大规模分布式系统第五次作业—— RDD&Streaming介绍&两表查询实践

16300200020 张言健

弹性分布式数据集

弹性分布式数据集（RDD）是spark重要概念，是一个可以并行操作的容错集合.

创建RDD的方法有两种，一种是并行化驱动现有的数据集合，另一种是引用外部的存储系统中的数据集，比如共享文件系统，HDFS，HBase或者其他Hadoop InputFormat的任何数据源。

其中的过程分为**Transformations**和**Actions**两种

Transformation

spark中数据进行整合的过程。常见的操作为map, filter, union, intersection, groupByKey 等一些系列数据整合的过程。Transformation是惰性的，他们不会立即返回计算结果，只会在Actions需要Transformation的结果的时候进行计算。这样的设计的原因应该是减少不必要的内存消耗。

Actions

spark中使用数据获得结果的函数行为。常见的操作有reduce，collect，count, first, sort等。spark代码中必须要有actions才能进行计算获得结果。

Transformations和Actions的组合可以使得Spark更加有效地运行，比如，我们可以使用map来创建一个数据集，然后直接用reduce返回最后的运算结果，这样就没有必要返回一个更加大的map操作后地数据集。

默认情况下，每次进行actions时，都可以重新计算每个转换后的RDD。但是，我们也可以使用persist的方法来在内存中保留RDD，在这种情况下，Spark会在集群中保留元素，以便下次更好地访问。

Spark Streaming

Spark Streaming是Spark核心API的拓展，用于实现数据流可拓展、高吞吐量、容错流的流处理。它的数据可以从许多来源中提取，举例如下。并且他们可以使用高级函数表示的的复杂算法进行处理，比如map, reduce, join, window，处理后地数据可以推送达文件系统，数据库或者可视化模块中。

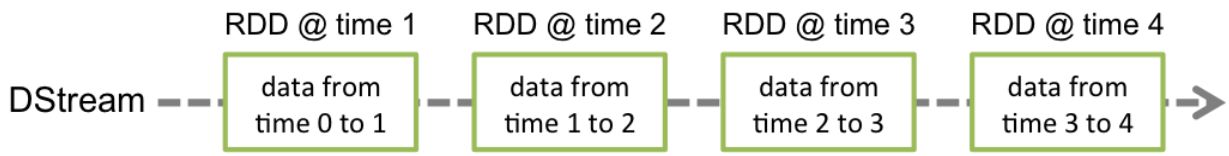
在内部，它的工作原理如下，Spark Streaming接收实时输入的数据流并将数据分成batch，然后又Spark引擎分batch处理生成最终结果流。

使用过程一般如下

1. .定义上下文`new StreamingContext`。
2. 通过创建输入DStreams来定义输入源。
3. 通过将转换和输出操作应用于DStream来定义流式计算。
4. 开始接收数据并使用它进行处理`streamingContext.start()`。
5. 等待处理停止（手动或由于任何错误）使用`streamingContext.awaitTermination()`。
6. 可以使用手动停止处理`streamingContext.stop()`。

离散流

离散流式Spark Streaming提供的基本的数据抽象，由于每个RDD都来自特定的时间间隔的数据，事实上每个RDD内包含的信息可以用三元组（start，end，data）来表示。这些底层的RDD的转换由Spark引擎来计算。 DStream操作隐藏了大部分细节，并为开发人员提供了更高级的API以方便使用。



Spark 两表查询

给定两张数据表，如下所示：

学号	语文成绩	数学成绩
101	99	25
102	56	68
103	74	88
104	36	75
105	88	44
106	65	78
107	44	99
108	78	65
109	60	76
110	100	47
111	54	88

学号	性别	身高
101	F	180
102	M	176
103	M	164
104	F	170
105	F	158
106	F	178
107	M	169
108	M	165
109	F	176
110	F	187
111	M	166

将以上两张表存入HDFS中，编写Spark应用程序，该程序对两张表进行联合统计，计算得到男性数学得分top3、男性身高高于175且语文及格的同学总数、男性数学及格的同学中最高身高同学的学号、高于170的女性同学的平均总分、男性同学平均总分、女性同学平均总分。

```
# 初始化 SPARK SQL
from __future__ import print_function,division
from pyspark import SparkConf,SparkContext
```

```
from pyspark.sql import SparkSession
from itertools import islice
from pyspark.sql import Row
# from pyspark.sql import HiveContext, Row

import re
conf = SparkConf().setAppName("Table").setMaster("local")
# hc = HiveContext(sc)
sc = SparkContext(conf=conf)
data1 = sc.textFile('hdfs://localhost:9001/user/form1.csv').map(lambda x :
re.split(",", x)).map(lambda x: Row(std = x[0], chinese = x[1], math = x[2]))
data2 = sc.textFile('hdfs://localhost:9001/user/form2.csv').map(lambda x :
re.split(",", x)).map(lambda x: Row(std = x[0], gender = x[1], height = x[2]))
spark = SparkSession(sc)
df1 = spark.createDataFrame(data1)
df2 = spark.createDataFrame(data2)
df1.createOrReplaceTempView("table1")
df2.createOrReplaceTempView("table2")
df1.show()
df2.show()
```

result:

```
+-------+----+---+
|chinese|math|std|
+-------+----+---+
|     99|  25|101|
|     56|  68|102|
|     74|  88|103|
|     36|  75|104|
|     88|  44|105|
|     65|  78|106|
|     44|  99|107|
|     78|  65|108|
|     60|  76|109|
|    100|  47|110|
|     54|  88|111|
+-------+----+---+

+------+------+---+
|gender|height|std|
+------+------+---+
|     F|   180|101|
|     M|   176|102|
|     M|   164|103|
|     F|   170|104|
|     F|   158|105|
|     F|   178|106|
|     M|   169|107|
|     M|   165|108|
|     F|   176|109|
|     F|   187|110|
|     M|   166|111|
```

```
+------+------+---+
```

Q1.

```
# 男性数学得分top3
result1 = spark.sql("SELECT TABLE1.STD, TABLE1.MATH,TABLE2.GENDER FROM TABLE1 INNER
JOIN TABLE2 ON TABLE1.STD = TABLE2.STD WHERE TABLE2.GENDER = 'M'  ORDER BY TABLE1.MATH
DESC")
result1.show(3)
```

result:

```
+---+----+------+
|STD|MATH|GENDER|
+---+----+------+
|107|  99|     M|
|103|  88|     M|
|111|  88|     M|
+---+----+------+
only showing top 3 rows
```

Q2.

```
# 男性身高高于175且语文及格的同学总数
result2 = spark.sql("SELECT TABLE1.STD, TABLE1.CHINESE, TABLE2.HEIGHT, TABLE2.GENDER
FROM TABLE1 INNER JOIN TABLE2 ON TABLE1.STD = TABLE2.STD WHERE TABLE2.GENDER = 'M' AND
TABLE1.CHINESE > 60 AND TABLE2.HEIGHT > 175 ")
result2.count()
```

result

```
0
```

Q3.

```
# 男性数学及格的同学中最高身高同学的学号
result3 = spark.sql("SELECT TABLE1.STD, TABLE1.MATH, TABLE2.HEIGHT, TABLE2.GENDER FROM
TABLE1 INNER JOIN TABLE2 ON TABLE1.STD = TABLE2.STD WHERE TABLE2.GENDER = 'M' AND
TABLE1.MATH > 60 ORDER BY TABLE2.HEIGHT DESC")
result3.show(1)
```

result

```
+---+----+------+------+
|STD|MATH|HEIGHT|GENDER|
+---+----+------+------+
|102|  68|   176|     M|
+---+----+------+------+
only showing top 1 row
```

Q4.

```
# 高于170的女性同学的平均总分
result4 = spark.sql("SELECT AVG(TABLE1.MATH + TABLE1.CHINESE) FROM TABLE1 INNER JOIN
TABLE2 ON TABLE1.STD = TABLE2.STD WHERE TABLE2.GENDER = 'F' AND TABLE2.HEIGHT > 170")
result4.show()
```

result

```
+-----------------------------------------------------+
|avg((CAST(MATH AS DOUBLE) + CAST(CHINESE AS DOUBLE)))|
+-----------------------------------------------------+
|                                                137.5|
+-----------------------------------------------------+
```

Q5.

```
# 男性同学的平均总分
result5 = spark.sql("SELECT AVG(TABLE1.MATH + TABLE1.CHINESE) FROM TABLE1 INNER JOIN
TABLE2 ON TABLE1.STD = TABLE2.STD WHERE TABLE2.GENDER = 'M'")
result5.show()
```

result

```
+-----------------------------------------------------+
|avg((CAST(MATH AS DOUBLE) + CAST(CHINESE AS DOUBLE)))|
+-----------------------------------------------------+
|                                                142.8|
+-----------------------------------------------------+
```

Q6.

```
# 女性同学的平均总分
result6 = spark.sql("SELECT AVG(TABLE1.MATH + TABLE1.CHINESE) FROM TABLE1 INNER JOIN
TABLE2 ON TABLE1.STD = TABLE2.STD WHERE TABLE2.GENDER = 'F'")
result6.show()
```

result

```
+-----------------------------------------------------+
|avg((CAST(MATH AS DOUBLE) + CAST(CHINESE AS DOUBLE)))|
+-----------------------------------------------------+
|                                 132.16666666666666|
+-----------------------------------------------------+
```