# Scala语法拾遗

## foldLeft等高级函数

应用场景：

val featureMap\_rdd: RDD[((Long, String), mutable.HashMap[Any, Long])] = new\_rdd.mapValues{

case (property\_arr, filsiz) => {

property\_arr.foldLeft(new mutable.HashMap[Any, Long]()){

case (muMap, property) if property != "" && property != -1L => {

muMap += property -> (muMap.getOrElse(property, 0L) + 1L)

}

muMap += "filsiz" -> (muMap.getOrElse("filsiz", 0L) + math.log(filsiz).toLong)

}

}

}

我们不想然所有的property参与foldLeft而需要加一个判断，这一点aggregate不行。

## Array[Array[T]]的转置重写

object Transposer{

implicit class TransArr[T](val matrix: Array[Array[T]]){

def transposeee(): Seq[Seq[T]] =

{

Array.range(0, matrix.head.length).map(i => matrix.view.map(\_(i)))

}

}

implicit class TransSeq[T](val matrix: Seq[Seq[T]]){

def transposeee(): Seq[Seq[T]] =

{

Array.range(0, matrix.head.length).map(i => matrix.view.map(\_(i)))

}

}

}

val matrix = Seq(Seq(0, 1, 0), Seq(0, 0, 1), Seq(1, 0, 0))

matrix.foreach(arr => println(arr.mkString(", ")))

// 转置

import Transposer.\_

matrix.transposeee().foreach(arr => println(arr.mkString(", ")))

## Array.slice源码

def slice(from: Int, until: Int): Repr = {

val lo = math.max(from, 0)

val hi = math.min(math.max(until, 0), length)

val elems = math.max(hi - lo, 0)

val b = newBuilder

b.sizeHint(elems)

var i = lo

while (i < hi) {

b += self(i)

i += 1

}

b.result

}

左闭右开