Go 语言中的指针

Go语言中的指针不能进行偏移和运算，是安全指针。

要搞明白Go语言中的指针需要先知道3个概念：指针地址、指针类型和指针取值。

Go语言中的函数传参都是值拷贝，当我们想要修改某个变量的时候，我们可以创建一个指向该变量地址的指针变量。传递数据使用指针，而无须拷贝数据。类型指针不能进行偏移和运算。

Go语言中的指针操作非常简单，只需要记住两个符号：`&`（取地址）和`*`（根据地址取值）。

指针地址和指针类型

每个变量在运行时都拥有一个地址，这个地址代表变量在内存中的位置。Go语言中使用`&`字符放在变量前面对变量进行“取地址”操作。 Go语言中的值类型`（int、float、bool、string、array、struct）`都有对应的指针类型，如：`*int、*int64、*string`等。

`&`指针地址，`*`指针类型

指针语法

一个指针变量指向了一个值的内存地址，也就是说我们声明了一个指针之后，可以像变量赋值一样，把一个值的内存地址放入到指针当中。

类似于变量常量，在使用指针前你需要声明指针。

指针声明格式：

```
var p *type
```

`p`：为指针变量

`type`：为指针类型

`*`：用于指定一个变量作为指针

程序在内存中存储它的值，每个内存块（或字）有一个地址，通常用十六进制数表示，如：`0x6b0820`或`0xf84001d7f0`。

示例：

```
var intP *int
var floatP *float32
```

指针取值

在对普通变量使用`&`操作符取地址后会获得这个变量的指针，然后可以对指针使用`*`操作，也就是指针取值.

```
package main

import "fmt"
```

```
func main() {
    //指针取值
    a := 10
    b := &a // 取变量a的地址，将指针保存到b中
    fmt.Printf("type of b:%T\n", b)
    c := *b // 指针取值（根据指针去内存取值）
    fmt.Printf("type of c:%T\n", c)
    fmt.Printf("value of c:%v\n", c)
}
#结果
type of b:*int
type of c:int
value of c:10
```

总结： 取地址操作符`&`和取值操作符`*`是一对互补操作符，`&`取出地址，`*`根据地址取出地址指向的值。

变量、指针地址、指针变量、取地址、取值的相互关系和特性如下：

1.对变量进行取地址（&）操作，可以获得这个变量的指针变量。
2.指针变量的值是指针地址。
3.对指针变量进行取值（*）操作，可以获得指针变量指向的原变量的值。

指针传值示例：

```
func modify1(x int) {
    x = 100
}

func modify2(x *int) {
    *x = 100
}

func main() {
    a := 10
    modify1(a)
    fmt.Println(a) // 10
    modify2(&a)
    fmt.Println(a) // 100
}
```

空指针

当一个指针被定义后没有分配到任何变量时，它的值为 nil。

空指针的判断：

```
package main

import "fmt"

func main() {
    var p *string
    fmt.Println(p)
    fmt.Printf("p的值是%v\n", p)
```

```
    if p != nil {
        fmt.Println("非空")
    } else {
        fmt.Println("空值")
    }
}
```

new和make

new

new是一个内置的函数，它的函数签名如下：

```
    func new(Type) *Type
```

- Type表示类型，new函数只接受一个参数，这个参数是一个类型
- *Type表示类型指针，new函数返回一个指向该类型内存地址的指针。

new函数不太常用，使用new函数得到的是一个类型的指针，并且该指针对应的值为该类型的零值。举个例子：

```
func main() {
    a := new(int)
    b := new(bool)
    fmt.Printf("%T\n", a) // *int
    fmt.Printf("%T\n", b) // *bool
    fmt.Println(*a)       // 0
    fmt.Println(*b)       // false
}
```

本节开始的示例代码中`var a *int`只是声明了一个指针变量a但是没有初始化，指针作为引用类型需要初始化后才会拥有内存空间，才可以给它赋值。应该按照如下方式使用内置的new函数对a进行初始化之后就可以正常对其赋值了：

```
func main() {
    var a *int
    a = new(int)
    *a = 10
    fmt.Println(*a)
}
```

make

make也是用于内存分配的，区别于new，它只用于slice、map以及chan的内存创建，而且它返回的类型就是这三个类型本身，而不是他们的指针类型，因为这三种类型就是引用类型，所以就没有必要返回他们的指针了。make函数的函数签名如下：

```
func make(t Type, size ...IntegerType) Type
```

make函数是无可替代的，我们在使用slice、map以及channel的时候，都需要使用make进行初始化，然后才可以对它们进行操作。

new与make的区别

- 二者都是用来做内存分配的。
- make只用于slice、map以及channel的初始化，返回的还是这三个引用类型本身；
- 而new用于类型的内存分配，并且内存对应的值为类型零值，返回的是指向类型的指针。

指向数组的指针

```
var ptr [n]*int; 表示数组里面的元素的类型是指针类型
```

示例：

```
package main

import "fmt"

func main() {
    a := [3]int{1, 2, 3}
    var pa [3]*int
    fmt.Printf("pa: %v\n", pa)

    for i := 0; i < len(a); i++ {
        pa[i] = &a[i]
    }
    fmt.Printf("pa: %v\n", pa)

    for i := 0; i < len(pa); i++ {
        fmt.Printf("pa: %v\n", *pa[i])
    }
}
#结果
pa: [<nil> <nil> <nil>]
pa: [0xc00001a138 0xc00001a140 0xc00001a148]
pa: 1
pa: 2
pa: 3
```