17 | Spark Streaming：从批处理走向流处理

今天，我们还是从**系统架构、流的描述、流的处理、流的状态、消息处理可靠性**这五个方面对 Spark Streaming 进行分析和讲解。

Spark Streaming

说到 Spark Streaming，还得从 Spark 谈起。如今在大数据的世界里，Spark 早已是众所周知的大数据处理和分析框架。Spark 在其诞生之初，由于采用内存计算和 DAG 简化处理流程的原因，使得大数据处理性能得到显著提升，一下子就将传统大数据批处理框架 Hadoop MapReduce 比了下去，取而代之成为大数据领域最耀眼的明星框架。

后来随着流计算技术的兴起，Spark 在批处理领域取得巨大成功之后，也开始将其触角延伸到流计算领域，于是诞生了 Spark Streaming。Spark Streaming 是一种建立在 Spark 批处理技术上的流计算框架，它提供了可扩展、高吞吐和错误容忍的流数据处理功能。

我们从 Spark Streaming 的系统架构中，就能看到 Spark Streaming 流计算技术和 Spark 批处理技术，这两者之间一脉相承的关系。

系统架构

下面的图 1 描述了 Spark Streaming 的工作原理。

图 1 Spark Streaming将流数据切分为块数据后进行处理

从上面的图 1 可以看出，当 Spark Streaming 接收到流数据时，先是将其切分成一个个的 RDD（Resilient Distributed Datasets，弹性分布式数据集），每个 RDD 实际是一个小的块数据。然后，这些 RDD 块数据再由 Spark 引擎进行各种处理。最后，处理完的结果同样是以一个个的 RDD 块数据依次输出。

所以，**Spark Streaming 本质上是将流数据分成一段段块数据后，进行连续不断地批处理**。

流的描述

接下来，我们就来看看在 Spark Streaming 中如何描述一个流计算过程。

由于 Spark Streaming 是构建在 Spark 之上，而 Spark 的核心是一个针对 RDD 块数据做批处理的执行引擎。所以 Spark Streaming 在描述流时，采用了“模版”的概念。具体如下图 2 所示。

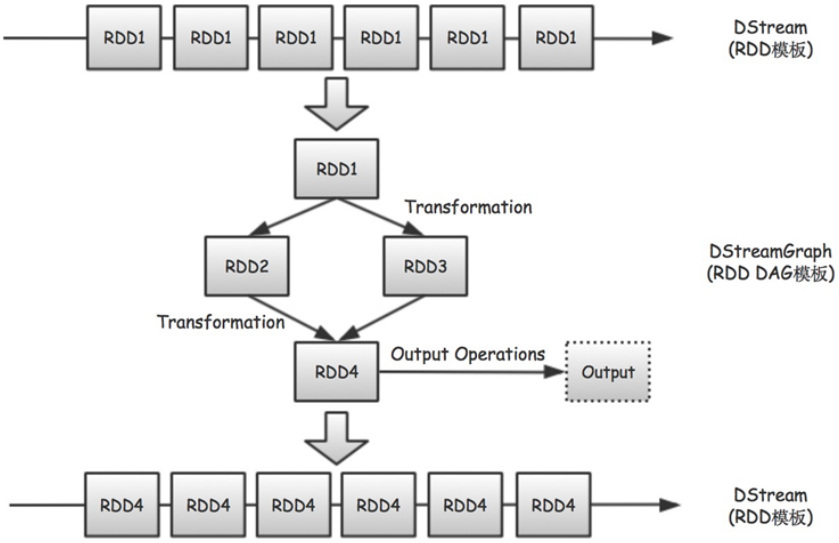

图 2　Spark Streaming 采用“模板”描述流计算过程

上面的图 2 说明了 Spark Streaming 是如何描述流计算过程的，具体如下：

- 首先是 RDD。它是 Spark 引擎的核心概念，代表了一个数据集合，是 Spark 进行数据处理的计算单元。
- 然后是 DStream。它是 Spark Streaming 对流的抽象，代表了连续数据流。**在 Spark Streaming 系统内部，DStream 是同一类 RDD 的模板**，因而我们也可以将 DStream 视为由同一类 RDD 组成的序列，每个 RDD 代表了一段间隔内的流数据。
- 然后是 Transformation。它代表了 Spark Streaming 对 DStream（也就是同一类 RDD） 的处理逻辑。目前， DStream 提供了很多 Transformation 相关 API，包括 map、flatMap、filter、reduce、union、join、transform 和 updateStateByKey 等。通过这些 API，可以对 DStream 做各种转换，从而将一个数据流变为另一个数据流。
- 接着是 DStreamGraph。它是 Spark Streaming 对流计算过程的描述，也就是 DAG。**在 Spark Streaming 系统内部，DStreamGraph 代表了 RDD 处理过程 DAG 的模板**。DStream 通过 Transformation 可以连接成 DStreamGraph，这就相当于用“点”和“线”画成 DAG 的过程。
- 最后是 Output Operations。它是 Spark Streaming 将 DStream 输出到控制台、数据库或文件系统等外部系统的操作。目前，DStream 支持的 Output Operations 包括 print、saveAsTextFiles、saveAsObjectFiles、saveAsHadoopFiles 和 foreachRDD。由于这些操作会触发外部系统访问，所以 DStream 各种转化的执行实际上是由这些操作触发的。

从上面 Spark Streaming 的概念可以看出，对应到我们在模块二中讨论过的流计算框架组成，DStreamGraph 对应了 DAG，DStream 相当于 DAG 中的“队列”， 而 Transformation 和 Output Operations 则相当于执行任务的“节点”。

流的处理

接下来，我们再来看 Spark Streaming 中的流是怎么被处理的。与 Storm 类似，我们从**流的输入、流的处理、流的输出和反向压力四个方面**来讨论。

首先是**流的输入**。Spark Streaming 提供了三种创建输入数据流的方式。

- 一是基础数据源，通过 StreamingContext 的相关 API，直接构建输入数据流。这类 API 通常是从 socket、文件或内存中构建输入数据流，比如 socketTextStream、textFileStream、queueStream 等。
- 二是高级数据源，通过外部工具类从 Kafka、Flume、Kinesis 等消息中间件或消息源构建输入数据流。
- 三是自定义数据源，当用户实现了 org.apache.spark.streaming.receiver 抽象类时，就可以实现一个自定义的数据源了。

由于 Spark Streaming 是用 DStream 来表示数据流，所以输入数据流也表示为 DStream。下面的代码（本课时完整代码请参考这里）演示了从 TCP 连接中构建文本数据输入流的过程。

```
SparkConf conf = new SparkConf().setMaster("local[2]").setAppName("WordCountExample");
JavaStreamingContext jssc = new JavaStreamingContext(conf, Durations.seconds(1));
JavaReceiverInputDStream<String> lines = jssc.socketTextStream("localhost", 9999);
```

在上面的代码中，我们用 socketTextStream 创建了一个从本地 9999 端口接收文本数据的输入流。

然后是**流的处理**。Spark Streaming 对流的处理，是通过 DStream 的各种转化操作 API 完成。DStream 的转换操作大体上包含了三类。

- **第一类是常用的流式处理操作**，比如 map、filter、reduce、count、transform 等。
- **第二类是流数据状态相关的操作**，比如 union、join、cogroup、window 等。
- **第三类则是流信息状态相关的操作**，目前有 updateStateByKey 和 mapWithState。

下面是一个对 DStream 进行转化处理的例子。

```
// 将每一行分割成单词，然后统计单词出现次数
JavaDStream<String> words = lines.flatMap(x -> Arrays.asList(x.split(" ")).iterator());
JavaPairDStream<String, Integer> pairs = words.mapToPair(s -> new Tuple2<>(s, 1));
JavaPairDStream<String, Integer> wordCounts = pairs.reduceByKey((i1, i2) -> i1 + i2);
```

在上面的代码中，先将从 socket 中读出的文本流 lines ，对每行文本分词后，用 flatMap 转化为单词流 words。然后用 mapToPair 将单词流 words 转化为计数元组流 pairs。最后，以单词为分组进行数量统计，通过 reduceByKey 转化为单词计数流 wordCounts。

接下来是**流的输出**。Spark Streaming 允许 DStream 输出到外部系统，这是通过 DStream 的各种输出操作完成的。DStream 的输出操作可以将数据输出到控制台、文件系统或数据库等。

目前 DStream 的输出操作有 print、saveAsTextFiles、saveAsHadoopFiles 和 foreachRDD 等。其中 foreachRDD 是一个通用的 DStream 输出接口，用户可以通过 foreachRDD 自定义各种 Spark Streaming 输出方式。下面的例子演示了将单词计数流打印到控制台。

```
wordCounts.print();
```

最后是**反向压力**。早期版本 Spark 不支持反向压力，但从 Spark 1.5 版本开始，Spark Streaming 也引入了反向压力功能，这是不是正说明了反向压力功能对流计算系统的必要性！默认情况下 Spark Streaming 的反向压力功能是关闭的。当要使用反向压力功能时，需要将 spark.streaming.backpressure.enabled 设置为 true。

整体而言，Spark 的反向压力借鉴了工业控制中 PID 控制器的思路，其工作原理如下。

- 首先，当 Spark 在处理完每批数据时，统计每批数据的处理结束时间、处理时延、等待时延、处理消息数等信息。
- 然后，根据统计信息估计处理速度，并将这个估计值通知给数据生产者。
- 最后，数据生产者根据估计出的处理速度，动态调整生产速度，最终使得生产速度与处理速度相匹配。

流的状态

接下来，我们再来看 Spark Streaming 中流的状态问题。Spark Streaming 关于流的状态管理，也是在 DStream 提供的转化操作中实现的。

我们先来看**流数据状态**。由于 DStream 本身就是将数据流分成 RDD 做批处理，所以 Spark Streaming 天然就需要对数据进行缓存和状态管理。换言之，组成 DStream 的一个个 RDD，就是一种流数据状态。DStream 提供了一些 window 相关的 API，实现了对流数据的窗口管理，并基于窗口实现了 count 和 reduce 这两类聚合功能。另外，DStream 还提供了union、join 和 cogroup 三种在多个流之间进行关联操作的 API。以上窗口和关联操作相关的 API，也属于 Spark 对流数据状态的支持。

说完流数据状态，我们再来看**流信息状态**。DStream 的 updateStateByKey 和 mapWithState 操作提供了流信息状态管理的方法。updateStateByKey 和 mapWithState 都可以基于 key 来记录历史信息，并在新的数据到来时，对这些信息进行更新。

不同的是，updateStateByKey 会返回记录的所有历史信息，而 mapWithState 只会返回处理当前一批数据时更新的信息。就好像，前者是在返回一个完整的直方图，而后者则只返回直方图中发生变化的柱条。

由此可见， mapWithState 比 updateStateByKey 的性能会优越很多。而且从功能上讲，如果不是用于报表生成的场景，大多数实时流计算应用中，使用 mapWithState 也会更合适。

消息处理可靠性

最后，我们来看下 Spark Streaming 中消息处理可靠性的问题。Spark Streaming 对消息可靠性的保证，是由数据接收、数据处理和数据输出共同决定的。从 1.2 版本开始，Spark 引入 WAL（write ahead logs）机制，可以将接收的数据先保存到错误容忍的存储上。当打开 WAL 机制后，再配合可靠的数据接收器（比如 Kafka），Spark Streaming 能够提供“至少一次”的消息接收。从 1.3 版本开始，Spark 又引入了 Kafka Direct API，进而可以实现“精确一次”的消息接收。

由于 Spark Streaming 对数据的处理是基于 RDD 完成，而 RDD 提供了“精确一次”的消息处理。所以在数据处理部分，Spark Streaming 天然具备“精确一次”的消息可靠性保证。

但是，Spark Streaming 的数据输出部分目前只具备“至少一次”的可靠性保证。也就是说，经过处理后的数据，可能会被多次输出到外部系统。

在一些场景下，这个不会有什么问题。比如输出数据是保存到文件系统，重复发送的结果只是覆盖掉之前写过一遍的数据。

但是在另一些场景下，比如需要根据输出增量更新数据库，那就需要做一些额外的去重处理了。一种可行的方法是，在各个 RDD 中新增一个唯一标志符来表示这批数据，然后在写入数据库时，使用这个唯一标志符来检查数据之前是否写入过。当然，这时写入数据库的动作需要使用事务来保证完整性了。

小结

总的来说，Spark Streaming 作为一种**从批处理框架发展而来的流计算框架**，代表了大数据领域的**两种趋势**。一是，**部分原本用批处理完成的任务，正逐渐转向由流处理来完成**。二是，这种 **“流”“批”一体的框架越来越成为主流**，毕竟用一个框架完成两种不同类型的计算任务，会极大地简化系统的复杂程度。

但是，Spark Streaming 这种将流数据按批处理的方式，还是存在一定的问题。问题主要是，Spark Streaming 将数据从流数据划分为 RDD 块数据时，时间间隔不能设置得太短，目前比较典型的最小时间间隔是 100 毫秒。这就意味着，Spark Streaming 处理流数据的时间延迟，最少会是 100 毫秒。

这对于实时要求更加严苛的场景是不适用的，同时这也意味着 Spark Streaming 并不能按逐条的方式，对流数据进行处理。这点需要你在实际开发过程中尤其注意，看你的业务场景是否可以接受 100 毫秒以上的延迟，以及是否不需要逐事件处理。

最后留一个小作业，如何分别使用 Spark Streaming 的流数据状态 API 和流信息状态 API 实现“过去 24 小时同一种商品的交易总金额”这种计算呢？可以将你的想法和问题在留言区写下来，我看到后会进行分析和解答。

下面是本课时的脑图，以帮助你理解。


流计算框架 Spark Streaming
分析流计算框架的整体思路
系统架构：理解一个流计算平台的设计架构，是使用这个流计算平台的基础
流的描述：用于描述流计算过程的 DAG 和相关的 API 接口
流的处理：与流的处理过程相关的 API，以及是否支持反向压力等
流的状态：流数据状态和流信息状态
消息处理可靠性：流计算系统对消息传递的保证如何，尽力而为 best-effort、至少一次 at-least-once、精确一次 exactly-once
系统架构
Spark Streaming 将流数据切分成一个个的 RDD 块数据
由 Spark 引擎对 RDD 块数据依次进行批次处理
处理完的结果同样以一个个的 RDD 块数据依次输出
流的描述
RDD 是 Spark 引擎的核心概念，代表了一个数据集合，是 Spark 进行数据处理的计算单元
Spark Streaming 代表了数据流，是由一系列的 RDD 构成，每个 RDD 代表了一段间隔内的流数据
Transformation 代表了对 DStream 的处理逻辑，包括 map、flatMap、filter、reduce、union、join、transform 和 updateStateByKey 等
DStreamGraph 是 Spark Streaming 对流计算过程的描述，也就是 DAG。DStream 通过 Transformation 连接成 DStreamGraph
Output Operations 是将 DStream 输出到控制台、数据库或文件系统等外部系统中的操作
流的处理
流的输入：包括基础数据源、高级数据源和自定义数据源共三种
流的处理：流式处理操作、流数据状态相关的操作、流信息状态相关的操作
流的输出：包括 print、saveAsTextFiles、saveAsHadoopFiles 和 foreachRDD 等
反向压力：通过将 spark.streaming.backpressure.enabled 设置为 true实现反向压力
流的状态
流数据状态：RDD本身、窗口和多流关联相关API
流信息状态：updateStateByKey 和 mapWithState
消息处理可靠性
Spark streaming 对消息可靠性的保证，是由数据接收、数据处理和数据输出共同决定的
在数据接受部分，使用 WAL 机制并配合可靠的数据接收器，Spark Streaming 能够提供“至少一次”和“精确一次”的可靠性保证
在数据处理部分，Spark Streaming 具备“精确一次”的消息可靠性保证
在数据输出部分，Spark Streaming 具备“至少一次”的可靠性保证