5 映射和元组

5.1 设置一个映射,其中包含你想要的一些装备，以及它们的价格。然后构建另一个映射，采用同一组键，但是价格上打 9 折

映射的简单操作

```
scala> val map = Map("book"->10,"gun"->18,"ipad"->1000)
map:
scala.collection.immutable.Map[java.lang.String,Int] =
Map(book -> 10, gun -> 18, ipad -> 1000)

scala> for((k,v) <- map) yield (k,v * 0.9)
res3:
scala.collection.immutable.Map[java.lang.String,Double]
= Map(book -> 9.0, gun -> 16.2, ipad -> 900.0)
```

5.2 编写一段程序，从文件中读取单词。用一个可变映射来清点每个单词出现的频率。读取这些单词的操作可以使用 java.util.Scanner:

val in = new java.util.Scanner(new java.io.File("myfile.txt")) while(in.hasNext()) 处理 in.next() 或者翻到第 9 章看看更 Scala 的做法。 最后，打印出所有单词和它们出现的次数。

当然使用 Scala 的方法啦。参考第 9 章

首先，创建一个文件 myfile.txt。输入如下内容

```
test test ttt test ttt t test sss s
```

Scala 代码如下

```
import scala.io.Source
import scala.collection.mutable.HashMap

//val source = Source.fromFile("myfile.txt")
//val tokens = source.mkString.split("\\s+")  //此写法
tokens 为空，不知为何

val source = Source.fromFile("myfile.txt").mkString
```

```
val tokens = source.split("\\s+")

val map = new HashMap[String,Int]

for(key <- tokens){
    map(key) = map.getOrElse(key,0) + 1
}

println(map.mkString(","))
```

5.3 重复前一个练习，这次用不可变的映射

不可变映射与可变映射的区别就是，每次添加元素，都会返回一个新的映射

```
import scala.io.Source

val source = Source.fromFile("myfile.txt").mkString

val tokens = source.split("\\s+")

var map = Map[String,Int]()

for(key <- tokens){
  map += (key -> (map.getOrElse(key,0) + 1))
}

println(map.mkString(","))
```

5.4 重复前一个练习，这次使用已排序的映射，以便单词可以按顺序打印出来

和上面的代码没有什么区别，只是将映射修改为 SortedMap

```
import scala.io.Source
import scala.collection.SortedMap

val source = Source.fromFile("myfile.txt").mkString

val tokens = source.split("\\s+")
```

```
var map = SortedMap[String,Int]()

for(key <- tokens){
  map += (key -> (map.getOrElse(key,0) + 1))
}

println(map.mkString(","))
```

5.5 重复前一个练习，这次使用 java.util.TreeMap 并使之适用于 Scala API

主要涉及 java 与 scala 的转换类的使用

```
import scala.io.Source
import scala.collection.mutable.Map
import scala.collection.JavaConversions.mapAsScalaMap
import java.util.TreeMap

val source = Source.fromFile("myfile.txt").mkString

val tokens = source.split("\\s+")

val map:Map[String,Int] = new TreeMap[String,Int]

for(key <- tokens){
  map(key) = map.getOrElse(key,0) + 1
}

println(map.mkString(","))
```

5.6 定义一个链式哈希映射,将"Monday"映射到 java.util.Calendar.MONDAY,依次类推加入其他日期。展示元素是以插入的顺序被访问的

LinkedHashMap 的使用

```
import scala.collection.mutable.LinkedHashMap
import java.util.Calendar
```

```
val map = new LinkedHashMap[String,Int]

map += ("Monday"->Calendar.MONDAY)
map += ("Tuesday"->Calendar.TUESDAY)
map += ("Wednesday"->Calendar.WEDNESDAY)
map += ("Thursday"->Calendar.THURSDAY)
map += ("Friday"->Calendar.FRIDAY)
map += ("Saturday"->Calendar.SATURDAY)
map += ("Sunday"->Calendar.SUNDAY)



println(map.mkString(","))
```

5.7 打印出所有 Java 系统属性的表格，

属性转 scala map 的使用

```
import
scala.collection.JavaConversions.propertiesAsScalaMap

val props:scala.collection.Map[String,String] =
System.getProperties()

val keys = props.keySet

val keyLengths = for( key <- keys ) yield key.length

val maxKeyLength = keyLengths.max

for(key <- keys) {
  print(key)
  print(" " * (maxKeyLength - key.length))
  print(" | ")
  println(props(key))
}
```

5.8 编写一个函数 minmax(values:Array[Int]),返回数组中最小值和最大值的对偶

```
def minmax(values:Array[Int])={
  (values.max,values.min)
}
```

5.9 编写一个函数 Iteqgt(values:Array[int],v:Int),返回数组中小于 v,等于 v 和大于 v 的数量，要求三个值一起返回

```
def iteqgt(values:Array[Int],v:Int)={
  val buf = values.toBuffer
  (values.count(_ < v),values.count(_ ==
v),values.count(_ > v))
}
```

5.10 当你将两个字符串拉链在一起，比如"Hello".zip("World")，会是什么结果？想出一个讲得通的用例

```
scala> "Hello".zip("World")
res0: scala.collection.immutable.IndexedSeq[(Char, Char)]
= Vector((H,W), (e,o), (l,r), (l,l), (o,d))
```

StringOps 中的 zip 定义如下

```
abstract def zip[B](that: GenIterable[B]): StringOps[(A,
B)]
```

GenIterable 是可遍历对象需要包含的 trait，对于 String 来说，它是可遍历的。但是它的遍历是遍历单个字母。 所以拉链就针对每个字母来进行。