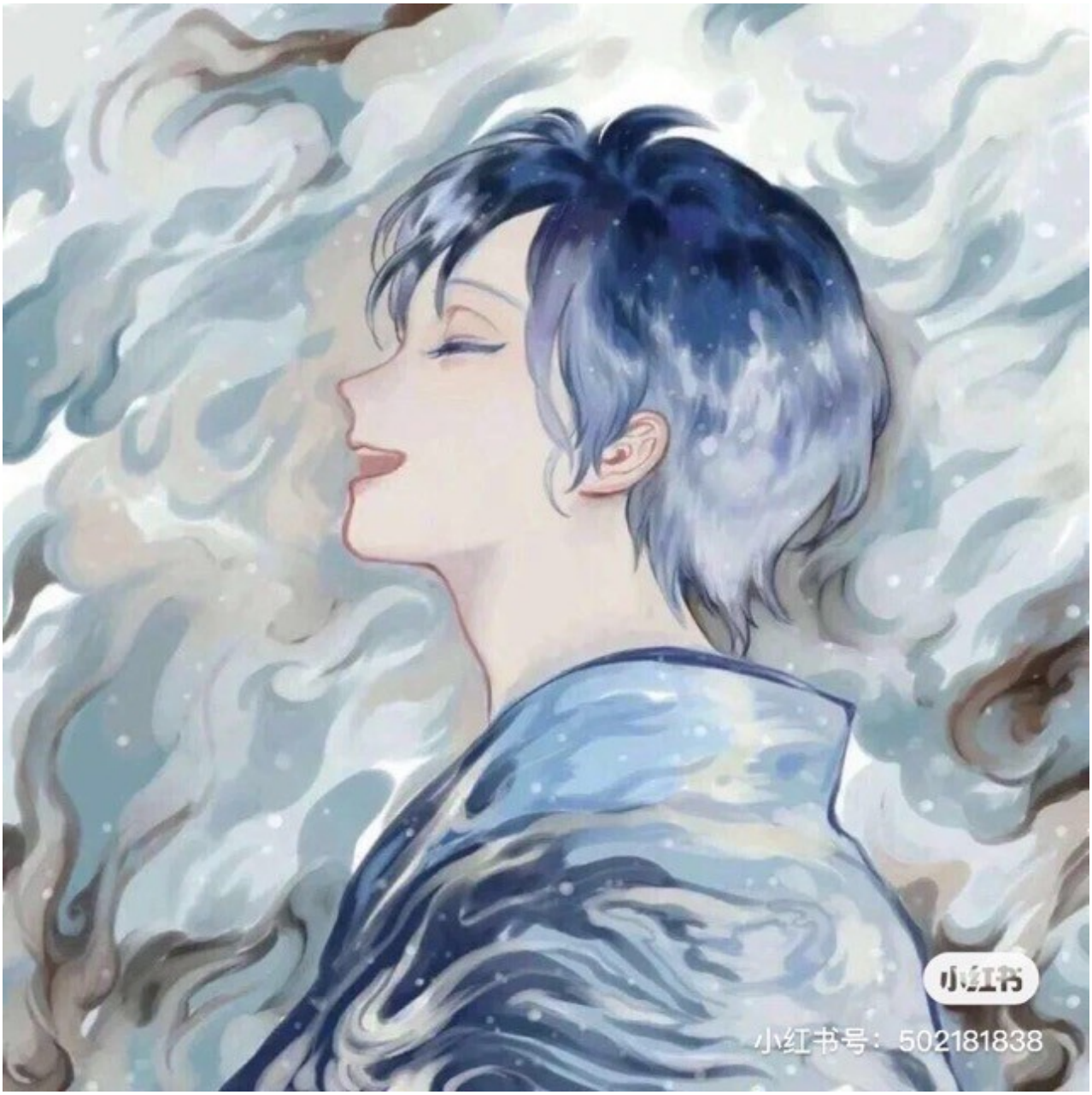
面试官：两个 nil 比较结果是什么？

脑子进煎鱼了 *2021-09-17 08:51*
编者荐语：

asong，非科班程序员，现就职于某电商外企，专研Go语言、微服务系统设计，欢迎关注asong 的公众号，一起学习进步！

以下文章来源于Golang梦工厂 ，作者AsongGo



Golang梦工厂.
asong是一名后端程序员，目前就职于一家电商公司，专注于Golang技术，定期分享Go语言、MySQL、Redis、Elasticsearch、计算机基础、微服务架构设计、面试等知识。这里不仅有技术，还有故事！

背景

> 哈喽，大家好，我是 `asong` 。前几天在一个交流群里看到了一道非常有意思的面试题，今天把它分享出来，我们先来看一下这个道题：

```
fmt.Println(nil== nil)
```

两个 `nil` 的比较结果是什么？

`true` 、 `false` 、还是无法编译？大家先思考着，文中揭晓答案。

> 写在开始：**建议你们看一下这个视频：https://www.youtube.com/watch?v=ynoY2xz-F8s 需要翻墙哈，看完这个你对 `nil` 会有一个新的理解。**

Go中 `nil` 的定义

在 `Go` 官方文档中，对 `nil` 的定义如下：

```
// nil is a predeclared identifier representing the zero value for a
// pointer, channel, func, interface, map, or slice type.
var nil Type // Type must be a pointer, channel, func, interface, map, or slic
```

`nil` 是一个预先声明的标识符，代表指针(`pointer`)、通道(`channel`)、函数(`func`)、接口(`interface`)、 `map` 、切片(`slice`)。也可以这么理解：指针、通道、函数、接口、map、切片的零值就是 `nil` ，就像布尔类型的零值是 `false` 、整型的零值是 `0` 。

深入理解 `nil`

`nil` 根本不是关键字

我们先来看一段代码：

```
func main()  {
 nil := "this is nil"
 fmt.Println(nil)
}
// 运行结果
this is nil
```

那再改成这样呢？

```
func main()  {
 nil := "this is nil"
 fmt.Println(nil)
  var slice []string = nil
 fmt.Println(slice)
}
// 运行结果
# command-line-arguments
./nil.go:10:6: cannot use nil (type string) as type []string in assignment
```

编译的时候直接报错了，因为这个 `nil` 是一个 `string` 类型，所以从这里确定 `nil` 在 `Go` 语言中并不是关键字，我们可以随意定义变量名为 `nil` （不过不建议这么用）。

`nil` 的默认类型

一般预声明标识符都会有一个默认类型，比如 `Go` 语言中的 `itoa` 默认类型就是 `int` ，那么 `nil` 的默认类型呢？我们写个例子来看一下：

```
func main()  {
 const val1 = iota
 fmt.Printf("%T\n",val1)
 var val2 = nil
 fmt.Printf("%T\n",val2)
}
// 运行结果
# command-line-arguments
./nil.go:10:6: use of untyped nil
```

在编译时就已经报错，编译器告诉我们使用了无类型的 `nil` ，所以我们可以得出结论：

> `nil` 是没有默认类型的，它的类型具有不确定性，我们在使用它时必须要提供足够的信息能够让编译器推断 `nil` 期望的类型。

`nil` 的比较

`nil` 的比较我们可以分为以下两种情况：

- `nil` 标识符的比较
- `nil` 的值比较

我们先来看一下 `nil` 标识符的比较，也就是我们开头那一道面试题，先看一下运行结果呢：

```
# command-line-arguments
./nil.go:8:18: invalid operation: nil == nil (operator == not defined on nil)
```

通过编译结果我们可以看出 `==` 符号对于 `nil` 来说是一种未定义的操作，所以是不可以比较两个 `nil` 的。

接着我们来看一看 `nil` 的值比较，因为 `nil` 是没有类型的，是在编译期根据上下文确定的，所以要比较 `nil` 的值也就是比较不同类型的 `nil` ，这又分为同一个类型的 `nil` 值比较和不同类型 `nil` 值的比较，分这两种情况我们分别来验证一下。

同一个类型的 `nil` 值比较

```
func main()  {
 // 指针类型的nil比较
 fmt.Println((*int64)(nil) == (*int64)(nil))
 // channel 类型的nil比较
 fmt.Println((chan int)(nil) == (chan int)(nil))
 // func类型的nil比较
 fmt.Println((func())(nil) == (func())(nil)) // func() 只能与nil进行比较
 // interface类型的nil比较
 fmt.Println((interface{})(nil) == (interface{})(nil))
 // map类型的nil比较
 fmt.Println((map[string]int)(nil) == (map[string]int)(nil)) // map 只能与nil进行比
较
 // slice类型的nil比较
 fmt.Println(([]int)(nil) == ([]int)(nil)) // slice 只能与nil进行比较
}
```

运行结果：

```
# command-line-arguments
./nil.go:13:28: invalid operation: (func())(nil) == (func())
(nil) (func can only be compared to nil)
./nil.go:17:36: invalid operation: (map[string]int)(nil) == (map[string]int)
(nil) (map can only be compared to nil)
./nil.go:19:27: invalid operation: ([]int)(nil) == ([]int)
(nil) (slice can only be compared to nil)
```

从运行结果我们可以看出，指针类型 `nil` 、 `channel` 类型的 `nil` 、 `interface` 类型可以相互比较，而 `func` 类型、 `map` 类型、 `slice` 类型只能与 `nil` 标识符比较，两个类型相互比较是不合法的。

不同类型的 `nil` 值比较

```
func main()  {
var ptr *int64 = nil
var cha chanint64 = nil
var fun func() = nil
var inter interface{} = nil
var ma map[string]string = nil
var slice []int64 = nil
 fmt.Println(ptr == cha)
 fmt.Println(ptr == fun)
 fmt.Println(ptr == inter)
 fmt.Println(ptr == ma)
 fmt.Println(ptr == slice)



 fmt.Println(cha == fun)
 fmt.Println(cha == inter)
 fmt.Println(cha == ma)
 fmt.Println(cha == slice)

 fmt.Println(fun == inter)
 fmt.Println(fun == ma)
 fmt.Println(fun == slice)

 fmt.Println(inter == ma)
 fmt.Println(inter == slice)

 fmt.Println(ma == slice)
}
```

运行结果：

```
# command-line-arguments
./nil.go:14:18: invalid operation: ptr == cha (mismatched types *int64 and chan int
./nil.go:15:18: invalid operation: ptr == fun (mismatched types *int64 and func())
./nil.go:17:18: invalid operation: ptr == ma (mismatched types *int64 and map[strin
./nil.go:18:18: invalid operation: ptr == slice (mismatched types *int64 and []int6
./nil.go:20:18: invalid operation: cha == fun (mismatched types chan int64 and func
./nil.go:22:18: invalid operation: cha == ma (mismatched types chan int64 and map[s
./nil.go:23:18: invalid operation: cha == slice (mismatched types chan int64 and []
./nil.go:25:18: invalid operation: fun == inter (operator == not defined on func)
./nil.go:26:18: invalid operation: fun == ma (mismatched types func() and map[strin
./nil.go:27:18: invalid operation: fun == slice (mismatched types func() and []int6
./nil.go:27:18: too many errors
```

从运行结果我们可以得出，只有指针类型和channel类型与接口类型可以比较，其他类型的之间是不可以相互比较的。为什么指针类型、channel类型可以和接口类型进行比较呢？

> 这个答案，先空着，因为我也没有想明白，不是说/任何类型都实现了interface{}类型吗？这里没想明白，期待你们的解答。

`nil` 在不同类型中使用需要注意的问题

`interface` 与 `nil` 比较要注意的一个点

我们先来看一个例子：

```
func main()  {
 err := Todo()
 fmt.Println(err == nil)
}

type Err interface {

}

type err struct {
 Code int64
 Msg string
}

func Todo() Err  {
 var res *err
 return res
}
// 运行结果
false
```

输出结果是 `false` ，在 `Todo` 方法内我们声明了一个变量 `res` ，这个变量是一个指针类型，零值是 `nil` ，返回的是接口类型，按理说返回值接口类型也应是 `nil` 才对，但是结果却不是这样。这是因为我们忽略了接口类型的一个概念，interface 不是单纯的值，而是分为类型和值。所以必须要类型和值同时都为 `nil` 的情况下， `interface` 的 `nil` 判断才会为 `true` 。

这是一个新手很容易出现的问题，大家一定要注意这个问题。

一个 `nil` 的 `map` 读写数据是否会发生 `panic`

对于这种问题，我们直接写个例子测试一下就好：

```
func main()  {
var m map[string]string
 fmt.Println(m["asoong"])
 m["asong"] = "Golang梦工厂"
}
// 运行结果
panic: assignment to entry in nilmap

  goroutine 1 [running]:
main.main()
        go/src/asong.cloud/Golang_Dream/code_demo/slice_demo/nil.go:10 +0xed
```

根据运行结果我们可以看出，一个 `nil` 的 `map` 可以读数据，但是不可以写入数据，否则会发生 `panic` ，所以要使用 `map` 一定要使用 `make` 进行初始化。

关闭 `nil` 的 `channel` 会引发 `panic`

```
func main()  {
 var cha chan int
 close(cha)
}
```

运行结果：

```
panic: close of nil channel

goroutine 1 [running]:
main.main()
       /go/src/asong.cloud/Golang_Dream/code_demo/slice_demo/nil.go:5 +0x2a
```

根据运行结果我们可以得出关闭一个 `nil` 的 `channel` 会导致程序 `panic` ，在使用上我们要注意这个问题，还有有一个需要注意的问题：**一个 `nil` 的 `channel` 读写数据都会造成永远阻塞。**

一个为 `nil` 的 `slice` 使用注意事项

```
func main()  {
var slice []int64 = nil
 fmt.Println(len(slice))
 fmt.Println(cap(slice))
forrange slice{



 }
 fmt.Println(slice[0])
}
// 运行结果
0
0
panic: runtime error: index out of range [0] with length 0

goroutine 1 [running]:
main.main()
        /go/src/asong.cloud/Golang_Dream/code_demo/slice_demo/nil.go:14 +0xf2
```

根据这个例子，我们可以得出如下结论：

> 一个为 `nil` 的索引，不可以进行索引，否则会引发 `panic` ，其他操作是可以。

方法接收者为 `nil` 时是否会引发 `panic`

```
func main()  {
var m *man
 fmt.Println(m.GetName())
}



type man struct {

}

func (m *man)GetName() string {
 return "asong"
}
// 运行结果
asong
```

根据运行结果我们可以看出，方法接收者为`nil`时，我们仍然可以访问对应的方法，但是要注意方法内的写法，否则也会引发`panic`。上面的代码改成这样：

```
func main()  {
var m *man
 fmt.Println(m.GetName())
}



type man struct {
 Name string
}

func (m *man)GetName() string {
return m.Name
}
// 运行结果
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0x10a6ec3]

 goroutine 1 [running]:
main.(*man).GetName(...)
        go/src/asong.cloud/Golang_Dream/code_demo/slice_demo/nil.go:18
main.main()
        go/src/asong.cloud/Golang_Dream/code_demo/slice_demo/nil.go:9 +0x23
```

这样就是直接引发`panic`，所以为了程序健壮性我们要做一次指针判空处理。

空指针是一个没有任何值的指针

```
func main()  {
 var a = (*int64)(unsafe.Pointer(uintptr(0x0)))
 fmt.Println(a == nil)  //true
}
// 运行结果
true
```

这里我们用了 `0x0` 做了一个小实验，正好证明了空指针就是一个没有指向任何值的指针。

总结

文章接近尾声啦，我们来揭晓一下文章开始的答案，用文中 `nil` 比较的知识点正好可以解答这个问题， `nil` 标识符是没有类型的，所以 `==` 对于 `nil` 来说是一种未定义的操作，不可以进行比较，而这个在 `python` 中是可以比较的，在 `python` 中，两个 `None` 值永远相等，不要弄混了朋友们。

Golang梦工厂

asong是一名后端程序员，目前就职于一家电商公司，专注于Golang技术，定期分享Go语言、MySQL、Redis、Elasticsearch、计算机基础、微服务架构设计、面试等知识。这里不仅有技术，还有故事！

74篇原创内容

公众号