Scala:fold,foldLeft和foldRight区别与联系

从本质上说，fold函数将一种格式的输入数据转化成另外一种格式返回。fold, foldLeft和foldRight这三个函数除了有一点点不同外，做的事情差不多。我将在下文解释它们的共同点并解释它们的不同点。

我将从一个简单的例子开始，用fold计算一系列整型的和。

```
val numbers = List(5, 4, 8, 6, 2)
numbers.fold(0) { (z, i) =>
  z + i
}
// result = 25
```

List中的fold方法需要输入两个参数：初始值以及一个函数。输入的函数也需要输入两个参数：累加值和当前item的索引。那么上面的代码片段发生了什么事？

代码开始运行的时候，初始值0作为第一个参数传进到fold函数中，list中的第一个item作为第二个参数传进fold函数中。

1、fold函数开始对传进的两个参数进行计算，在本例中，仅仅是做加法计算，然后返回计算的值；

2、Fold函数然后将上一步返回的值作为输入函数的第一个参数，并且把list中的下一个item作为第二个参数传进继续计算，同样返回计算的值；

3、第2步将重复计算，直到list中的所有元素都被遍历之后，返回最后的计算值，整个过程结束；

4、这虽然是一个简单的例子，让我们来看看一些比较有用的东西。早在后面将会介绍foldLeft函数，并解释它和fold之间的区别，目前，你只需要想象foldLeft函数和fold函数运行过程一样。

下面是一个简单的类和伴生类：

```
class Foo(val name: String, val age: Int, val sex: Symbol)

object Foo {
  def apply(name: String, age: Int, sex: Symbol) = new Foo(name, age, sex)
}
```

假如我们有很多的Foo实例，并存在list中：

```
val fooList = Foo("Hugh Jass", 25, 'male) ::
              Foo("Biggus Dickus", 43, 'male) ::
              Foo("Incontinentia Buttocks", 37, 'female) ::
              Nil
```

我们想将上面的list转换成一个存储[title] [name], [age]格式的String链表：

```
val stringList = fooList.foldLeft(List[String]()) { (z, f) =>
  val title = f.sex match {
    case 'male => "Mr."
    case 'female => "Ms."
  }
  z :+ s"$title ${f.name}, ${f.age}"
}

// stringList(0)
// Mr. Hugh Jass, 25

// stringList(2)
// Ms. Incontinentia Buttocks, 37
```

和第一个例子一样，我们也有个初始值，这里是一个空的String list，也有一个操作函数。在本例中，我们判断了性别，并构造了我们想要的String，并追加到累加器中（这里是一个list）。

fold, foldLeft, and foldRight之间的区别

主要的区别是fold函数操作遍历问题集合的顺序。foldLeft是从左开始计算，然后往右遍历。foldRight是从右开始算，然后往左遍历。而fold遍历的顺序没有特殊的次序。来看下这三个函数的实现吧（在TraversableOnce特质里面实现）

```
def fold[A1 >: A](z: A1)(op: (A1, A1) => A1): A1 = foldLeft(z)(op)

def foldLeft[B](z: B)(op: (B, A) => B): B = {
  var result = z
  this.seq foreach (x => result = op(result, x))
  result
}

def foldRight[B](z: B)(op: (A, B) => B): B =
  reversed.foldLeft(z)((x, y) => op(y, x))
```

由于fold函数遍历没有特殊的次序，所以对fold的初始化参数和返回值都有限制。在这三个函数中，初始化参数和返回值的参数类型必须相同。

第一个限制
是初始值的类型必须是list中元素类型的超类。在我们的例子中，我们的对List[Int]进行fold计算，而初始值是Int类型的，它是List[Int]的超类。

第二个限制
是初始值必须是中立的(neutral)。也就是它不能改变结果。比如对加法来说，中立的值是0；而对于乘法来说则是1，对于list来说则是Nil。

顺便说下，其实foldLeft和foldRight函数还有两个缩写的函数：

```
 def /:[B](z: B)(op: (B, A) => B): B = foldLeft(z)(op)

def :₩[B](z: B)(op: (A, B) => B): B = foldRight(z)(op)

scala> (0/:(1 to 100))(_+_)
res32: Int = 5050

scala> ((1 to 100):    )(_+_)
res24: Int = 5050
```