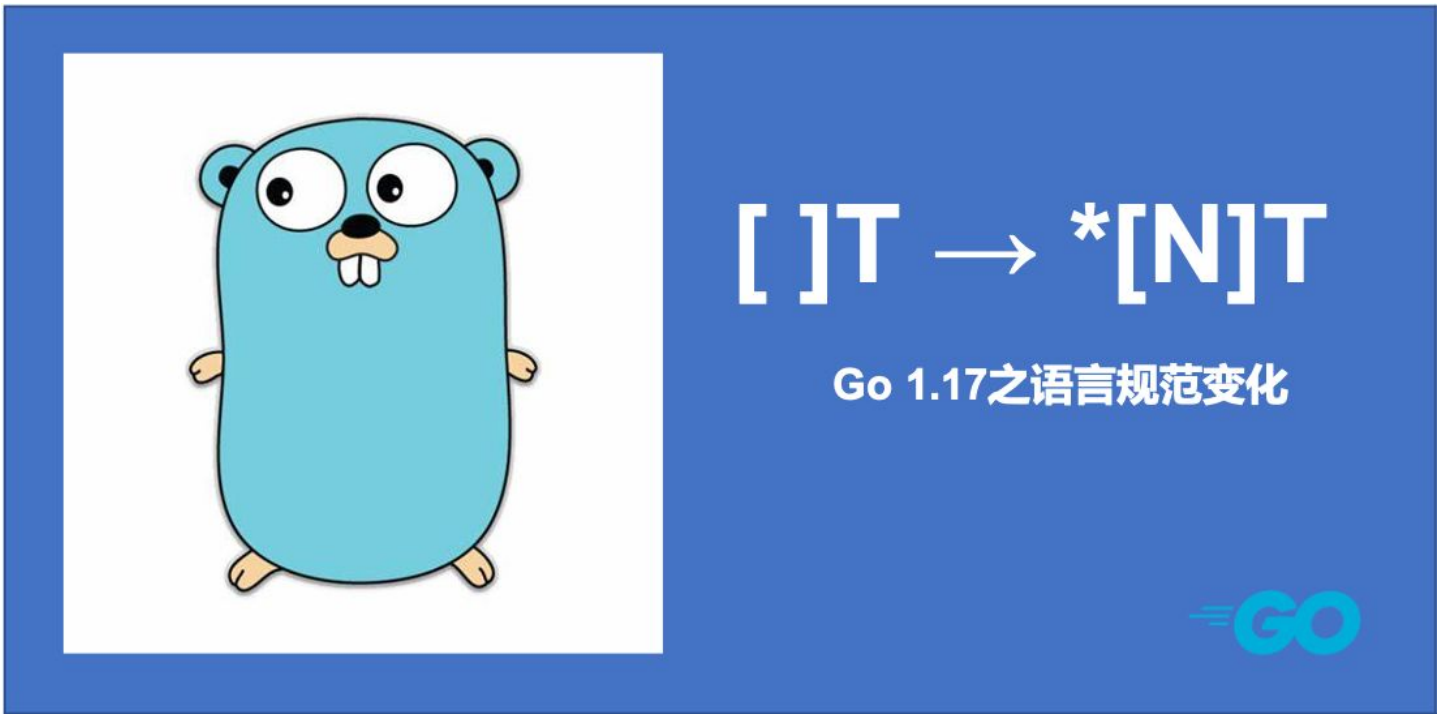

Go 1.17新特性详解：支持将切片转换为数组指针

bigwhite

Go属于那种极简的语言，从诞生到现在语言自身特性变化很小，不会像其他主流语言那样走“你有的我也要有”的特性融合路线。因此新语言特性对于Gopher来说属于“稀缺品”，属于“供不应求”那类事物^_^。这也直接导致了每次Go新版本发布，我们都要首先看看语言特性是否有变更，每个新加入语言的特性都值得我们去投入更多关注，去深入研究。下面我们就来深入**Go 1.17版本**中语言规范的一些变化！

1. 支持将切片转换为数组指针

在Go 1.17版本之前，我们可以将数组转换为切片，数组将成为转换后的切片底层存储数组，因此，通过切片可以直接改变数组中的元素，就像下面代码这样：

```
// github.com/bigwhite/experiments/tree/master/go1.17-examples/lang/slice2arrayptr/main.go
func array2slice() {
 var a = [5]int{11, 12, 13, 14, 15}
 var b = a[0:len(a)] // or var b = a[:]
 b[1] += 10
 fmt.Printf("%v\n", b) // [11 22 13 14 15]
}
```

但反过来则不行，Go不支持将切片再转换回数组类型，编译器会报下面错误信息：

```
// github.com/bigwhite/experiments/tree/master/go1.17-examples/lang/slice2arrayptr/main.go
func slice2array() {
 var b = []int{11, 12, 13}
 var a = [3]int(b) // cannot convert b (type []int) to type [3]int
 fmt.Printf("%v\n", a)
}
```

那么在Go中我们就没法将切片转换为数组了么？也不是绝对的。我们可以通过unsafe包以hack的方式实现这样的转换，如下面代码所示：

```
// github.com/bigwhite/experiments/tree/master/go1.17-examples/lang/slice2arrayptr/main.go
func slice2arrayWithHack() {
 var b = []int{11, 12, 13}
 var a = *(*[3]int)(unsafe.Pointer(&b[0]))
 a[1] += 10
```

```
 fmt.Printf("%v\n", b) // [11 12 13]
}
```

上面代码中，我们实际上得到是切片底层数组的一份拷贝，修改该拷贝中的元素值，切片中的元素将不会受到影响。如果想通过数组修改切片中元素，我们还得通过获取数组指针的方式，如下面代码所示。

```
// github.com/bigwhite/experiments/tree/master/go1.17-examples/lang/slice2arrayptr/main.go
func slice2arrayptrWithHack() {
 var b = []int{11, 12, 13}
 var p = (*[3]int)(unsafe.Pointer(&b[0]))
 p[1] += 10
 fmt.Printf("%v\n", b) // [11 22 13]
}
```

但是使用unsafe，一如其名，其安全性没有编译器和runtime层的保证，只能由开发者自己保证，Gopher在通常情况下应该避免使用。

于是在2009年末，也就是**Go语言宣布开源**后不久（那时Go 1.0版本尚未发布），**Roger Peppe**便提出一个issue（那时go的开发还没有如今这么规范，没有proposal流程）：**“spec: use (*[4]int)(x) to convert slice x into array pointer”**。最初该issue的提出仅仅是因为语法层面缺失了从切片到数组的转换语法，同时希望这种转换以及转换后的数组使用时的下标边界能得到编译器和runtime的协助检查。这个issue得到了当时Go核心开发组成员的支持，Russ Cox还提出将Roger Peppe提议的语法形式做如下变动：

```
从
b := a.[0:4]

变为

b := (*[4]int)(a[0:4])
```

但不知何故，该issue始终没有被纳入Go主干中，直到Go 1.17版本，该issue又被重新提出来了。Go 1.17直接**支持将切片转换为数组指针**，我们可以在Go 1.17中编写和运行如下面这样的代码，而无需再借助unsafe的hack：

```
// github.com/bigwhite/experiments/tree/master/go1.17-examples/lang/slice2arrayptr/main.go
func slice2arrayptr() {
 var b = []int{11, 12, 13}
 var p = (*[3]int)(b)
 p[1] = p[1] + 10
 fmt.Printf("%v\n", b) // [11 22 13]
}
```

Go通过运行时对这类切片到数组指针的转换代码做检查，如果发现越界行为，就会通过运行时panic予以处理。Go运行时实施检查的一条原则就是“转换后的数组长度不能大于原切片的长度”，注意这里是切片的长度(len)，而不是切片的容量(cap)，于是下面的转换有些合法，有些非法：

```
// github.com/bigwhite/experiments/tree/master/go1.17-examples/lang/slice2arrayptr/main.go

var b = []int{11, 12, 13}
var p = (*[4]int)(b) // cannot convert slice with length 3 to pointer to array with length 4
var p = (*[0]int)(b) // ok，*p = []
var p = (*[1]int)(b) // ok，*p = [11]
var p = (*[2]int)(b) // ok，*p = [11, 12]
var p = (*[3]int)(b) // ok，*p = [11, 12, 13]
var p = (*[3]int)(b[:1]) // cannot convert slice with length 1 to pointer to array with length 3
```

关于这个语言特性的应用场合，目前还待Go社区挖掘，不过已经有人提出提出**利用该特性优化go编译器的可行性评估**了。

2. unsafe包新增了两个“语法糖”函数

Go 1.17中增加了两个“语法糖”函数：**Add**和**Slice**。这两个函数原型如下：

```
// $GOROOT/src/unsafe.go
func Add(ptr Pointer, len IntegerType) Pointe
func Slice(ptr *ArbitraryType, len IntegerType) []ArbitraryType
```

之所以这两个函数能进入unsafe包，和其他已经存在于unsafe包中的函数的目的是一样的，那就是将Go开发人员一些经常使用的“代码片段模式”升级为unsafe包内置的函数，这样不仅可以降低开发人员误用的比例，还可以让Go runtime提供一些检查，增加类型安全性。

unsafe.Add函数

由于go原生不允许指针加减操作，因此我们在特定场景下不得不使用unsafe包来做指针加减，比如下面代码：

```
// github.com/bigwhite/experiments/tree/master/go1.17-examples/lang/unsafe/add/main.go
const intLen = unsafe.Sizeof(int(8))

func foo() {
    var a = [5]int{11, 12, 13, 14, 15}
    for i := 0; i < 5; i++ {
        p := (*int)(unsafe.Pointer(uintptr(unsafe.Pointer(&a[0])) + uintptr(uintptr(i)*intLen)))
        *p = *p + 10
    }
    fmt.Println(a)// [21 22 23 24 25]
}
```

上面代码中间变量p声明同时赋值那行是在Go 1.17之前unsafe包最常见的一种用法和代码模式。大家都这么用，但用起来还那么繁琐，于是便有了unsafe.Add。如果用unsafe.Add改造上面代码，便能简略一些，如下面代码所示：

```
// github.com/bigwhite/experiments/tree/master/go1.17-examples/lang/unsafe/add/main.go
const intLen = unsafe.Sizeof(int(8))

func bar() {
    var a = [5]int{11, 12, 13, 14, 15}
    for i := 0; i < 5; i++ {
        p := (*int)(unsafe.Add(unsafe.Pointer(&a[0]), uintptr(i)*intLen))
        *p = *p + 10
    }
    fmt.Println(a)
}
```

本质上unsafe.Add(ptr, len) 就等价于unsafe.Pointer(uintptr(ptr) + uintptr(len))。在之前版本中，runtime的stubs.go中也有个类似的实现：

```
$GOROOT/src/runtime/stubs.go

  // Should be a built-in for unsafe.Pointer?
  //go:nosplit
  func add(p unsafe.Pointer, x uintptr) unsafe.Pointer {
      return unsafe.Pointer(uintptr(p) + x)
  }
```

Go 1.17有了这个Add函数后，建议大家就多多使用该函数，而尽量不要自己去拼那个“大长串”了。

unsafe.Slice函数

unsafe.Slice函数支持基于一个数组创建一个切片，该数组将作为切片的底层存储，它也可以理解为等价于下面常用“代码片段”语法糖函数：

```
func Slice(ptr *ArbitraryType, len IntegerType) []ArbitraryType

<=>

(*[len]ArbitraryType)(unsafe.Pointer(ptr))[:]
```

下面是unsafe.Slice的一个应用例子：

```
// github.com/bigwhite/experiments/tree/master/go1.17-examples/lang/unsafe/slice/main.go
func main() {
 var a = [5]int{11, 12, 13, 14, 15}
 s1 := a[:]
 s2 := unsafe.Slice(&a[0], 5)

 fmt.Println(s1) // [11 12 13 14 15]
 fmt.Println(s2) // [11 12 13 14 15]
 fmt.Printf("the type of s2 is %T\n", s2)

 s2[2] += 10
 fmt.Println(a)  // [11 12 23 14 15]
 fmt.Println(s1) // [11 12 23 14 15]
 fmt.Println(s2) // [11 12 23 14 15]
}
```

我们看到基于unsafe.Slice与基于数组进行切片得到的两个切片一样的，它们的底层数组都是数组a。因此，无论通过修改哪个切片元素，都会反映到另外一个切片中并反映到底层数组上。

3. 小结

在本文中，我们了解到了Go 1.17新增的很少的语言特性，这些个性更多从语言的易用性、安全性等方面考虑才添加的，相较于以往版本，这些新增特性算是不少了。如果要期待语言特性的巨大变更，那还是一起等Go 1.18吧。Go 1.18保证让你爽歪歪。泛型(类型参数)的加入必然让go代码变得比以前更烧脑一些。

本文涉及代码可以在**这里**下载：github.com/bigwhite/exp...

- 作者：tonybai.com

发布于 2021-08-19 21:50

Go 语言 | Go 编程 | Golang 最佳实践