链滴

golang 二维数组 / 切片排序

作者：hzylyh

原文链接：https://ld246.com/article/1553248083961

来源网站：链滴

二维数组/切片排序

前言

最近在做接口自动化的时候，碰到二维数组排序问题，java中用的stream的sorted方法，可以实现指字段排序（先根据第一个字段，如果第一个字段相同，再根据第二个字段，以此类推），但是最近在golang，就想着把项目用go重新实现，到这里有点卡住了，没找到现有的方法，就自己用go的sort口重新写了下，写此篇备忘。

实现

初版

java版

```
// 最初java中是用Comparator接口自己写的实现，后来不止怎么地，可能想试试新特性，就用的strem的sorted
// 有点搓，凑合理解，就是挨个比第一，第二，第三个字段，进行排序
public class HollTest {

    public static List sortList (List list) {
        List<List<String>> stream = (List<List<String>>) list.stream().sorted(Comparator.comparing(HollTest::comparingOne).thenComparing(HollTest::comparingTwo).thenComparing(HollTest::comparingThree))
                .collect(Collectors.toList());
        return stream;
    }

    private static String comparingOne(List<String> mylist){
        return mylist.get(0);
    }

    private static String comparingTwo(List<String> mylist){
        return mylist.get(1);
    }
    private static String comparingThree(List<String> mylist){
        return mylist.get(mylist.size() - 1);
    }
}
```

golang版

```
//最开始准备自己定义方法，后来一想，有现成的sort接口，为啥不用，就实现了下sort接口
package main

import (
    "sort"
    "fmt"
)
```

```
type myvalue2 [][]string

func (p myvalue2) Len() int {
    return len(p)
}

// 沿袭java当时的实现思路，挨个字段比，忽然发现怎么这么别扭，就出现了第二版
func (p myvalue2) Less(i, j int) bool {
    if p[i][0] > p[j][0] {
        return true
    } else if p[i][0] == p[j][0] && len(p[i]) > 0 {
        if p[i][1] > p[j][1] {
            return true
        } else if p[i][1] > p[j][1] && len(p[i]) > 1 {
            if p[i][2] > p[j][2] {
                return true
            } else {
                return false
            }
        } else {
            return false
        }
    } else {
        return false
    }
}

func (p myvalue2) Swap(i, j int) {
    p[i], p[j] = p[j], p[i]
}

func main()  {
    var m myvalue2
    m = [][]string{
        {"a", "a", "g", "c", "z", "d"},
        {"a", "c", "g", "v", "z", "d"},
        {"r", "a", "g", "c", "z", "a"},
        {"a", "a", "g", "b", "a", "v"},
        {"a", "a", "g", "b", "a", "a"},
        {"a", "a", "g", "b", "z", "a"},
    }
    sort.Sort(m)
    fmt.Println(m)
}
```

第二版

```
package main

import (
    "sort"
    "fmt"
)
```

```
//type myvalue []string

type myvalue2 [][]string

func (p myvalue2) Len() int {
    return len(p)
}

// 想了想能用循环，实现所有字段比对一次，更精确点，免得二维数组的元素前面n个元素都是一样的
func (p myvalue2) Less(i, j int) bool {
    for k := 0; k < len(p[i]); k++ {
        if p[i][k] > p[j][k] {
            return true
        } else if p[i][k] == p[j][k] {
            continue
        } else {
            return false
        }
    }
    return true
}

func (p myvalue2) Swap(i, j int) {
    p[i], p[j] = p[j], p[i]
}

func main()  {
    var m myvalue2
    m = [][]string{
        {"a", "a", "g", "c", "z", "d"},
        {"a", "c", "g", "v", "z", "d"},
        {"r", "a", "g", "c", "z", "a"},
        {"a", "a", "g", "b", "a", "v"},
        {"a", "a", "g", "b", "a", "a"},
        {"a", "a", "g", "b", "z", "a"},
    }
    sort.Sort(m)
    fmt.Println(m)
}
```