1、基本介绍

sort包提供了`排序切片和用户自定义数据集的函数`。

2、Interface接口

这个接口是所有需要调用包中的排序函数或者方法需要实现的接口

type Interface

```
type Interface interface {
    // Len方法返回集合中的元素个数
    Len() int
    // Less方法报告索引i的元素是否比索引j的元素小
    Less(i, j int) bool
    // Swap方法交换索引i和j的两个元素
    Swap(i, j int)
}
```

一个满足sort.Interface接口的（集合）类型可以被本包的函数进行排序。方法要求集合中的元素可以被整数索引。

3、包中的三个类型

type `IntSlice、Float64Slice和StringSlice`。这三个类型分别是[]int、[]float64和[]string的别名。这三个类型都实现了Interface。

type IntSlice

```
type IntSlice []int
```

IntSlice给[]int添加方法以满足Interface接口，以便排序为递增序列。

func (IntSlice) Len

```
func (p IntSlice) Len() int
```

func (IntSlice) Less

```
func (p IntSlice) Less(i, j int) bool
```

func (IntSlice) Swap

```
func (p IntSlice) Swap(i, j int)
```

func (IntSlice) Sort

```
func (p IntSlice) Sort()
```

Sort等价于调用Sort(p)

func (IntSlice) Search

```
func (p IntSlice) Search(x int) int
```

Search等价于调用SearchInts(p, x)

type Float64Slice

```
type Float64Slice []float64
```

Float64Slice给[]float64添加方法以满足Interface接口，以便排序为递增序列。

func (Float64Slice) Len

```
func (p Float64Slice) Len() int
```

func (Float64Slice) Less

```
func (p Float64Slice) Less(i, j int) bool
```

func (Float64Slice) Swap

```
func (p Float64Slice) Swap(i, j int)
```

func (Float64Slice) Sort

```
func (p Float64Slice) Sort()
```

Sort等价于调用Sort(p)

func (Float64Slice) Search

```
func (p Float64Slice) Search(x float64) int
```

Search等价于调用SearchFloat64s(p, x)

type StringSlice

```
type StringSlice []string
```

StringSlice给[]string添加方法以满足Interface接口，以便排序为递增序列。

func (StringSlice) Len

```
func (p StringSlice) Len() int
```

func (StringSlice) Less ¶

```
func (p StringSlice) Less(i, j int) bool
```

func (StringSlice) Swap

```
func (p StringSlice) Swap(i, j int)
```

func (StringSlice) Sort

```
func (p StringSlice) Sort()
```

Sort等价于调用Sort(p)

func (StringSlice) Search

```
func (p StringSlice) Search(x string) int
```

Search等价于调用SearchStrings(p, x)

除了实现接口的方法，其他两个方法分别是排序和搜索。

```go
func main() {

    // IntSlice
    var slice sort.IntSlice = []int{1, 8, 5, 12, 2, 9}

    slice.Sort() // 排序

    fmt.Println(slice) // [1 2 5 8 9 12]

    index := slice.Search(12) // 乱序也能被搜索到

    fmt.Println(index) // 5

}
```


4、包中的函数--操作[]int[]float64[]string

包中还有一部分函数，用于给[]int、[]float64和[]string三种类型做排序查找，以及判断是否已经升序排列了。
如果不需要使用上面的对象操作，可以直接使用以下函数操作。

func Ints

```
func Ints(a []int)
```

Ints函数将a排序为递增顺序。

Example

```
s := []int{5, 2, 6, 3, 1, 4} // unsorted
sort.Ints(s)
fmt.Println(s)
```

Output:

```
[1 2 3 4 5 6]
```

func IntsAreSorted

```
func IntsAreSorted(a []int) bool
```

IntsAreSorted检查a是否已排序为递增顺序。

func SearchInts

```
func SearchInts(a []int, x int) int
```

SearchInts在递增顺序的a中搜索x，返回x的索引。如果查找不到，返回值是x应该插入a的位置（以保证a的递增顺序），返回值可以是len(a)。


```go
func main() {

    var data []int = []int{1, 8, 5, 12, 2, 9}

    fmt.Println(sort.IntsAreSorted(data)) // false 判断是否已经递增排序了

    sort.Ints(data) // [1 2 5 8 9 12]

    fmt.Println(sort.IntsAreSorted(data)) // true

    index := sort.SearchInts(data, 5)
    fmt.Println(index) // 2

    fmt.Println(data) // [1 2 5 8 9 12]

}
```

func Float64s

```
func Float64s(a []float64)
```

Float64s函数将a排序为递增顺序。

func Float64sAreSorted ¶

```
func Float64sAreSorted(a []float64) bool
```

Float64sAreSorted检查a是否已排序为递增顺序。

func SearchFloat64s

```
func SearchFloat64s(a []float64, x float64) int
```

SearchFloat64s在递增顺序的a中搜索x，返回x的索引。如果查找不到，返回值是x应该插入a的位置（以保证a的递增顺序），返回值可以是len(a)。

func Strings

```
func Strings(a []string)
```

Strings函数将a排序为递增顺序。

func StringsAreSorted

```
func StringsAreSorted(a []string) bool
```

StringsAreSorted检查a是否已排序为递增顺序。

func SearchStrings

```
func SearchStrings(a []string, x string) int
```

SearchStrings在递增顺序的a中搜索x，返回x的索引。如果查找不到，返回值是x应该插入a的位置（以保证a的递增顺序），返回值可以是len(a)。

5、包中的函数--操作Interface

另一部分函数就是传入实现了Interface接口的对象，就可以实现相关的排序、查找等操作。

func Sort

```
func Sort(data Interface)
```

Sort排序data。它调用1次data.Len确定长度，调用O(n*log(n))次data.Less和data.Swap。本函数不能保证排序的稳定性（即不保证相等元素的相对次序不变）。

func Stable

```
func Stable(data Interface)
```

Stable排序data，并保证排序的稳定性，相等元素的相对次序不变。

它调用1次data.Len，O(n*log(n))次data.Less和O(n*log(n)*log(n))次data.Swap。

func IsSorted

```
func IsSorted(data Interface) bool
```

IsSorted报告data是否已经被排序。

func Reverse

```
func Reverse(data Interface) Interface
```

Reverse包装一个Interface接口并返回一个新的Interface接口，对该接口排序可生成递减序列。

Example

```
s := []int{5, 2, 6, 3, 1, 4} // unsorted
sort.Sort(sort.Reverse(sort.IntSlice(s)))
fmt.Println(s)
```

Output:

```
[6 5 4 3 2 1]
```


func Search

```
func Search(n int, f func(int) bool) int
```

Search函数采用二分法搜索找到[0, n)区间内最小的满足f(i)==true的值i。也就是说，Search函数希望f在输入位于区间[0, n)的前面某部分（可以为空）时返回假，而在输入位于剩余至结尾的部分（可以为空）时返回真；Search函数会返回满足f(i)==true的最小值i。如果没有该值，函数会返回n。注意，未找到时的返回值不是-1，这一点和strings.Index等函数不同。Search函数只会用区间[0, n)内的值调用f。

一般使用Search找到值x在插入一个有序的、可索引的数据结构时，应插入的位置。这种情况下，参数f（通常是闭包）会捕捉应搜索的值和被查询的数据集。

例如，给定一个递增顺序的切片，调用Search(len(data), func(i int) bool { return data[i] >= 23 })会返回data中最小的索引i满足data[i] >= 23。如果调用者想要知道23是否在切片里，它必须另外检查data[i] == 23。

搜索递减顺序的数据时，应使用<=运算符代替>=运算符。

下列代码尝试在一个递增顺序的整数切片中找到值x：

```
x := 23
i := sort.Search(len(data), func(i int) bool { return data[i] >= x })
if i < len(data) && data[i] == x {
        // x is present at data[i]
} else {
        // x is not present in data,
        // but i is the index where it would be inserted.
}
```

6、自定义结构体实现排序

```
package main

import (
    "fmt"
    "sort"
)

// 定义一个结构体
type Student struct {
    name   string
    age    int
    weight float64
}

// 让Student结构体实现Interface接口
type Students []Student

func (stu Students) Len() int {

    return len(stu)
}

func (stu Students) Less(i, j int) bool {

    if stu[i].name < stu[j].name { // 使用姓名排序
        return true
    }
    return false
}

func (stu Students) Swap(i, j int) {

    stu[i], stu[j] = stu[j], stu[i]
}

func main() {

    students := Students([]Student{
        Student{
            "kiko",
            12,
            120.12,
        },
        Student{
            "yoyo",
            19,
            110.12,
        },
        Student{
            "jerry",
            10,
            90.12,
        },
```

```
    })

    fmt.Println(students)

    sort.Sort(students)

    fmt.Println(students)
}
```