19 | Apache Flink：最惊艳的开源流计算框架

今天，我们来看第四种开源流计算框架 Apache Flink。我们继续从**系统架构、流的描述、流的处理、流的状态、消息处理可靠性**这五个方面来进行分析和讲解。

Apache Flink

在流计算技术不断发展的今天，有关流计算的理论和模型已经构建得比较清晰和完善。Apache Flink（后简称 Flink） 就是这些流计算领域最新理论和模型的优秀实践。相比 Spark 在批处理领域的火爆流行，Flink 可以说是目前流计算领域最耀眼的明星框架了。

Flink 和 Spark 都是“流”“批”一体的分布式计算框架，但不同于**Spark 的核心是“批处理”引擎，Flink 的核心是“流计算”引擎**。

系统架构

我们先来看 Flink 的系统架构。

Flink 是一个主从（Master/Worker）架构的分布式系统。主节点负责调度流计算作业，管理和监控任务执行。当主节点从客户端接收到作业相关的 JAR 包和资源后，进行分析和优化，生成执行计划，也就是需要执行的任务，然后将相关的任务分配给各个 Worker，由 Worker 负责任务的具体执行。

图 1 展示了 Flink 的系统架构。

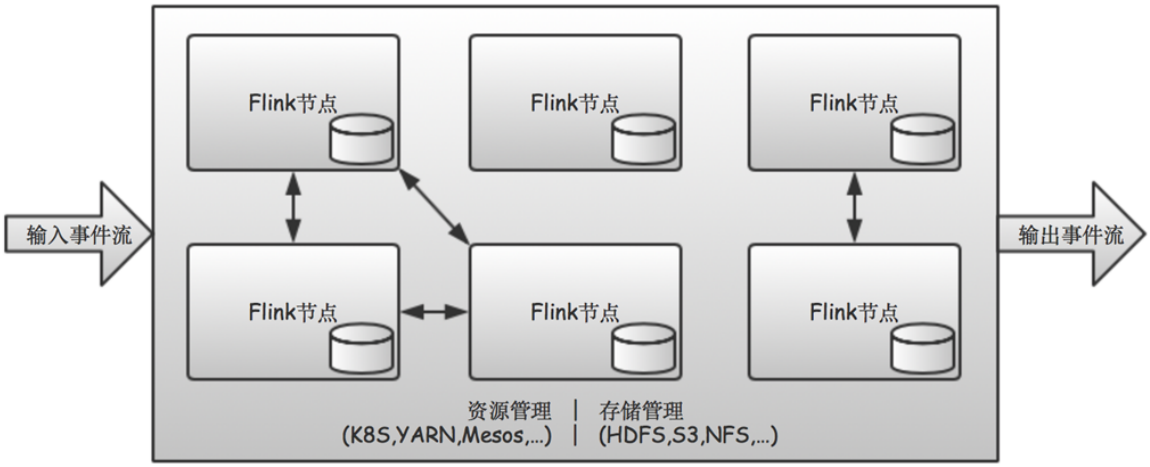

图 1 Flink系统架构图

可以看出，Flink 可以部署在诸如 Yarn、Mesos 和 K8S 等分布式资源管理器上。其整体架构与 Storm 和 Spark Streaming 等分布式流计算框架类似，但与这些流计算框架不同的是，**Flink 明确地把状态管理（尤其是流信息状态管理）纳入到了其系统架构中**。

在 Flink 计算节点执行任务的过程中，可以将状态保存到本地。通过 checkpoint 机制，再配合诸如 HDFS、S3 和 NFS 这样的分布式文件系统，Flink 在不降低性能的同时，实现了状态的分布式管理。

流的描述

接下来，我们再来看看在 Flink 中如何描述一个流计算过程。下面的图 2 展示了 Flink 用于描述流计算过程的 DAG 组成。

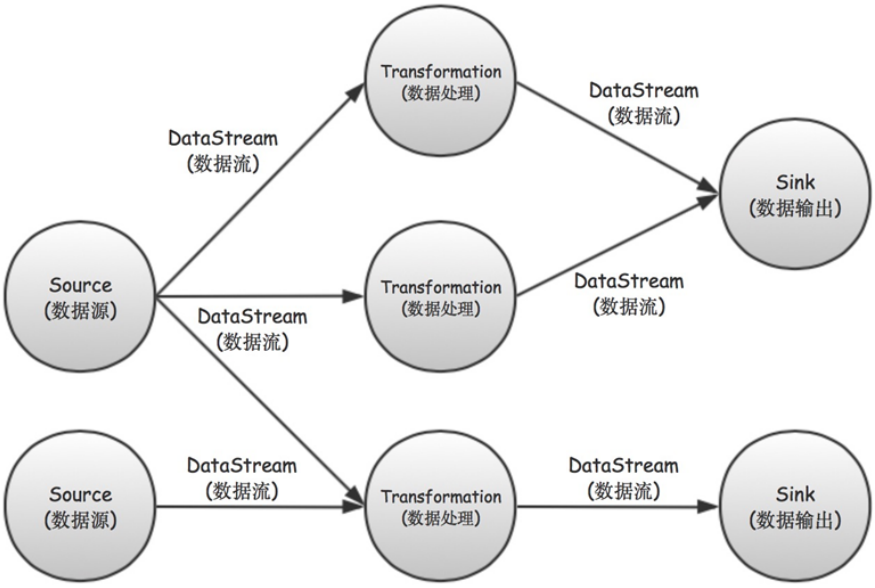

图 2 Flink 描述流计算过程的 DAG 组成

从图 2 可以看出，Flink 使用了 DataStream 来描述数据流。Flink 的数据输入、数据处理和数据输出均与 DataStream 有关，具体来说是这样的。

- 首先是**Source**。 它用于描述 Flink 流数据的输入源。当流数据输入 Flink 系统后，就被表示为 DataStream。Flink 的 Source 可以是消息中间件、数据库、文件系统或其他各种数据源。
- 然后是**Transformation**。 Transformation 将一个或多个 DataStream 转化为一个新的DataStream，是 Flink 对流数据进行处理的地方。目前，Flink 提供 map、flatMap、filter、keyBy、reduce、fold、aggregations、window、union、join、split、select 和 iterate 等类型的 Transformation。
- 最后是**Sink**。Sink 是 Flink 将 DataStream 输出到外部系统的地方，比如写入控制台、数据库、文件系统或消息中间件等。

总的来说，DataStream 在 Flink 中扮演的角色犹如 Spark 中的 RDD，它们都是各自框架中的核心概念。通过对 DataStream 进行各种 Transformation 操作，就形成了描述 Flink 流计算过程的 DAG。

另外值得一提的是，Flink 也支持批处理 DataSet 的概念。不过 Flink 的 DataSet 内部同样是由 DataStream 构成。Flink 这种**将批处理视为流处理**特殊情况的做法，与 Spark Streaming 中**将流处理视为连续批处理**的做法**截然相反**。

流的处理

接下来，我们再来看 Flink 中的流是怎么被处理的。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 Flink 中流的处理过程。

1. 流的输入

Flink 使用 StreamExecutionEnvironment.addSource 设置流的数据源 Source。为了使用方便，Flink 在 StreamExecutionEnvironment.addSource 的基础上提供了一些内置的数据源实现。

这些数据源主要包含了四类。

- 一是，基于文件的输入。从文件中读入数据作为流数据源，比如 readTextFile 和 readFile 等。
- 二是，基于套接字的输入。从 TCP 套接字中读入数据作为流数据源，比如 socketTextStream 等。
- 三是，基于集合的输入。用集合作为流数据源，比如 fromCollection、fromElements、fromParallelCollection 和 generateSequence 等。
- 四是，自定义输入。StreamExecutionEnvironment.addSource 是最通用的流数据源生成方法，用户可以其为基础开发自己的流数据源。一些第三方数据源，比如 flink-connector-kafka 中的 FlinkKafkaConsumer08 就是针对 Kafka 消息中间件开发的流数据源。

Flink 将从数据源读出的数据流表示为 DataStream。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。

```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
DataStream<String> text = env.socketTextStream("localhost", 9999, "\n");
```

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。

2. 流的处理

Flink 对流的处理，是通过 DataStream 的各种转化操作完成的。相比 Spark 中 DStream 的转化操作混淆了流数据状态管理和流信息状态管理，**Flink 的设计思路更加清晰，明确地将流信息状态从流数据状态的管理中分离出去**。

DataStream 转化操作只包含了两类操作，一类是常规的流式处理操作，比如 map、filter、reduce、count、transform 等。另一类是流数据状态相关的操作，比如 union、join、coGroup、window 等。

这两类操作都是针对流本身的处理和管理。从设计模式中单一职责原则的角度来看，**Flink 关于流的设计显然更胜一筹**。

下面是一个对 DataStream 进行转化处理的例子。

```
DataStream<WordWithCount> windowCounts = text
    .flatMap(new FlatMapFunction<String, WordWithCount>() {
        @Override
        public void flatMap(String value, Collector<WordWithCount> out) {
            for (String word : value.split("\\s")) {
                out.collect(new WordWithCount(word, 1L));
            }
        }
    })
    .keyBy("word")
    .timeWindow(Time.seconds(5), Time.seconds(1))
    .reduce(new ReduceFunction<WordWithCount>() {
        @Override
        public WordWithCount reduce(WordWithCount a, WordWithCount b) {
            return new WordWithCount(a.word, a.count + b.count);
        }
    });
```

在上面的代码中，我们先将从 socket 中读出的文本流 text，用 flatMap 函数对每行文本进行分词，从而将文本流转化为了单词流。然后，我们使用 keyBy 函数，将单词流按照单词（也就是 word）进行分组，从而形成了单词分区流。最后，我们使用 timeWindow 函数，在单词分区流上以 5 秒钟为时间窗口，进行具有“求和”功能的 reduce 聚合计算，这样就得到了每五秒钟内各个单词出现的次数。

3. 流的输出

Flink 使用 DataStream.addSink 设置数据流的输出方法。同时 Flink 在 DataStream.addSink 的基础上提供了一些**内置的流数据输出实现**。

这些流数据输出实现主要包含了四类：

- 一是，输出到文件系统。将流数据输出到文件系统，比如 writeAsText、writeAsCsv 和 writeUsingOutputFormat。
- 二是，输出到控制台。将流数据打印到控制台，比如 print 和 printToErr。
- 三是，输出到套接字。将流数据输出到 TCP 套接字，比如 writeToSocket。

- 四是，自定义输出。DataStream.addSink 是最通用的流数据输出方法，用户可以其为基础开发自己的流数据输出方法。比如 flink-connector-kafka 中的 FlinkKafkaProducer011 就是针对 Kafka 消息中间件开发的流数据输出方法。

下面的示例演示了将 DataStream 表示的流数据打印到控制台。

```
windowCounts.print().setParallelism(1);
```

将上面输入、处理和输出三个部分的代码片段整合起来，就能实现一个具备单词计数功能的 Flink 流计算应用了。

4. 反向压力

Flink 对反向压力的支持非常好，不仅实现了反向压力功能，还直接内置了反向压力的监控功能。Flink 采用容量有限的分布式阻塞队列来进行数据传递，当下游任务从队列中读取消息过慢时，就非常自然地减慢了上游任务往队列中写入消息的速度。这种反向压力的实现思路，和我们在前面 03 课时中使用 JDK 自带的 BlockingQueue 实现反向压力的方法基本一致。

另外值得一提的是，与 Storm 和 Spark Streaming 需要明确打开开关才能使用反向压力功能不一样的是，**Flink 的反向压力功能是天然地包含在了其数据传送方案内的，不需特别再去实现，使用时也无须特别打开开关**。

流的状态

接下来，我们再来看 Flink 中流的状态问题。

Flink 是第一个明确地**将流信息状态管理从流数据状态管理中剥离开来**的流计算框架。大多数其他的流计算框架要么没有流信息状态管理，要么实现的流信息状态管理非常有限，要么流信息状态管理混淆在了流数据状态管理中，使用起来并不方便和明晰。

在**流数据状态**方面，Flink 有关流数据状态的管理，都集中在 DataStream 的转化操作上。这是非常合理的，因为流数据状态管理本身是属于流转化和管理的一部分。比如，流按窗口分块处理、多流的合并、事件乱序处理等功能的实现，虽然也涉及数据缓存和状态操作，但这些功能原本就应该由流计算框架来处理。

DataStream 上与**窗口管理**相关的 API 包括 Window 和 WindowAll。其中 Window 是针对KeyedStream，而 WindowAll 是针对非 KeyedStream。DataStream 基于窗口提供了一系列聚合计算相关的方法，比如 reduce、fold、sum、min、max 和 apply 等。另外，DataStream 还提供了一系列有关流与流之间关联计算的操作，比如 union、join、coGroup 和 connect 等。

除了以上这些对流数据状态的支持方法外，Flink 还提供了非常有特色的 KeyedStream。所谓 KeyedStream 是指将流按照指定的键值（key 值），在逻辑上分成多个独立的流。这些逻辑流在计算时，状态彼此独立、互不影响，但是在物理上这些独立的流可能是合并在同一条物理的数据流中。因此在 KeyedStream 具体实现时，Flink 会在处理每个消息前，将当前运行时上下文切换到键值（key 值）所指定流的上下文，就和切换线程上下文一样，这样优雅地避免了不同逻辑流在运算时的相互干扰。

在**流信息状态**方面，Flink 对流信息状态管理的支持，是它相比当前其他流计算框架更显优势的地方。Flink 在 DataStream 之外，提供了独立的状态管理接口。可以说，**实现流信息状态管理，并将其从对流数据本身的管理中分离出来，是 Flink 在洞悉流计算本质后的明智之举**。因为，**如果说 DataStream 是对数据在时间维度的管理，那么状态接口其实是对数据在空间维度的管理**。Flink 之前的流数据框架对这两个概念的区分可以说并不是非常明确，这也导致它们关于状态的设计不是非常完善，甚至根本没有。

在 Flink 中有两种类型的状态接口：Keyed State 和 Operator State。它们既可以用于流信息状态管理，也可以用于流数据状态管理。

我们先来看**Keyed State**。Keyed State 与 KeyedStream 相关。前面提到，KeyedStream 是对流按照 key 值做出的逻辑划分。每个逻辑流都有自己的上下文，就像每个线程都有自己的线程栈一样。当我们需要在逻辑流中记录一些状态信息时，就可以使用 Keyed State。比如“统计不同 IP 上出现的不同设备数”，就可以将流按照 IP 分成 KeyedStream，这样来自不同 IP 的设备事件，会分发到不同 IP 独有的逻辑流中。然后在逻辑流处理过程中，使用 Keyed State 来记录不同设备数。如此一来，就非常方便地实现了“统计不同 IP 上出现的不同设备数”的功能。

我们再来看**Operator State**。Operator State 与算子有关。其实与 Keyed State 的设计思路非常一致，Keyed State是按 key 值划分状态空间，而 Operator State 则是按照算子的并行度划分状态空间。每个 Operator State 绑定到算子的一个并行实例上，因而这些并行实例在运行时可以维护各自的状态。这有点像线程局部量，每个线程都维护自己的一个状态对象，在运行时互不影响。比如当 Kafka Consumer 在消费同一个 topic 的不同 partition 时，可以用 Operator State 来维护各自消费 partition 的 offset。

另外，从 Flink 1.6 版本开始，Flink 引入了状态生存时间值（State Time-To-Live），这为实际开发中，淘汰过期的状态提供了极大的便利。不过美中不足的是，Flink 虽然针对状态存储，提供了 TTL 机制，但是 TTL 本身实际是一种非常底层的功能。如果 Flink 能够针对状态管理也提供诸如窗口管理这样的功能，会使 Flink 的流信息状态管理更加完善和方便。

消息处理可靠性

最后，我们来看下 Flink 中消息处理可靠性的问题。

Flink 基于 snapshot 和 checkpoint 的故障恢复机制，在 Flink 内部提供了 exactly-once 级别的消息处理可靠性保证。当然，**得到这个保证的前提是，在 Flink 应用中保存状态时必须使用 Flink 内部的状态机制，比如 Keyed State 和 Operator State**。

因为这些 Flink 内部状态的保存和恢复都是包含在 Flink 的故障机制内，由 Flink 系统保证了状态的一致性。如果使用不包含在 Flink 故障恢复机制内的状态存储方案，比如用另外独立的 Redis 记录 PV/UV 统计状态，那么就不能获得 exactly-once 级别的消息处理可靠性保证，而只能得到 at-least-once 级别的可靠性保证了。

要想在 Flink 中，实现从流数据“输入”到“输出”这种“端到端”的 exactly-once 级别的消息处理可靠性保证，还必须有 Flink connectors 配合才行。不同的 Flink connectors 提供了不同级别的可靠性保证。比如在 Source 端，Apache Kafka 提供了 exactly once 保证，Twitter Streaming API 提供了 at most once 保证。在 Sink 端，HDFS rolling sink 提供了 exactly once 保证，Kafka producer 则只提供了 at least once 的保证。

小结：如何面对琳琅满目的流计算框架

今天，我们讨论了 Flink。至此，我们就讨论完了四种主流的开源流计算框架。

下面的表 1 是对模块四所讨论四种流计算框架的比较。

表1 四种开源流计算框架比较

比较项\流计算框架	Apache Storm	Spark Streaming	Apache Samza	Apache Flink
是否流批一体	仅流处理	流批一体	仅流处理	流批一体
是否支持 DAG	是	是	是	是
是否支持流式API（如filter、map、reduce等）	是	是	是	是
是否支持 SQL	是	是	是	是
处理延迟（越低越好）	低	中	低	低
窗口支持	滑动窗口、翻滚窗口	滑动窗口	翻滚窗口、会话窗口	翻滚窗口、滑动窗口 会话窗口、自定义窗口
是否支持消息乱序处理	是	是	否	是
是否原生支持 CEP	否	否	否	是
反向压力	支持	支持	不支持	支持
状态管理（越高越好）	中	中	较高	超高
消息处理可靠性	尽力而为、至少一次、限于Trident的精确一次	数据处理是“精确一次” 数据输出是“至少一次”	至少一次	精确一次、至少一次
上手难度（越低越容易上手）	中	低	中	低
适用场景	处理时延要求较高，但吞吐量不大的场景；只需要流处理而不用批处理的场景；部署和维护成本较高	处理时延要求不是非常严格的场景，比如可以接受秒级时延的业务；流批一体处理的场景；需要使用较多的内存资源	只要求“至少一次”级别消息处理可靠性的场景；只需要流处理而不用批处理的场景；能够部署 Kafka 的场景	处理时延要求较高、吞吐量大、可靠性要求较高的业务场景；**各种流计算场景都可以优先考虑 Flink**
入门是否推荐	否	是	否	是

从上面的表中可以看出，在流计算领域，Flink 在各个方面都更有优势，所以它不愧是当前最好的流计算框架。如果你是刚刚入门流计算领域的话，我最推荐的就是 Flink 了，其次是 Spark Streaming。至于 Storm 和 Samza，你可以了解一下，以帮助理解流计算概念和技术即可。

另外，除了上面四种开源流计算框架外，其实还有一些其他的流计算框架或平台，比如 Akka Streaming、Apache Beam 等，这些流计算框架也各有特色。比如 Akka Streaming 支持丰富灵动的流计算编程 API，可谓是惊艳。再比如 Apache Beam 则是流计算模式

的集大成者，可以在底层集成多种不同的流计算框架，试图一统流计算江湖。

既然有这么多的流计算框架，**那我们该如何面对琳琅满目的流计算框架呢**？我认为可以从两个角度来看待这个问题。

从**横向功能特征的角度**来看，其实所有流计算框架的核心概念都是相同的，我们在前面的模块一和模块二中已经详细地讲解了这些概念，比如 DAG、队列、线程、流数据状态、流信息状态、反向压力等。只要我们掌握了这些流计算中的核心概念，把握住了流计算框架中各种问题的关键所在，那么在面对这些流计算框架时，就不会感到眼花缭乱，乱了阵脚。

从**纵向发展历史的角度**来看，以 Flink 为代表的新一代流计算框架，在理论和实践上都已日趋完善和成熟。**当掌握了流计算中的核心概念后，不妨一开始就站在 Flink 这个巨人的肩膀上，开始在流计算领域的探索和实践**。而作为有希望统一流计算领域的 Apache Beam，实际上是构建在各种具体流计算框架上的更高一层统一编程模式，它对流计算中的各种概念和问题做出了总结，是我们追踪流计算领域最新进展的一个好切入点。

到这里为止，我们有关流计算的各种核心概念和对几种开源流计算框架的讲解就完成了。在后面的课程中，我将用 Flink 演示两个应用案例，来帮助你理解如何将流计算技术运用到具体的业务中。

最后留一个小作业，你知道 Flink 的分布式快照是怎么回事吗？它是如何帮助 Flink 实现 exactly-once 语义的呢？可以将你的想法或问题写在留言区。

下面是本课时的脑图，以帮助你理解。