**14.1** 编写一个函数，给定字符串，产出一个包含所有字符的下标的映射。举例来说：**indexes("Mississippi")** 应返回一个映射，让**'M'**对应集**{0}**，**'i'**对应集**{1,4,7,10}**，依此类推。使用字符到可变集的映射。另外，你如何保证集是经过排序的？

更新 scala 到版本 2.10.0。有可变的可排序的 Set，实际上还是 TreeSet

```
import collection.mutable.{Map,HashMap,SortedSet}

def indexs(str:String):Map[Char,SortedSet[Int]]={
  var map = new HashMap[Char, SortedSet[Int]]();
  var i = 0;
  str.foreach{
    c=>
      map.get(c) match{
        case Some(result) => map(c) = result + i
        case None => map += (c-> SortedSet{i})
      }
      i += 1
  }
  map
}

println(indexs("Mississippi"))
```

**14.2** 重复前一个练习，这次用字符到列表的不可变映射。

```
import collection.immutable.HashMap
import collection.mutable.ListBuffer

def indexs(str:String):Map[Char,ListBuffer[Int]]={
  var map = new HashMap[Char, ListBuffer[Int]]()
  var i = 0
  str.foreach{
    c=>
      map.get(c) match{
        case Some(result) => result += i
        case None => map += (c-> ListBuffer{i})
      }
      i += 1
  }
  map
}

println(indexs("Mississippi"))
```

**14.3** 编写一个函数，从一个整型链表中去除所有的零值。

```
def removeZero(nums : List[Int]):List[Int]={
```

```
  nums.filter(_ != 0)
}

println(removeZero(List(3,5,0,2,7,0)))
```

**14.4** 编写一个函数，接受一个字符串的集合，以及一个从字符串到整数值的映射。返回整型的集合，其值为能和集合中某个字符串相对应的映射的值。举例来说，给定 **Array("Tom","Fred","Harry")**和 **Map("Tom"->3,"Dick"->4,"Harry"->5)**，返回 **Array(3,5)**。提示：用 **flatMap** 将 **get** 返回的 **Option** 值组合在一起

```
def strMap(strArr : Array[String],map : Map[String,Int]) : Array[Int] = {
  strArr.flatMap(map.get(_))
}

val a = Array("Tom","Fred","Harry")
val m = Map("Tom"->3,"Dick"->4,"Harry"->5)
println(strMap(a,m).mkString(","))
```

**14.5** 实现一个函数，作用与 **mkString** 相同，使用 **reduceLeft**。

```
import collection.mutable

trait MyMkString{
  this:mutable.Iterable[String]=>
  def myMkString = if( this != Nil) this.reduceLeft(_ + _)
}

var a = new mutable.HashSet[String] with MyMkString

a += "1"
a += "2"
a += "3"

println(a.myMkString)
```

**14.6** 给定整型列表 **lst**,**(lst :\ List[Int]())(_ :: _ )**得到什么**?(List[Int]() /: lst)(_ :+ _)**又得到什么？如何修改它们中的一个，以对原列表进行反向排序？

得到的结果和 **lst** 相同

```
val lst = List(1,2,3,4,5)

println((lst :\ List[Int]())(_ :: _))

println((List[Int]() /: lst)((a,b) => b :: a))
```

**14.7** 在 **13.11** 节中，表达式**(prices zip quantities) map { p => p._1 * p._2}**有些不够优雅。我们不能用 **(prices zip quantities) map { _ * _}**，因为 **_ * _** 是一个带两个参数的函数，而我们需要的是一个带单个类型为元组的参数的函数，**Function** 对象的 **tupled** 方法可以将带两个参数的函数改为以元组为参数的函数。将 **tupled** 应用于乘法函数，以使我们可以用它来映射由对偶组成的列表。

```
val prices = List(5.0,20.0,9.95)
val quantities = List(10,2,1)

println((prices zip quantities) map { Function.tupled(_ * _) })
```

**14.8** 编写一个函数。将 **Double** 数组转换成二维数组。传入列数作为参数。举例来说，**Array(1,2,3,4,5,6)** 和三列，返回 **Array(Array(1,2,3),Array(4,5,6))**。用 **grouped** 方法。

```
def divArr(arr:Array[Double],i:Int)={
  arr.grouped(i).toArray
}

val arr = Array(1.0,2,3,4,5,6)

divArr(arr,3).foreach(a => println(a.mkString(",")))
```

**14.9 Harry Hacker** 写了一个从命令行接受一系列文件名的程序。对每个文件名，他都启动一个新的线程来读取文件内容并更新一个字母出现频率映射，声明为：

```
val frequencies = new scala.collection.multable.HashMap[Char,Int] with
scala.collection.mutable.SynchronizedMap[Char,Int]
```

当读到字母 **c** 时，他调用

```
frequencies(c) = frequencies.getOrElse(c,0) + 1
```

为什么这样做得不到正确答案？如果他用如下方式实现呢：

```
import scala.collection.JavaConversions.asScalaConcurrentMap
val frequencies:scala.collection.mutable.ConcurrentMap[Char,Int] = new
java.util.concurrent.ConcurrentHashMap[Char,Int]
```

并发问题，并发修改集合不安全.修改后的代码和修改前的代码没有什么太大的区别.

**14.10 Harry Hacker** 把文件读取到字符串中，然后想对字符串的不同部分用并行集合来并发地更新字母出现频率映射。他用了如下代码：

```
val frequencies = new scala.collection.mutable.HashMap[Char,Int]
for(c <- str.par) frequencies(c) = frequencies.getOrElse(c,0) + 1
```

为什么说这个想法很糟糕？要真正地并行化这个计算，他应该怎么做呢？（提示：用 **aggregate**） 并行修改共享变量，结果无法估计。

```
import scala.collection.immutable.HashMap

val str = "abcabcac"

val frequencies = str.par.aggregate(HashMap[Char,Int]())(
        {
                (a,b) =>
                a + (b -> (a.getOrElse(b,0) + 1))
    }
        ,
        {
        (map1,map2) =>
```

```
        (map1.keySet ++ map2.keySet).foldLeft( HashMap[Char,Int]() ) {
         (result,k) =>
           result + ( k -> ( map1.getOrElse(k,0 ) + map2.getOrElse(k,0) ) )
       }
    }
)

println(frequencies)
```