复合数据类型

前言

复合数据类型是由基本数据类型以各种方式组合而成的

- 数组
  - 元素具有相同的类型
- 结构体
  - 元素数据类型可以不同
- 固定长度
- slice
- map
- 动态数据结构，长度在元素添加到结构中的时候可以动态增长

4.1 数组

数组是具有固定长度且拥有零个或多个相同数据类型元素的序列

定义数组
- var a [3]int

初始化数组
数组字面量
- var q [3]int = [3]int{1, 2, 3}
- var p [3]int = [3]int{1, 2}
- q :=[...]int{1, 2, 3}
- symbol := [...]int{1: 2, 3: 5, 99: 2}
  - 可以给出索引和索引的值

如果出现省略号，则数组长度由初始化数组的元素个数确定

数组长度是数据类型一部分
[3]int and [4]int 是两种不同类型的数组
- 数组的长度必须是常量表达式

如果一个数组的元素类型是可比较的
->数组也是可比较的

因为 Go 是值传递，将数组传给一个函数形参是一个副本
如果想要修改数组，使用指针

由于数组长度不可变的特性，除了特定情况
我们很少使用数组

4.2slice

slice 表示一个拥有相同类型元素的可变长度的序列
slice 通常写成[]T,其中的元素类型都是 T

slice是一种轻量级的数据结构，可以用来访问数组的部分或全部元素
这个数组称为 slice 的底层数组

slice的三个属性
- 指针
  - 指针指向数组的第一个可访问元素，这个元素不一定是第一个数组元素
- 长度
  - 长度是 slice的元素个数，不能超过 slice 的容量
- 容量
  - 容量的大小通常是 slice 的起始元素到底层数组的最后一个元素间元素的个数

一个底层数组可以对应多个 slice

创建slice的方式
- 声明 var a []int
- 字面量方式 b := []int{1, 2} 注意[...]变成了数组
- make 方式 c := make([]int, 3)

slice 包含了指向数组元素的指针
将一个 slice 传递给函数的时候，可以在函数内部修改底层数组的元素

slice 无法做比较（即使用==），必须自己写函数来比较
- slice 有可能包含自身
- slice 元素不是直接的
  如果底层数组元素改变
  同一个 slice 在不同时间会有不同元素
- 散列表（如 map）的键必须在散列表的生命周期保持不变
  slice 不能作为散列表的键
- slice的零值是 nil
  - 值为 nil 的散列表长度和容量都是 0
  - []int{} 的长度和容量是 0，但不是 nil
- 检查散列表是否为空
  使用 len(slice)==0

make([]T,len,cap) 分别指定 len 和 cap
make([]T,len) len 和 cap 都是 len

4.2.1 append 函数

- append 用来将元素追加到 slice 后面
- copy 函数用来为两个拥有相同类型元素的 slice 复制元素
- 我们不清楚一次 append 调用会不会导致一次新的内存分配，所以我们不能假设原来的 slice 用 append 后的结果 slice 指向同一个底层数组，也无法证明它们指向不同的底层数组。同样，无法假设旧 slice 上对元素的操作会不会影响新的 slice 元素。所以，通常我们将 append 的调用结果再次赋值给传入 append 的 slice
- 不仅仅是在调用 append 函数的情况下需要更新 slice 变量。对于任何函数，只要有可能改变 slice 的长度或容量，或让 slice 指向不同的底层数组，都需要更新 slice 变量。
- 虽然 slice 的底层数组是间接引用的
  但是长度和容量不是
- ... 表示可变长度参数列表

4.2.2slice 就地修改

slice 可以用来实现栈
- stack=append(stack,v) //入栈
- top:=stack[len(stack)-1]// 取栈顶
- stack=stack[:len(stack)-1] //出栈

4.3 map

它是一个拥有键值对元素的无序集合
- 键的值是唯一的
  对应的值可以通过键来获取、更新或移除
- map 是散列表的引用
  map 类型是 map[K]V,其中 K 和 V 是字典的键和值对应的数据结构
- map 所有的键都有相同的数据类型
  所有的值也都有相同的数据类型
- map 的键必须是可以通过操作符==来比较的数据类型

创建
- ages:=make([string]int)
  - // 内置函数 make 创建 map
- ages := map[string]int{
  " lihua": 3,
  " zhangsan": 2,
  }
  - // 使用 map 字面量创建带初始化键值对元素的字典
- ages := map[string]int{}
  - // 另一种空 map 创建的方式

删除
- delete(ages, "lihua")
  - // 使用内置的 delete 来从字典根据键移除一个元素

即使键在 map 中，也是安全的
- 如果 map 对应的元素不存在，就返回值类型的零值

map 元素不是一个变量，不可以获取它的地址
- 其中一个原因是 map 的增长可能导致已有元素被重新散列到新的位置
  这样可能使获取的地址无效
- _ = &ages["bob"] // 报错

如果要按照某种方式来遍历 map 的某种顺序，必须显示的给键排序

```
var name []string
for n:=range ages{
    name=append(name,n)
}
for _,n:=range name{
    fmt.Println(map[n])
}
```

map 中的元素迭代顺序是不正确的

map 的零值是 nil

Go 的集合可以使用 map 来实现

```
var m map[string]int

func k(list []string) string {
r       eturn fmt.Sprintf("%q", list)
}

func Add(list []string) {
m       D[k(list)]++
}

func Count(list []string) int {
r       eturn m[k(list)]
}
```

有时候我们需要一个 map 并且它的键是 slice，我们需要一个转换函数
- 同样的方法可以用到任何不可以直接比较的函数

map 的值类型可以是符合数据类型
例如 map 或 slice

4.4 结构体

结构体是将零个或多个任意类型的命名变量组合在一起的聚合数据类型

每个变量都叫做结构体的成员

访问方式
- 结构体的每一个成员都用 "." 点号来表示
- 或者获取成员变量的地址，然后使用指针来访问它
  - position := &dilbert.Position
    *position = "Senior" + *position
  - 等价于
    (*e).Positon = "CEO"
  - var e *E= &dilibert
    e.Position = "CEO"
  - 点号同样可以用在结构体指针上

成员变量的顺序对于结构体很重要
如果我们将两个变量互换，我们就是定义了一个不同的结构体

一个结构体不可以包含它自己，但是可以定义一个 S 的指针类型，即*S

结构体也可以使用 new 来获取一个结构体指针

4.4.1 结构体字面量

没有任何成员变量的结构体称为空结构体
- 没有长度，也不携带任何信息
  - struct{}{}

结构体的值可以通过结构体字面量来社会
- type Point struct{X,Y int}
- p:=Point{1,2} //设置方法 1，需要按照正确的顺序
- p:=Point{X:1} //指定变量的名称和初始值
- 两种方式不可以混用

大型的结构体通常使用结构体指针的方式直接传递给函数或从函数中返回

4.4.2 结构体比较

如果结构体的所有成员变量都可以比较，那么结构体可以比较

可以比较的结构体类型可以作为 map 类型

4.4.3 结构体嵌套和匿名成员

我们可以将一个命名结构体当做另一个结构体类型的匿名成员使用
- type Point struct {
  X   , Y int
  }
- type Circle struct{
  P   oi nt
  R   adi us int
  }
- var c Circle
  c.X //等价于 c.Point.X

Go 允许我们定义不带名称的结构体成员，只要指定其类型即可
这种结构体成员称为匿名成员

结构体嵌套没有办法便捷初始化
- c:=&Circle{Point{1,2,},3}

删词 # 使得 Printf 格式化符合%v 按照 Go 语法输出
- c := &Circle{Point: Point{X: 1, Y: 2}, Radius: 3}
  fmt.Printf("%#v", c)
- 输出 Circle{Point:main.Point{X:1, Y:2}, Radius:3}

4.5 JSON

JavaScript 对象表示法（JSON）是一种发送和接收格式化信息的标准

成员的标签 使用 ``

把 Go 的数据结构转换为 JSON 称为 marshal

MarshalIndent 可以输出整齐格式化过的结果

只有可导出的成员才可以转换为 JSON 字段，这就是为什么我们将 Go 结构体里的所有成员定义为首字母大写

marshal 逆操作将 JSON 字符串解码为 Go 数据结构，这个过程叫做 unmarshal