Go 语言结构体 Go 语言范围(Range)

Go 语言切片(Slice)

Go 语言切片是对数组的抽象。

Go 数组的长度不可改变，在特定场景中这样的集合就不太适用，Go中提供了一种灵活，功能强悍的内置类型切片("动态数组"),与数组相比切片的长度是不固定的，可以追加元素，在追加时可能使切片的容量增大。

定义切片

你可以声明一个未指定大小的数组来定义切片：

```
var identifier []type
```

切片不需要说明长度。

或使用make()函数来创建切片:

```
var slice1 []type = make([]type, len)

也可以简写为

slice1 := make([]type, len)
```

也可以指定容量，其中capacity为可选参数。

```
make([]T, length, capacity)
```

这里 len 是数组的长度并且也是切片的初始长度。

切片初始化

```
s :=[] int {1,2,3 }
```

直接初始化切片，[]表示是切片类型，{1,2,3}初始化值依次是1,2,3.其cap=len=3

```
s := arr[:]
```

初始化切片s,是数组arr的引用

```
s := arr[startIndex:endIndex]
```

将arr中从下标startIndex到endIndex-1 下的元素创建为一个新的切片

```
s := arr[startIndex:]
```

缺省endIndex时将表示一直到arr的最后一个元素

```
s := arr[:endIndex]
```

缺省startIndex时将表示从arr的第一个元素开始

```
s1 := s[startIndex:endIndex]
```

通过切片s初始化切片s1

```
s :=make([]int,len,cap)
```

通过内置函数make()初始化切片s,[]int 标识为其元素类型为int的切片

---

len() 和 cap() 函数

切片是可索引的，并且可以由 len() 方法获取长度。

切片提供了计算容量的方法 cap() 可以测量切片最长可以达到多少。

以下为具体实例：

实例

```
package main

import "fmt"

func main() {
   var numbers = make([]int,3,5)

   printSlice(numbers)
}

func printSlice(x []int){
   fmt.Printf("len=%d cap=%d slice=%v\n",len(x),cap(x),x)
}
```

以上实例运行输出结果为:

```
len=3 cap=5 slice=[0 0 0]
```

空(nil)切片

一个切片在未初始化之前默认为 nil，长度为 0，实例如下：

实例

```
package main

import "fmt"

func main() {
   var numbers []int

   printSlice(numbers)

   if(numbers == nil){
      fmt.Printf("切片是空的")
   }
}

func printSlice(x []int){
   fmt.Printf("len=%d cap=%d slice=%v\n",len(x),cap(x),x)
}
```

以上实例运行输出结果为:

```
len=0 cap=0 slice=[]
切片是空的
```

切片截取

可以通过设置下限及上限来设置截取切片 *[lower-bound:upper-bound]*，实例如下：

实例

```
package main

import "fmt"

func main() {
   /* 创建切片 */
   numbers := []int{0,1,2,3,4,5,6,7,8}
   printSlice(numbers)

   /* 打印原始切片 */
   fmt.Println("numbers ==", numbers)

   /* 打印子切片从索引1(包含) 到索引4(不包含)*/
   fmt.Println("numbers[1:4] ==", numbers[1:4])

   /* 默认下限为 0*/
   fmt.Println("numbers[:3] ==", numbers[:3])
```

```
   /* 默认上限为 len(s)*/
   fmt.Println("numbers[4:] ==", numbers[4:])

   numbers1 := make([]int,0,5)
   printSlice(numbers1)

   /* 打印子切片从索引  0(包含) 到索引 2(不包含) */
   number2 := numbers[:2]
   printSlice(number2)

   /* 打印子切片从索引 2(包含) 到索引 5(不包含) */
   number3 := numbers[2:5]
   printSlice(number3)

}

func printSlice(x []int){
   fmt.Printf("len=%d cap=%d slice=%v\n",len(x),cap(x),x)
}
```

执行以上代码输出结果为：

```
len=9 cap=9 slice=[0 1 2 3 4 5 6 7 8]
numbers == [0 1 2 3 4 5 6 7 8]
numbers[1:4] == [1 2 3]
numbers[:3] == [0 1 2]
numbers[4:] == [4 5 6 7 8]
len=0 cap=5 slice=[]
len=2 cap=9 slice=[0 1]
len=3 cap=7 slice=[2 3 4]
```

append() 和 copy() 函数

如果想增加切片的容量，我们必须创建一个新的更大的切片并把原分片的内容都拷贝过来。

下面的代码描述了从拷贝切片的 copy 方法和向切片追加新元素的 append 方法。

实例

```
package main

import "fmt"

func main() {
   var numbers []int
   printSlice(numbers)

   /* 允许追加空切片 */
   numbers = append(numbers, 0)
   printSlice(numbers)
```

```
   /* 向切片添加一个元素 */
   numbers = append(numbers, 1)
   printSlice(numbers)

   /* 同时添加多个元素 */
   numbers = append(numbers, 2,3,4)
   printSlice(numbers)

   /* 创建切片 numbers1 是之前切片的两倍容量*/
   numbers1 := make([]int, len(numbers), (cap(numbers))*2)

   /* 拷贝 numbers 的内容到 numbers1 */
   copy(numbers1,numbers)
   printSlice(numbers1)
}

func printSlice(x []int){
   fmt.Printf("len=%d cap=%d slice=%v\n",len(x),cap(x),x)
}
```

以上代码执行输出结果为：

```
len=0 cap=0 slice=[]
len=1 cap=1 slice=[0]
len=2 cap=2 slice=[0 1]
len=5 cap=6 slice=[0 1 2 3 4]
len=5 cap=12 slice=[0 1 2 3 4]
```

← Go 语言结构体 Go 语言范围(Range) →

6 篇笔记 写笔记