键值对RDD

Key-Value RDD

键值对操作

键值对操作

- 键值对RDD是Spark中许多操作所需要的常见操作
- 键值对RDD是很多程序的重要构成要素，因为他们提供了并行操作各个键或跨节点重新进行数据分组的操作接口。例如：
  - 键值对RDD提供reduceByKey()方法，可以分别归约每个键对应的数据
  - 还有join操作，它可以把两个RDD中键相同的元素组合到一起，生成一个新的RDD
- 我们通常从一个RDD中提取某些字段（例如用户ID，某些事件的事件）作为pari RDD的键

创建键值对RDD

创建键值对RDD

- 在读取外部数据的时候会转换为键值对RDD
- 将普通的RDD转换为键值对RDD可以使用map方法

创建键值对RDD

RDD 创建

构建pair rdd的方式在不同语言中有所不同。

Python中，为了让提取键之后的数据能够在函数中应用，需要返回一个由二元组组成的RDD。

[5]:
```
list = ["Hadoop", "Spark", "Python", "Spark", "MachineLearning", "MachineLearning"]
rdd = sc.parallelize(list)
pairRDD = rdd.map(lambda word : (word, 1))
pairRDD.collect()
```

[5]:
```
[('Hadoop', 1),
 ('Spark', 1),
 ('Python', 1),
 ('Spark', 1),
 ('MachineLearning', 1),
 ('MachineLearning', 1)]
```

键值对RDD的转换操作

键值对RDD的转换操作

- 键值对RDD可以使用标准RDD上的可用的转换操作
- 同样可以对键值对RDD进行函数的传递。
- 由于Pari RDD操作的是元组，所以传递的函数需要操作的是元组而不是单个元素

聚合操作

- 当数据以键值对形式组织的时候，聚合具有相同键的元素进行一些统计是很常见的操作。

- Spark有一组类似的操作，可以组合具有相同键的值，这些操作返回RDD，因此是转换操作，而不是行动操作

reduceByKey(func)

- reduceByKey与reduce类似，他们都接受一个函数，并使用该函数对值进行合并。

- reduceByKey会为数据集中的每一个键进行操作，每个操作会将相同key的值合并起来。

- 因为数据集中往往存在大量的key，所以reduceByKey没有实现一个向用户程序返回一个值的操作

reduceByKey(func)

reduceByKey(func)

reduceByKey(func)的功能是，使用func函数合并具有相同键的值。例如如下代码，对具有相同key的的value求合

[25]:
```
list = ["Hadoop", "Spark", "Python", "Spark", "MachineLearning", "MachineLearning"]
rdd = sc.parallelize(list)
pairRDD = rdd.map(lambda word : (word, 1))
pairRDD.reduceByKey(lambda a, b : a + b).collect()
```

[25]:
```
[('Hadoop', 1), ('Spark', 2), ('Python', 1), ('MachineLearning', 2)]
```

reduceByKey(func)

使用reduceByKey与mapValues求key对应的均值

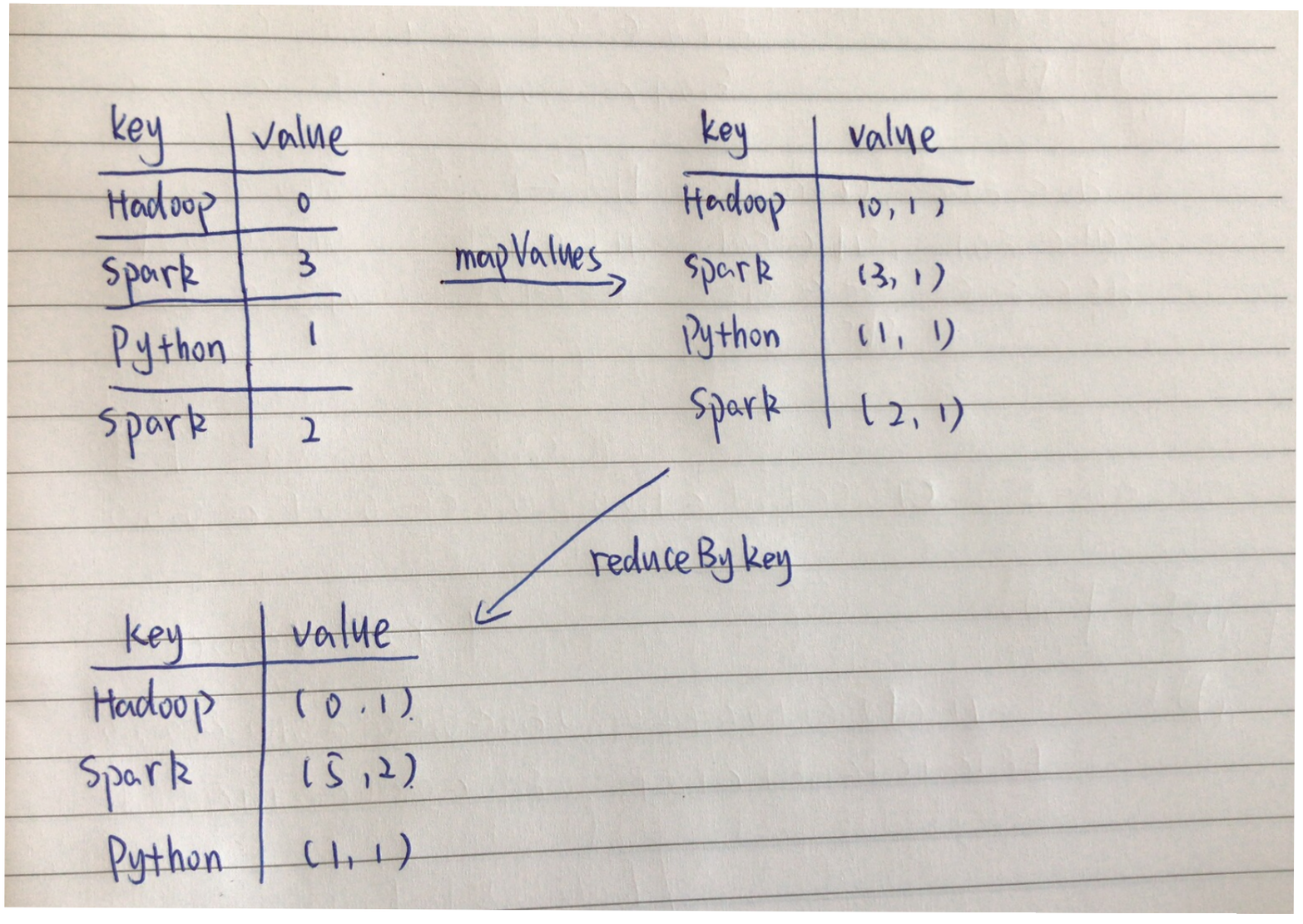
]:
```
rdd = sc.parallelize([('Hadoop', 0), ('Spark', 3), ('Python', 1), ('Spark', 2)])
# 转换为pairRDD形式
pairRDD = rdd.map(lambda t: (t[0], t[1]))
# mapValues(func)是对每个value进行操作
sum_num = pairRDD.mapValues(lambda x: (x, 1)).reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
sum_num.collect()
```

]:
```
[('Hadoop', (0, 1)), ('Spark', (5, 2)), ('Python', (1, 1))]
```

]:
```
sum_num.map(lambda x: x[1][0] / x[1][1]).collect()
```

]:
```
[0.0, 2.5, 1.0]
```

求均值的数据走向


key
value
Hadoop
0
Spark
3
Python
1
Spark
2
mapValues
key
value
Hadoop
(0, 1)
Spark
(3, 1)
Python
(1, 1)
Spark
(2, 1)
reduceBy key
key
value
Hadoop
(0, 1)
Spark
(5, 2)
Python
(1, 1)

reduceByKey(func)

- reduceByKey会在计算全局的结果之前，在每个worker上进行本地的合并

reduceByKey(func)

使用Jupyter 统计文本中单词出现的个数

1. 导入pyspark包

[1]:
```
from pyspark import SparkContext
```

2. 创建SparkContext

[2]:
```
sc = SparkContext('local', 'Wordcount')
```

[4]:
```
textFile = sc.textFile('../data/words')
word_count = textFile.flatMap(lambda line: line.split(" "))\
                     .map(lambda word: (word,1))\
                     .reduceByKey(lambda a, b : a + b)
```

[5]:
```
# foreach会打印到后台
#word_count.foreach(print)
word_count.collect()
```

[5]:
```
[('Path:', 1),
 ('cantaloupe', 1),
 ('srv', 1),
 ('cs', 1),
 ('cmu', 3)
```

数据分组

- 对于有键的数据，一个常见的操作是将数据根据键进行分组

- 比如，查看一个顾客的所有订单

- groupByKey会使用RDD中的键对数据进行分组，如果一个由类型K的键和类型V的值组成的RDD，所得到的结果RDD类型会是[K, Iterable[V]]

groupByKey()

groupByKey()

groupByKey()的功能是，对具有相同键的值进行分组。例如对[('Hadoop', 1), ('Spark', 1), ('Python', 1), ('Spark', 1), ('MachineLearning', 1), ('MachineLearning', 1)] 进行分组的结果是[('Hadoop', (1)), ('Spark', (1, 1)), ('Python', (1)), ('MachineLearning', (1, 1))]

[26]:
```
list = ["Hadoop", "Spark", "Python", "Spark", "MachineLearning", "MachineLearning"]
rdd = sc.parallelize(list)
pairRDD = rdd.map(lambda word : (word, 1))
pairRDD.groupByKey().collect()
```

[26]:
```
[('Hadoop', <pyspark.resultiterable.ResultIterable at 0x7feb00175668>),
 ('Spark', <pyspark.resultiterable.ResultIterable at 0x7feb00175dd8>),
 ('Python', <pyspark.resultiterable.ResultIterable at 0x7feb001750f0>),
 ('MachineLearning',
  <pyspark.resultiterable.ResultIterable at 0x7feb00175cf8>)]
```

连接操作

- 将有键的数据与另一组有键的数据一起使用是对键值对数据最常用的使用方式

- 连接的方式有多样：右外连接、左外连接和内连接

连接操作

join是内连接

[13]:
```
rdd_1 = sc.parallelize([('Hadoop', 0), ('Spark', 3), ('Python', 1), ('Spark', 2)])
rdd_2 = sc.parallelize([('Hadoop', 0), ('Spark', 3)])
rdd_1.join(rdd_2).collect()
```

[13]:
```
[('Spark', (3, 3)), ('Spark', (2, 3)), ('Hadoop', (0, 0))]
```

leftOuterJoin() 是左外连接

[15]:
```
rdd_1 = sc.parallelize([('Hadoop', 0), ('Spark', 3), ('Python', 1), ('Spark', 2)])
rdd_2 = sc.parallelize([('Hadoop', 0), ('Spark', 3)])
rdd_1.leftOuterJoin(rdd_2).collect()
```

[15]:
```
[('Spark', (3, 3)),
 ('Spark', (2, 3)),
 ('Python', (1, None)),
 ('Hadoop', (0, 0))]
```

rightOuterJoin() 是右外连接

[18]:
```
rdd_1 = sc.parallelize([('Hadoop', 0), ('Spark', 3), ('Python', 1), ('Spark', 2)])
rdd_2 = sc.parallelize([('Hadoop', 0), ('Spark', 3), ('Java', 3)])
rdd_1.rightOuterJoin(rdd_2).collect()
```

[18]:
```
[('Spark', (3, 3)), ('Spark', (2, 3)), ('Java', (None, 3)), ('Hadoop', (0, 0))]
```

数据排序

- 很多时候，让数据排好序是很有用的，尤其在生成下游输出时。如果键有已定义的顺序，就可以对这种键值对RDD进行排序，在对数据进行collect（）或者save（）的话，就会显示有序的数据

数据排序

排序 ¶

我们使用sortByKey()对键值对RDD进行排序操作。

[23]:
```
rdd = sc.parallelize([('Hadoop', 0), ('Spark', 3), ('Python', 1), ('Spark', 2)])
# ascending = True, 升序
# ascending = False, 降序
rdd.sortByKey(ascending=False).collect()
```

[23]:
```
[('Spark', 3), ('Spark', 2), ('Python', 1), ('Hadoop', 0)]
```

[24]:
```
# 同样可以传递函数
# 按照字符串的顺序排序
rdd.sortByKey(ascending=False, keyfunc = lambda x: str(x)).collect()
```

[24]:
```
[('Spark', 3), ('Spark', 2), ('Python', 1), ('Hadoop', 0)]
```

其他转换方法-keys()

keys()

keys()只会把键值对RDD中的key返回形成一个新的RDD

[27]:
```
list = ["Hadoop", "Spark", "Python", "Spark", "MachineLearning", "MachineLearning"]
rdd = sc.parallelize(list)
pairRDD = rdd.map(lambda word : (word, 1))
pairRDD.keys().collect()
```

[27]:
```
['Hadoop', 'Spark', 'Python', 'Spark', 'MachineLearning', 'MachineLearning']
```

其他转换方法-values()

values()

values()会把键值对RDD中的value返回形成一个新的RDD

[28]:
```
list = ["Hadoop", "Spark", "Python", "Spark", "MachineLearning", "MachineLearning"]
rdd = sc.parallelize(list)
pairRDD = rdd.map(lambda word : (word, 1))
pairRDD.values().collect()
```

[28]:
```
[1, 1, 1, 1, 1, 1]
```

键值对RDD的行动操作

键值对RDD的行动操作

- 与转换操作一样，所有标准RDD支持的转换操作也同样可以使用在Pair RDD

- Pari RDD也提供了一些特殊的行动操作

键值对RDD的行动操作

countByKey() 对每个键的值的个数进行计数

```
rdd = sc.parallelize([('Hadoop', 0), ('Spark', 3), ('Python', 1), ('Spark', 2)])
pairRDD = rdd.map(lambda t: (t[0], t[1]))
pairRDD.countByKey()
```

```
defaultdict(int, {'Hadoop': 1, 'Spark': 2, 'Python': 1})
```

键值对RDD的行动操作

lookup() 返回给定键对应的值

```
rdd = sc.parallelize([('Hadoop', 0), ('Spark', 3), ('Python', 1), ('Spark', 2)])
pairRDD = rdd.map(lambda t: (t[0], t[1]))
pairRDD.lookup('Spark')
```

```
[3, 2]
```

数据分区

数据分区

- 在分布式系统中，通信的代价是很大的，因此控制数据分布以获得最少的网络传输可以极大的提高系统的整体性能

- Spark可以通过控制RDD分区来提高系统的性能

- 分区并不是对所有程序都有好处的，比如说RDD只需要扫描一次，那根本没有分区的必要。

- 只有数据集多次在基于键的相关操作时，分区才会体现出他的作用，例如连接这种操作

数据分区

- Spark中所有的键值对RDD都可以进行分区。
- 系统会根据一个针对键的函数对元素进行分组。
- 尽管Spark没有给出显示控制每个键具体落在哪个workder哪个分区的方法，但是它会保证同一组的键出现在同一个节点上

数据分区

- 内存中保存着一个巨大用户信息表(UserID, UserInfo), UserInfo包含一个用户订阅的主题列表

- 该表会周期性的与一个小文件进行组合，该小文件存放着过去五分钟内发生的事件(UserID, LinkInfo)，存放着过去5分钟内用户访问的网站。

- 我们需要统计用户订阅又没有访问的网站

数据分区

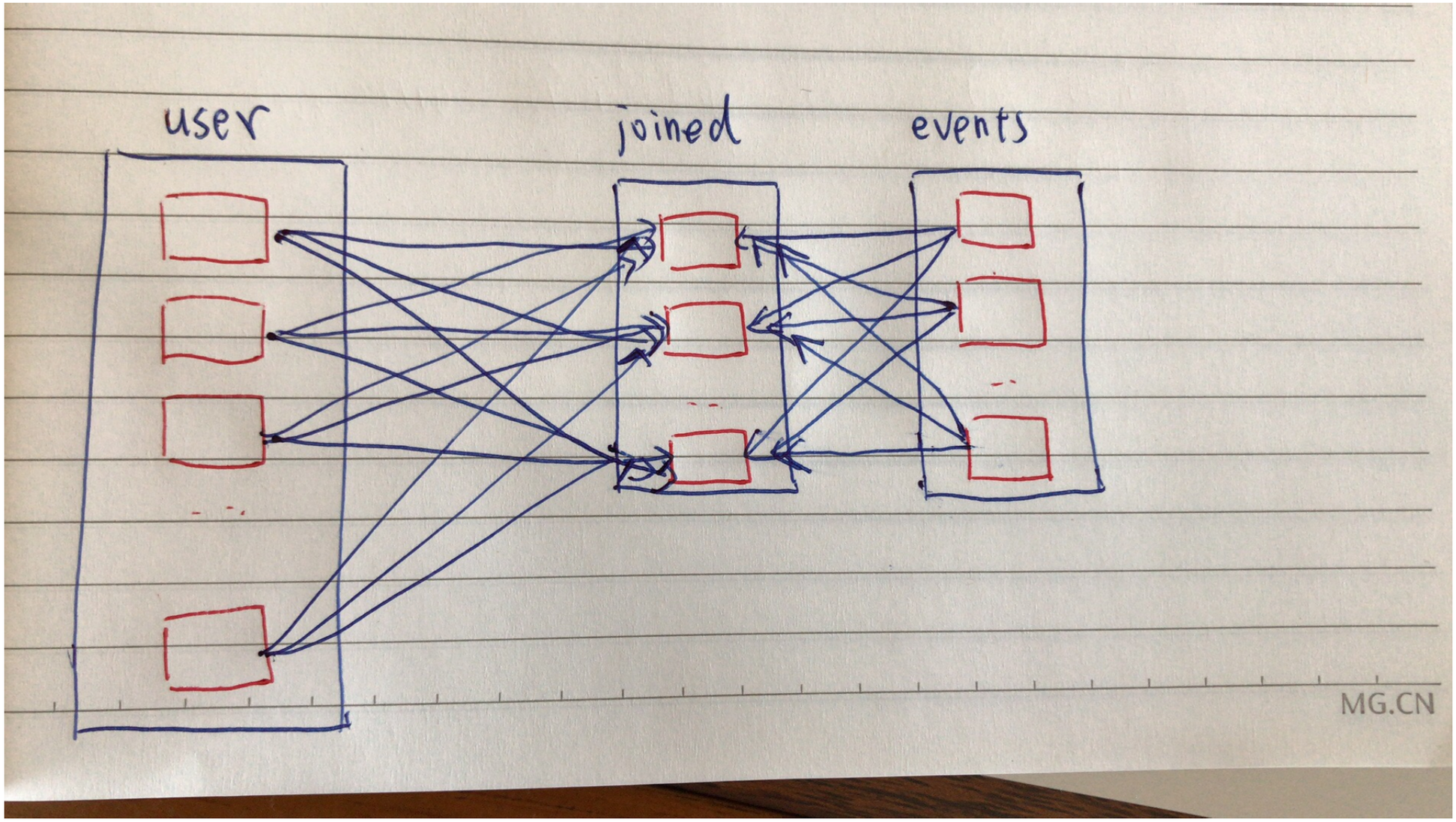
- 代码可以运行，但是效率不高，每次调用processNewLogs都会用到join。默认情况下，连接操作会讲两个数据集中的所有键的哈希值都求出来，将该哈希值相同的记录通过网络传到同一台机器上，然后在那台机器上对键相同的记录进行连接操作

- userData比event要大的多，每次调用时，都需对userData进行哈希值计算和跨界点数据混洗，虽然userData数据基本不发生变化

```
# 从HDFS上读取一个读取用户信息表(UserID, UserInfo)
userData = sc.sequenceFile('hdfs://...')

# 周期性的调用函数来处理过去五分钟产生的事件日志
# 也假设是来自于HDFS
def processNewLogs(logFileName):
    events = sc.sequenceFile(logFileName)
    joined = userData.join(events).filter(lambda topics: topics not in LinkInfo.topics )
    print('Number of visits to non-subscribed topics:', joined.count())
```

数据分区



数据分区

- 要解决这一问题，我们指定对RDD使用哈希分区即可。使用partitionBy（）可以进行分区，但是要记得持久化，否则每次都会对RDD进行分区

```
# 从HDFS上读取一个读取用户信息表(UserID, UserInfo)
# 构造100个分区
userData = sc.sequenceFile('hdfs://...').partitionBy(100).persist()
# 周期性的调用函数来处理过去五分钟产生的事件日志
# 也假设是来自于HDFS
def processNewLogs(logFileName):
    events = sc.sequenceFile(logFileName)
    joined = userData.join(events).filter(lambda topics: topics not in LinkInfo.topics )
    print('Number of visits to non-subscribed topics:', joined.count())
```

数据分区