Golang 中的指针

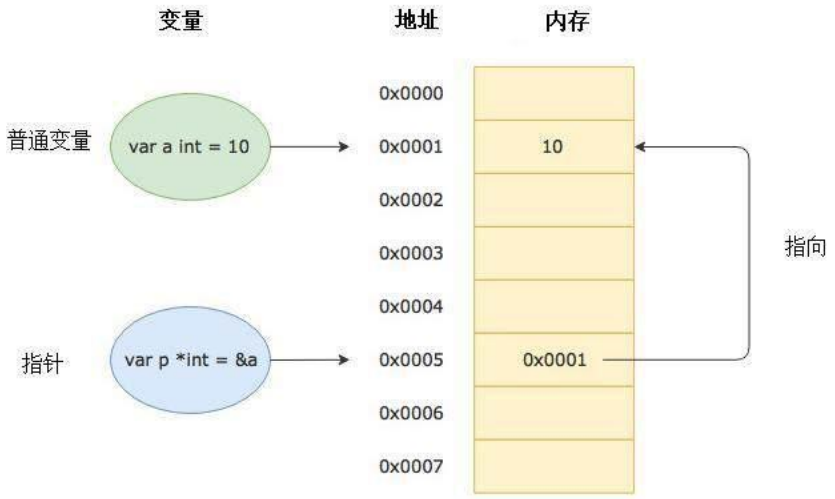
主讲教师：（大地）
合作网站：www.itying.com （IT 营）
我的专栏：https://www.itying.com/category-79-b0.html

一、关于指针.....1
二、指针地址和指针类型.....2
三、指针取值.....3
四、指针传值示例.....3
五、new 和 make.....4

一、关于指针

通过前面的教程我们知道变量是用来存储数据的，变量的本质是给存储数据的内存地址起了一个好记的别名。比如我们定义了一个变量 a := 10 ,这个时候可以直接通过 a 这个变量来读取内存中保存的 10 这个值。在计算机底层 a 这个变量其实对应了一个内存地址。

指针也是一个变量，但它是一种特殊的变量，它存储的数据不是一个普通的值，而是另一个变量的内存地址。



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

二、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。

取变量指针的语法如下：

```
ptr := &v    // 比如v的类型为T
```

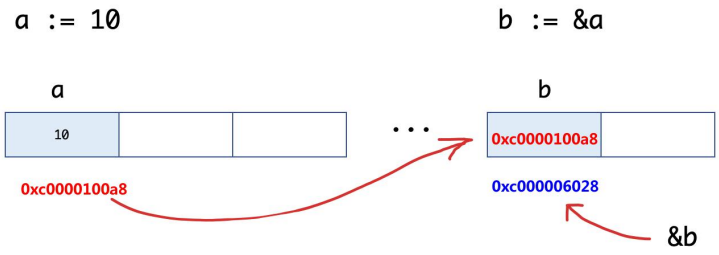
则其中：

- **v**：代表被取地址的变量，类型为 T
- **ptr**：用于接收地址的变量，**ptr** 的类型就为*T，称做 T 的指针类型。*代表指针。

举个例子：

```
package main
import "fmt"
func main() {
    var a = 10
    var b = &a
    fmt.Printf("a:%d ptr:%p\n", a, &a) // a:10 ptr:0xc0000100a8
    fmt.Printf("b:%v type:%T\n", b, b) // b:0xc0000100a8 type:*int
    fmt.Println("取 b 的地址：", &b)          // 0xc000006028
}
```

我们来看一下 b := &a 的图示：



三、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，代码如下。

```
func main() {
    a := 10
    b := &a // 取变量 a 的地址，将地址保存到指针 b 中
    fmt.Printf("type of b:%T\n", b)
    c := *b // 指针取值（根据指针的值去内存取值）
    fmt.Printf("type of c:%T\n", c)
    fmt.Printf("value of c:%v\n", c)
}
```

输出如下：

```
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：

- 对变量进行取地址（&）操作，可以获得这个变量的指针变量。
- 指针变量的值是指针地址。
- 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。

四、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
}
```

五、new 和 make

我们先来看一个例子：

```
func main() {
    var userinfo map[string]string
    userinfo["username"] = "张三"
    fmt.Println(userinfo)
}
```

```
func main() {
    var a *int
    *a = 100
    fmt.Println(*a)
}
```

执行上面的代码会引发 panic，为什么呢？ 在 Go 语言中对于引用类型的变量，我们在使用的时候不仅要声明它，还要为它分配内存空间，否则我们的值就没办法存储。而对于值类型的声明不需要分配内存空间，是因为它们在声明的时候已经默认分配好了内存空间。要分配内存，就引出来今天的 new 和 make。 Go 语言中 new 和 make 是内建的两个函数，主要用来分配内存。

1、new 函数分配内存

new 是一个内置的函数，它的函数签名如下：

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

其中：

- 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。

实际想开发中 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

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

new 函数对 a 进行初始化之后就可以正常对其赋值了：

2、make 函数分配内存

make 也是用于内存分配的，区别于 new，它只用于 slice、map 以及 channel 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。这个我们在前面的教程中都有说明，关于 channel 我们会在后续的章节详细说明。

本节开始的示例中 var b map[string]int 只是声明变量 b 是一个 map 类型的变量，需要像下面的示例代码一样使用 make 函数进行初始化操作之后，才能对其进行键值对赋值：

```
func main() {
  var userinfo map[string]string
  userinfo = make(map[string]string)
  userinfo["username"] = "张三"
  fmt.Println(userinfo)
}
```

3、new 与 make 的区别

1. 二者都是用来做内存分配的。
2. 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
3. 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。