

Go 语言指针

Go 语言中指针是很容易学习的，Go 语言中使用指针可以更简单的执行一些任务。

接下来让我们来一步步学习 Go 语言指针。

我们都知道，变量是一种使用方便的占位符，用于引用计算机内存地址。

Go 语言的取地址符是 &，放到一个变量前使用就会返回相应变量的内存地址。

以下实例演示了变量在内存中地址：

实例

```
package main

import "fmt"

func main() {
    var a int = 10

    fmt.Printf("变量的地址: %x\n", &a )
}
```

执行以上代码输出结果为：

```
变量的地址: 20818a220
```

现在我们已经了解了什么是内存地址和如何去访问它。接下来我们将具体介绍指针。

什么是指针

一个指针变量指向了一个值的内存地址。

类似于变量和常量，在使用指针前你需要声明指针。指针声明格式如下：

```
var var_name *var-type
```

var-type 为指针类型，var_name 为指针变量名，* 号用于指定变量是作为一个指针。以下是有效的指针声明：

```
var ip *int      /* 指向整型*/
var fp *float32  /* 指向浮点型 */
```

本例中这是一个指向 int 和 float32 的指针。

如何使用指针

指针使用流程：

- 定义指针变量。
- 为指针变量赋值。
- 访问指针变量中指向地址的值。

在指针类型前面加上 * 号（前缀）来获取指针所指向的内容。

实例

```
package main

import "fmt"

func main() {
    var a int = 20    /* 声明实际变量 */
    var ip *int       /* 声明指针变量 */

    ip = &a /* 指针变量的存储地址 */

    fmt.Printf("a 变量的地址是: %x\n", &a )

    /* 指针变量的存储地址 */
    fmt.Printf("ip 变量储存的指针地址: %x\n", ip )

    /* 使用指针访问值 */
    fmt.Printf("*ip 变量的值: %d\n", *ip )
}
```

以上实例执行输出结果为：

```
a 变量的地址是: 20818a220
ip 变量储存的指针地址: 20818a220
*ip 变量的值: 20
```

Go 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。

nil 指针也称为空指针。

nil 在概念上和其它语言的 null、None、nil、NULL 一样，都指代零值或空值。

一个指针变量通常缩写为 ptr。

查看以下实例：

实例

```
package main

import "fmt"

func main() {
    var ptr *int
```

```
fmt.Printf("ptr 的值为 : %x\n", ptr )
}
```

以上实例输出结果为：

```
ptr 的值为 : 0
```

空指针判断：

```
if(ptr != nil)    /* ptr 不是空指针 */
if(ptr == nil)    /* ptr 是空指针 */
```


Go指针更多内容


接下来我们将为大家介绍Go语言中更多的指针应用：

内容	描述
Go 指针数组	你可以定义一个指针数组来存储地址
Go 指向指针的指针	Go 支持指向指针的指针
Go 向函数传递指针参数	通过引用或地址传参，在函数调用时可以改变其值

← Go 语言向函数传递数组

Go 语言指针数组 →

 1 篇笔记

 写笔记



测试实例：

```
package main

import "fmt"

func main() {
    var a int = 10
    var ip *int
    fmt.Printf("变量的地址: %x\n", &a)
    fmt.Println("变量的地址: ", &a)
    ip = &a
    fmt.Println("ip 变量存储的指针地址:", ip)
    fmt.Println("ip 变量存储的指针地址的值:", *ip)
    fmt.Println("ip 变量存储的指针地址的地址:", &ip)
    var ptr *int
```

```
if (ptr != nil) {
    if (ip != nil) {
        fmt.Println("ptr不是空指针")
        fmt.Println("ip不是空指针")
    }else {
        fmt.Println("ptr不是空指针")
        fmt.Println("ip是空指针")
    }
} else {
    if(ip != nil){
        fmt.Println("ptr是空指针")
        fmt.Println("ip不是空指针")
    }else{
        fmt.Println("ptr是空指针")
        fmt.Println("ip是空指针")
    }
}
/* 自学的时候想到能不能使用 switch 优化 for 繁琐的写法, 但是发现 case 匹配到后会自动跳出 switch。
   查了一下 select 等方法发现并不适用, 最后发现了 fallthrough 可以很好的用在这里 (不过要注意 fallthrough 存在的位置, 避免产生逻辑混乱) */
switch {
    case ptr != nil:
        fmt.Println("ptr不是空指针")
        fallthrough
    case ptr == nil:
        fmt.Println("ptr是空指针")
        fallthrough
    case ip != nil:
        fmt.Println("ip不是空指针")
    default:
        fmt.Println("ip是空指针")
}
}
```

以上代码执行结果为：

```
变量的地址: c420080008
变量的地址: 0xc420080008
ip 变量存储的指针地址: 0xc420080008
ip 变量存储的指针地址的值: 10
ip 变量存储的指针地址的地址: 0xc42008a018
ptr是空指针
ip不是空指针
ptr是空指针
ip不是空指针
```

千树 2个月前 [01-29]

