①
                          "RU": "俄罗斯", "US": "美国", "JP": "日本"}

    fmt.Printf("%v\n", countryCodeMap)                                    ②

    // 2.通过 make()函数声明映射

    var classMap = make(map[int]string)                                   ③
    classMap[102] = "张三"                                                ④
    classMap[105] = "李四"
    classMap[109] = "王五"
    classMap[110] = "董六"                                                ⑤

    fmt.Printf("%v\n", classMap)

}
```

上述代码第①行通过 map 关键字声明映射变量 countryCodeMap，采用键-值对初始化，键和值之间用冒号分隔，键-值对之间用逗号分隔。

代码第②行打印映射变量 countryCodeMap，它的格式化转换符是%v，采用默认格式表示。

代码第③行通过 map()函数声明映射变量 classMap，它的键是 int 类型，值是字符串类型。

代码第④～⑤行添加键-值对。

上述代码执行结果如下：

```
map[CN:中国 JP:日本 RU:俄罗斯 US:俄罗斯]
map[102:张三 105:李四 109:王五 110:董六]
```


微课视频

5.4.2 访问映射元素

访问映射中的值是通过键实现的，键是放在中括号“[]”里的，语法格式如下：

```
value, result = 映射变量名[key]
```

其中，key 是要从映射里访问的键。表达式“映射变量名[key]”返回值有两个，第一个是通过 key 从映射中返回的值 value；第二个是返回值 result，表示是否有与键对应的值，如果没有，则 result 是 false，如果有，则 result 是 true。

示例代码如下：

```
// 5.4.2 访问映射元素

package main

import "fmt"

func main() {
      // 通过 map()函数声明映射

      var classMap = make(map[int]string)
      classMap[102] = "张三"
      classMap[105] = "李四"
      classMap[109] = "王五"
      classMap[110] = "董六"

      fmt.Printf("%v\n", classMap)

      name, ok := classMap[102]       ①

      if ok {
        fmt.Printf("学号 102 是:%s\n", name)
      } else {
        fmt.Println("学号 102 不存在!")
      }
}
```

上述代码第①行通过键 102 访问映射 classMap,其中 value 是要返回的值。

上述代码执行结果如下:

```
map[102:张三 105:李四 109:王五 110:董六]
学号 102 是:张三
```

5.4.3 删除元素


微课视频

删除元素可以通过 delete()函数实现,该函数按照键删除对应的值。

示例代码如下:

```
// 5.4.3 删除元素

package main

import "fmt"

func main() {
      // 通过 map()函数声明映射

      var classMap = make(map[int]string)
      classMap[102] = "张三"
      classMap[105] = "李四"
      classMap[109] = "王五"
```

```
    classMap[110] = "董六"

    fmt.Printf("修改前：%v %d\n", classMap, len(classMap))
    // 109 键已经存在,替换原值"王五"
    classMap[109] = "李四"
    fmt.Printf("修改后：%v %d\n", classMap, len(classMap))
    //删除键-值对
    fmt.Printf("删除前：%v %d\n", classMap, len(classMap))
    delete(classMap, 102)                                          ①
    fmt.Printf("删除后：%v %d\n", classMap, len(classMap))
}
```

上述代码第①行通过 delete()函数删除 102 键-值对。注意，delete()函数的第一个参数是要删除的映射变量，第二个参数是要删除的键。

另外，len()函数可以获得映射的长度，即键-值对的个数。


微课视频

5.5 遍历容器

数组、切片和映射都属于容器数据，它们包含很多元素，因此遍历容器中的元素是常用的操作。Go 语言提供了 range 关键字，它可以帮助迭代数组、切片、映射和通道(channel)等容器数据中的元素。

使用 range 关键字迭代不同类型的数据时，会返回不同的数据。下面根据不同的容器类型分别介绍。

(1) 数组、切片：返回两个值，其中第一个值是索引，第二个值是元素。

(2) 映射：返回两个值，其中第一个值是键，第二个值是值。

(3) 字符串：返回两个值，其中第一个值是索引，第二个值是字符。

示例代码如下：

```
// 5.5 遍历容器

package main

import "fmt"

func main() {
    // 声明数组
    odd := [7]int{1, 3, 5, 7, 9, 11, 13}

    fmt.Println("-----遍历数组 odd------")

    for i, item := range odd {                                     ①

        // 打印索引和元素
        fmt.Printf("odd[ %d] = %d \n", i, item)
    }
```

```
    var str1 = "Hello world."
    fmt.Println("----- 遍历字符串 str1 ------")
    for i, item := range str1 {                          ②
        fmt.Printf("str1[%d] = %c \n", i, item)
    }

    var classMap = make(map[int]string)
    classMap[102] = "张三"
    classMap[105] = "李四"
    classMap[109] = "王五"
    classMap[110] = "董六"

    fmt.Println("----- 遍历映射 classMap ------")

    for k, v := range classMap {                         ③
        fmt.Printf("%v->%v\n", k, v)
    }
}
```

上述代码第①行遍历数组 odd，其中返回的 i 是元素的索引，item 是数组中的元素。代码第②行遍历字符串，因为字符串底层也是切片，所以也可以使用 range 关键字进行遍历。代码第③行遍历映射，其中第一个返回值 k 是键，第二个返回值 v 是值。

上述代码执行结果如下：

```
----- 遍历数组 odd ------
odd[0] = 1
odd[1] = 3
odd[2] = 5
odd[3] = 7
odd[4] = 9
odd[5] = 11
odd[6] = 13
----- 遍历字符串 str1 ------
str1[0] = H
str1[1] = e
str1[2] = l
str1[3] = l
str1[4] = o
str1[5] =
str1[6] = w
str1[7] = o
str1[8] = r
str1[9] = l
str1[10] = d
str1[11] = .
----- 遍历映射 classMap ------
102->张三
105->李四
109->王五
110->董六
```

5.6 动手练一练

1. 选择题

(1) array1 [10]float64 的元素类型是(　　)。

A. int　　B. string　　C. float32　　D. float64

(2) 数组声明为 var array1[20]float32，则其元素索引的最大值是(　　)。

A. 19　　B. 20　　C. 21　　D. 22

(3) 函数 make()可以用来创建(　　)数据类型实例。

A. 数组　　B. 切片　　C. 字符串　　D. 以上都可以

2. 判断题

(1) 可以用关键字 range 遍历映射，它返回两个值，其中第一个值是键，第二个值是值。(　　)

(2) 可以用关键字 range 遍历数组和切片，它返回两个值，其中第一个值是索引，第二个值是元素。(　　)