Spark Core开发

RDD（Transform & Action）：弹性数据集，是Spark里对所处理数据的抽象。

Spark Sql 开发

Spark Streaming

Spark可以简单概括为3点：

Scala语法

RDD操作（Transform & Action）

分布式化

做Spark开发，其实就是用Scala语言进行RDD操作，Spark会自动将RDD分发到集群上，并将操作并行化执行。

Java工程师做分布式开发的话，常用Mapreduce。

而Scala工程师做分布式开发，用Spark，Scala编程语言具有很多优秀特性，实现相同功能时代码量是Java的1/5，而且更加易读易懂。

Spark v2.2.0 在线Api地址：

<http://spark.apache.org/docs/2.2.0/api/scala/index.html#org.apache.spark.package>

大家可以网上搜chm格式的api，分享出来。

## RDD

Spark 对数据的核心抽象——弹性分布式数据集（Resilient Distributed Dataset，简

称RDD）。

在Spark 中，对数据的所有操作不外乎创建RDD和操作RDD 。

对数据（Seq）的操作，比如List：

转换（Transform），map、filter ，返回List类型，数据转换/加工过程。

Action：head、tail、count ，返回不同类型，即我们需要的结果。

RDD就是类似集合类（Iterable），具有和集合类几乎完全相同的操作（Transform和Action）。

而在这一切背后，Spark 会自动将RDD 中的数据分发到集群上，并将操作并行化执行。

Spark 中的RDD 就是一个不可变的分布式对象集合。每个RDD 都被分为多个分区，这些

分区运行在集群中的不同节点上。RDD 可以包含Python、Java、Scala 中任意类型的对象，

甚至可以包含用户自定义的对象。

RDD是类似Iterator的数据结构。

## 创建RDD

用户可以使用两种方法创建RDD：

·用SparkContext 的parallelize(Seq) 把Seq转为RDD。该方式常用于学习和实验

·读外部数据，通常是读HDFS、消息队列等。

在前边学习中，我们通过SparkContext 的parallelize(Seq)的方法创建RDD，并进行各种学习。

先看看parallelize方法：

def parallelize[T](seq: Seq[T], numSlices: Int = defaultParallelism) : RDD[T]

numSlices是并行度，具有初始值所以调用时可以只给一个参数，这是Scala语言的特性。

比如可以parallelize(seq) ，可以parallelize(seq ,10) ，并行度为10意味着Spark把数据分割为10份，放在集群上运行。

defaultParallelism 是机器CPU个数。

# 查看CPU的个数

cat /proc/cpuinfo| grep"processor"| wc -l

## RDD 操作

RDD 两种类型的操作： Transform转化操作 和Action行动操作。

注：就是Iterable类中的函数， Transform返回Iterable本身类型，Action返回新类型。

Iterable: Seq、 Map

RDD： 单元素RDD、 PairRDD

Transform操作会由一个RDD 生成一个新的RDD，这个过程中不进行实质计算，只有当第一次Action操作时才会真正计算。 称作Lazy计算，惰性计算。

比如：

scala> val a = sc.parallelize(1 to 9, 3)

scala> val b = a.map(x => x\*2) // map() 是Transform函数

scala> b.first // first() 是Action函数，此时才进行计算。

行动操作会对RDD 计算出一个结果，可以把结果返回，或把结果存储到外部存储系统（如HDFS）中。

RDD是类似Iterable的数据结构，也具有Iterator类的Map()、filter()、flatMap()等高阶函数，这些函数是Scala里常用的。

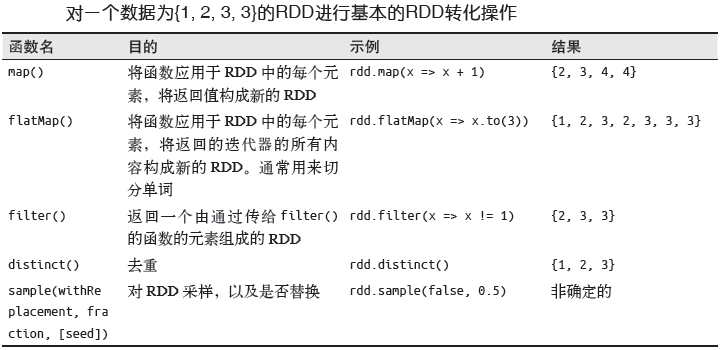
## Transform操作

RDD：分为单元素RDD和k-v元素PairRDD两种。

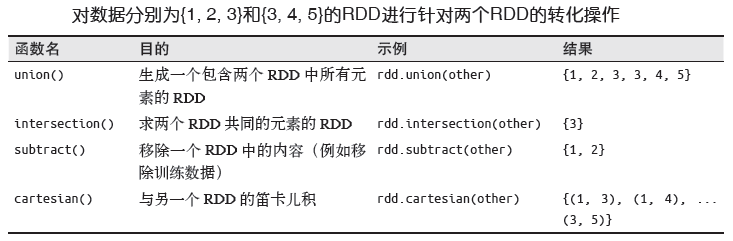
简单的有Map()、filter()、flatMap()，见下图：

flatMap() 对RDD中每个元素进行函数处理，和map函数不同的是，flatMap返回的可以是不同数据类型。

对单个RDD的Transform函数：



两个RDD之间的Transform函数：



## Action操作

前边提到的first() 、collect() 都是Action操作。常用的有：

collect()：把数据返回驱动器程序中最简单、最常见的操作, 通常在单元测试中使用，数据量不能 太大，因为放在内存中，数据量大会内存溢出。

reduce()：类似sum() ，如：val sum = rdd.reduce((x, y) => x + y) ，结果同sum

fold()：和reduce() 类似，多一个“初始值”，当初始值为零时效果同reduce(). fold(0) = reduce()

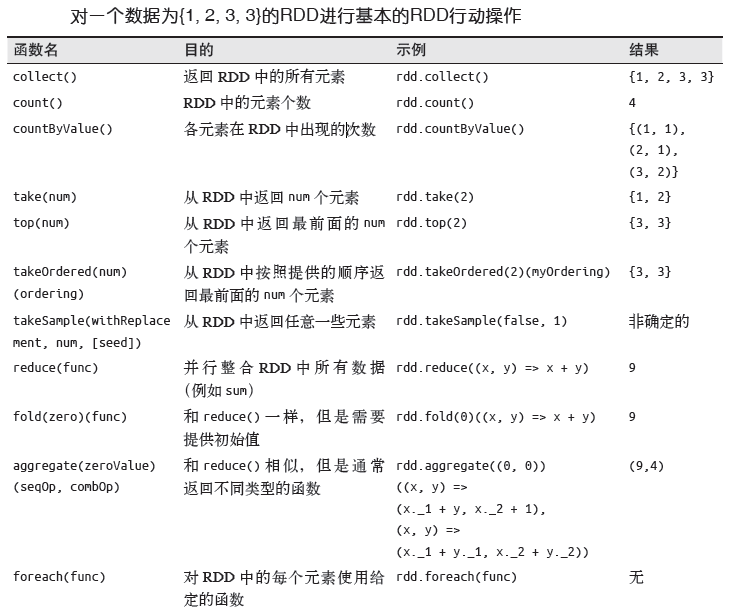
take(n) ：返回RDD 中的第n 个元素，并且尝试只访问尽量少的分区。

top(n) ：从RDD 中获取前几个元素

count() ：用来返回元素的个数

countByValue() ：返回一个从各值到值对应的计数的映射表

sum()：返回汇总



fold(n) 的执行原理：

每个分区里进行这样的计算：初始值+sum(元素)

最后进行：初始值+sum(分区sum值)

初始值累加次数为分区数+1次

scala> val rdd = sc.parallelize(1 to 5,2)

scala> rdd.fold(1)((x,y)=>x+y)

res8: Int = 18

scala> rdd.fold(2)((x,y)=>x+y)

res9: Int = 21

## 持久化函数persist()

RDD 是惰性(Lazy)求值的，当我们希望能多次使用同一个RDD时，RDD 调用行动操作，Spark 每次都会重算RDD 以及它的所有依赖, 这在迭代算法中消耗格外大。

如：

val rdd1 = rdd.map(x => x+1)

println(rdd1.first())

println(rdd1.count())

println(rdd1.sum())

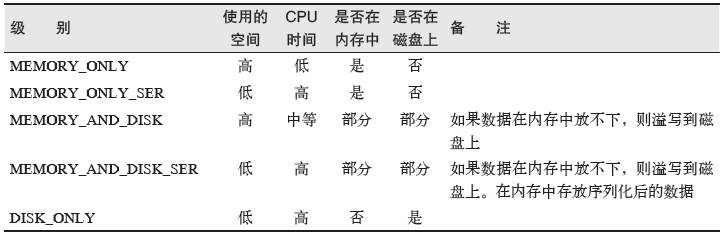
println(rdd1.collect().mkString(","))

如果不做处理的话，每个Action函数执行时都会执行一遍rdd.map(x => x+1) ，消耗很大。

Spark提供rdd的persist()函数来解决这个重复计算的问题，persist()把需要重复使用的rdd存起来，这样仅第一个Action操作才会计算，其他Action操作不需要再计算。

当我们执行rdd的persist()时，计算出RDD 的节点会分别保存它们所求出的分区数据。如果一个有持久化数据的节点发生故障，Spark 会在需要用到缓存的数据时重算丢失的数据分区。

rdd的persist()有5种持久化级别，分别是：



来自org.apache.spark.storage.StorageLevel 的定义。

默认情况下persist() 会把数据以序列化的形式缓存在JVM 的堆空间中。

这样上方案例可以优化为：

val rdd1 = rdd.map(x => x+1)

rdd1.persist(StorageLevel.DISK\_ONLY)

println(rdd1.first())

println(rdd1.count())

println(rdd1.sum())

println(rdd1.collect().mkString(","))

rdd1.unpersist() //释放缓存，必须手工释放

如果觉得数据过于重要，怕存一份有风险，则可以存2份：

rdd1.persist(StorageLevel.*MEMORY\_ONLY\_2*)

**值得注意：**

如果要缓存的数据太多，内存中放不下，Spark 会自动利用最近最少使用（LRU）的缓存

策略把最老的分区从内存中移除。但是对于仅把数据存放在内存中的缓存级别，下一次要用到已经被移除的分区时，这些分区就需要重新计算。

不必担心你的作业因为缓存了太多数据而被打断。

另外，如果*MEMORY\_ONLY 不足的时候，Spark会自动用硬盘来承载。*

**可能报错：**

Idea或Eclipse中执行Job报错：

System memory 259522560 must be at least 471859200（450M）

解决办法：

conf.set(**"spark.testing.memory"**, **"471859200"**)