Flink编程模型

2019-03-04 | 2021-04-14 | Flink | |

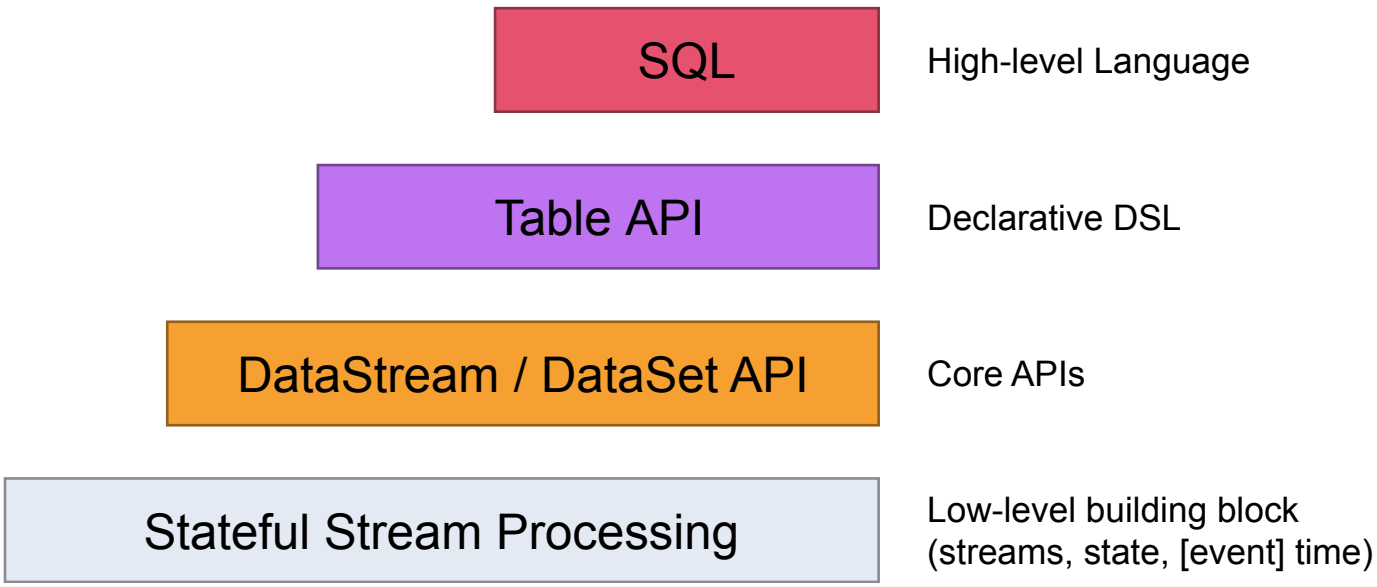
Flink 的编程模型为数据流编程模型（Dataflow Programming Model），这里介绍编程模型里面的几个概念。

本文译自Flink官网：

Apache Flink 1.7 Documentation: Dataflow Programming Model

数据流编程模型（Dataflow Programming Model）

抽象层次（Levels of Abstraction）



*Flink 提供几种不同层次的抽象来开发 **流/批（streaming/batch）**程序*

- 最低级的抽象仅提供**状态流（stateful streaming）**，它通过 Process Function （处理函数）内嵌在 DataStream API 中。它容许用户自由地处理来自一个或多个流的事件，并且使用一致的容错状态。此外，用户也可以给事件时间和处理时间注册回调，使得程序可以实现复杂的计算。

- 实践中，多数的应用程序不需要使用上述的低级的抽象，仅需要使用**核心接口（Core API）**来编码，比如 DataStream API（数据流接口，有界/无界流）和 DataSet API （数据集接口，有界数据集）。这些流畅的接口为数据处理提供了通用构建流程，诸如用户指定的转换（transformation）、连接（join）、聚合（aggregation）、窗口（window）、状态（state）等不同形式。这些接口处理的数据类型在不同的编程语言中以类（class）的形式呈现。

  低层次的*处理函数（Process Function）*与*数据流接口*（DataStream API）的交互，使得某些特定的操作可以抽象为更低的层次成为可能。*数据集接口（DataSet API）*在有界的数据集上提供额外的原始操作，例如循环和迭代（loops/iterations）。

- **表接口（Table API）**使以表为中心的声明性 DSL，可以动态地改变表（当展示流的时候）。Table API遵循（扩展）关系型模型：表附加了一个模式(schema)（类似于关系型数据库中的表），此API提供了可比较的操作，例如select，project，join，group-by，aggregate等。Table API程序以声明方式定义应该执行的逻辑操作，而不是准确地指定操作代码。 尽管Table API可以通过各种类型的用户定义函数进行扩展，但它的表现力不如Core API，但使用起来更简洁（编写的代码更少）。 此外，Table API程序还会通过优化程序，在执行之前应用优化规则。

  可以在表和DataStream/ DataSet之间无缝转换，允许在程序中混合Table API以及DataStream和DataSet API。

- Flink提供的最高级抽象是**SQL**。 这种抽象在语义和表达方面类似于Table API，但是将程序表示为SQL查询表达式。 SQL抽象与Table API紧密交互，SQL查询可以在Table API中定义的表上执行。

程序和数据流（Programs and Dataflows）

Flink程序的基本构建块是**流（streams）**和**转换（transformations）**。 （请注意，Flink的DataSet API中使用的DataSet也是内部流，稍后会详细介绍。）从概念上讲，*流*是（可能永无止境的）数据记录流，而*转换*是将一个或多个流作为输入，并产生一个或多个输出流的操作。

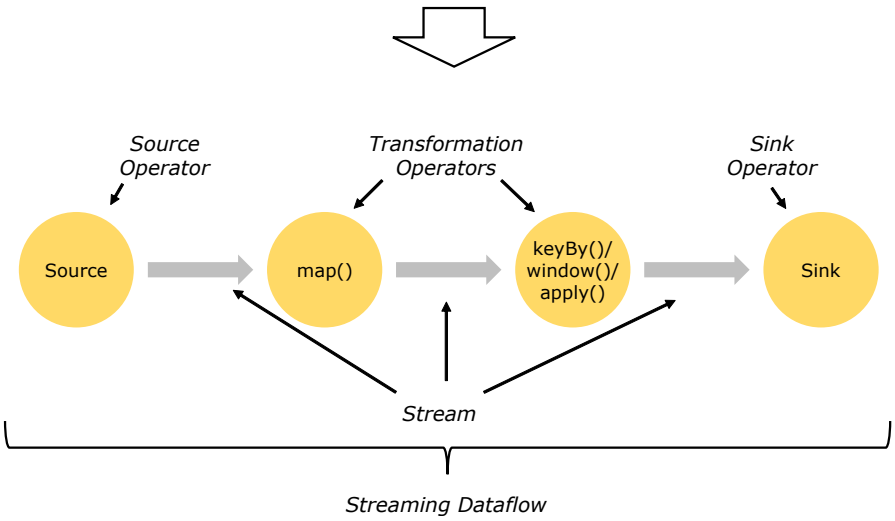
执行时，Flink程序映射到**流数据流（streaming dataflows）**，由**流(streams)和转换运算符(operators)组成。 每个数据流都以一个或多个源(sources)开头，并以一个或多个接收器(sinks)结束。 数据流类似于任意有向无环图（DAGs, Directed acyclic graphs）**。尽管通过*迭代结构*允许特殊形式的循环，但为了简单起见，我们将在大多数情况下对其进行掩饰简化。

```
DataStream<String> lines = env.addSource(                          } Source
                                new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));       } Transformation

DataStream<Statistics> stats = events                              }
        .keyBy("id")                                               } Transformation
        .timeWindow(Time.seconds(10))                              }
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));                              } Sink
```



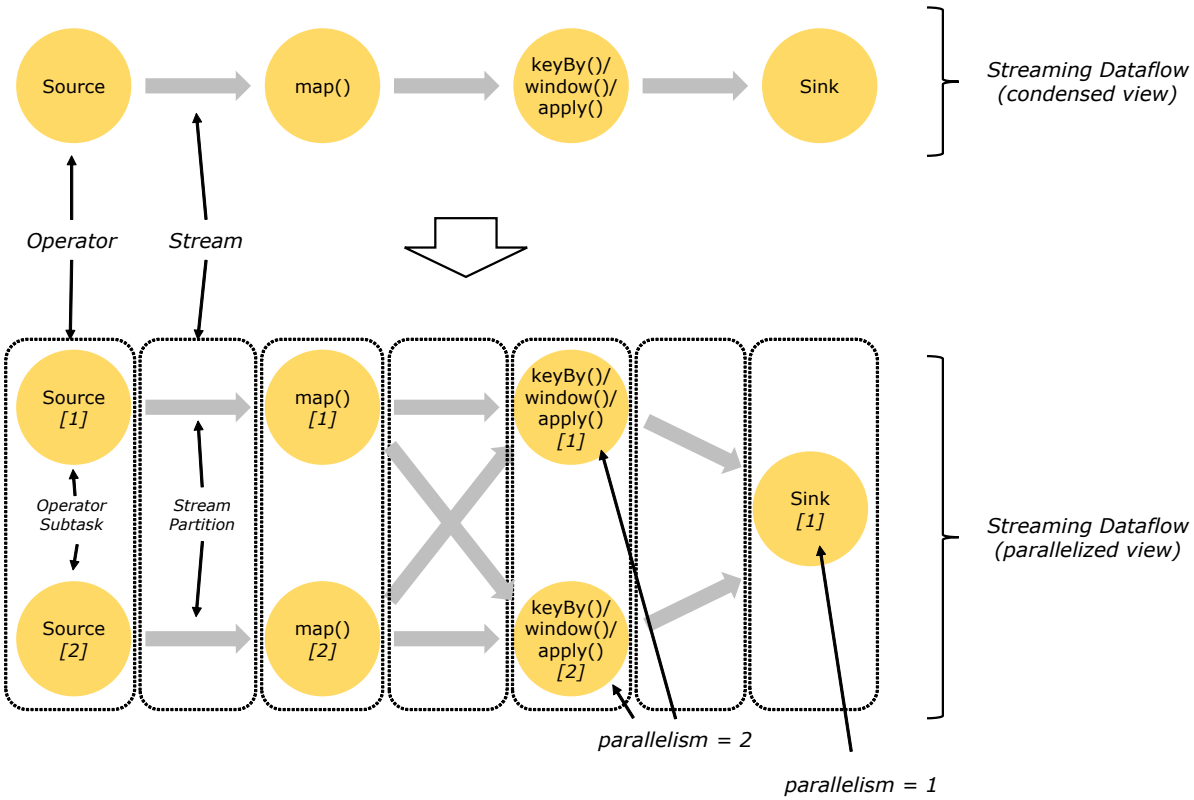
通常，程序中的转换与数据流中的运算符之间存在一对一的对应关系。 但是，有时一个转换可能包含多个转换运算符。

源（soruces）和接收器（sinks）被记录在流连接器和批处理连接器文档中。 转换（transformation）被记录在DataStream运算符和DataSet转换中。

并行数据流

Flink中的程序本质上是并行（parallel）和分布式的（distributed）。 在执行期间，*流*具有一个或多个*流分区（stream partitions）*，并且每个*运算符*具有一个或多个*运算子任务(operator subtasks)*。 运算子任务彼此独立，并且可以在不同的线程中执行，也可能是在不同的机器或容器上执行。

运算子任务的数量就是某个特定运算符的**并行度（parallelism）**。 流的并行度始终是其生成的运算符的并行度。 同一程序的不同运算符可能具有不同的并行级别。



流可以以*一对一（或转发）*的模式或以*重新分发*的模式在两个运算符之间传输数据：

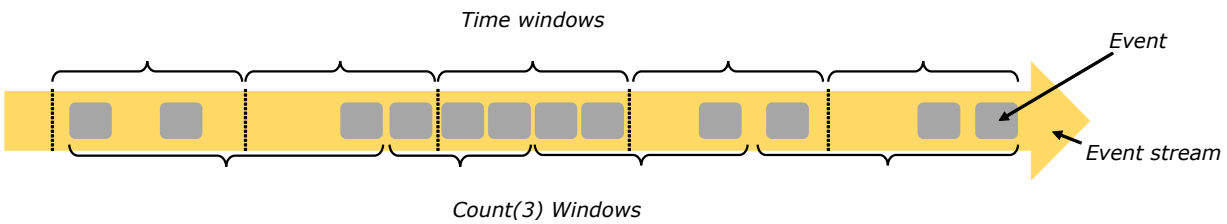
- **一对一（One-to-one）**流（例如，在上图中的*Source*和*map()运算符之间）保留元素的分区和排序。这意味着map()运算符的subtask[1]看到的元素与Source*运算符的subtask[1]生成的元素顺序相同。
- **重新分发（Redistributing）**流（在上面的*map()和keyBy/window*之间，以及*keyBy/window*和*Sink*之间）重新分配流的分区。每个*运算子任务*将数据发送到不同的目标子任务，具体取决于所选的转换。示例是*keyBy()（通过散列键重新分区），broadcast()或rebalance()（随机重新分区）。在重新分发*的交换中，元素之间的排序仅保留在每对发送和接收子任务中（例如，*map()*的subtask[1]和keyBy/window的subtask[2]）。因此，在此示例中，保留了每个键的排序，但并行度确实带来了不同键的聚合结果到达sink的顺序的不确定性。

有关配置和控制并行性的详细信息，请参阅并行执行的文档。

窗口（Windows）

聚合事件（如，counts，sums）在流上的工作方式与批处理方式不同。 例如，不可能计算流中的所有元素，因为流通常是无限的（无界）。 相反，流上的聚合（counts，sums等）由**窗口(windows)**限定，例如“在最后5分钟内计数”或“最后100个元素的总和”。

Windows可以是时间驱动的（例如：每30秒）或数据驱动（例如：每100个元素）。 人们通常区分不同类型的窗口，例如***翻滚窗口(tumbling windows)（***没有重叠***），滑动窗口(sliding windows)（***具有重叠***）和会话窗口(session windows)***（由不活动间隙打断）。

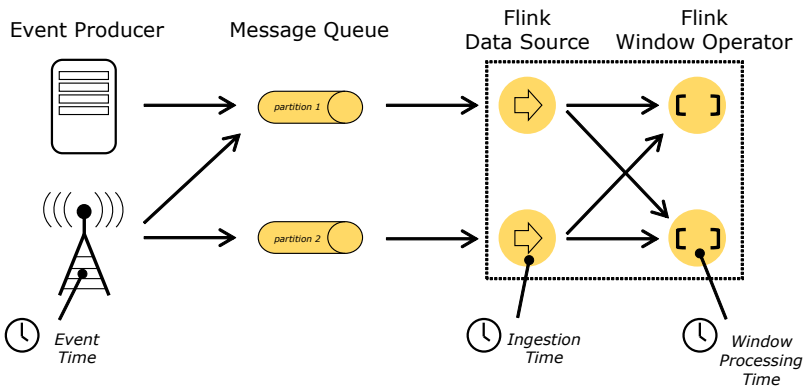


可以在此博客文章中找到更多窗口示例。 更多详细信息可参阅窗口文档。

时间（Time）

当在流程序中引用时间（例如定义窗口）时，可以参考不同的时间概念：

- **事件时间（Event Time）**是创建事件的时间。 它通常由事件中的时间戳描述，例如由生产传感器或生产服务生成。 Flink通过时间戳分配器（timestamp assigners）访问事件时间戳。
- **接收时间(Ingestion Time)**是事件在源操作符处进入Flink数据流的时间。
- **处理时间（Processing Time）**是每个操作符执行基于时间的操作时的本地时间。



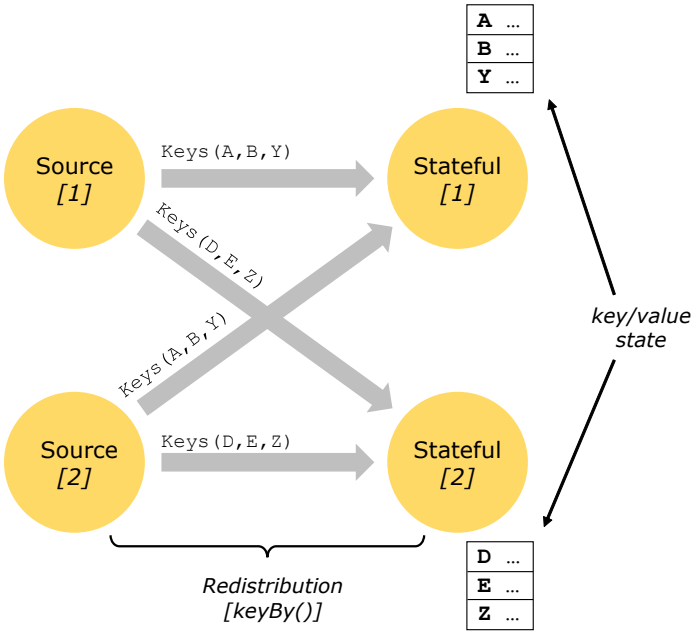
事件时间，接收时间和处理时间

有关如何处理时间的更多详细信息，请参阅事件时间文档。

状态运算（Stateful Operations）

虽然数据流中的许多运算只是一次查看一个单独的事件（例如事件解析器），但某些运算会记住多个事件（例如窗口运算符）的信息。这些操作称为**stateful**。

状态运算的状态可以被认为是由内嵌的键/值存储来维护。状态和状态运算符读取的流被严格地分区和分发。因此，只有在*keyBy()*函数之后才能在keyed stream上访问键/值状态，并且限制为与当前事件的键相关联的值。对齐流和状态的键可确保所有状态更新都是本地操作，从而保证一致性而无需事务开销。对齐操作还允许Flink重新分配状态并透明地调整流分区。



状态和分区

有关更多信息，请参阅有关状态的文档。

容错检查点(Checkpoints for Fault Tolerance)

Flink使用**stream replay**和**检查点(checkpointng)**的组合来实现容错。检查点与每个输入流中的特定点以及每个运算符的对应状态相关。通过恢复运算符的状态并从检查点重新执行（replay）事件，可以从检查点恢复流数据流并保持一致性（exactly-once processing semantics）。

检查点间隔是执行期间的容错和恢复时间（需要重放的事件的数量）之间的折衷方法。

容错的内部机制中的描述提供了有关Flink如何管理检查点和相关主题的更多信息。有关启用和配置检查点的详细信息，请参阅检查点API文档。

批处理流

Flink执行批处理程序作为流程序的一种特殊情况，即流是有界的（有限数量的元素）。 *DataSet*在内部被视为数据流。因此，上述概念以相同的方式应用于批处理程序，并且它们适用于流程序，除了少数例外：

- 批处理程序的容错不使用检查点。通过完全重新执行流来进行恢复，因为输入是有限的。这会使资源更多地用于恢复，且使得常规处理资源消耗更少，因为它避免了检查点。
- DataSet API中的有状态操作（stateful operations）使用简化的内存/核外(in-memory/out-of-core)数据结构，而不是键/值索引。
- DataSet API引入了特殊的同步（ superstep-based）迭代，这些迭代只能在有界流上进行。有关详细信息，请查看迭代文档。

Donate

> **Post author:** Jiezhi.G

Welcome to my other publishing channels

Telegram RSS

Apache # Flink # Big Data # Translate

分享学习大数据相关笔记，并邀请您加入

解决 macos systemuiserver 无响应的问题