数组和切片和map

数组

定义方式：var name [length]type 例子

数组定义和初始化

```
1：定义
var str[10] string //创建了包含10个元素的string类型的元素
var ints[5] int //创建了包含5个int类型的元素
2：初始化
var str =[5]string{"ju","liu","zhang","xue","oh"} == str:=
[5]string{"ju","liu","zhang","xue","oh"}
var balance = [...]float32{1000.0, 2.0, 3.4, 7.0, 50.0}
```

多维数组和数组在函数中使用

1:多维数组的使用

```
func main(){
    var str=[3][3]uint{{1,2,3}{4,5,6}{7,8,9}}
    for i:=0;i<3;i++{
        for j:=0;j<3;j++{
            fmt.Println(str[i][j])
        }
    }
}
```

2:数组在函数中使用

```
func fun1(arr *[5]int, size int) {
for i := 0; i < size; i++ {
arr[i] = arr[i] + 5
}
}
func display(arr [5]int, size int) {   //func diaplay(arr *[5]int)
for i := 0; i < size; i++ {
fmt.Printf("%d ", arr[i])
}
}
func main(){
var arr = [5]int{1, 1, 1, 1, 1}
display(arr, 5)
fun1(&arr, 5)
```

```
fmt.Printf("\n")
display(arr, 5)
}
//在实验中我发现，在Go语言里面值和指针是分开的，而且分开的很彻底，在C/C++里面的话当传送数组名的话，那么默认的就会把它的地址给传送过去，我们对
//其相似的地址里面的值进行操控的时候，在调用函数里面就会改变，但是在Go语言里面的话就不会，除非我们加指针
```

切片

why?

分片是一种抽象的数组，相对于数组死板的无法动态的去增加长度，Go的分片很好的解决了这个问题。

切片的定义和初始化

方式1:就是在数组的长度里面什么都不需要填写。var arr []int 方式2：使用内置的make去设置一个切片。 func make([]T, len, cap) []T (arr:=make([]int，5))

切片的lens()和cap()

长度是切片引用的元素数目。容量是底层数组的元素数目（从切片指针开始) 切片操作并不复制切片指向的元素。它创建一个新的切片并复用原来切片的底层数组

例子：

```
func main(){
str := make([]int, 5)
fmt.Println(len(str))
fmt.Println(cap(str))
str = str[2:4]
fmt.Println(len(str))
fmt.Println(cap(str))
}
//结果是5 5 2 3
```

切片的append()和copy()

append函数是在切片后面追加一些元素 函数签名 func append(s []T, x ...T) []T copy函数将源切片的元素复制到目的切片.它返回复制元素的数目. 函数签名： func copy(dst, src []T) int 使用append增加元素的时候 可以使用append(des,"woaini"...).加三个点的方式。 ！！增加切片的方式如下： s = s[:cap(s)] 对于切片的一些理解： 切片是引用类型，它用一个切片段指向了那个切片。当我们赋值给另一个切片的时候，其实他们底层指向的同一个切片数组。为什么在函数调用之后(函数里面使用append)，我们打印原来的切片 的值会发现发现在调用函数里面压入的值并没有被展示出来。 原因有二： 1：append的时候，超过了cap的值，然后新的切片的底层指向了新的空间，这样的改变就分别是两个不同的改变。 2：当我们发生并没有超出cab的值，但是打印在函数里面增加的数据不见了。原因为：因为作用域的问题，其实也就是切片段的不同，出作用域以后，函数里面的那个切片的切片段就不见了。 当然只留下了调用函数区域的切片段。

map

键值对的存在 map使用的是无序的hash，并不是像红黑树一样有顺序的结构。

```
key := make(map[string]int)
key["ju"] = 1
key["wen"] = 2
key["jie"] = 3
key["liu"] = 4
key["jia"] = 5
key["li"] = 6
for val := range key {
fmt.Printf("map value is %d\n", key[val])
}
```